User's Manual

32

Renesas Peripheral Driver Library

User's Manual

RX63T Group

Renesas Electronics
www.renesas.com

Rev.2.11 Sept 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.

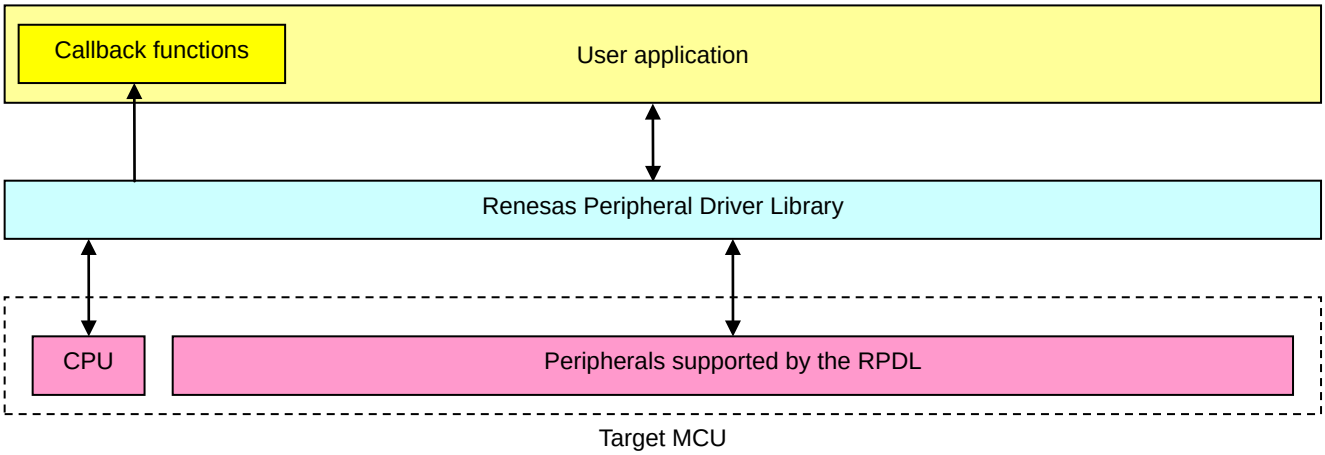


Figure 1.1: System configuration, with all peripherals supported by RPDL

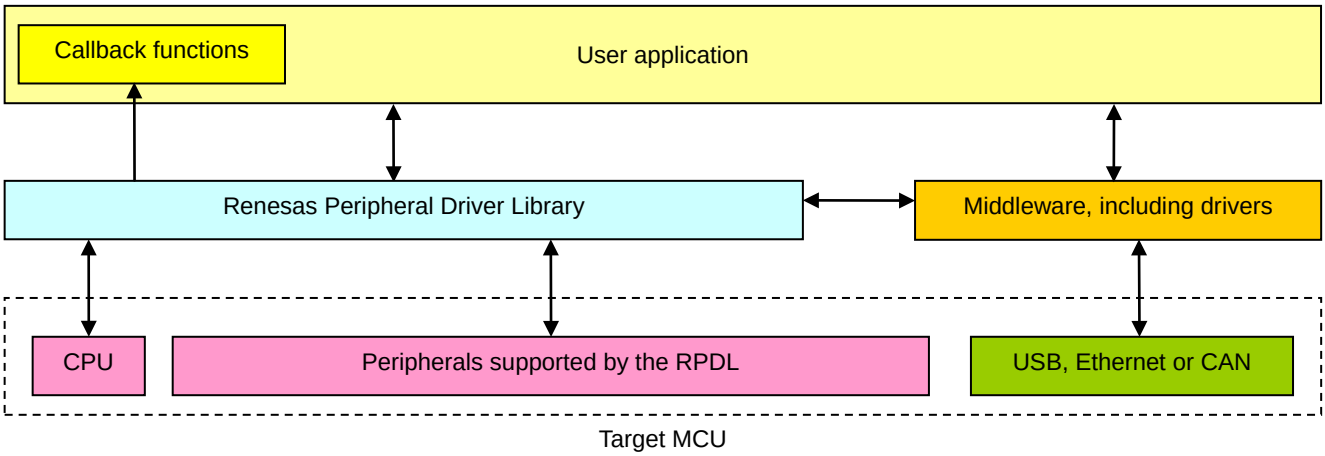


Figure 1.2: System configuration, with middleware taking direct control of some peripherals

The library is packaged as:

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

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

i. The hardware schematic diagram
ii. The RX63T 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 = rx600
-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.

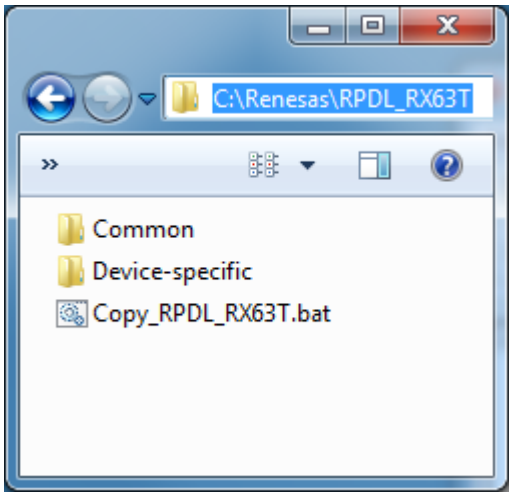
The instructions to follow for stand-alone use start are given below.

1) Unzip the RPDL files

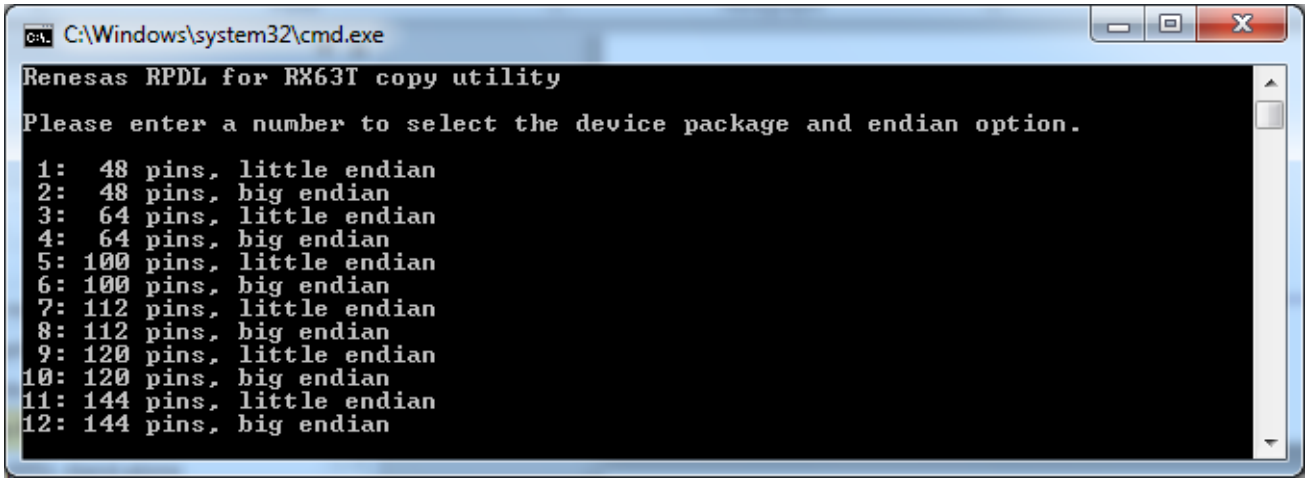
Double-click on the file RPDL_RX63T.exe to unpack the files.
The default location is C:\Renesas\RPDL_RX63T.

2) Copy the files into your project area

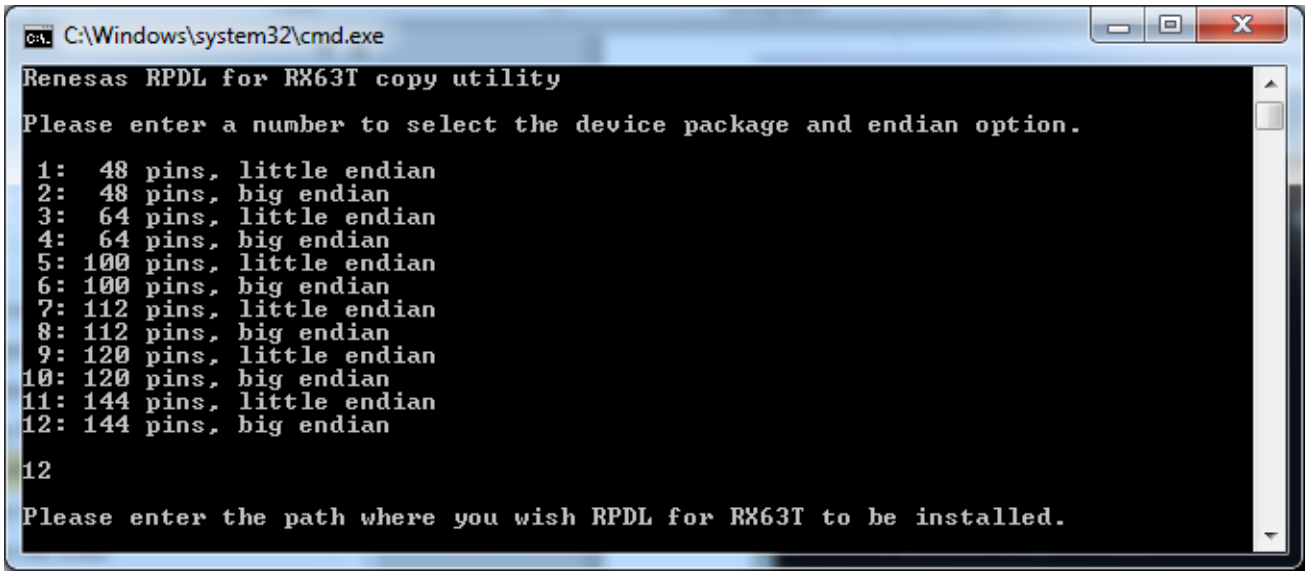
Navigate to where the RPDL files were unpacked.



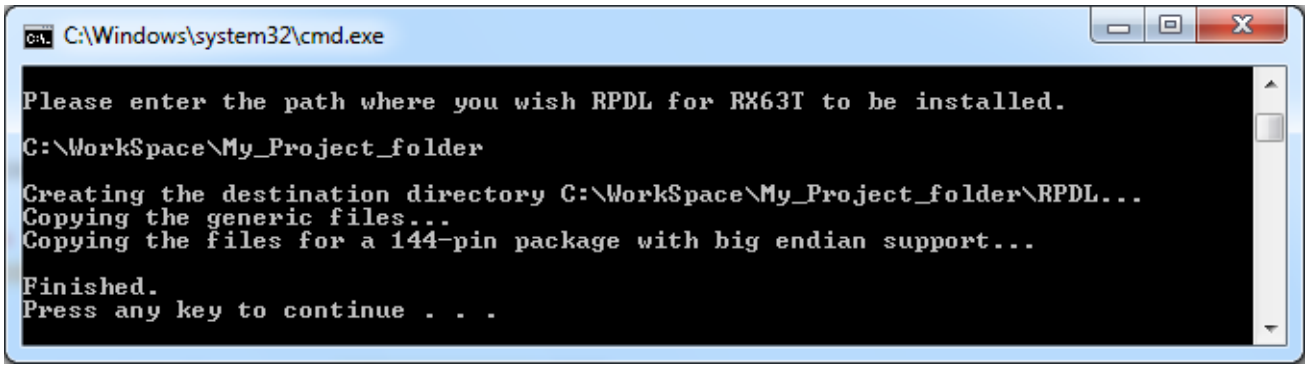
Double-click on "Copy_RPDL_RX63T.bat" to start the copy process.



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.
NOTE: To avoid a problem with long pathnames enclose the path in quotes.
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”)

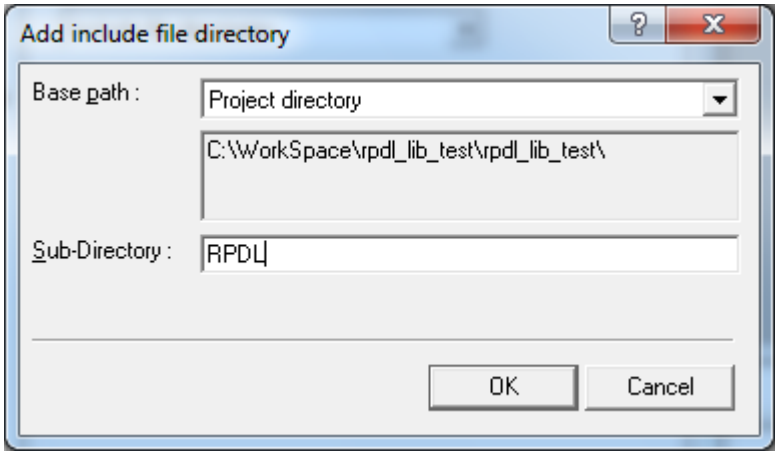
3) Include the new directory

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

Select the C/C++ tab.

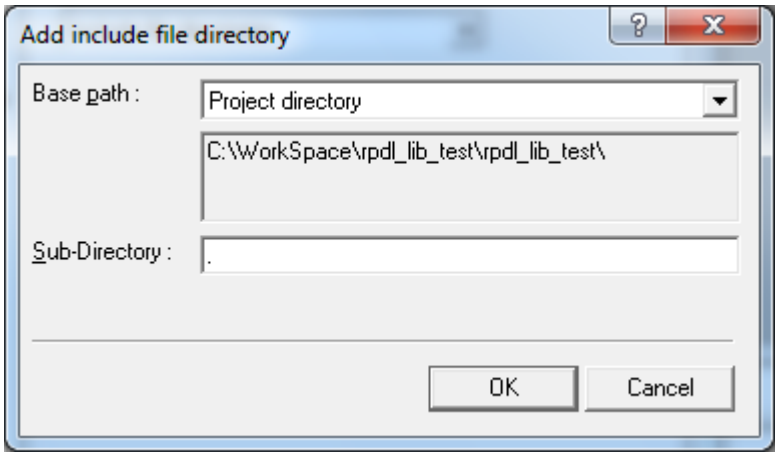
Use the key sequence S, I to show the included file directories.

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:



Click on “OK” to close the window.

4) Add the RPDL library file

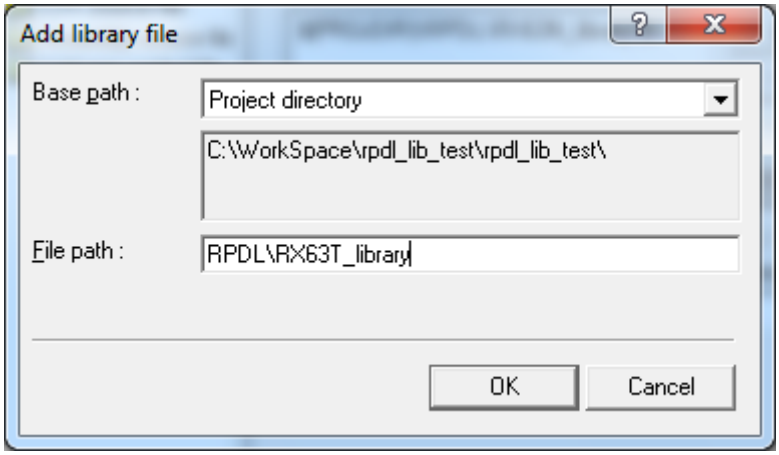
The library file is added to the list used by the linker application.

Select the Link/Library tab.

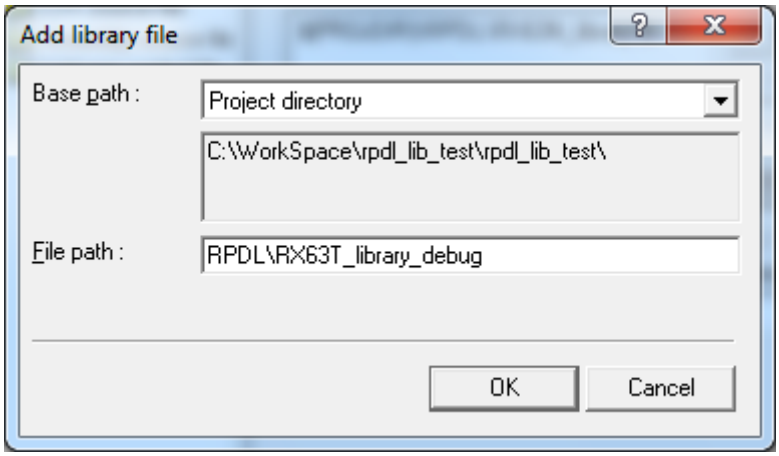
From the “Show entries for :” drop-down menu, select “Library files”.

Click on the “Add…” button.

In the “Add library file” window, select “Project directory” and enter “RPDL\RX63T_library” as the File path.



To use library with debug information, enter “RPDL\RX63T_library_debug” as the File path.



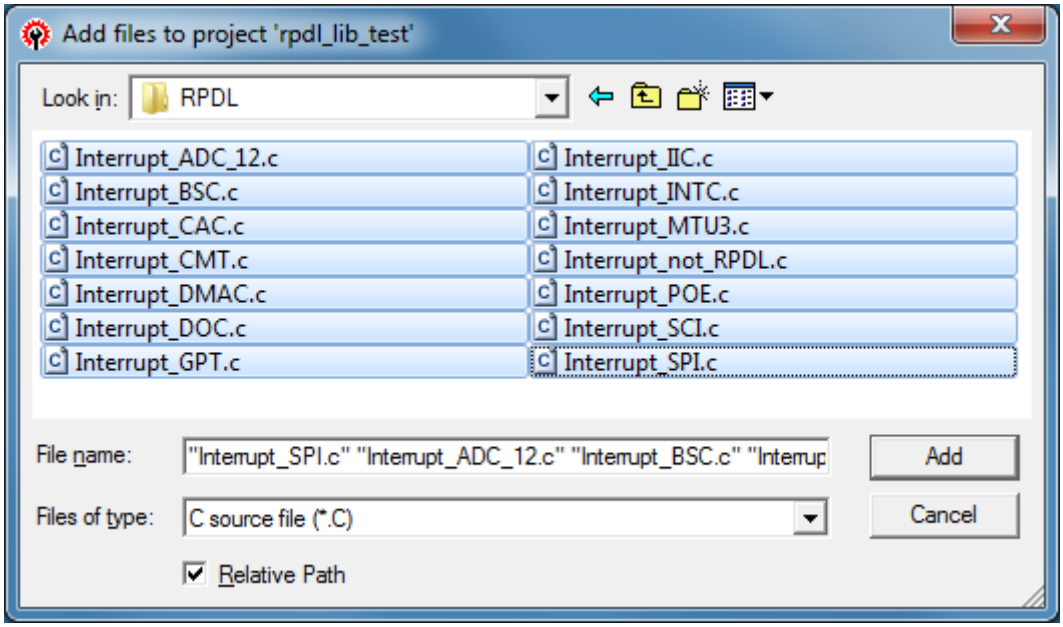
Click on “OK” to close the window.
Click on “OK” to return to the main HEW window.

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).

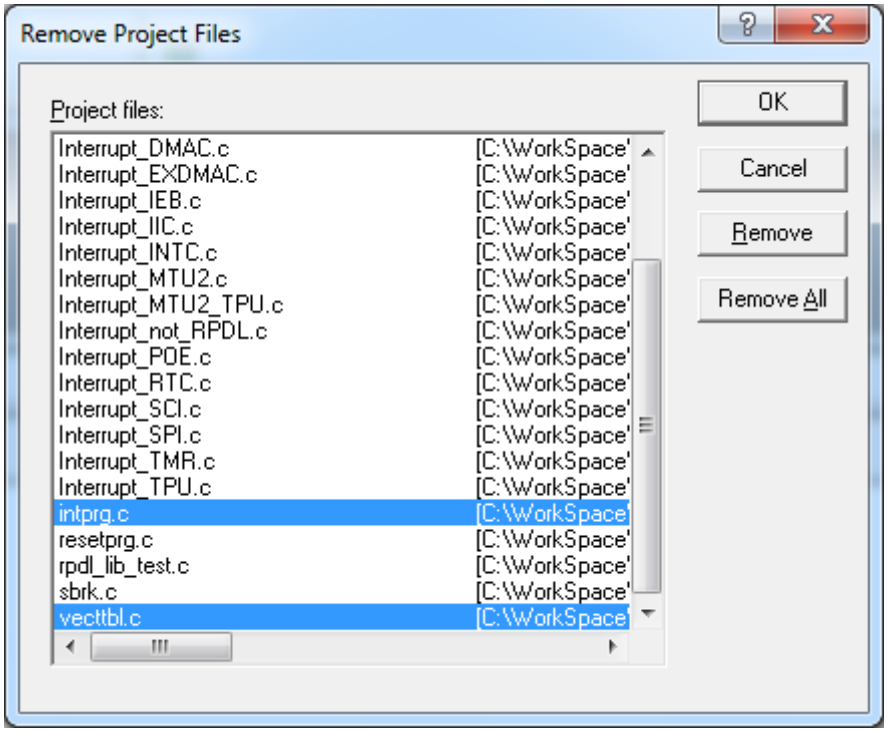
8) Avoid conflicts with standard project files

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.

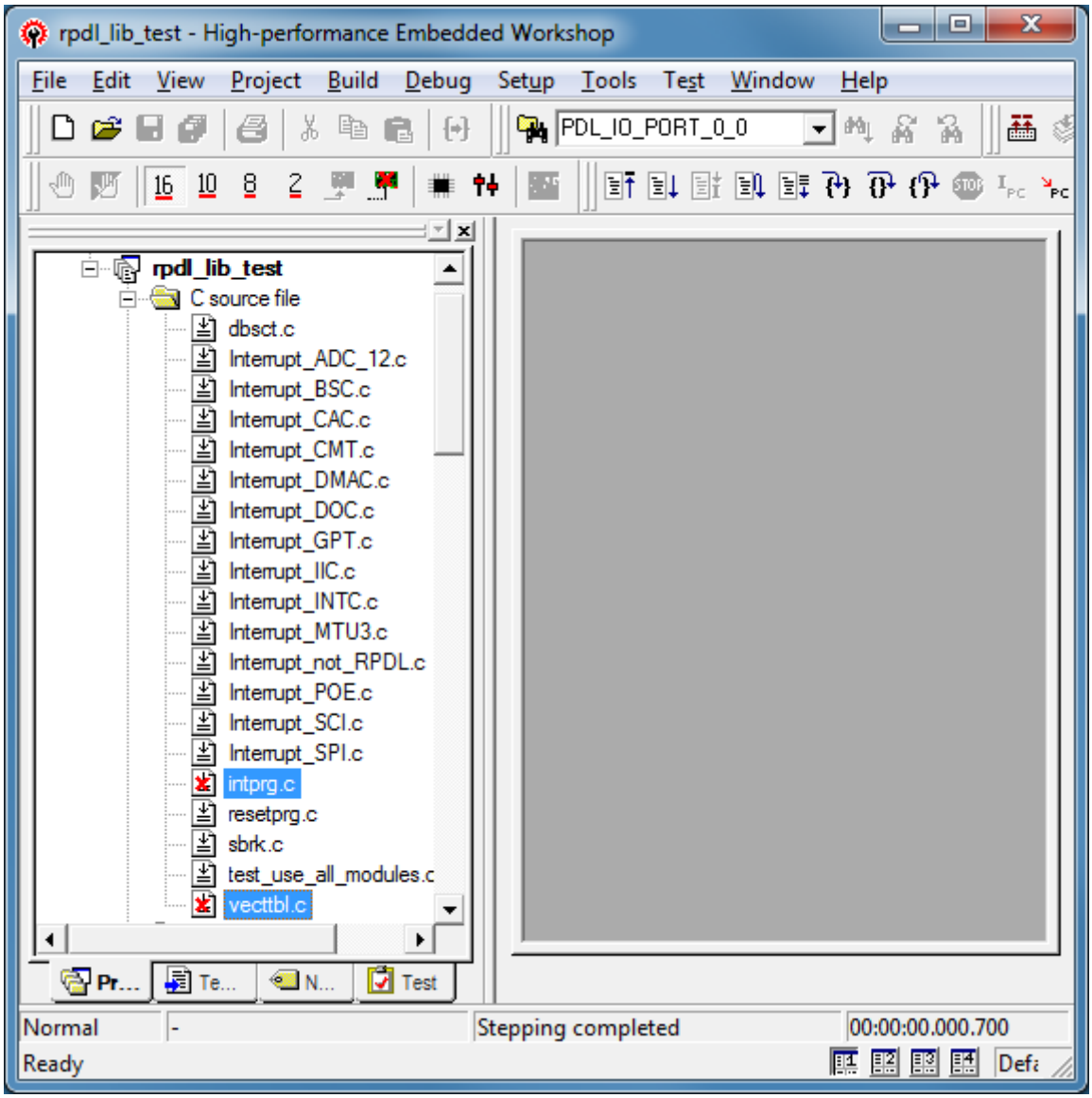


Figure 1.3: intprg.c and vecttbl.c have been excluded

9) Set the build options.

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

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

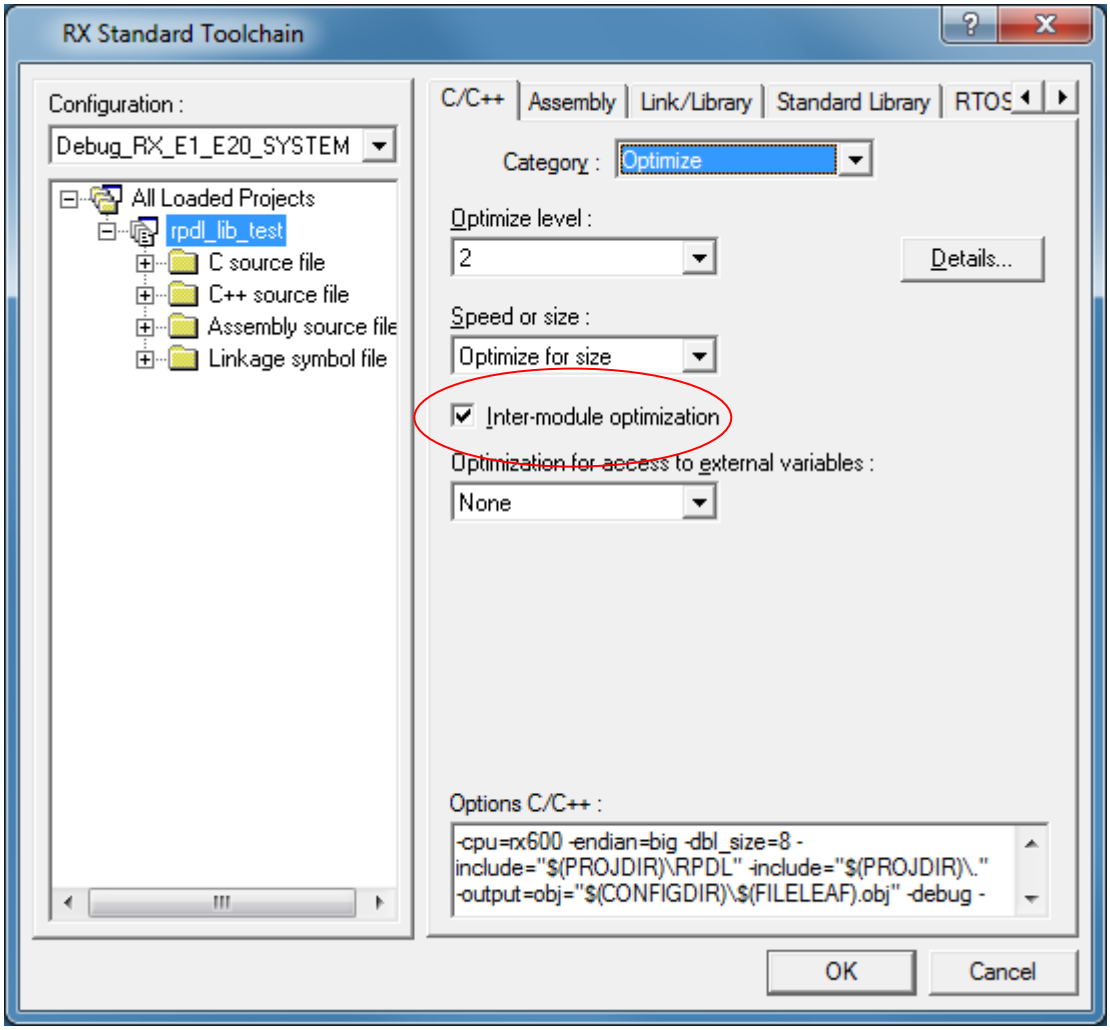
To avoid linking unused RPDL functions, adjust the Compiler and Linker settings.

(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.

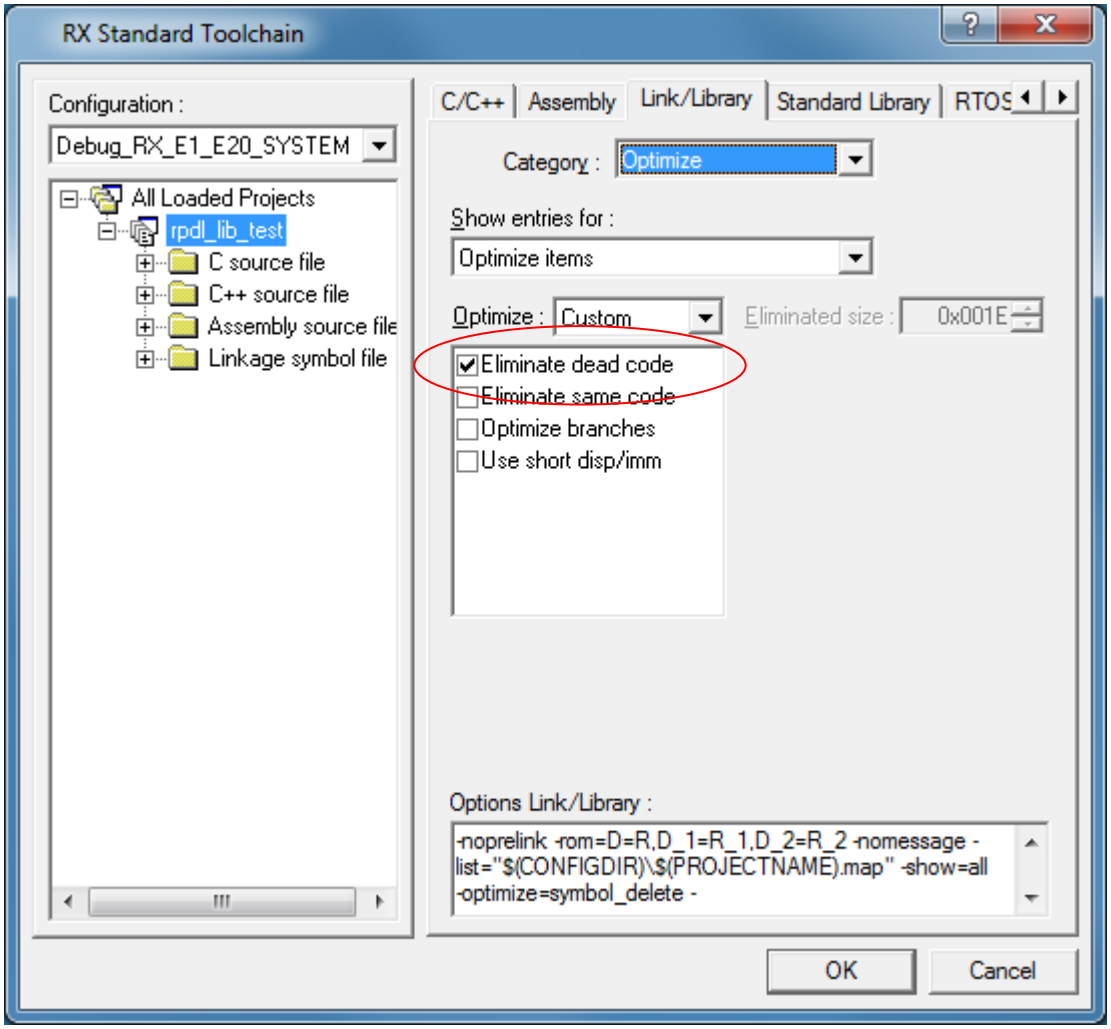


(ii) Linker

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.



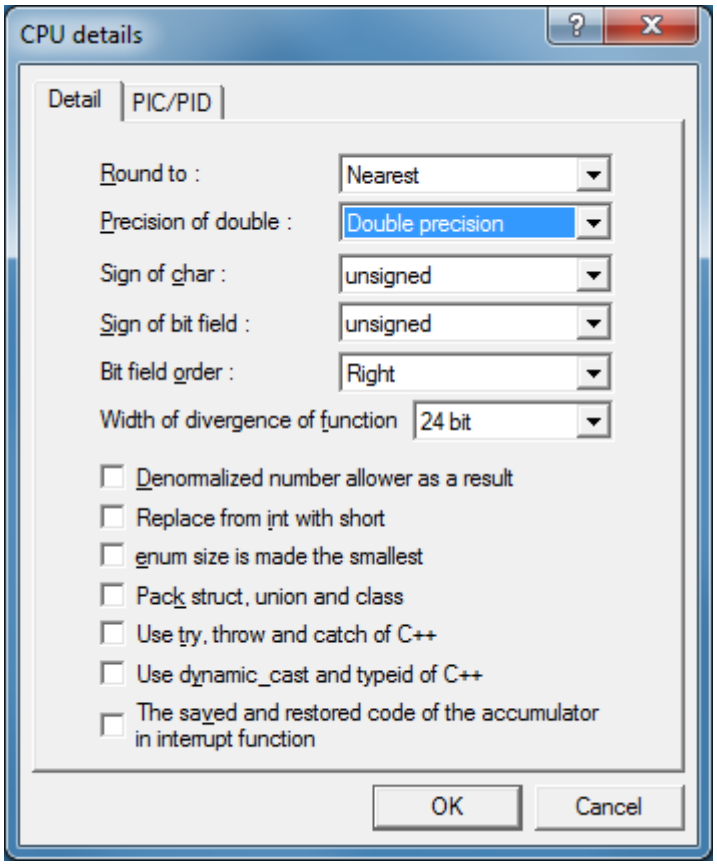
(b) Set the floating point precision

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

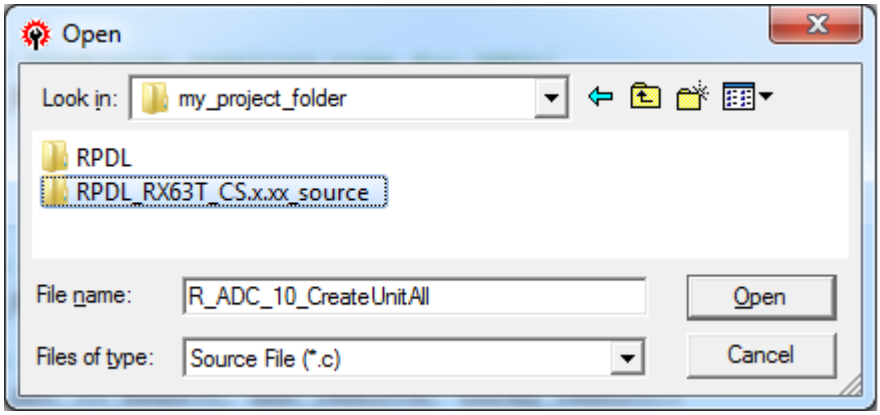
RPDL library with debug information should be chosen, in order to step in the RPDL source code for debugging.

Unzip the RPDL source zip file (e.g. “RPDL_RX63T_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 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();
```

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.

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. Acronyms and abbreviations

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
GPT	General PWM Timer
GSM	Global System for Mobile communications
HEW	High-performance Embedded Workbench
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
SDRAM	Synchronous Dynamic RAM
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) Port Function
These driver functions are used for configuring the I/O pin optional functions.

(5) MCU Operation
These driver functions are used for configuring the MCU operation.

(6) 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 controlling the clock frequency accuracy measurement circuit.

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

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

(10) Register Write Proctection
These driver functions are used for controlling the register right protection.

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

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

(15) General PWM Timer
These driver functions are used for configuring and controlling the timers.

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

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

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

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

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

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

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

(23) 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.

(24) 10-bit Analog to Digital Converter
These driver functions are used for configuring the 10-bit ADC units, controlling the units and reading the conversion results.

(25) 10-bit Digital to Analog converter
These driver functions are used for configuring the DAC module and setting the output voltages.

(26) Data Operation Circuit
These driver functions are used for configuring the data operation circuit.

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 and 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.
11. Configuring a group of interrupt sources.
12. Controlling a group of interrupt sources.
13. Reading the status of a group of interrupt sources.

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 and RAM.
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. Configuring the detection circuits for use, including:
    a. Setting voltage thresholds.
    b. Defining a voltage event.
    c. Setting up interrupts when a voltage event is detected.
    d. Configuring a reset when supply voltage drops below a voltage threshold.
2. Controlling the detection circuits.
3. Reading the status of the detection circuits.

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. Writing data to the backup memory area.
4. Reading data from the backup memory area.
5. Determining the cause of the exit from the lowest power mode.

2.11. 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 internal bus, providing the following operations.

1. Setting the internal and External bus operation.
2. Configuration of the controller.
3. Configuration of the eight address space areas.
4. Disabling an area that is not required.
5. Controlling the bus controller.
6. 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. Multi-Function Timer Pulse Unit Driver

The driver functions support the use of the eight 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.

2.16. 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.

2.17. General PWM Timer Driver

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

1. Selection of the GPT pins for use.
2. Configuration of a GPT unit (the IWDTCLK count module).
3. Configuration of a GPT channel, including
   - Access to all control bits.
   - Automatic interrupt control
4. Disabling channels that are no longer required and enabling low-power mode.
5. Control of a GPT channel.
6. Control of a GPT unit.
7. Reading the status flags and registers of a GPT channel.
8. Reading the status flags and registers of a GPT unit.
9. Enable the PWM Edge Delay circuit.
10. Control the PWM Edge Delay circuit.
11. Disable the Edge Delay circuit.

2.18. 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.

2.19. Watchdog Timer Driver

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

1. Configuring the timer for use, including

   - Clock selection.
   - Time-out period.
   - Window position.
   - Reset or NMI Interrupt selection when timer overflows.

2. Control of the timer, including

   - Counter refresh to prevent timeout.

3. Reading the timer status including counter value.

2.20. 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.

2.21. 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. Transferring data in SPI mode, with polling or interrupt mode automatically selected.
7. Transmitting data in IIC mode, with polling or interrupt mode automatically selected.
8. Receiving data in IIC mode, with polling or interrupt mode automatically selected.
9. Receiving the last byte of data in 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.

2.22. 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.

2.23. Serial Peripheral Interface Driver

The driver functions support the use of the SPI channel, 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.

2.24. 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.

2.25. 12-bit Analog to Digital Converter Driver

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

1. I/O pin configuration
2. Unit specific configuration for use, including
   - Automatic clock setting using sampling time as an input
   - Automatic interrupt control
3. Channel specific configuration for use, including
   - Double trigger control
   - Sample-and-hold control
   - Sampling time control
   - Comparator 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.

2.26. 10-bit Analog to Digital Converter Driver

The driver functions support the use of the 10-bit ADC unit 0, providing the following operations.

1. I/O configuration.
2. Configuration of the ADC unit
3. Channel specific configuration for use, including
    - Value addition 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 interrupt.

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

2.27. 10-bit Digital to Analog Converter Driver

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

1. Configuring a channel for use, including
   - Data alignment
   - D/A A/D synchronous conversion
2. Disabling channels that are no longer required and enabling low-power mode.
3. Writing data to a channel.

2.28. Data Operation Circuit

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

3.2.1. PDL_NO_FUNC

Used as a parameter when there is no applicable function.

3.2.2. PDL_NO_PTR

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

3.2.3. PDL_NO_DATA

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

3.2.4. PDL_MCU_GROUP

The MCU group supported by this build of the driver library. It is defined as RX63T.

A usage example is:

```
#if PDL_MCU_GROUP != RX63T
 #error "Wrong RPDL !"
#endif
```

3.2.5. 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;
```

3.2.6. Bit definitions

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

4. Library Reference

4.1. API List by Peripheral Function

Table 4.1 lists the Renesas Embedded APIs by peripheral function.

Table 4.1 Renesas Embedded API List

Category	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.
	11	R_INTC_CreateGroup	Configure an interrupt source group.
	12	R_INTC_ControlGroup	Control an interrupt source group.
	13	R_INTC_GetStatusGroup	Read the status of an interrupt source group.
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	Modify the clock accuracy circuit operation.
	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.
	3	R_LPC_WriteBackup	Write to the Backup registers.
	4	R_LPC_ReadBackup	Read from the Backup registers.
	5	R_LPC_GetStatus	Read the status flags.
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.
	3	R_BSC_CreateArea	Configure an external bus area.
	4	R_BSC_Destroy	Stop the Bus Controller.
	5	R_BSC_Control	Modify the Bus Controller operation.
	6	R_BSC_GetStatus	Read the Bus Controller status registers.
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.
Multi-function Timer pulse unit	1	R_MTU3_Set	Configure the Multi-function Timer Pulse Units.
	2	R_MTU3_Create	Configure a MTU channel.
	3	R_MTU3_Destroy	Disable a Multi-function Timer Pulse Unit.
	4	R_MTU3_ControlChannel	Control an MTU channel.
	5	R_MTU3_ControlUnit	Control a Multi-function Timer Pulse Unit.
	6	R_MTU3_ReadChannel	Read from MTU channel registers.
	7	R_MTU3_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 Port Output Enable module status.
General PWM Timer	1	R_GPT_Set	Select the I/O pins for the GPT unit.
	2	R_GPT_CreateUnit	Configure the GPT unit.
	3	R_GPT_CreateChannel	Configure a GPT channel.
	4	R_GPT_Destroy	Disable the GPT unit.
	5	R_GPT_ControlChannel	Control a GPT channel.
	6	R_GPT_ControlUnit	Control the GPT unit.
	7	R_GPT_ReadChannel	Read from GPT channel registers.
	8	R_GPT_ReadUnit	Read the GPT unit flags.
	9	R_GPT_EdgeDelay_Create	Enable the PWM Edge Delay circuit.
	10	R_GPT_EdgeDelay_Control	Control the PWM Edge Delay circuit.
	11	R_GPT_EdgeDelay_Destroy	Disable the Edge Delay circuit.
Compare Match Timer	1	R_CMT_Create	Configure a CMT channel.
	2	R_CMT_CreateOneShot	Configure a CMT channel as a one-shot event.
	3	R_CMT_Destroy	Disable a CMT unit.
	4	R_CMT_Control	Control CMT operation.
	5	R_CMT_Read	Read CMT channel status and registers.
Watchdog Timer	1	R_WDT_Set	Configure the Watchdog timer operation.
	2	R_WDT_Control	Control the Watchdog operation.
	3	R_WDT_Read	Read the Watchdog timer status and registers.
Independent Watchdog Timer	1	R_IWDT_Set	Configure the Independent Watchdog operation.
	2	R_IWDT_Control	Control the Independent Watchdog operation.
	3	R_IWDT_Read	Read the watchdog timer status and counter.
Serial Communication Interface	1	R_SCI_Set	Configure the SCI pin selection.
	2	R_SCI_Create	SCI channel setup.
	3	R_SCI_Destroy	Shut down a SCI channel.
	4	R_SCI_Send	Send a string of characters.
	5	R_SCI_Receive	Receive a string of characters.
	6	R_SCI_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_Create	I²C channel setup.
	2	R_IIC_Destroy	Disable an I²C channel.
	3	R_IIC_MasterSend	Write data to a slave device.
	4	R_IIC_MasterReceive	Read data from a slave device.
	5	R_IIC_MasterReceiveLast	Complete a DMAC or DTC-based read process.
	6	R_IIC_SlaveMonitor	Monitor the bus and receive data from a master.
	7	R_IIC_SlaveSend	Write data to a master device.
	8	R_IIC_Control	I²C channel control.
	9	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.
10-bit Analog to Digital converter	1	R_ADC_10_Set	Select the I/O pins for the 10-bit ADC.
	2	R_ADC_10_CreateUnit	Configure the 10-bit ADC unit.
	3	R_ADC_10_CreateChannel	Configure 10-bit ADC analog channels.
	4	R_ADC_10_Destroy	Shut down the ADC unit.
	5	R_ADC_10_Control	Start or stop the ADC unit.
	6	R_ADC_10_Read	Read the ADC conversion results.
10-bit Digital to Analog converter	1	R_DAC_10_Create	Configure the 10-bit DAC module.
	2	R_DAC_10_Destroy	Disable a DAC channel.
	3	R_DAC_10_Write	Write data to a DAC channel.
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 A frequency
    double data6,      // Peripheral module clock B frequency
    double data7,      // Peripheral module clock C frequency
    double data8,      // Peripheral module clock D frequency
    double data9,      // Flash interface clock frequency
    double data10,     // External bus clock frequency
    double data11      // USB clock frequency
);
```

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_MAIN or PDL_CGC_CLK_PLL or PDL_CGC_CLK_IWDTLOCO	Select the low-speed on-chip oscillator (LOCO), main clock oscillator, Phase-locked loop (PLL) circuit or IWDT-dedicated low-speed clock on-chip oscillator (IWDTLOCO).

[data2]
Configuration settings.

- BCLK pin output control (ignored if the device package does not support the external bus)

PDL_CGC_BCLK_DIV_1 or PDL_CGC_BCLK_DIV_2 or PDL_CGC_BCLK_DISABLE	Output the external bus clock (BCLK), BCLK ÷ 2 or disable the BCLK signal.

[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 A clock (PCLKA), in Hertz.

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

[data7]
The desired frequency of the Peripheral module C clock (PCLKC), in Hertz.
Ignored for packages with 48 or 64 pins.

[data8]
The desired frequency of the Peripheral module D clock (PCLKD), in Hertz.

[data9]
The desired frequency of the Flash memory interface clock (FCLK), in Hertz.

[data10]
The desired frequency of the External Bus clock (BCLK), in Hertz.
If the external bus will not be used, specify PDL_NO_DATA.
Ignored for packages with 48 or 64 pins.

Description (2/2)

[data11]
The desired frequency of the USB clock (UCLK), in Hertz.
If the USB will not be used, specify PDL_NO_DATA.
Ignored for packages with 48 or 64 pins.

Return value

True if all parameters are valid and exclusive; otherwise false.
For RX63T, the following rules are checked where applicable:

- For 48- and 64-pin packages:
  $f_{MAIN\_CLOCK\_OSCILLATOR} \leq 20$ MHz (between 4 and 16 MHz if a resonator is used).
- For packages with 100 pins or more:
  $f_{MAIN\_CLOCK\_OSCILLATOR} \leq 14$ MHz (between 8 and 12.5 MHz if a resonator is used).
- f_{PLL} = 104 to 200 MHz
- $f_{ICLK} \leq 100$ MHz
- $f_{PCLKA} \leq 100$ MHz
- $f_{PCLKB} \leq 50$ MHz
- $f_{PCLKC} \leq 100$ MHz
- $f_{PCLKD} \leq 50$ MHz
- $f_{FCLK} \leq 50$ MHz
- $f_{BCLK} \leq 50$ MHz
- $f_{BCLK\_PIN} \leq 25$ MHz
- $f_{BCLK} \leq f_{ICLK}$
- f_{USB_CLOCK} = 48 MHz
- The frequency of the PLL is achievable: (main clock ÷ 1, 2 or 4) x 8, 10, 12, 16, 20, 24, 25 or 50.
- The frequencies of the internal clocks (ICLK, PCLKA, PCLKB, PCLKC, PCLKD, FCLK and BCLK) are achievable: (selected clock source) ÷ 1, 2, 4, 8, 16, 32 or 64.
- The frequency of the USB clock (UCLK) is achievable: (selected clock source) ÷ 2, 3 or 4.

Category

Clock generation circuit

References

R_CGC_Control, R_MCU_GetStatus, R_LPC_Create

Remarks

- Call this function once for each clock source that will be used.
- If the current clock source is selected in parameter data1, the frequencies of the internal clocks will be changed by this function.
- 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.
- The ratios of f_{ICLK}, f_{PCLKA}, f_{PCLKB}, f_{PCLKC}, f_{PCLKD} and f_{FCLK} have some restrictions. Please refer to the notes of SCKCR register in the hardware manual.
- The registers MOSCWTCR (main clock) and PLLWTCR (PLL) 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 PLL will be used, first use this function to configure the main clock oscillator settings.
- If the PLL will be used, the frequencies of the internal clocks ICLK, PCLKA, PCLKB, PCLKC, PCLKD, FCLK and BCLK must be no more than the PLL output clock frequency ÷ 2.
- If the main clock will be used, the frequencies of the internal clocks ICLK, PCLKA, PCLKB, PCLKC, PCLKD, FCLK and BCLK must be no more than the main clock frequency ÷ 4.
- If the PLL output frequency is to be changed while the PLL is enabled, before calling this function use R_CGC_Control to select another clock source and stop the PLL.
- If the IWDTLOCO is selected, specify PDL_NO_DATA for parameters data2 and data4 to data11.
- The external bus is not available on the 48- and 64-pin packages.
- The BCLK pin output will not be active until the external bus is enabled (using R_BSC_Control).
- When operation of the external bus clock is selected, the PE5 I/O port pin function is not available.
- If low-speed operating mode 1 is selected, do not call this function to configure the PLL.
- When USB module is used, UCLK must be set to 48MHz.

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 12.0 MHz crystal */
    /* ICLK = 3 MHz, PCLKA = 3 MHz, PCLKB = 3 MHz */
    /* PCLKC = 3 MHz, PCLKD = 3 MHz, FCLK = 3 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_CGC_BCLK_DISABLE, /* PDL_NO_DATA for 64 and 48 pin package */
        12E6,
        3E6,
        3E6,
        3E6,
        3E6, /* PDL_NO_DATA for 64 and 48 pin package */
        3E6,
        3E6,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Configure PLL operation.  The PLL will be set to 120 MHz */
    /* ICLK = 60 MHz, PCLKA = 60 MHz, PCLKB = 15 MHz */
    /* PCLKC = 15 MHz, PCLKD = 15 MHz, FCLK = 15 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_PLL,
        PDL_CGC_BCLK_DISABLE, /* PDL_NO_DATA for 64 and 48 pin package */
        120E6,
        60E6,
        60E6,
        15E6,
        15E6, /* PDL_NO_DATA for 64 and 48 pin package */
        15E6,
        15E6,
        PDL_NO_DATA,
        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

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_MAIN or PDL_CGC_CLK_PLL	Select the low-speed on-chip oscillator (LOCO), main clock oscillator or Phase-locked loop (PLL) circuit.

[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.

- BCLK pin output control (ignored if the device package does not support the external bus)

PDL_CGC_BCLK_ENABLE or PDL_CGC_BCLK_DISABLE	Enable or disable the BCLK pin output.

- Low-speed on-chip oscillator control

PDL_CGC_LOCO_ENABLE or PDL_CGC_LOCO_DISABLE	Enable or disable the LOCO.

- Main clock oscillator control

PDL_CGC_MAIN_ENABLE or PDL_CGC_MAIN_DISABLE	Enable or disable the main clock oscillator.

- Main clock oscillator forced oscillation control

PDL_CGC_MAIN_FORCED_ENABLE or PDL_CGC_MAIN_FORCED_DISABLE	Enable or disable forced oscillation of 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.

- PLL control

PDL_CGC_PLL_ENABLE or PDL_CGC_PLL_DISABLE	Enable or disable the PLL circuit.

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

PDL_CGC_IWDTLOCO_ENABLE or PDL_CGC_IWDTLOCO_DISABLE	Enable or disable the IWDTLOCO.

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 selecting it.
- 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 1 is selected, do not enable the PLL.
- If this function is used to enable a clock oscillator, wait for the required settling time before selecting the clock source.

Program example

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

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

void func(void)
{
    /* Stop the low-speed on-chip oscillator */
    R_CGC_Control(
        PDL_NO_DATA,
        PDL_CGC_LOCO_DISABLE,
        PDL_NO_DATA
    );

    /* Select the PLL */
    R_CGC_Control(
        PDL_CGC_CLK_PLL,
        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	Clock control					
	-	IWDTLOCO	LOCO	-	Main clock	PLL
	0: Operating 1: Stopped					

b7	b6 - b4	b3 - b2	b1	b0
0	Selected clock source	0	Main clock oscillation stop detection	
	000b: LOCO 010b: Main clock 100b: PLL		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

Assign the external interrupt pins.

[data]
Allocate the pins for signals IRQ0 to IRQ5 for 64 and 48 pin packages.
Allocate the pins for signals IRQ0 to IRQ7 for 144, 120, 112 and 100 pin packages.
All selections are optional.
If multiple selections are required, use "|" to separate each selection.
Please refer to Table 21.1 at the "Multifunction Pin Controller (MPC)" section in the RX63T Hardware Manual for details of pin-package.

PDL_INTC_IRQ0_P10 or PDL_INTC_IRQ0_PB5	Select the pins to be used for signals IRQ0 to IRQ5 for 64 and 48 pin packages.
PDL_INTC_IRQ1_P11 or PDL_INTC_IRQ1_P93	
PDL_INTC_IRQ2_P00	
PDL_INTC_IRQ3_PB4	
PDL_INTC_IRQ4_P01	
PDL_INTC_IRQ5_P70	

PDL_INTC_IRQ0_P10 or PDL_INTC_IRQ0_PE5 or PDL_INTC_IRQ0_PG0	Select the pins to be used for signals IRQ0 to IRQ7 for 144, 120, 112 and 100 pin packages.
PDL_INTC_IRQ1_P11 or PDL_INTC_IRQ1_PE4 or PDL_INTC_IRQ1_PG1	
PDL_INTC_IRQ2_PE3 or PDL_INTC_IRQ2_PB6 or PDL_INTC_IRQ2_PG2	
PDL_INTC_IRQ3_PB4 or PDL_INTC_IRQ3_P34 or PDL_INTC_IRQ3_P82	
PDL_INTC_IRQ4_P96 or PDL_INTC_IRQ4_PB1 or PDL_INTC_IRQ4_P24	
PDL_INTC_IRQ5_P70 or PDL_INTC_IRQ5_PF2 or PDL_INTC_IRQ5_P80	
PDL_INTC_IRQ6_P21 or PDL_INTC_IRQ6_PD5 or PDL_INTC_IRQ6_PG4	
PDL_INTC_IRQ7_P20 or PDL_INTC_IRQ7_P03 or PDL_INTC_IRQ7_PE0	

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.
- Some IRQ pins (labelled in the hardware manual with the suffix –DS) can also be used to exit from Deep Software Standby mode. Please refer to the Low Power Consumption section for details.

Program example

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

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

void func(void)
{
    /* Select P10 for IRQ0, P11 for IRQ1 and P70 for IRQ5 */
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ0_P10 | PDL_INTC_IRQ1_P11 | PDL_INTC_IRQ5_P70
    );
}
```

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 or PDL_INTC_NMI	IRQn (n = 0 to 5 for 64 and 48 pin package, n = 0 to 7 for 144, 120, 112 and 100 pin package) interrupt pin or NMI interrupt pin.

[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_WDT_DISABLE or PDL_INTC_WDT_ENABLE	Disable or enable the NMI signal when a WDT underflow 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.

Description (2/2)

[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.

[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 WDT only (no NMI pin) */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_WDT_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.
Some interrupt vectors are not existed in 64 and 48 pin packages. Refer to Hardware manual section 15, Interrupt controller, Table 15.3, Interrupt Vector Table for details.

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_D0FIFO0	USB channel 0	D0FIFO transfer request
PDL_INTC_VECTOR_D1FIFO0		D1FIFO transfer request
PDL_INTC_VECTOR_USBI0		Event detection
PDL_INTC_VECTOR_USBR0		Resume
PDL_INTC_VECTOR_FERRF	Clock frequency accuracy measurement	Frequency error
PDL_INTC_VECTOR_MENDF		Measurement end
PDL_INTC_VECTOR_OVFF		Overflow
PDL_INTC_VECTOR_SPRI0	SPI channel 0	Receive buffer full
PDL_INTC_VECTOR_SPTI0		Transmit buffer empty
PDL_INTC_VECTOR_SPII0		Idle
PDL_INTC_VECTOR_SPRI1	SPI channel 1	Receive buffer full
PDL_INTC_VECTOR_SPTI1		Transmit buffer empty
PDL_INTC_VECTOR_SPII1		Idle
PDL_INTC_VECTOR_RXF1	CAN channel 1	Receive FIFO
PDL_INTC_VECTOR_TXF1		Transmit FIFO
PDL_INTC_VECTOR_RXM1		Reception complete
PDL_INTC_VECTOR_TXM1		Transmission complete
PDL_INTC_VECTOR_GTCIA7	General PWM timer, channel 7	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB7		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC7		Compare match C or D
PDL_INTC_VECTOR_GTCIE7		Compare match E or F
PDL_INTC_VECTOR_GTCIV7		Counter limit match
PDL_INTC_VECTOR_CMP4	12-bit ADC unit 1	Comparator event on input AN100
PDL_INTC_VECTOR_CMP5		Comparator event on input AN101
PDL_INTC_VECTOR_CMP6		Comparator event on input AN102
PDL_INTC_VECTOR_DOPCF	Data operation	Condition detection
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_GROUP0		Group 0 event
PDL_INTC_VECTOR_GROUP12		Group 12 event
PDL_INTC_VECTOR_ADI0	10-bit ADC	Conversion completed
PDL_INTC_VECTOR_S12ADI	12-bit ADC unit 0	Conversion completed
PDL_INTC_VECTOR_S12GBADI		Group B conversion completed
PDL_INTC_VECTOR_S12ADI1	12-bit ADC unit 1	Conversion completed
PDL_INTC_VECTOR_S12GBADI1		Group B conversion completed

Description (2/3)

PDL_INTC_VECTOR_SCIX0	SCI channel 12	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_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
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 or underflow
PDL_INTC_VECTOR_TGIU5	Multi-function Timer Pulse Unit channel 5	Compare match or Input capture U
PDL_INTC_VECTOR_TGIV5		Compare match or Input capture V
PDL_INTC_VECTOR_TGIW5		Compare match or Input capture W
PDL_INTC_VECTOR_TGIA6	Multi-function Timer Pulse Unit channel 6	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB6		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC6		Compare match or Input capture C
PDL_INTC_VECTOR_TGID6		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV6		Overflow
PDL_INTC_VECTOR_TGIA7	Multi-function Timer Pulse Unit channel 7	Compare match or Input capture A
PDL_INTC_VECTOR_TGIB7		Compare match or Input capture B
PDL_INTC_VECTOR_TGIC7		Compare match or Input capture C
PDL_INTC_VECTOR_TGID7		Compare match or Input capture D
PDL_INTC_VECTOR_TCIV7		Overflow
PDL_INTC_VECTOR_OEI1	Port Output Enable	Input-level sampling or output-level comparison detection
PDL_INTC_VECTOR_OEI2		
PDL_INTC_VECTOR_OEI3		
PDL_INTC_VECTOR_OEI4		
PDL_INTC_VECTOR_OEI5		
PDL_INTC_VECTOR_CMP0	12-bit ADC unit 0	Comparator event on input AN000
PDL_INTC_VECTOR_CMP1		Comparator event on input AN001
PDL_INTC_VECTOR_CMP2		Comparator event on input AN002
PDL_INTC_VECTOR_GTCIA4	General PWM timer, channel 4	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB4		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC4		Compare match C or D
PDL_INTC_VECTOR_GTCIE4		Compare match E or F
PDL_INTC_VECTOR_GTCIV4		Counter limit match
PDL_INTC_VECTOR_LOCOI4		LOCO count function event
PDL_INTC_VECTOR_GTCIA5	General PWM timer, channel 5	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB5		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC5		Compare match C or D
PDL_INTC_VECTOR_GTCIE5		Compare match E or F
PDL_INTC_VECTOR_GTCIV5		Counter limit match

Description (3/3)

PDL_INTC_VECTOR_GTCIA6	General PWM timer, channel 6	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB6		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC6		Compare match C or D
PDL_INTC_VECTOR_GTCIE6		Compare match E or F
PDL_INTC_VECTOR_GTCIV6		Counter limit match
PDL_INTC_VECTOR_ICEEI1	I²C bus interface channel 1	Transfer error or event generation
PDL_INTC_VECTOR_ICRXI1		Data received
PDL_INTC_VECTOR_ICTXI1		Start of next data transfer
PDL_INTC_VECTOR_ICTEI1		End of data transfer
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
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_RXI0	SCI channel 0	Data received
PDL_INTC_VECTOR_TXI0		Start of next data transfer
PDL_INTC_VECTOR_TEI0		End of data transfer
PDL_INTC_VECTOR_RXI1	SCI channel 1	Data received
PDL_INTC_VECTOR_TXI1		Start of next data transfer
PDL_INTC_VECTOR_TEI1		End of data transfer
PDL_INTC_VECTOR_RXI2	SCI channel 2	Data received
PDL_INTC_VECTOR_TXI2		Start of next data transfer
PDL_INTC_VECTOR_TEI2		End of data transfer
PDL_INTC_VECTOR_RXI3	SCI channel 3	Data received
PDL_INTC_VECTOR_TXI3		Start of next data transfer
PDL_INTC_VECTOR_TEI3		End of data transfer
PDL_INTC_VECTOR_RXI12	SCI channel 12	Data received
PDL_INTC_VECTOR_TXI12		Start of next data transfer
PDL_INTC_VECTOR_TEI12		End of data transfer
PDL_INTC_VECTOR_GTCIA0	General PWM timer, channel 0	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB0		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC0		Compare match C or D
PDL_INTC_VECTOR_GTCIE0		Compare match E or F
PDL_INTC_VECTOR_GTCIV0		Counter limit match
PDL_INTC_VECTOR_LOCOI0		LOCO count function event
PDL_INTC_VECTOR_GTCIA1	General PWM timer, channel 1	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB1		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC1		Compare match C or D
PDL_INTC_VECTOR_GTCIE1		Compare match E or F
PDL_INTC_VECTOR_GTCIV1		Counter limit match
PDL_INTC_VECTOR_GTCIA2	General PWM timer, channel 2	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB2		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC2		Compare match C or D
PDL_INTC_VECTOR_GTCIE2		Compare match E or F
PDL_INTC_VECTOR_GTCIV2		Counter limit match
PDL_INTC_VECTOR_GTCIA3	General PWM timer, channel 3	Compare match or input capture A
PDL_INTC_VECTOR_GTCIB3		Compare match or input capture B
PDL_INTC_VECTOR_GTCIC3		Compare match C or D
PDL_INTC_VECTOR_GTCIE3		Compare match E or F
PDL_INTC_VECTOR_GTCIV3		Counter limit match

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
    void * func3,     // Callback function
    void * func4      // 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 access exception is detected.
Specify PDL_NO_FUNC if no callback function is required.

[func3]
The function to be called when an undefined instruction is detected.
Specify PDL_NO_FUNC if no callback function is required.

[func4]
The function to be called when a floating point exception is detected.
Specify PDL_NO_FUNC if no callback function is required.

Return value

True.

Category

Interrupt control

Reference

Remarks

- Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.

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 FloatingPointError ( void );

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

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 or PDL_INTC_NMI	IRQn (n = 0 to 5 for 64 and 48 pin package, n = 0 to 7 for 144, 120, 112 and 100 pin package) 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_WDT_FLAG	Clear the WDT event 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 or PDL_INTC_NMI	IRQn (n = 0 to 5 for 64 and 48 pin package, n = 0 to 7 for 144, 120, 112 and 100 pin package) 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	WDT				
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 (Some IR registers are not existed in 64 and 48 pin packages. Refer to Hardware manual section 15, Interrupt controller, Table 15.3, Interrupt Vector Table for details.)

PDL_INTC_REG_IR_BSC_BUSERR	PDL_INTC_REG_IR_MTU1_TGIA
PDL_INTC_REG_IR_FCU_FIFERR	PDL_INTC_REG_IR_MTU1_TGIB
PDL_INTC_REG_IR_FCU_FRDYI	PDL_INTC_REG_IR_MTU1_TCIV
PDL_INTC_REG_IR_ICU_SWINT	PDL_INTC_REG_IR_MTU1_TCIU
PDL_INTC_REG_IR_CMT0_CMI	PDL_INTC_REG_IR_MTU2_TGIA
PDL_INTC_REG_IR_CMT1_CMI	PDL_INTC_REG_IR_MTU2_TGIB
PDL_INTC_REG_IR_CMT2_CMI	PDL_INTC_REG_IR_MTU2_TCIV
PDL_INTC_REG_IR_CMT3_CMI	PDL_INTC_REG_IR_MTU2_TCIU
PDL_INTC_REG_IR_USB0_D0FIFO	PDL_INTC_REG_IR_MTU3_TGIA
PDL_INTC_REG_IR_USB0_D1FIFO	PDL_INTC_REG_IR_MTU3_TGIB
PDL_INTC_REG_IR_USB0_USBI	PDL_INTC_REG_IR_MTU3_TGIC
PDL_INTC_REG_IR_USB0_USBR	PDL_INTC_REG_IR_MTU3_TGID
PDL_INTC_REG_IR_CAC_FERRF	PDL_INTC_REG_IR_MTU3_TCIV
PDL_INTC_REG_IR_CAC_MENDF	PDL_INTC_REG_IR_MTU4_TGIA
PDL_INTC_REG_IR_CAC_OVFF	PDL_INTC_REG_IR_MTU4_TGIB
PDL_INTC_REG_IR_SPI0_SPRI	PDL_INTC_REG_IR_MTU4_TGIC
PDL_INTC_REG_IR_SPI0_SPTI	PDL_INTC_REG_IR_MTU4_TGID
PDL_INTC_REG_IR_SPI0_SPII	PDL_INTC_REG_IR_MTU4_TCIV
PDL_INTC_REG_IR_SPI1_SPRI	PDL_INTC_REG_IR_MTU5_TGIU
PDL_INTC_REG_IR_SPI1_SPTI	PDL_INTC_REG_IR_MTU5_TGIV
PDL_INTC_REG_IR_SPI1_SPTI	PDL_INTC_REG_IR_MTU5_TGIW
PDL_INTC_REG_IR_CAN1_RXF	PDL_INTC_REG_IR_MTU6_TGIA
PDL_INTC_REG_IR_CAN1_TXF	PDL_INTC_REG_IR_MTU6_TGIB
PDL_INTC_REG_IR_CAN1_RXM	PDL_INTC_REG_IR_MTU6_TGIC
PDL_INTC_REG_IR_CAN1_TXM	PDL_INTC_REG_IR_MTU6_TGID
PDL_INTC_REG_IR_GPT7_GTCIA	PDL_INTC_REG_IR_MTU6_TCIV
PDL_INTC_REG_IR_GPT7_GTCIB	PDL_INTC_REG_IR_MTU7_TGIA
PDL_INTC_REG_IR_GPT7_GTCIC	PDL_INTC_REG_IR_MTU7_TGIB
PDL_INTC_REG_IR_GPT7_GTCIE	PDL_INTC_REG_IR_MTU7_TGIC
PDL_INTC_REG_IR_GPT7_GTCIV	PDL_INTC_REG_IR_MTU7_TGID
PDL_INTC_REG_IR_CMP4	PDL_INTC_REG_IR_MTU7_TCIV
PDL_INTC_REG_IR_CMP5	PDL_INTC_REG_IR_POE_OEI1
PDL_INTC_REG_IR_CMP6	PDL_INTC_REG_IR_POE_OEI2
PDL_INTC_REG_IR_DOC_DOPCF	PDL_INTC_REG_IR_POE_OEI3
PDL_INTC_REG_IR_ICU_IRQ0	PDL_INTC_REG_IR_POE_OEI4
PDL_INTC_REG_IR_ICU_IRQ1	PDL_INTC_REG_IR_POE_OEI5
PDL_INTC_REG_IR_ICU_IRQ2	PDL_INTC_REG_IR_CMP0
PDL_INTC_REG_IR_ICU_IRQ3	PDL_INTC_REG_IR_CMP1
PDL_INTC_REG_IR_ICU_IRQ4	PDL_INTC_REG_IR_CMP2
PDL_INTC_REG_IR_ICU_IRQ5	PDL_INTC_REG_IR_GPT4_GTCIA
PDL_INTC_REG_IR_ICU_IRQ6	PDL_INTC_REG_IR_GPT4_GTCIB
PDL_INTC_REG_IR_ICU_IRQ7	PDL_INTC_REG_IR_GPT4_GTCIC
PDL_INTC_REG_IR_AD_ADI	PDL_INTC_REG_IR_GPT4_GTCIE
PDL_INTC_REG_IR_S12AD_S12ADI	PDL_INTC_REG_IR_GPT4_GTCIV
PDL_INTC_REG_IR_S12AD_S12GBADI	PDL_INTC_REG_IR_GPT4_LOCO
PDL_INTC_REG_IR_S12AD1_S12ADI	PDL_INTC_REG_IR_GPT5_GTCIA
PDL_INTC_REG_IR_S12AD1_S12GBADI	PDL_INTC_REG_IR_GPT5_GTCIB
PDL_INTC_REG_IR_ICU_GROUP0	PDL_INTC_REG_IR_GPT5_GTCIC
PDL_INTC_REG_IR_ICU_GROUP12	PDL_INTC_REG_IR_GPT5_GTCIE
PDL_INTC_REG_IR_SCI12_SCIX0	PDL_INTC_REG_IR_GPT5_GTCIV
PDL_INTC_REG_IR_SCI12_SCIX1	PDL_INTC_REG_IR_GPT6_GTCIA
PDL_INTC_REG_IR_SCI12_SCIX2	PDL_INTC_REG_IR_GPT6_GTCIB
PDL_INTC_REG_IR_SCI12_SCIX3	PDL_INTC_REG_IR_GPT6_GTCIC
PDL_INTC_REG_IR_MTU0_TGIA	PDL_INTC_REG_IR_GPT6_GTCIE
PDL_INTC_REG_IR_MTU0_TGIB	PDL_INTC_REG_IR_GPT6_GTCIV
PDL_INTC_REG_IR_MTU0_TGIC	PDL_INTC_REG_IR_IIC1_EEI
PDL_INTC_REG_IR_MTU0_TGID	PDL_INTC_REG_IR_IIC1_RXI
PDL_INTC_REG_IR_MTU0_TCIV	PDL_INTC_REG_IR_IIC1_TXI
PDL_INTC_REG_IR_MTU0_TGIE	PDL_INTC_REG_IR_IIC1_TEI
PDL_INTC_REG_IR_MTU0_TGIF	PDL_INTC_REG_IR_IIC0_EEI

PDL_INTC_REG_IR_IIC0_RXI	PDL_INTC_REG_IR_GPT0_GTCIE
PDL_INTC_REG_IR_IIC0_TXI	PDL_INTC_REG_IR_GPT0_GTCIV
PDL_INTC_REG_IR_IIC0_TEI	PDL_INTC_REG_IR_GPT0_LOCO
PDL_INTC_REG_IR_DMAC_DMAC0I	PDL_INTC_REG_IR_GPT1_GTCIA
PDL_INTC_REG_IR_DMAC_DMAC1I	PDL_INTC_REG_IR_GPT1_GTCIB
PDL_INTC_REG_IR_DMAC_DMAC2I	PDL_INTC_REG_IR_GPT1_GTCIC
PDL_INTC_REG_IR_DMAC_DMAC3I	PDL_INTC_REG_IR_GPT1_GTCIE
PDL_INTC_REG_IR_SCI0_RXI	PDL_INTC_REG_IR_GPT1_GTCIV
PDL_INTC_REG_IR_SCI0_TXI	PDL_INTC_REG_IR_GPT2_GTCIA
PDL_INTC_REG_IR_SCI0_TEI	PDL_INTC_REG_IR_GPT2_GTCIB
PDL_INTC_REG_IR_SCI1_RXI	PDL_INTC_REG_IR_GPT2_GTCIC
PDL_INTC_REG_IR_SCI1_TXI	PDL_INTC_REG_IR_GPT2_GTCIE
PDL_INTC_REG_IR_SCI2_RXI	PDL_INTC_REG_IR_GPT2_GTCIV
PDL_INTC_REG_IR_SCI2_TXI	PDL_INTC_REG_IR_GPT3_GTCIA
PDL_INTC_REG_IR_SCI2_TEI	PDL_INTC_REG_IR_GPT3_GTCIB
PDL_INTC_REG_IR_SCI3_RXI	PDL_INTC_REG_IR_GPT3_GTCIC
PDL_INTC_REG_IR_SCI3_TXI	PDL_INTC_REG_IR_GPT3_GTCIE
PDL_INTC_REG_IR_SCI3_TEI	PDL_INTC_REG_IR_GPT3_GTCIV
PDL_INTC_REG_IR_GPT0_GTCIA	PDL_INTC_REG_IR_SCI12_RXI
PDL_INTC_REG_IR_GPT0_GTCIB	PDL_INTC_REG_IR_SCI12_TXI
PDL_INTC_REG_IR_GPT0_GTCIC	PDL_INTC_REG_IR_SCI12_TEI

IER register definitions

PDL_INTC_REG_IER02	PDL_INTC_REG_IER12
PDL_INTC_REG_IER03	PDL_INTC_REG_IER13
PDL_INTC_REG_IER04	PDL_INTC_REG_IER14
PDL_INTC_REG_IER05	PDL_INTC_REG_IER15
PDL_INTC_REG_IER07	PDL_INTC_REG_IER18
PDL_INTC_REG_IER08	PDL_INTC_REG_IER19
PDL_INTC_REG_IER0B	PDL_INTC_REG_IER1A
PDL_INTC_REG_IER0C	PDL_INTC_REG_IER1B
PDL_INTC_REG_IER0E	PDL_INTC_REG_IER1C
PDL_INTC_REG_IER0F	PDL_INTC_REG_IER1D
PDL_INTC_REG_IER10	PDL_INTC_REG_IER1E
PDL_INTC_REG_IER11	PDL_INTC_REG_IER1F

IPR register definitions (Some IPR registers are not existed in 64 and 48 pin packages. Refer to Hardware manual section 15, Interrupt controller, Table 15.3, Interrupt Vector Table for details.)

PDL_INTC_REG_IPR_BSC_BUSERR	PDL_INTC_REG_IPR_MTU0_TGIAD
PDL_INTC_REG_IPR_FCU_FIFERR	PDL_INTC_REG_IPR_MTU0_TGIVEF
PDL_INTC_REG_IPR_FCU_FRDYI	PDL_INTC_REG_IPR_MTU1_TGIAB
PDL_INTC_REG_IPR_ICU_SWINT	PDL_INTC_REG_IPR_MTU1_TCIVU
PDL_INTC_REG_IPR_CMT0_CMI	PDL_INTC_REG_IPR_MTU2_TGIAB
PDL_INTC_REG_IPR_CMT1_CMI	PDL_INTC_REG_IPR_MTU2_TCIVU
PDL_INTC_REG_IPR_CMT2_CMI	PDL_INTC_REG_IPR_MTU3_TGIAD
PDL_INTC_REG_IPR_CMT3_CMI	PDL_INTC_REG_IPR_MTU3_TCIV
PDL_INTC_REG_IPR_USB0_D0FIFO0	PDL_INTC_REG_IPR_MTU4_TGIAD
PDL_INTC_REG_IPR_USB0_D1FIFO0	PDL_INTC_REG_IPR_MTU4_TCIV
PDL_INTC_REG_IPR_USB0_USBI0	PDL_INTC_REG_IPR_MTU5
PDL_INTC_REG_IPR_USB0_USBR0	PDL_INTC_REG_IPR_MTU6_TGIAD
PDL_INTC_REG_IPR_CAC	PDL_INTC_REG_IPR_MTU6_TCIV
PDL_INTC_REG_IPR_SPI0_SPRI	PDL_INTC_REG_IPR_MTU7_TGIAB
PDL_INTC_REG_IPR_SPI0_SPTI	PDL_INTC_REG_IPR_MTU7_TGICD
PDL_INTC_REG_IPR_SPI0_SPII	PDL_INTC_REG_IPR_MTU7_TCIV
PDL_INTC_REG_IPR_SPI1_SPRI	PDL_INTC_REG_IPR_POE
PDL_INTC_REG_IPR_SPI1_SPTI	PDL_INTC_REG_IPR_CMP0
PDL_INTC_REG_IPR_SPI1_SPII	PDL_INTC_REG_IPR_CMP1
PDL_INTC_REG_IPR_CAN1	PDL_INTC_REG_IPR_CMP2
PDL_INTC_REG_IPR_GPT7_GTCIAC	PDL_INTC_REG_IPR_GPT4_GTCIAC
PDL_INTC_REG_IPR_GPT7_GTCIEV	PDL_INTC_REG_IPR_GPT4_GTCIEVLOCO
PDL_INTC_REG_IPR_CMP4	PDL_INTC_REG_IPR_GPT5_GTCIAC
PDL_INTC_REG_IPR_CMP5	PDL_INTC_REG_IPR_GPT5_GTCIEV
PDL_INTC_REG_IPR_CMP6	PDL_INTC_REG_IPR_GPT6_GTCIAC
PDL_INTC_REG_IPR_DOC	PDL_INTC_REG_IPR_GPT6_GTCIEV
PDL_INTC_REG_IPR_ICU_IRQ0	PDL_INTC_REG_IPR_IIC1
PDL_INTC_REG_IPR_ICU_IRQ1	PDL_INTC_REG_IPR_IIC0
PDL_INTC_REG_IPR_ICU_IRQ2	PDL_INTC_REG_IPR_DMAC_DMAC0I
PDL_INTC_REG_IPR_ICU_IRQ3	PDL_INTC_REG_IPR_DMAC_DMAC1I
PDL_INTC_REG_IPR_ICU_IRQ4	PDL_INTC_REG_IPR_DMAC_DMAC2I
PDL_INTC_REG_IPR_ICU_IRQ5	PDL_INTC_REG_IPR_DMAC_DMAC3I
PDL_INTC_REG_IPR_ICU_IRQ6	PDL_INTC_REG_IPR_SCI0
PDL_INTC_REG_IPR_ICU_IRQ7	PDL_INTC_REG_IPR_SCI1
PDL_INTC_REG_IPR_ICU_GROUP0	PDL_INTC_REG_IPR_SCI2
PDL_INTC_REG_IPR_ICU_GROUP12	PDL_INTC_REG_IPR_SCI3
PDL_INTC_REG_IPR_AD_ADI	PDL_INTC_REG_IPR_GPT0_GTCIAC
PDL_INTC_REG_IPR_S12AD_S12ADI	PDL_INTC_REG_IPR_GPT0_GTCIEVLOCO
PDL_INTC_REG_IPR_S12AD_S12GBADI	PDL_INTC_REG_IPR_GPT1_GTCIAC
PDL_INTC_REG_IPR_S12AD_S12ADI1	PDL_INTC_REG_IPR_GPT1_GTCIEV
PDL_INTC_REG_IPR_S12AD_S12GBADI1	PDL_INTC_REG_IPR_GPT2_GTCIAC
PDL_INTC_REG_IPR_SCI12_SCIX	PDL_INTC_REG_IPR_GPT2_GTCIEV
	PDL_INTC_REG_IPR_GPT3_GTCIAC
	PDL_INTC_REG_IPR_GPT3_GTCIEV
	PDL_INTC_REG_IPR_SCI12

DTCER register definitions (Some DTCER registers are not existed in 64 and 48 pin packages. Refer to Hardware manual section 15, Interrupt controller, Table 15.3, Interrupt Vector Table for details.)

PDL_INTC_REG_DTCER_ICU_SWINT	PDL_INTC_REG_DTCER_CMP_CMP0
PDL_INTC_REG_DTCER_CMT0_CMI	PDL_INTC_REG_DTCER_CMP_CMP1
PDL_INTC_REG_DTCER_CMT1_CMI	PDL_INTC_REG_DTCER_CMP_CMP2
PDL_INTC_REG_DTCER_CMT2_CMI	PDL_INTC_REG_DTCER_GPT4_GTCIA4
PDL_INTC_REG_DTCER_CMT3_CMI	PDL_INTC_REG_DTCER_GPT4_GTCIB4
PDL_INTC_REG_DTCER_USB0_D0FIFO	PDL_INTC_REG_DTCER_GPT4_GTCIC4
PDL_INTC_REG_DTCER_USB0_D1FIFO	PDL_INTC_REG_DTCER_GPT4_GTCIE4
PDL_INTC_REG_DTCER_SPI1_SPRI	PDL_INTC_REG_DTCER_GPT4_GTCIV4
PDL_INTC_REG_DTCER_SPI1_SPTI	PDL_INTC_REG_DTCER_GPT4_LOCO
PDL_INTC_REG_DTCER_SPI0_SPRI	PDL_INTC_REG_DTCER_GPT5_GTCIA5
PDL_INTC_REG_DTCER_SPI0_SPTI	PDL_INTC_REG_DTCER_GPT5_GTCIB5
PDL_INTC_REG_DTCER_GPT7_GTCIA7	PDL_INTC_REG_DTCER_GPT5_GTCIC5
PDL_INTC_REG_DTCER_GPT7_GTCIB7	PDL_INTC_REG_DTCER_GPT5_GTCIE5
PDL_INTC_REG_DTCER_GPT7_GTCIC7	PDL_INTC_REG_DTCER_GPT5_GTCIV5
PDL_INTC_REG_DTCER_GPT7_GTCIE7	PDL_INTC_REG_DTCER_GPT6_GTCIA6
PDL_INTC_REG_DTCER_GPT7_GTCIV7	PDL_INTC_REG_DTCER_GPT6_GTCIB6
PDL_INTC_REG_DTCER_CMP_CMP4	PDL_INTC_REG_DTCER_GPT6_GTCIC6
PDL_INTC_REG_DTCER_CMP_CMP5	PDL_INTC_REG_DTCER_GPT6_GTCIE6
PDL_INTC_REG_DTCER_CMP_CMP6	PDL_INTC_REG_DTCER_GPT6_GTCIV6
PDL_INTC_REG_DTCER_ICU_IRQ0	PDL_INTC_REG_DTCER_IIC1_RXI
PDL_INTC_REG_DTCER_ICU_IRQ1	PDL_INTC_REG_DTCER_IIC1_TXI
PDL_INTC_REG_DTCER_ICU_IRQ2	PDL_INTC_REG_DTCER_IIC0_RXI
PDL_INTC_REG_DTCER_ICU_IRQ3	PDL_INTC_REG_DTCER_IIC0_TXI
PDL_INTC_REG_DTCER_ICU_IRQ4	PDL_INTC_REG_DTCER_DMAC_DMAC0I
PDL_INTC_REG_DTCER_ICU_IRQ5	PDL_INTC_REG_DTCER_DMAC_DMAC1I
PDL_INTC_REG_DTCER_ICU_IRQ6	PDL_INTC_REG_DTCER_DMAC_DMAC2I
PDL_INTC_REG_DTCER_ICU_IRQ7	PDL_INTC_REG_DTCER_DMAC_DMAC3I
PDL_INTC_REG_DTCER_AD_ADI	PDL_INTC_REG_DTCER_SCI0_RXI
PDL_INTC_REG_DTCER_S12AD_S12ADI	PDL_INTC_REG_DTCER_SCI0_TXI
PDL_INTC_REG_DTCER_S12AD_S12GBADI	PDL_INTC_REG_DTCER_SCI1_RXI
PDL_INTC_REG_DTCER_S12AD1_S12ADI	PDL_INTC_REG_DTCER_SCI1_TXI
PDL_INTC_REG_DTCER_S12AD1_S12GBADI	PDL_INTC_REG_DTCER_SCI2_RXI
PDL_INTC_REG_DTCER_MTU0_TGIA	PDL_INTC_REG_DTCER_SCI2_TXI
PDL_INTC_REG_DTCER_MTU0_TGIB	PDL_INTC_REG_DTCER_SCI3_RXI
PDL_INTC_REG_DTCER_MTU0_TGIC	PDL_INTC_REG_DTCER_SCI3_TXI
PDL_INTC_REG_DTCER_MTU0_TGID	PDL_INTC_REG_DTCER_GPT0_GTCIA0
PDL_INTC_REG_DTCER_MTU1_TGIA	PDL_INTC_REG_DTCER_GPT0_GTCIB0
PDL_INTC_REG_DTCER_MTU1_TGIB	PDL_INTC_REG_DTCER_GPT0_GTCIC0
PDL_INTC_REG_DTCER_MTU2_TGIA	PDL_INTC_REG_DTCER_GPT0_GTCIE0
PDL_INTC_REG_DTCER_MTU2_TGIB	PDL_INTC_REG_DTCER_GPT0_GTCIV0
PDL_INTC_REG_DTCER_MTU3_TGIA	PDL_INTC_REG_DTCER_GPT0_LOCO
PDL_INTC_REG_DTCER_MTU3_TGIB	PDL_INTC_REG_DTCER_GPT1_GTCIA1
PDL_INTC_REG_DTCER_MTU3_TGIC	PDL_INTC_REG_DTCER_GPT1_GTCIB1
PDL_INTC_REG_DTCER_MTU3_TGID	PDL_INTC_REG_DTCER_GPT1_GTCIC1
PDL_INTC_REG_DTCER_MTU4_TGIA	PDL_INTC_REG_DTCER_GPT1_GTCIE1
PDL_INTC_REG_DTCER_MTU4_TGIB	PDL_INTC_REG_DTCER_GPT1_GTCIV1
PDL_INTC_REG_DTCER_MTU4_TGIC	PDL_INTC_REG_DTCER_GPT2_GTCIA2
PDL_INTC_REG_DTCER_MTU4_TGID	PDL_INTC_REG_DTCER_GPT2_GTCIB2
PDL_INTC_REG_DTCER_MTU4_TCIV	PDL_INTC_REG_DTCER_GPT2_GTCIC2
PDL_INTC_REG_DTCER_MTU5_TGIU	PDL_INTC_REG_DTCER_GPT2_GTCIE2
PDL_INTC_REG_DTCER_MTU5_TGIV	PDL_INTC_REG_DTCER_GPT2_GTCIV2
PDL_INTC_REG_DTCER_MTU5_TGIW	PDL_INTC_REG_DTCER_GPT3_GTCIA3
PDL_INTC_REG_DTCER_MTU6_TGIA	PDL_INTC_REG_DTCER_GPT3_GTCIB3
PDL_INTC_REG_DTCER_MTU6_TGIB	PDL_INTC_REG_DTCER_GPT3_GTCIC3
PDL_INTC_REG_DTCER_MTU6_TGIC	PDL_INTC_REG_DTCER_GPT3_GTCIE3
PDL_INTC_REG_DTCER_MTU6_TGID	PDL_INTC_REG_DTCER_GPT3_GTCIV3
PDL_INTC_REG_DTCER_MTU7_TGIA	PDL_INTC_REG_DTCER_SCI12_RXI
PDL_INTC_REG_DTCER_MTU7_TGIB	PDL_INTC_REG_DTCER_SCI12_TXI
PDL_INTC_REG_DTCER_MTU7_TGIC	
PDL_INTC_REG_DTCER_MTU7_TGID	
PDL_INTC_REG_DTCER_MTU7_TCIV	

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
    );
}
```

11) R_INTC_CreateGroup

Synopsis

Configure a group of peripheral interrupt requests.

Prototype

```
bool R_INTC_CreateGroup(
    uint8_t data1,      // Group selection
    void * func,        // Callback function
    uint8_t data2       // Interrupt priority level
);
```

Description

Set up the grouped interrupt request callback function.

[data1]
The interrupt request group n to be configured.
For 64 and 48 pin packages, n must be 12; for other pin packages, n can be 0 or 12.

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

[data2]
The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).

Return value

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

Category

Interrupt control

Reference

R_INTC_ControlGroup

Remarks

- Do not use this function if RPDL functions will be used to configure the applicable peripheral.
- Use R_INTC_ControlGroup to enable the required peripheral interrupt requests.
- 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"

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

void func( void )
{
    /* Assign a handler for group 12 */
    R_INTC_CreateGroup(
        12,
        CallBackFunc,
        10
    );
}
```

12) R_INTC_ControlGroup

Synopsis

Control an interrupt request group.

Prototype

```
bool R_INTC_ControlGroup(
    uint8_t data1,     // Group selection
    uint8_t data2,     // Interrupt control operation
    uint32_t data3     // Interrupt source selection
);
```

Description

Control an interrupt request group.

[data1]
The interrupt request group n to be configured.
For 64 and 48 pin packages, n must be 12; for other pin packages, n can be 0 or 12.

[data2]
The logical operations to be applied to the interrupt request group.
If multiple selections are required, use "|" to separate each selection.

PDL_INTC_GROUP_DISABLE	Disable the interrupt requests.
PDL_INTC_GROUP_CLEAR	Clear the interrupt request flags. This option does not exist in group 12.
PDL_INTC_GROUP_ENABLE	Enable the interrupt requests.

[data3]
Choose the peripheral interrupt request sources (for the group specified in parameter data1) to be modified.
If multiple selections are required, use "|" to separate each selection.
PDL_INTC_GRPn_ALL can be used to specify all applicable selections.

- Group 0 selections

PDL_INTC_GRP0_ERS1	Error on CAN channel 1.

- Group 12 selections.

PDL_INTC_GRP12_ERI0	Reception error on SCI channels 0, 1, 2, 3 or 12. Note: SCI channel 2 and 3 are not available for 64 and 48 pin packages.
PDL_INTC_GRP12_ERI1	
PDL_INTC_GRP12_ERI2	
PDL_INTC_GRP12_ERI3	
PDL_INTC_GRP12_ERI12	
PDL_INTC_GRP12_SPEI0	Error on SPI channel 0 or 1. Note: SPI channel 1 is not available for 64 and 48 pin packages.
PDL_INTC_GRP12_SPEI1	

Return value

False if the group number is invalid; otherwise true.

Category

Interrupt control

Reference

R_INTC_CreateGroup

Remarks

- Do not use this function if RPDL functions will be used to control the applicable peripheral.
- Call R_INTC_CreateGroup before calling this function.
- Group 12 interrupts use level detection, so clearing must be done by clearing the source of the interrupt.

Program example

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

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

void func( void )
{
    /* Disable the interrupt for SCI channel 0 */
    R_INTC_ControlGroup(
        12,
        PDL_INTC_GROUP_DISABLE,
        PDL_INTC_GRP12_ERI0
    );

    /* Enable all of the Group 12 interrupt sources */
    R_INTC_ControlGroup(
        12,
        PDL_INTC_GROUP_ENABLE,
        PDL_INTC_GRP12_ALL
    );
}
```

13) R_INTC_GetStatusGroup

Synopsis

Read the status of an interrupt request group.

Prototype

```
bool R_INTC_GetStatusGroup(
    uint8_t data1,      // Group selection
    uint32_t * data2    // Data storage location
);
```

Description

Read the grouped interrupt request status flags.

[data1]
The interrupt request group n to be configured.
For 64 and 48 pin packages, n must be 12; for other pin packages, n can be 0 or 12.

[data2]
The status flags shall be stored in the format below.

- Group 0

b31 – b2	B1	b0
0	CAN error interrupts Channel 1 0: Not requested 1: Requested	0

- Group 12

b31 – b7	b6	b5
0	SPI error	
	Channel 1	Channel 0
	0: Not requested 1: Requested	

b4	b3	b2	b1	b0
SCI reception error				
Channel 12	Channel 3	Channel 2	Channel 1	Channel 0
0: Not requested 1: Requested				

Return value

True if all parameters are valid; otherwise false.

Category

Interrupt control

References

R_INTC_ControlGroup

Remarks

- For 64 and 48 pin packages, SCI channel 2 and 3, and SPI channel 1 are not available.

Program example

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

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

void func(void)
{
    uint32_t Status_flags;

    /* Read the group 12 status flags */
    R_INTC_GetStatusGroup(
        12,
        &Status_flags
    );
}
```

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_6	Port P6	PDL_IO_PORT_C	Port PC
PDL_IO_PORT_1	Port P1	PDL_IO_PORT_7	Port P7	PDL_IO_PORT_D	Port PD
PDL_IO_PORT_2	Port P2	PDL_IO_PORT_8	Port P8	PDL_IO_PORT_E	Port PE
PDL_IO_PORT_3	Port P3	PDL_IO_PORT_9	Port P9	PDL_IO_PORT_F	Port PF
PDL_IO_PORT_4	Port P4	PDL_IO_PORT_A	Port PA	PDL_IO_PORT_G	Port PG
PDL_IO_PORT_5	Port P5	PDL_IO_PORT_B	Port PB		

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_0	Port pin $P0_0$	PDL_IO_PORT_6_0	Port pin $P6_0$	PDL_IO_PORT_C_0	Port pin PC_0
PDL_IO_PORT_0_1	Port pin $P0_1$	PDL_IO_PORT_6_1	Port pin $P6_1$	PDL_IO_PORT_C_1	Port pin PC_1
PDL_IO_PORT_0_2	Port pin $P0_2$	PDL_IO_PORT_6_2	Port pin $P6_2$	PDL_IO_PORT_C_2	Port pin PC_2
PDL_IO_PORT_0_3	Port pin $P0_3$	PDL_IO_PORT_6_3	Port pin $P6_3$	PDL_IO_PORT_C_3	Port pin PC_3
PDL_IO_PORT_0_4	Port pin $P0_4$	PDL_IO_PORT_6_4	Port pin $P6_4$	PDL_IO_PORT_C_4	Port pin PC_4
PDL_IO_PORT_0_5	Port pin $P0_5$	PDL_IO_PORT_6_5	Port pin $P6_5$	PDL_IO_PORT_C_5	Port pin PC_5
PDL_IO_PORT_1_0	Port pin $P1_0$	PDL_IO_PORT_7_0	Port pin $P7_0$	PDL_IO_PORT_D_0	Port pin PD_0
PDL_IO_PORT_1_1	Port pin $P1_1$	PDL_IO_PORT_7_1	Port pin $P7_1$	PDL_IO_PORT_D_1	Port pin PD_1
PDL_IO_PORT_1_2	Port pin $P1_2$	PDL_IO_PORT_7_2	Port pin $P7_2$	PDL_IO_PORT_D_2	Port pin PD_2
PDL_IO_PORT_1_3	Port pin $P1_3$	PDL_IO_PORT_7_3	Port pin $P7_3$	PDL_IO_PORT_D_3	Port pin PD_3
PDL_IO_PORT_1_4	Port pin $P1_4$	PDL_IO_PORT_7_4	Port pin $P7_4$	PDL_IO_PORT_D_4	Port pin PD_4
		PDL_IO_PORT_7_5	Port pin $P7_5$	PDL_IO_PORT_D_5	Port pin PD_5
PDL_IO_PORT_2_0	Port pin $P2_0$	PDL_IO_PORT_7_6	Port pin $P7_6$	PDL_IO_PORT_D_6	Port pin PD_6
PDL_IO_PORT_2_1	Port pin $P2_1$			PDL_IO_PORT_D_7	Port pin PD_7
PDL_IO_PORT_2_2	Port pin $P2_2$	PDL_IO_PORT_8_0	Port pin $P8_0$		
PDL_IO_PORT_2_3	Port pin $P2_3$	PDL_IO_PORT_8_1	Port pin $P8_1$	PDL_IO_PORT_E_0	Port pin PE_0
PDL_IO_PORT_2_4	Port pin $P2_4$	PDL_IO_PORT_8_2	Port pin $P8_2$	PDL_IO_PORT_E_1	Port pin PE_1
PDL_IO_PORT_2_5	Port pin $P2_5$			PDL_IO_PORT_E_2	Port pin PE_2
PDL_IO_PORT_2_6	Port pin $P2_6$	PDL_IO_PORT_9_0	Port pin $P9_0$	PDL_IO_PORT_E_3	Port pin PE_3
		PDL_IO_PORT_9_1	Port pin $P9_1$	PDL_IO_PORT_E_4	Port pin PE_4
PDL_IO_PORT_3_0	Port pin $P3_0$	PDL_IO_PORT_9_2	Port pin $P9_2$	PDL_IO_PORT_E_5	Port pin PE_5
PDL_IO_PORT_3_1	Port pin $P3_1$	PDL_IO_PORT_9_3	Port pin $P9_3$		
PDL_IO_PORT_3_2	Port pin $P3_2$	PDL_IO_PORT_9_4	Port pin $P9_4$	PDL_IO_PORT_F_0	Port pin PF_0
PDL_IO_PORT_3_3	Port pin $P3_3$	PDL_IO_PORT_9_5	Port pin $P9_5$	PDL_IO_PORT_F_1	Port pin PF_1
PDL_IO_PORT_3_4	Port pin $P3_4$	PDL_IO_PORT_9_6	Port pin $P9_6$	PDL_IO_PORT_F_2	Port pin PF_2
PDL_IO_PORT_3_5	Port pin $P3_5$			PDL_IO_PORT_F_3	Port pin PF_3
		PDL_IO_PORT_A_0	Port pin PA_0	PDL_IO_PORT_F_4	Port pin PF_4
PDL_IO_PORT_4_0	Port pin $P4_0$	PDL_IO_PORT_A_1	Port pin PA_1		
PDL_IO_PORT_4_1	Port pin $P4_1$	PDL_IO_PORT_A_2	Port pin PA_2	PDL_IO_PORT_G_0	Port pin PG_0
PDL_IO_PORT_4_2	Port pin $P4_2$	PDL_IO_PORT_A_3	Port pin PA_3	PDL_IO_PORT_G_1	Port pin PG_1
PDL_IO_PORT_4_3	Port pin $P4_3$	PDL_IO_PORT_A_4	Port pin PA_4	PDL_IO_PORT_G_2	Port pin PG_2
PDL_IO_PORT_4_4	Port pin $P4_4$	PDL_IO_PORT_A_5	Port pin PA_5	PDL_IO_PORT_G_3	Port pin PG_3
PDL_IO_PORT_4_5	Port pin $P4_5$	PDL_IO_PORT_A_6	Port pin PA_6	PDL_IO_PORT_G_4	Port pin PG_4
PDL_IO_PORT_4_6	Port pin $P4_6$			PDL_IO_PORT_G_5	Port pin PG_5
PDL_IO_PORT_4_7	Port pin $P4_7$	PDL_IO_PORT_B_0	Port pin PB_0	PDL_IO_PORT_G_6	Port pin PG_6
		PDL_IO_PORT_B_1	Port pin PB_1		
PDL_IO_PORT_5_0	Port pin $P5_0$	PDL_IO_PORT_B_2	Port pin PB_2		
PDL_IO_PORT_5_1	Port pin $P5_1$	PDL_IO_PORT_B_3	Port pin PB_3		
PDL_IO_PORT_5_2	Port pin $P5_2$	PDL_IO_PORT_B_4	Port pin PB_4		
PDL_IO_PORT_5_3	Port pin $P5_3$	PDL_IO_PORT_B_5	Port pin PB_5		
PDL_IO_PORT_5_4	Port pin $P5_4$	PDL_IO_PORT_B_6	Port pin PB_6		
PDL_IO_PORT_5_5	Port pin $P5_5$	PDL_IO_PORT_B_7	Port pin PB_7		
PDL_IO_PORT_5_6	Port pin $P5_6$				
PDL_IO_PORT_5_7	Port pin $P5_7$				

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.
if any pins of PORT 4, 5, 6 or C are selected,these pins will not be used as analog pins.

[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.
Specify PDL_NO_DATA if any pins of PORT 4, 5, 6 or C are selected for data1.

- 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	Select CMOS push-pull output or N-channel open-drain.

- Drive capacity control

PDL_IO_PORT_DRIVE_BUS_NORMAL or PDL_IO_PORT_DRIVE_BUS_HIGH or PDL_IO_PORT_DRIVE_SPI_NORMAL or PDL_IO_PORT_DRIVE_SPI_HIGH	Normal or high-current drive. Valid on packages with 100 pins or more. Multiple pins may be affected. Refer to the I/O Port DSCR1 and DSCR2 register descriptions in the hardware manual.

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 PE2 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.

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 P22 as an input port */
    R_IO_PORT_Set(
        PDL_IO_PORT_2_2,
        PDL_IO_PORT_INPUT
    );
}
```

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_DRIVE_BUS or	Drive capacity control (related to external bus pins) Valid on packages with 100 pins or more.
PDL_IO_PORT_DRIVE_SPI	Drive capacity control (related to SPI pins). Valid on packages with 100 pins or more.

[data3]
The address where the register value shall be stored, using one of the formats below.
Pin control:

b15 – b1	b0
0	0 or 1

Port (not open-drain) control:

b15 – b8	b7 – b0
0	Register

Port (open-drain) control:

b15 – b8	b7 – b0
Register ODR1	Register ODR0

Port (Driving Ability) control: for option PDL_IO_PORT_DRIVE_BUS

b15 – b8	b7 – b0
0	Register DSCR1

Port (Driving Ability) control: for option PDL_IO_PORT_DRIVE_SPI

b15 – b8	b7 – b0
0	Register DSCR2

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.
- If port is input for the data1, PDL_IO_PORT_DRIVE_BUS or PDL_IO_PORT_DRIVE_SPI is selected for data2, data3 will read whole register value for Driving Ability Control Register. (Refer to register DSCR1 and DSCR2 description in hardware manual.)

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 B */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_B,
        PDL_IO_PORT_DIRECTION,
        &result
    );

    /* Read the output type for pin P30 */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_3_0,
        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_DRIVE_BUS or	Drive capacity control (related to external bus pins) Valid on packages with 100 pins or more.
PDL_IO_PORT_DRIVE_SPI	Drive capacity control (related to SPI pins). Valid on packages with 100 pins or more.

- 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 control:

b15 – b1	b0
Do not care	0 or 1

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

Port (Driving Ability) control: for option PDL_IO_PORT_DRIVE_BUS

b15 – b8	b7 – b0
Do not care	Register DSCR1

Port (Driving Ability) control: for option PDL_IO_PORT_DRIVE_SPI

b15 – b8	b7 – b0
Do not care	Register DSCR2

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 PE2 is fixed as an input and can only be modified by option PDL_IO_PORT_MODE.
- If port is input for the data1, PDL_IO_PORT_DRIVE_BUS or PDL_IO_PORT_DRIVE_SPI is selected for data2, data3 will modify whole register value for Driving Ability Control Register. (Refer to register DSCR1 and DSCR2 description in hardware manual.)

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 PB to output */
    R_IO_PORT_ModifyControl(
        PDL_IO_PORT_B,
        PDL_IO_PORT_DIRECTION | PDL_IO_PORT_OR,
        0x0F
    );
}
```

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 P22 */
    R_IO_PORT_Read(
        PDL_IO_PORT_2_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 P22 */
    R_IO_PORT_Write(
        PDL_IO_PORT_2_2,
        0
    );

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

6) R_IO_PORT_Compare

Synopsis

Check the pin states on an I/O port.

Prototype

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

Description

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

[data1]
Use either one of the following definition values (from §4.2.3):
- One port definition or
- One port pin definition.

[data2]
The value to be compared with; Between 0x00 and 0xFF for a port, 0 or 1 for a pin.

[func]
The function to be called if a match occurs.

Return value

True if the parameters are valid; otherwise false.

Category

I/O port

References

R_IO_PORT_Set

Remarks

- If an invalid port or pin is specified, the operation of the function cannot be guaranteed.
- 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){}
void IoHandler2(void){}

void func( void )
{
    /* Call function IoHandler1 if port pin P22 is high */
    R_IO_PORT_Compare(
        PDL_IO_PORT_2_2,
        1,
        IoHandler1
    );

    /* Call function IoHandler2 if port 4 reads as 0x55 */
    R_IO_PORT_Compare(
        PDL_IO_PORT_4,
        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; 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 )
{
    /* Invert port pin P22 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_2_2,
        PDL_IO_PORT_XOR,
        1
    );

    /* And the value port 4 with 0x55 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_4,
        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 P22 reads as 0 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_2_2,
        0
    );

    /* Wait until port 4 reads as 0x55 */
    R_IO_PORT_Wait(
        PDL_IO_PORT_4,
        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

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_P00PFS	PDL_MPC_REG_P60PFS	PDL_MPC_REG_PC0PFS
PDL_MPC_REG_P01PFS	PDL_MPC_REG_P61PFS	PDL_MPC_REG_PC1PFS
PDL_MPC_REG_P02PFS	PDL_MPC_REG_P62PFS	PDL_MPC_REG_PC2PFS
PDL_MPC_REG_P03PFS	PDL_MPC_REG_P63PFS	PDL_MPC_REG_PC3PFS
PDL_MPC_REG_P10PFS	PDL_MPC_REG_P64PFS	PDL_MPC_REG_PC4PFS
PDL_MPC_REG_P11PFS	PDL_MPC_REG_P65PFS	PDL_MPC_REG_PC5PFS
PDL_MPC_REG_P12PFS	PDL_MPC_REG_P70PFS	PDL_MPC_REG_PD0PFS
PDL_MPC_REG_P13PFS	PDL_MPC_REG_P71PFS	PDL_MPC_REG_PD1PFS
PDL_MPC_REG_P14PFS	PDL_MPC_REG_P72PFS	PDL_MPC_REG_PD2PFS
PDL_MPC_REG_P20PFS	PDL_MPC_REG_P73PFS	PDL_MPC_REG_PD3PFS
PDL_MPC_REG_P21PFS	PDL_MPC_REG_P74PFS	PDL_MPC_REG_PD4PFS
PDL_MPC_REG_P22PFS	PDL_MPC_REG_P75PFS	PDL_MPC_REG_PD5PFS
PDL_MPC_REG_P23PFS	PDL_MPC_REG_P76PFS	PDL_MPC_REG_PD6PFS
PDL_MPC_REG_P24PFS	PDL_MPC_REG_P80PFS	PDL_MPC_REG_PD7PFS
PDL_MPC_REG_P25PFS	PDL_MPC_REG_P81PFS	PDL_MPC_REG_PE0PFS
PDL_MPC_REG_P26PFS	PDL_MPC_REG_P82PFS	PDL_MPC_REG_PE1PFS
PDL_MPC_REG_P30PFS	PDL_MPC_REG_P90PFS	PDL_MPC_REG_PE2PFS
PDL_MPC_REG_P31PFS	PDL_MPC_REG_P91PFS	PDL_MPC_REG_PE3PFS
PDL_MPC_REG_P32PFS	PDL_MPC_REG_P92PFS	PDL_MPC_REG_PE4PFS
PDL_MPC_REG_P33PFS	PDL_MPC_REG_P93PFS	PDL_MPC_REG_PE5PFS
PDL_MPC_REG_P34PFS	PDL_MPC_REG_P94PFS	PDL_MPC_REG_PF2PFS
PDL_MPC_REG_P35PFS	PDL_MPC_REG_P95PFS	PDL_MPC_REG_PF3PFS
PDL_MPC_REG_P40PFS	PDL_MPC_REG_P96PFS	PDL_MPC_REG_PG0PFS
PDL_MPC_REG_P41PFS	PDL_MPC_REG_PA0PFS	PDL_MPC_REG_PG1PFS
PDL_MPC_REG_P42PFS	PDL_MPC_REG_PA1PFS	PDL_MPC_REG_PG2PFS
PDL_MPC_REG_P43PFS	PDL_MPC_REG_PA2PFS	PDL_MPC_REG_PG3PFS
PDL_MPC_REG_P44PFS	PDL_MPC_REG_PA3PFS	PDL_MPC_REG_PG4PFS
PDL_MPC_REG_P45PFS	PDL_MPC_REG_PA4PFS	PDL_MPC_REG_PG5PFS
PDL_MPC_REG_P46PFS	PDL_MPC_REG_PA5PFS	PDL_MPC_REG_PG6PFS
PDL_MPC_REG_P47PFS	PDL_MPC_REG_PA6PFS	
PDL_MPC_REG_P50PFS	PDL_MPC_REG_PB0PFS	
PDL_MPC_REG_P51PFS	PDL_MPC_REG_PB1PFS	
PDL_MPC_REG_P52PFS	PDL_MPC_REG_PB2PFS	
PDL_MPC_REG_P53PFS	PDL_MPC_REG_PB3PFS	
PDL_MPC_REG_P54PFS	PDL_MPC_REG_PB4PFS	
PDL_MPC_REG_P55PFS	PDL_MPC_REG_PB5PFS	
PDL_MPC_REG_P56PFS	PDL_MPC_REG_PB6PFS	
PDL_MPC_REG_P57PFS	PDL_MPC_REG_PB7PFS	

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

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 PFCSE */
    R_MPC_Read(
        PDL_MPC_REG_P40PFS,
        &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 PD3PFS */
    R_MPC_Write(
        PDL_MPC_REG_PD3PFS,
        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 7 in P72PFS to 1 */
    R_MPC_Modify(
        PDL_MPC_REG_P72PFS,
        PDL_MPC_OR,
        0x80
    );
}
```

4.2.5. MCU operation

1) R_MCU_Control

Synopsis

Control the operation of the MCU.

Prototype

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

Description

Modify the MCU control registers.

[data]
Select the operation states. All selections are optional.
If multiple selections are required, use "|" to separate each selection.

- On-chip ROM control

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

- 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

None.

Remarks

- None.

Program example

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

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

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

2) R_MCU_GetStatus

Synopsis

Read the MCU status.

Prototype

```
bool R_MCU_GetStatus(
    uint16_t * 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.

For packages with 100 pins or more in b15~b8

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

For packages with 64-pins and 48-pins in b15~b8

b15 – b8
0

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 – b14	b8
0	Start type 0: Cold 1: Warm

b7	b6	b5	b4	b3	b2	b1	b0
Reset detection flags (0: not detected; 1: detected)							
Exit from deep software standby	Software	WDT	IWDT	Voltage monitor			1: Power-on
				2	1	0	

[data3]
Where the OFS0 register contents shall be stored.
Specify PDL_NO_PTR if they are not required.

[data4]
Where the OFS1 register contents shall be stored.
Specify PDL_NO_PTR if they are not required.

Return value

True.

Category

MCU registers

References

None.

Remarks

- If a reset status 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,    // WDT configuration options
    uint32_t data3     // LVD 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 percent 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 percent 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 WDT configuration settings.
If multiple selections are required, use "|" to separate each selection.

- Auto-start control

PDL_MCU_OFS_WDT_HALTED or PDL_MCU_OFS_WDT_AUTOSTART	Disable or enable the WDT auto-start mode.

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

- Timeout period

PDL_MCU_OFS_WDT_TIMEOUT_1024 or PDL_MCU_OFS_WDT_TIMEOUT_4096 or PDL_MCU_OFS_WDT_TIMEOUT_8192 or PDL_MCU_OFS_WDT_TIMEOUT_16384	Timeout period specified in cycles of the divided clock as specified in the Clock Selection below.

- Clock division

PDL_MCU_OFS_WDT_CLOCK_PCLK_4 or PDL_MCU_OFS_WDT_CLOCK_PCLK_64 or PDL_MCU_OFS_WDT_CLOCK_PCLK_128 or PDL_MCU_OFS_WDT_CLOCK_PCLK_512 or PDL_MCU_OFS_WDT_CLOCK_PCLK_2048 or PDL_MCU_OFS_WDT_CLOCK_PCLK_8192	The selected clock. The PCLKB ÷ 4, 64, 128, 512, 2048 or 8192.

- Window end position

PDL_MCU_OFS_WDT_WIN_END_75 or PDL_MCU_OFS_WDT_WIN_END_50 or PDL_MCU_OFS_WDT_WIN_END_25 or PDL_MCU_OFS_WDT_WIN_END_0	The window end position specified as a percent 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_WDT_WIN_START_25 or PDL_MCU_OFS_WDT_WIN_START_50 or PDL_MCU_OFS_WDT_WIN_START_75 or PDL_MCU_OFS_WDT_WIN_START_100	The window start position specified as a percent 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_WDT_NMI or PDL_MCU_OFS_WDT_RESET	Select an NMI or reset when the WDT down-counter underflows.

[data3]
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.

Category

MCU registers

References

R_IWDT_Set, R_WDT_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. */
/* Leave the WDT disabled. */
/* Enable reset at Vdet0. */
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_WDT_HALTED,
    PDL_MCU_OFS_LVD_0_ENABLE
);
```

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
    uint8_t data2,      // Monitor 1 Voltage selection
    uint16_t data3,     // Monitor 2 Configuration selection
    uint8_t data4       // Monitor 2 Voltage selection
);
```

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	Select no action, a reset on low voltage detection or non-maskable 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.
For device packages with 48 or 64 pins this fixed at 2.95V, specify PDL_NO_DATA.

- Voltage levels applicable for 3V products. Specify PDL_NO_DATA to use the default

PDL_LVD_VOLTAGE_LEVEL_288 or PDL_LVD_VOLTAGE_LEVEL_285 or PDL_LVD_VOLTAGE_LEVEL_290	Set the voltage detection level.

- Voltage levels applicable for 5V products. Specify PDL_NO_DATA to use the default

PDL_LVD_VOLTAGE_LEVEL_450 or PDL_LVD_VOLTAGE_LEVEL_423 or PDL_LVD_VOLTAGE_LEVEL_477	Set the voltage detection level.

[data3]
Monitor 2 voltage detection configurations.
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	Select no action, a reset on low voltage detection or non-maskable interrupt when a specified voltage event is detected.

Description(1/2)

- 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.

[data4]
Monitor 2 voltage detection levels.
For device packages with 48 or 64 pins this fixed at 2.95V, specify PDL_NO_DATA.

- Voltage levels applicable for 3V products. Specify PDL_NO_DATA to use the default

PDL_LVD_VOLTAGE_LEVEL_288 or PDL_LVD_VOLTAGE_LEVEL_285 or PDL_LVD_VOLTAGE_LEVEL_290	Set the voltage detection level.

- Voltage levels applicable for 5V products. Specify PDL_NO_DATA to use the default

PDL_LVD_VOLTAGE_LEVEL_450 or PDL_LVD_VOLTAGE_LEVEL_423 or PDL_LVD_VOLTAGE_LEVEL_477	Set the voltage detection level.

Return value

True if the parameters are valid; otherwise false.

Category

Voltage detection circuit

References

R_INTC_CreateExtInterrupt, R_CGC_Set, R_LPC_Create, R_CGC_Control

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.
- 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.
- Ensure Monitor 1 and 2 are in PDL_LVD_MONITOR_ONLY during flash memory programming / erasure.
- Disable the digital filter circuit when using voltage monitoring 1 and 2 circuit in software standby mode or deep software standby mode.
- Do not use the voltage detection 1 and 2 circuit in deep software standby mode, with PDL_LPC_DEEPCUT_RAM_USB_LVD. See function R_LPC_Create.
- If using both LVD1 and LVD2, 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 2.95 V*/
    R_LVD_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_LVD_INTERRUPT_NMI_DETECT_FALL | PDL_LVD_FILTER_DISABLE,
        PDL_NO_DATA
    );
}
```

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
0	Monitor 2		0	Monitor 1	
	Status	Change		Status	Change
	0: VCC < Vdet2 1: VCC ≥ Vdet2, or the monitor is disabled	0: None 1: Detected		0: VCC < Vdet1 1: VCC ≥ Vdet1, or the monitor is disabled.	0: None 1: Detected

Return value

True.

Category

Voltage detection circuit

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
    void* func2,       // Callback function
    void* func3,       // Callback function
    uint8_t data6      // 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_PCLK or PDL_CAC_REFERENCE_IWDTLOCO or PDL_CAC_REFERENCE_CACREF	Select the Main clock oscillator, Peripheral module clock (PCLKB), 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 applicable when the CACREF input is selected as the reference signal.

- Measured clock selection and division

PDL_CAC_MEASURE_MAIN PDL_CAC_MEASURE_PCLK or PDL_CAC_MEASURE_IWDTLOCO	Select the Main clock oscillator, Peripheral module clock (PCLKB) 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.

[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_0_0 or PDL_CAC_CACREF_PORT_0_1 or PDL_CAC_CACREF_PORT_2_3 or PDL_CAC_CACREF_PORT_B_3	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.

Description (2/2)

[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.

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

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

[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 parameters func1, func2 and 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%.
- The PDL_CAC_CACREF_PORT_0_0 pin is valid on packages with 100 pins or more.
- The PDL_CAC_CACREF_PORT_0_1 pin is valid on packages with 64 pins.
- There is a CACREF pin restriction when using the 48 pin device package.
- After applying the selected dividers, the clock to be measured, divided by the reference clock, must be > 0 and < h'10000.
  Furthermore, if specifying a permissible deviation the resulting permissible range must also fit within these limits. This function will return false if this is not the case.
- 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 IWDTLOCO accuracy */
    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,
        PDL_NO_DATA,
        10,
        10,
        CAC_frequency_error,
        CAC_measurement_complete,
        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_PCLK or PDL_CAC_REFERENCE_IWDTLOCO or PDL_CAC_REFERENCE_CACREF	Select the Main clock oscillator, Peripheral module clock, 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_PCLK or PDL_CAC_MEASURE_IWDTLOCO	Select the Main clock oscillator, Peripheral module clock or IWDT low-speed on-chip oscillator for measurement.

- 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.

Description (2/2)

- 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 pin selection and 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(
    uint8_t data1,     // Configuration options
    uint32_t data2,    // Select deep standby interrupt
    uint32_t data3,    // Select deep standby interrupt
    uint16_t data4,    // Main oscillator waiting times
    uint16_t data5     // PLL waiting times
);
```

Description (1/2)

Load the registers that control module or CPU operation.

[data1]
Select the required settings.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- I/O port retention control.

PDL_LPC_IO_SAME or PDL_LPC_IO_DELAY	Select whether I/O port retention is cancelled when deep software standby mode is ended, or when CPU operation has resumed.

- Output port retention control.
  This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

PDL_LPC_OUTPUT_PORT_RETAINED or PDL_LPC_OUTPUT_PORT_NOT_RETAINED	Select whether the address bus and bus control signals retain the output state in software standby mode or deep software standby mode.

[data2]
Select the interrupt to cancel deep software standby mode.
The default settings are shown in bold. Specify PDL_NO_DATA to use the defaults.

- Deep software standby cancel control.
  IRQ6-DS pin and IRQ7-DS pin are supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

PDL_LPC_CANCEL_IRQ0_DISABLE or PDL_LPC_CANCEL_IRQ0_FALLING or PDL_LPC_CANCEL_IRQ0_RISING	Prevent or allow an edge on the IRQ0-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ1_DISABLE or PDL_LPC_CANCEL_IRQ1_FALLING or PDL_LPC_CANCEL_IRQ1_RISING	Prevent or allow an edge on the IRQ1-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ2_DISABLE or PDL_LPC_CANCEL_IRQ2_FALLING or PDL_LPC_CANCEL_IRQ2_RISING	Prevent or allow an edge on the IRQ2-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ3_DISABLE or PDL_LPC_CANCEL_IRQ3_FALLING or PDL_LPC_CANCEL_IRQ3_RISING	Prevent or allow an edge on the IRQ3-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ4_DISABLE or PDL_LPC_CANCEL_IRQ4_FALLING or PDL_LPC_CANCEL_IRQ4_RISING	Prevent or allow an edge on the IRQ4-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ5_DISABLE or PDL_LPC_CANCEL_IRQ5_FALLING or PDL_LPC_CANCEL_IRQ5_RISING	Prevent or allow an edge on the IRQ5-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ6_DISABLE or PDL_LPC_CANCEL_IRQ6_FALLING or PDL_LPC_CANCEL_IRQ6_RISING	Prevent or allow an edge on the IRQ6-DS pin to cancel deep software standby mode.
PDL_LPC_CANCEL_IRQ7_DISABLE or PDL_LPC_CANCEL_IRQ7_FALLING or PDL_LPC_CANCEL_IRQ7_RISING	Prevent or allow an edge on the IRQ7-DS pin to cancel deep software standby mode.

Description (2/2)

[data3]
Select the interrupt to cancel deep software standby mode.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Deep software standby cancel control

PDL_LPC_CANCEL_LVD1_DISABLE or PDL_LPC_CANCEL_LVD1_FALLING or PDL_LPC_CANCEL_LVD1_RISING	Prevent or allow an edge on the LVD1 pin to cancel deep software standby mode.
PDL_LPC_CANCEL_LVD2_DISABLE or PDL_LPC_CANCEL_LVD2_FALLING or PDL_LPC_CANCEL_LVD2_RISING	Prevent or allow an edge on the LVD2 pin to cancel deep software standby mode.
PDL_LPC_CANCEL_NMI_DISABLE or PDL_LPC_CANCEL_NMI_FALLING or PDL_LPC_CANCEL_NMI_RISING	Prevent or allow an edge on the NMI pin to cancel deep software standby mode.

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

- Software Standby 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_64 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.

[data5]
Select the PLL waiting times.
If no selections are required, specify PDL_NO_DATA.

- Deep Software Standby waiting time

PDL_LPC_PLL_16 or PDL_LPC_PLL_32 or PDL_LPC_PLL_64 or PDL_LPC_PLL_512 or PDL_LPC_PLL_1024 or PDL_LPC_PLL_2048 or PDL_LPC_PLL_4096 or PDL_LPC_PLL_16384 or PDL_LPC_PLL_32768 or PDL_LPC_PLL_65536 or PDL_LPC_PLL_131072 or PDL_LPC_PLL_262144 or PDL_LPC_PLL_524288 or PDL_LPC_PLL_1048576 or PDL_LPC_PLL_2097152 or PDL_LPC_PLL_4194304	Select the oscillation settling time of the PLL before the CPU resumes after exiting from software standby mode. When updating this value, the PLL circuit 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, R_LPC_Create

Remarks

- If PDL_LPC_IO_DELAY is specified, use R_LPC_Control with the PDL_LPC_IO_RELEASE option to cancel the I/O port state retention.
- The IRQn-DS pins are the only IRQ pins that can be used to exit from deep software standby mode.
- When the flash memory is in program or erase mode, do not call this function if it will result in the power mode changing. This function will return false is this situation.
- 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.
- If the NMI pin is enabled for cancelling deep software standby mode, it cannot be disabled.
- Use R_CGC_Control to stop and start the clocks as required.

Program example

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

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

void func( void )
{
    /*Allow a falling edge on IRQ2-DS to cancel deep software standby*/
    R_LPC_Create(
        PDL_NO_DATA,
        PDL_LPC_CANCEL_IRQ2_FALLING,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

2) R_LPC_Control

Synopsis

Select a low power consumption mode.

Prototype

```
bool R_LPC_Control(
    uint32_t data       // Mode selection
);
```

Description

Transition to one of the low power modes.

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

- Mode selection

PDL_LPC_MODE_SLEEP or PDL_LPC_MODE_ALL_MODULE_CLOCK_STOP or PDL_LPC_MODE_SOFTWARE_STANDBY or PDL_LPC_MODE_DEEP_SOFTWARE_STANDBY	Select the mode to be entered. Check the Remarks section for any restrictions.

- I/O port retention cancellation

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

Return value

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

Category

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 or deep software standby mode, the oscillation stop detection function is disabled. The detection is re-enabled if software standby mode is interrupted.
- On exit from deep software standby mode, the MCU is reset.
- Do not set up the DMACA 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 DMACA 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 the flash memory is in program or erase mode, do not call this function if it will result in the power mode changing. This function will return false is this situation.

Program example

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

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

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

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

3) R_LPC_WriteBackup

Synopsis

Write to the Backup registers.

Prototype

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

Description

Write data into the backup registers.

[data1]
The data to be written to the backup area.

[data2]
The number of bytes to be written to the backup area. Valid from 1 to 32.

Return value

True if all parameters are valid; otherwise false.

Category

LPC

References

None.

Remarks

- The definition R_PDL_LPC_BACKUP_AREA_SIZE specifies the number of bytes that are available

Program example

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

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

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

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

4) R_LPC_ReadBackup

Synopsis

Read from the Backup registers.

Prototype

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

Description

Read data from the backup registers.

[data1]
The storage area for the data read from the backup area.

[data2]
The number of bytes to be read from the backup area. Valid from 1 to 32.

Return value

True if all parameters are valid; otherwise false.

Category

LPC

References

R_LPC_WriteBackup

Remarks

- The definition R_PDL_LPC_BACKUP_AREA_SIZE specifies the number of bytes that are available

Program example

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

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

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

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

5) R_LPC_GetStatus

Synopsis

Read the status flags.

Prototype

```
bool R_LPC_GetStatus(
    uint32_t * data      // Data pointer
);
```

Description

Read the Low power status flags.

[data]
The status flags shall be stored in the format below.
Interrupt flags of IRQ6-DS pin and IRQ7-DS pin are supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

b31 – b24
0

b23	b22 – b20	b19	b18	b17	b16
Event detection flags (0: not detected; 1: detected)					
An interrupt has caused an exit from deep software standby mode, followed by an internal reset	0	LVD2	LVD1	LVD0	Power-on reset

b15 - b13	b12	b11 - b10	b9	b8
Deep Software Standby cancel request detection				
0: No activity 1: The exit from deep software standby was caused by one of the following signals.				
0	NMI	0	LVD2	LVD1

b7	b6	b5	b4	b3	b2	b1	b0
Deep Software Standby cancel request detection							
0: No activity 1: The exit from deep software standby was caused by one of the following signals.							
IRQ7-DS	IRQ6-DS	IRQ5-DS	IRQ4-DS	IRQ3-DS	IRQ2-DS	IRQ1-DS	IRQ0-DS

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.

Program example

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

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

void func( void )
{
    uint32_t status_flags;

    /* Find out what caused the exit from deep software standby */
    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 LPC, Main clock oscillator forced oscillation, 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

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
	LVD		Mode and Reset	CGC
0	0: Write Disabled 1: Write Enabled	0	0: Write Disabled 1: Write Enabled	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	PFSWE	
0: Writing to the PFSWE bit is enabled 1: Writing to the PFSWE bit is disabled	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 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_PB23_MB2 or PDL_BSC_PRIORITY_PB23_MB1	Peripheral 2 and 3	
PDL_BSC_PRIORITY_PB45_MB2 or PDL_BSC_PRIORITY_PB45_MB1	Peripheral 4 and 5	
PDL_BSC_PRIORITY_PB6_MB2 or PDL_BSC_PRIORITY_PB6_MB1	Peripheral 6	
PDL_BSC_PRIORITY_EB_MB2 or PDL_BSC_PRIORITY_EB_MB1	External	Fixed to external main bus 2, or toggled with external main bus 1.

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.
- External Bus is not valid on packages with 64-pin and 48-pin package

Program example

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

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

void func(void)
{
    /* Give internal main bus 1 priority access to the internal peripheral
bus 1 */
    R_BSC_Set(
          PDL_BSC_PRIORITY_PB23_MB1
    );
}
```

2) R_BSC_Create

Synopsis

Configure the external bus operation.

Prototype

```
bool R_BSC_Create(
        uint32_t data1,     // Bus control pin selection
        uint32_t data2,     // Bus address pin selection
        uint16_t data3,     // Recovery cycle insertion
        uint8_t data4,      // Error control
        void * func,        // Callback function
        uint8_t data5       // Interrupt priority level
);
```

Description (1/2)

Configure the I/O pins, cycle insertion, error detection and register the callback function

[data1]
Configure the bus control signals. Use "|" to separate each selection.
Ignored for the 64-pin and 48-pin device package (specify PDL_NO_DATA).

- Chip select pin selection (only required for each external memory area that will be enabled).
- Please refer to Table 21.39 at the "Multifunction Pin Controller (MPC)" section in the RX63T Hardware Manual for details of pin-package.

PDL_BSC_CS0_P26 or PDL_BSC_CS0_PD1	Select the port pin to be used for signal CS0#.
PDL_BSC_CS1_P00 or PDL_BSC_CS1_P25 or PDL_BSC_CS1_PF2	Select the port pin to be used for signal CS1#.
PDL_BSC_CS2_PD2 or PDL_BSC_CS2_PG6or PDL_BSC_CS2_P05	Select the port pin to be used for signal CS2#.
PDL_BSC_CS3_P12 or PDL_BSC_CS3_PF4 or PDL_BSC_CS3_PA6	Select the port pin to be used for signal CS3#.

- WAIT pin selection (only required if the WAIT# signal is to be used).

PDL_BSC_WAIT_P05 or PDL_BSC_WAIT_P82 or PDL_BSC_WAIT_PE0	Select the port pin to be used for signal WAIT#. NOTE: PDL_BSC_WAIT_P05 is only available in the 144-pin and 112-pin package.

- ALE signal control (only required if the ALE signal is to be used).

PDL_BSC_ALE_ENABLE	Enable the ALE signal on pin P11.

[data2]

- Address output control.
  The signals are **enabled** by default, unless the pin is allocated to a bus control signal.
  If multiple selections are required, use "|" to separate each selection.
  Specify PDL_NO_DATA for no change.
  Ignored for the 64-pin and 48-pin device package (specify PDL_NO_DATA).

PDL_BSC_A7_A0_DISABLE	Disable the output of the A7 to A0 signals.
PDL_BSC_A8_DISABLE	Disable the output of the A8 signal.
PDL_BSC_A9_DISABLE	Disable the output of the A9 signal.
PDL_BSC_A10_DISABLE	Disable the output of the A10 signal.
PDL_BSC_A11_DISABLE	Disable the output of the A11 signal.
PDL_BSC_A12_DISABLE	Disable the output of the A12 signal.
PDL_BSC_A13_DISABLE	Disable the output of the A13 signal.
PDL_BSC_A14_DISABLE	Disable the output of the A14 signal.
PDL_BSC_A15_DISABLE	Disable the output of the A15 signal.
PDL_BSC_A16_DISABLE	Disable the output of the A16 signal.
PDL_BSC_A17_DISABLE	Disable the output of the A17 signal.
PDL_BSC_A18_DISABLE	Disable the output of the A18 signal.
PDL_BSC_A19_DISABLE	Disable the output of the A19 signal.

Description (2/2)

[data3]

- Recovery cycle insertion control.
  The controls are disabled by default. Specify PDL_NO_DATA to use the defaults.
  If multiple selections are required, use "|" to separate each selection.
  Ignored for the 64-pin and 48-pin device package (specify PDL_NO_DATA).

	Bus type	Bus access		Area
		Current	Next	
PDL_BSC_RCV_SRRS_ENABLE	Separate	Read	Read	Same
PDL_BSC_RCV_SRRD_ENABLE				Different
PDL_BSC_RCV_SRWS_ENABLE			Write	Same
PDL_BSC_RCV_SRWD_ENABLE				Different
PDL_BSC_RCV_SWRS_ENABLE		Write	Read	Same
PDL_BSC_RCV_SWRD_ENABLE				Different
PDL_BSC_RCV_SWWS_ENABLE			Write	Same
PDL_BSC_RCV_SWWD_ENABLE				Different
PDL_BSC_RCV_MRRS_ENABLE	Multiplexed	Read	Read	Same
PDL_BSC_RCV_MRRD_ENABLE				Different
PDL_BSC_RCV_MRWS_ENABLE			Write	Same
PDL_BSC_RCV_MRWD_ENABLE				Different
PDL_BSC_RCV_MWRS_ENABLE		Write	Read	Same
PDL_BSC_RCV_MWRD_ENABLE				Different
PDL_BSC_RCV_MWWS_ENABLE			Write	Same
PDL_BSC_RCV_MWWD_ENABLE				Different

[data4]

- Error monitoring

PDL_BSC_ERROR_ILLEGAL_ADDRESS_DISABLE or PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE	Disable or enable illegal address access detection.
PDL_BSC_ERROR_TIME_OUT_DISABLE or PDL_BSC_ERROR_TIME_OUT_ENABLE	Disable or enable bus time-out detection.

[func]

The function to be called when a bus error occurs. Specify 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 parameter func.

Return value

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

Category

Bus Controller

Reference

R_BSC_Set, R_BSC_CreateArea, 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.
- External Bus is not valid on packages with 64-pin and 48-pin package.
- Call this function after all calls of function R_BSC_CreateArea.
- After calling this function, use, R_BSC_Control to start the external bus operation.
- Multifunction Pin Control registers are modified by this function.

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)
{
    /* Select CS2 on pin PD2, all address signals, enable interrupts and
register the callback function */
    R_BSC_Create(
          PDL_BSC_CS2_PD2,
          PDL_NO_DATA,
          PDL_NO_DATA,
          PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE | \
          PDL_BSC_ERROR_TIME_OUT_ENABLE,
          BusErrorFunc,
          5
    );
}
```

3) R_BSC_CreateArea

Synopsis

Configure an external bus area.

Prototype

```
bool R_BSC_CreateArea(
        uint8_t data1,      // Area selection
        uint16_t data2,     // Configuration selection
        uint8_t data3,      // RRCV cycles
        uint8_t data4,      // WRCV cycles
        uint8_t data5,      // CSPRWAIT cycles
        uint8_t data6,      // CSPWWAIT cycles
        uint8_t data7,      // CSRWAIT cycles
        uint8_t data8,      // CSWWAIT cycles
        uint8_t data9,      // CSROFF cycles
        uint8_t data10,     // CSWOFF cycles
        uint8_t data11,     // WDOFF cycles
        uint8_t data12,     // AWAIT cycles
        uint8_t data13,     // RDON cycles
        uint8_t data14,     // WRON cycles
        uint8_t data15,     // WDON cycles
        uint8_t data16      // CSON cycles
);
```

Description (1/2)

Set up an external bus area.

[data1]
The address area n (where n = 0 to 3).

[data2]
Configure the operation of area CSn.
If multiple selections are required, use “|” to separate each selection.
The default settings are shown in **bold**.

- External bus width

PDL_BSC_WIDTH_8 or PDL_BSC_WIDTH_16	Select 8, 16 bit data bus width.

- Endian mode

PDL_BSC_ENDIAN_SAME or PDL_BSC_ENDIAN_OPPOSITE	Set the bus endian mode to be the same or opposite to that of the CPU.

- Multiplexed mode

PDL_BSC_SEPARATE or PDL_BSC_MULTIPLEXED	Select separate or multiplexed address and data pins.

- Write access mode

PDL_BSC_WRITE_BYTE or PDL_BSC_WRITE_SINGLE	Select byte or single write strobe mode.

- External wait control

PDL_BSC_WAIT_DISABLE or PDL_BSC_WAIT_ENABLE	Disable or enable external wait control (using the WAIT# signal).

- Page access control

PDL_BSC_PAGE_READ_DISABLE or PDL_BSC_PAGE_READ_NORMAL or PDL_BSC_PAGE_READ_CONTINUOUS	Disable or enable page read accesses using normal access compatible mode or continuous assertion mode.
PDL_BSC_PAGE_WRITE_DISABLE or PDL_BSC_PAGE_WRITE_ENABLE	Disable or enable page write accesses.

[data3]
The number of read recovery cycles (RRCV). Valid between 0 and 15.

[data4]
The number of write recovery cycles (WRCV). Valid between 0 and 15.

Description (2/2)

[data5]
The number of wait cycles used for second and subsequent accesses during a page read sequence (CSPRWAIT). Valid between 0 and 7.

[data6]
The number of wait cycles used for second and subsequent accesses during a page write sequence (CSPWWAIT). Valid between 0 and 7.

[data7]
The number of wait cycles for the first access during a normal or page read sequence (CSRWAIT). Valid between 0 and 31.

[data8]
The number of wait cycles for the first access during a normal or page write sequence (CSWWAIT). Valid between 0 and 31.

[data9]
The number of cycles that the CS signal is left asserted after the read strobe is negated (CSROFF). Valid between 0 and 7.

[data10]
The number of cycles that the CS signal is left asserted after the write strobe is negated (CSWOFF). Valid between 0 and 7.

[data11]
The number of cycles that the data output is left asserted after the write strobe is negated (WDOFF). Valid between 0 and 7.

[data12]
The number of wait cycles to be inserted into a multiplexed address output cycle (AWAIT). Valid between 0 and 3.

[data13]
The number of cycles before the read strobe is asserted (RDON). Valid between 0 and 7.

[data14]
The number of cycles before the write strobe is asserted (WRON). Valid between 0 and 7.

[data15]
The number of cycles before the write data is output (WDON). Valid between 0 and 7.

[data16]
The number of cycles before the chip select is asserted (CSON). Valid between 0 and 7.

Return value

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

Category

Bus Controller

Reference

R_BSC_Create

Remarks

- Use this function to set up each required area and then call R_BSC_Create.
- This function is not required when using 64-pin and 48-pin package.
- The endian mode of the CPU is selected by the MDE bits in the MDES or MDEB registers.
- Multifunction Pin Control registers are modified by this function.
- The cycle count parameters are not checked for validity. Use the hardware manual to check these values.
- Setting single write strobe mode is prohibited in the 8-bit bus space.

Program example

```
/* RPDL definitions */
#include “r_pdl_bsc.h”

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

void func(void)
{
    /* Configure CS2: 8-bit width, maximum cycle counts */
    R_BSC_CreateArea(
         2,
         PDL_BSC_WIDTH_8,
         15,
         15,
         7,
         7,
         31,
         31,
         7,
         7,
         7,
         3,
         7,
         7,
         7,
         7
    );
}
```

4) R_BSC_Destroy

Synopsis

Stop the External Bus Controller.

Prototype

```
bool R_BSC_Destroy(
        uint8_t data     // Area selection
);
```

Description

Disable an external bus area.

[data]
Select the external bus area CSn (where n = 0 to 3) to be disabled.

Return value

True.

Category

Bus Controller

Reference

R_BSC_Control

Remarks

- The bus error interrupt request will not be disabled by this function. Use R_BSC_Control to disable it.
- Multifunction Pin Control registers are modified by this function.
- This function is not required when using 64-pin and 48-pin package..

Program example

```
/* RPDL definitions */
#include “r_pdl_bsc.h”

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

void func(void)
{
    /* Disable the CS3 area */
    R_BSC_Destroy(
         3
    );
}
```

5) 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.

- Start / stop operation

PDL_BSC_ENABLE or PDL_BSC_DISABLE	Enable or disable BSC operation. Ignored for the 64-pin and 48-pin device package.

- 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 if success; False if invalid parameters are selected.

Category

Bus Controller

Reference

R_BSC_Create

Remarks

- Before enabling the BSC operation, call R_BSC_Create.
- 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
    );
}
```

6) R_BSC_GetStatus

Synopsis

Read the 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 – b2	b1	b0
0	The bus master that caused the error 000 : CPU 011 : DTC or DMAC	0	Timeout 0: None 1: Occurred	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 (1/4)

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 (2/4)

- 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_USB0_D0 or	D0FIFO transfer request on USB port 0
PDL_DMAC_TRIGGER_USB0_D1 or	D1FIFO transfer request on USB port 0
PDL_DMAC_TRIGGER_SPI0_RX or	Receive buffer full on SPI channel n (n = 0 to 1).
PDL_DMAC_TRIGGER_SPI1_RX or	
PDL_DMAC_TRIGGER_SPI0_TX or	Transmit buffer empty on SPI channel n (n = 0 to 1).
PDL_DMAC_TRIGGER_SPI1_TX or	
PDL_DMAC_TRIGGER_IRQ0 or	Valid edge detected on pin IRQn (n = 0 to 7).
PDL_DMAC_TRIGGER_IRQ1 or	
PDL_DMAC_TRIGGER_IRQ2 or	
PDL_DMAC_TRIGGER_IRQ3 or	
PDL_DMAC_TRIGGER_IRQ4 or	
PDL_DMAC_TRIGGER_IRQ5 or	
PDL_DMAC_TRIGGER_IRQ6 or	
PDL_DMAC_TRIGGER_IRQ7 or	
PDL_DMAC_TRIGGER_ADC10 or	Conversion completed on the 10-bit ADC unit.
PDL_DMAC_TRIGGER_S12ADI or	Conversion completed on the 12-bit ADC unit n (n = 0 to 1).
PDL_DMAC_TRIGGER_S12ADI1 or	
PDL_DMAC_TRIGGER_S12GBADI or	Conversion completed on group B of the 12-bit ADC unit n (n = 0 to 1).
PDL_DMAC_TRIGGER_S12GBADI1 or	
PDL_DMAC_TRIGGER_MTUA0 or	Compare match or input capture A on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DMAC_TRIGGER_MTUA1 or	
PDL_DMAC_TRIGGER_MTUA2 or	
PDL_DMAC_TRIGGER_MTUA3 or	
PDL_DMAC_TRIGGER_MTUA4 or	
PDL_DMAC_TRIGGER_MTUA6 or	
PDL_DMAC_TRIGGER_MTUA7 or	
PDL_DMAC_TRIGGER_MTUB0 or	Compare match or input capture B on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DMAC_TRIGGER_MTUB1 or	
PDL_DMAC_TRIGGER_MTUB2 or	
PDL_DMAC_TRIGGER_MTUB3 or	
PDL_DMAC_TRIGGER_MTUB4 or	
PDL_DMAC_TRIGGER_MTUB6 or	
PDL_DMAC_TRIGGER_MTUB7 or	
PDL_DMAC_TRIGGER_MTUC0 or	Compare match or input capture C on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DMAC_TRIGGER_MTUC3 or	
PDL_DMAC_TRIGGER_MTUC4 or	
PDL_DMAC_TRIGGER_MTUC6 or	
PDL_DMAC_TRIGGER_MTUC7 or	

Description (3/4)

PDL_DMAC_TRIGGER_MTUD0 or PDL_DMAC_TRIGGER_MTUD3 or PDL_DMAC_TRIGGER_MTUD4 or PDL_DMAC_TRIGGER_MTUD6 or PDL_DMAC_TRIGGER_MTUD7 or	Compare match or input capture D on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DMAC_TRIGGER_MTUU5 or	Compare match or input capture U on MTU channel 5.
PDL_DMAC_TRIGGER_MTUV4 or PDL_DMAC_TRIGGER_MTUV5 or PDL_DMAC_TRIGGER_MTUV7 or	Compare match or input capture V on MTU channel 5 Counter over or underflow V on MTU channel 4 and 7.
PDL_DMAC_TRIGGER_MTUW5 or	Compare match or input capture W on MTU channel 5.
PDL_DMAC_TRIGGER_CMP0 or	Interrupt request from Comparator 0.
PDL_DMAC_TRIGGER_CMP1 or	Interrupt request from Comparator 1.
PDL_DMAC_TRIGGER_CMP2 or	Interrupt request from Comparator 2
PDL_DMAC_TRIGGER_CMP4 or	Interrupt request from Comparator 4
PDL_DMAC_TRIGGER_CMP5 or	Interrupt request from Comparator 5
PDL_DMAC_TRIGGER_CMP6 or	Interrupt request from Comparator 6
PDL_DMAC_TRIGGER_IIC0_RX or PDL_DMAC_TRIGGER_IIC1_RX or	Receive buffer full on I²C channel n (n = 0 to 1).
PDL_DMAC_TRIGGER_IIC0_TX or PDL_DMAC_TRIGGER_IIC1_TX or	Transmit buffer empty on I²C channel n (n = 0 to 1).
PDL_DMAC_TRIGGER_GPTA0 or PDL_DMAC_TRIGGER_GPTA1 or PDL_DMAC_TRIGGER_GPTA2 or PDL_DMAC_TRIGGER_GPTA3 or PDL_DMAC_TRIGGER_GPTA4 or PDL_DMAC_TRIGGER_GPTA5 or PDL_DMAC_TRIGGER_GPTA6 or PDL_DMAC_TRIGGER_GPTA7 or	Compare match or input capture A on GPT channel n (n = 0 to 7).
PDL_DMAC_TRIGGER_GPTB0 or PDL_DMAC_TRIGGER_GPTB1 or PDL_DMAC_TRIGGER_GPTB2 or PDL_DMAC_TRIGGER_GPTB3 or PDL_DMAC_TRIGGER_GPTB4 or PDL_DMAC_TRIGGER_GPTB5 or PDL_DMAC_TRIGGER_GPTB6 or PDL_DMAC_TRIGGER_GPTB7 or	Compare match or input capture B on GPT channel n (n = 0 to 7).
PDL_DMAC_TRIGGER_GPTC0 or PDL_DMAC_TRIGGER_GPTC1 or PDL_DMAC_TRIGGER_GPTC2 or PDL_DMAC_TRIGGER_GPTC3 or PDL_DMAC_TRIGGER_GPTC4 or PDL_DMAC_TRIGGER_GPTC5 or PDL_DMAC_TRIGGER_GPTC6 or PDL_DMAC_TRIGGER_GPTC7 or	Compare match C or D on GPT channel n (n = 0 to 7).
PDL_DMAC_TRIGGER_GPTE0 or PDL_DMAC_TRIGGER_GPTE1 or PDL_DMAC_TRIGGER_GPTE2 or PDL_DMAC_TRIGGER_GPTE3 or PDL_DMAC_TRIGGER_GPTE4 or PDL_DMAC_TRIGGER_GPTE5 or PDL_DMAC_TRIGGER_GPTE6 or PDL_DMAC_TRIGGER_GPTE7 or	Compare match E or F on GPT channel n (n = 0 to 7).
PDL_DMAC_TRIGGER_GPTV0 or PDL_DMAC_TRIGGER_GPTV1 or PDL_DMAC_TRIGGER_GPTV2 or PDL_DMAC_TRIGGER_GPTV3 or PDL_DMAC_TRIGGER_GPTV4 or PDL_DMAC_TRIGGER_GPTV5 or PDL_DMAC_TRIGGER_GPTV6 or PDL_DMAC_TRIGGER_GPTV7 or	Counter limit match V on GPT channel n (n = 0 to 7).

Description (4/4)

PDL_DMAC_TRIGGER_LOCOI0 or	LOCO count function event on GPT channel 0.
PDL_DMAC_TRIGGER_LOCOI4 or	LOCO count function event on GPT channel 4.
PDL_DMAC_TRIGGER_SCI0_RX or	Receive buffer full on SCI channel n (n = 0, 1, 2, 3, 12).
PDL_DMAC_TRIGGER_SCI1_RX or	
PDL_DMAC_TRIGGER_SCI2_RX or	
PDL_DMAC_TRIGGER_SCI3_RX or	
PDL_DMAC_TRIGGER_SCI12_RX or	
PDL_DMAC_TRIGGER_SCI0_TX or	Transmit buffer empty on SCI channel n (n = 0, 1, 2, 3, 12).
PDL_DMAC_TRIGGER_SCI1_TX or	
PDL_DMAC_TRIGGER_SCI2_TX or	
PDL_DMAC_TRIGGER_SCI3_TX or	
PDL_DMAC_TRIGGER_SCI12_TX	

[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).

[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_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 RX63T";
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 a 1 kB 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 (1/4)

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
  Some triggers are not existed in 64 and 48 pin packages. Refer to Hardware manual section 15, Interrupt controller, Table 15.3, Interrupt Vector Table for details.

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_USB_D0 or	FIFO interrupt from USB0.
PDL_DTC_TRIGGER_USB_D1 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 (2/2)

PDL_DTC_TRIGGER_SPI1_RX or	Receive buffer full on SPI channel 1
PDL_DTC_TRIGGER_SPI1_TX or	Transmit buffer empty on SPI channel 1
PDL_DTC_TRIGGER_CMP4 or	Interrupt request from Comparator 4.
PDL_DTC_TRIGGER_CMP5 or	Interrupt request from Comparator 5.
PDL_DTC_TRIGGER_CMP6 or	Interrupt request from Comparator 6.
PDL_DTC_TRIGGER_IRQ0 or	Valid edge detected on pin IRQn (n = 0 to 7).
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	
PDL_DTC_TRIGGER_ADI0 or	Conversion completed on the 10-bit ADC unit.
PDL_DTC_TRIGGER_S12ADI or	Conversion completed on the 12-bit ADC unit 0.
PDL_DTC_TRIGGER_S12GBADI or	Conversion completed on group B of the 12-bit ADC unit 0.
PDL_DTC_TRIGGER_S12ADI1 or	Conversion completed on the 12-bit ADC unit 1.
PDL_DTC_TRIGGER_S12GBADI1 or	Conversion completed on group B of the 12-bit ADC unit 1.
PDL_DTC_TRIGGER_MTUA0 or	Compare match or input capture A on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DTC_TRIGGER_MTUA1 or	
PDL_DTC_TRIGGER_MTUA2 or	
PDL_DTC_TRIGGER_MTUA3 or	
PDL_DTC_TRIGGER_MTUA4 or	
PDL_DTC_TRIGGER_MTUA6 or	
PDL_DTC_TRIGGER_MTUA7 or	
PDL_DTC_TRIGGER_MTUB0 or	Compare match or input capture B on MTU channel n (n = 0 to 4 or 6 to 7).
PDL_DTC_TRIGGER_MTUB1 or	
PDL_DTC_TRIGGER_MTUB2 or	
PDL_DTC_TRIGGER_MTUB3 or	
PDL_DTC_TRIGGER_MTUB4 or	
PDL_DTC_TRIGGER_MTUB6 or	
PDL_DTC_TRIGGER_MTUB7 or	
PDL_DTC_TRIGGER_MTUC0 or	Compare match or input capture C on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DTC_TRIGGER_MTUC3 or	
PDL_DTC_TRIGGER_MTUC4 or	
PDL_DTC_TRIGGER_MTUC6 or	
PDL_DTC_TRIGGER_MTUC7 or	
PDL_DTC_TRIGGER_MTUD0 or	Compare match or input capture D on MTU channel n (n = 0, 3, 4, 6 or 7).
PDL_DTC_TRIGGER_MTUD3 or	
PDL_DTC_TRIGGER_MTUD4 or	
PDL_DTC_TRIGGER_MTUD6 or	
PDL_DTC_TRIGGER_MTUD7 or	
PDL_DTC_TRIGGER_MTUU5 or	Compare match or input capture U on MTU channel 5.
PDL_DTC_TRIGGER_MTUV4 or	Compare match or input capture V on MTU channel 5 Counter over or underflow V on MTU channel 4 and 7.
PDL_DTC_TRIGGER_MTUV5 or	
PDL_DTC_TRIGGER_MTUV7 or	
PDL_DTC_TRIGGER_MTUW5 or	Compare match or input capture W on MTU channel 5.
PDL_DTC_TRIGGER_CMP0 or	Interrupt request from Comparator 0.
PDL_DTC_TRIGGER_CMP1 or	Interrupt request from Comparator 1.
PDL_DTC_TRIGGER_CMP2 or	Interrupt request from Comparator 2
PDL_DTC_TRIGGER_IIC1_RX or	Receive buffer full on I²C channel 1
PDL_DTC_TRIGGER_IIC1_TX or	Transmit buffer empty on I²C channel 1
PDL_DTC_TRIGGER_IIC0_RX or	Receive buffer full on I²C channel 0
PDL_DTC_TRIGGER_IIC0_TX or	Transmit buffer empty on I²C channel 0
PDL_DTC_TRIGGER_DMAC0I	Transfer complete on DMAC channel n (n = 0 to 3).
PDL_DTC_TRIGGER_DMAC1I	
PDL_DTC_TRIGGER_DMAC2I	
PDL_DTC_TRIGGER_DMAC3I	

Description (3/4)

PDL_DTC_TRIGGER_GPTA0 or PDL_DTC_TRIGGER_GPTA1 or PDL_DTC_TRIGGER_GPTA2 or PDL_DTC_TRIGGER_GPTA3 or PDL_DTC_TRIGGER_GPTA4 or PDL_DTC_TRIGGER_GPTA5 or PDL_DTC_TRIGGER_GPTA6 or PDL_DTC_TRIGGER_GPTA7 or	Compare match or input capture A on GPT channel n (n = 0 to 7).
PDL_DTC_TRIGGER_GPTB0 or PDL_DTC_TRIGGER_GPTB1 or PDL_DTC_TRIGGER_GPTB2 or PDL_DTC_TRIGGER_GPTB3 or PDL_DTC_TRIGGER_GPTB4 or PDL_DTC_TRIGGER_GPTB5 or PDL_DTC_TRIGGER_GPTB6 or PDL_DTC_TRIGGER_GPTB7 or	Compare match or input capture B on GPT channel n (n = 0 to 7).
PDL_DTC_TRIGGER_GPTC0 or PDL_DTC_TRIGGER_GPTC1 or PDL_DTC_TRIGGER_GPTC2 or PDL_DTC_TRIGGER_GPTC3 or PDL_DTC_TRIGGER_GPTC4 or PDL_DTC_TRIGGER_GPTC5 or PDL_DTC_TRIGGER_GPTC6 or PDL_DTC_TRIGGER_GPTC7 or	Compare match C or D on GPT channel n (n = 0 to 7).
PDL_DTC_TRIGGER_GPTE0 or PDL_DTC_TRIGGER_GPTE1 or PDL_DTC_TRIGGER_GPTE2 or PDL_DTC_TRIGGER_GPTE3 or PDL_DTC_TRIGGER_GPTE4 or PDL_DTC_TRIGGER_GPTE5 or PDL_DTC_TRIGGER_GPTE6 or PDL_DTC_TRIGGER_GPTE7 or	Compare match E or F on GPT channel n (n = 0 to 7).
PDL_DTC_TRIGGER_GPTV0 or PDL_DTC_TRIGGER_GPTV1 or PDL_DTC_TRIGGER_GPTV2 or PDL_DTC_TRIGGER_GPTV3 or PDL_DTC_TRIGGER_GPTV4 or PDL_DTC_TRIGGER_GPTV5 or PDL_DTC_TRIGGER_GPTV6 or PDL_DTC_TRIGGER_GPTV7 or	Counter limit match V on GPT channel n (n = 0 to 7).
PDL_DTC_TRIGGER_LOCOI0 or PDL_DTC_TRIGGER_LOCOI4 or	LOCO count function event
PDL_DTC_TRIGGER_SCI0_RX or PDL_DTC_TRIGGER_SCI1_RX or PDL_DTC_TRIGGER_SCI2_RX or PDL_DTC_TRIGGER_SCI3_RX or PDL_DTC_TRIGGER_SCI12_RX or	Receive buffer full on SCI channel n (n = 0, 1, 2, 3, 12).
PDL_DTC_TRIGGER_SCI0_TX or PDL_DTC_TRIGGER_SCI1_TX or PDL_DTC_TRIGGER_SCI2_TX or PDL_DTC_TRIGGER_SCI3_TX or PDL_DTC_TRIGGER_SCI12_TX	Transmit buffer empty on SCI channel n (n = 0, 1, 2, 3, 12).

[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).

Description (4/4)

[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 */
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. Multi-Function Timer Pulse Unit

1) R_MTU3_Set

Synopsis

Configure the Multi-function Timer Pulse Unit.

Prototype

```
bool R_MTU3_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 7).

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

- Valid when n = 0

PDL_MTU3_PIN_0A_P31 or PDL_MTU3_PIN_0A_PB3	Select the P31 or PB3 pin for MTIOC0A.
PDL_MTU3_PIN_0B_P30 or PDL_MTU3_PIN_0B_PB2	Select the P30 or PB2 pin for MTIOC0B.
PDL_MTU3_PIN_0C_PB1	Select the PB1 pin for MTIOC0C.
PDL_MTU3_PIN_0D_PB0	Select the PB0 pin for MTIOC0D.

- Valid when n = 1

PDL_MTU3_PIN_1A_PA5	Select the PA5 pin for MTIOC1A.
PDL_MTU3_PIN_1B_PA4	Select the PA4 pin for MTIOC1B.

- Valid when n = 2

PDL_MTU3_PIN_2A_PA3	Select the PA3 pin for MTIOC2A.
PDL_MTU3_PIN_2B_PA2	Select the PA2 pin for MTIOC2B.

- Valid when n = 3

PDL_MTU3_PIN_3A_P33	Select the P33 pin for MTIOC3A.
PDL_MTU3_PIN_3B_P71	Select the P71 pin for MTIOC3B.
PDL_MTU3_PIN_3C_P32	Select the P32 pin for MTIOC3C.
PDL_MTU3_PIN_3D_P74	Select the P74 pin for MTIOC3D.

- Valid when n = 4

PDL_MTU3_PIN_4A_P72	Select the P72 pin for MTIOC4A.
PDL_MTU3_PIN_4B_P73	Select the P73 pin for MTIOC4B.
PDL_MTU3_PIN_4C_P75	Select the P75 pin for MTIOC4C.
PDL_MTU3_PIN_4D_P76	Select the P76 pin for MTIOC4D.

- Valid when n = 5

PDL_MTU3_PIN_5U_P24 or PDL_MTU3_PIN_5U_P82	Select the P24 or P82 pin for MTIOC5U.
PDL_MTU3_PIN_5V_P23 or PDL_MTU3_PIN_5V_P81	Select the P23 or P81 pin for MTIOC5V.
PDL_MTU3_PIN_5W_P22 or PDL_MTU3_PIN_5W_P80	Select the P22 or P80 pin for MTIOC5W.

- Valid when n = 6

PDL_MTU3_PIN_6A_P33 or PDL_MTU3_PIN_6A_PA1	Select the P33 or PA1 pin for MTIOC6A.
PDL_MTU3_PIN_6B_P71 or PDL_MTU3_PIN_6B_P95	Select the P71 or P95 pin for MTIOC6B.
PDL_MTU3_PIN_6C_P32 or PDL_MTU3_PIN_6C_PA0	Select the P32 or PA0 pin for MTIOC6C.
PDL_MTU3_PIN_6D_P74 or PDL_MTU3_PIN_6D_P92	Select the P74 or P92 pin for MTIOC6D.

Description (2/2)

- Valid when n = 7

PDL_MTU3_PIN_7A_P72 or PDL_MTU3_PIN_7A_P94	Select the P72 or P94 pin for MTIOC7A.
PDL_MTU3_PIN_7B_P73 or PDL_MTU3_PIN_7B_P93	Select the P73 or P93 pin for MTIOC7B.
PDL_MTU3_PIN_7C_P75 or PDL_MTU3_PIN_7C_P91	Select the P75 or P91 pin for MTIOC7C.
PDL_MTU3_PIN_7D_P76 or PDL_MTU3_PIN_7D_P90	Select the P76 or P90 pin for MTIOC7D.

- Valid when n = 0 to 4

PDL_MTU3_PIN_CLKA_P22 or PDL_MTU3_PIN_CLKA_PB3 or PDL_MTU3_PIN_CLKA_P33 or PDL_MTU3_PIN_CLKA_P21	Select the P22, PB3, P33 or P21 pin for MTCLKA.
PDL_MTU3_PIN_CLKB_P23 or PDL_MTU3_PIN_CLKB_PB2 or PDL_MTU3_PIN_CLKB_P32 or PDL_MTU3_PIN_CLKB_P20	Select the P23, PB2, P32 or P20 pin for MTCLKB.

- Valid when n = 0 or 2

PDL_MTU3_PIN_CLKC_P11 or PDL_MTU3_PIN_CLKC_P24 or PDL_MTU3_PIN_CLKC_P31 or PDL_MTU3_PIN_CLKC_PE4	Select the P11, P24, P31, or E4 pin for MTCLKC.
PDL_MTU3_PIN_CLKD_P10 or PDL_MTU3_PIN_CLKD_P30 or PDL_MTU3_PIN_CLKD_PE3	Select the P10, P30 or E3 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_MTU3_Create

Remarks

- Before calling R_MTU3_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.
- There are some pin restrictions when not using the 64 pin device package, or not using the package that is greater than 64-pin.

Program Example

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

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

void func(void)
{
    /* Configure the MTU pins */
    R_MTU3_Set(
        0,
        PDL_MTU3_PIN_0A_P31
    );
}
```

2) R_MTU3_Create

Synopsis

Configure an MTU channel.

Prototype

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

R_MTU3_Create_structure members:

```
uint16_t channel_mode               // Configuration selection
uint32_t event_trigger_operation    // Configuration selection
uint32_t counter_operation          // Configuration selection
uint32_t ADC_trigger_operation      // Configuration selection
uint32_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 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 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
uint8_t interrupt_priority_3        // Interrupt priority level
```

Description (1/9)

Set up a 16-bit MTU3 channel.

[data1]
The channel number n (where n = 0 to 7).

Description (2/9)

[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 ≠ 5, unless stated otherwise.

PDL_MTU3_MODE_NORMAL or	Normal operation.
PDL_MTU3_MODE_PWM1 or	Pulse Width Modulation (PWM) mode 1.
PDL_MTU3_MODE_PWM2 or	Pulse Width Modulation (PWM) mode 2. Valid for n = 0, 1, and 2.
PDL_MTU3_MODE_PHASE1 or PDL_MTU3_MODE_PHASE2 or PDL_MTU3_MODE_PHASE3 or PDL_MTU3_MODE_PHASE4 or	Phase counting mode 1, 2, 3 or 4. Valid for n = 1 and 2.
PDL_MTU3_MODE_PWM_RS or	Reset-synchronised PWM mode. Valid for n = 3 and 6. See Remarks for setting up associate channels 4 and 7.
PDL_MTU3_MODE_PWM_COMP1 or PDL_MTU3_MODE_PWM_COMP2 or PDL_MTU3_MODE_PWM_COMP3	Complementary PWM mode 1, 2 or 3. Valid for n = 3 and 6. See Remarks for setting up associate channels 4 and 7.

- Synchronous mode. Valid for n ≠ 5.

PDL_MTU3_SYNC_DISABLE or PDL_MTU3_SYNC_ENABLE	Disable or enable synchronous presetting / clearing.

[event_trigger_operation]
Configure the event 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.

- DMAC / DTC event trigger control. Valid for n ≠ 5, unless stated otherwise.

PDL_MTU3_TGRA_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRA_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRA_DTC_TRIGGER_ENABLE	TGRA compare match or input capture.
PDL_MTU3_TGRB_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRB_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRB_DTC_TRIGGER_ENABLE	TGRB compare match or input capture.
PDL_MTU3_TGRC_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRC_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRC_DTC_TRIGGER_ENABLE	TGRC compare match or input capture. Valid for n = 0, 3, 4, 6 and 7.
PDL_MTU3_TGRD_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRD_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRD_DTC_TRIGGER_ENABLE	TGRD compare match or input capture. Valid for n = 0, 3, 4, 6 and 7.
PDL_MTU3_TCIV_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TCIV_DMAC_TRIGGER_ENABLE or PDL_MTU3_TCIV_DTC_TRIGGER_ENABLE	Counter overflow or underflow. Valid for n = 4 and 7.

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

PDL_MTU3_TGRU_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRU_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRU_DTC_TRIGGER_ENABLE	TGRU compare match or input capture.
PDL_MTU3_TGRV_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRV_DMAC_TRIGGER_ENABLE or PDL_MTU3_TGRV_DTC_TRIGGER_ENABLE	TGRV compare match or input capture.
PDL_MTU3_TGRW_DMAC_DTC_TRIGGER_DISABLE or PDL_MTU3_TGRW_DTC_TRIGGER_ENABLE or PDL_MTU3_TGRW_DTC_TRIGGER_ENABLE	TGRW compare match or input capture.

Description (3/9)

[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 ≠ 5, unless stated otherwise.

PDL_MTU3_CLK_PCLKA_DIV_1 or PDL_MTU3_CLK_PCLKA_DIV_4 or PDL_MTU3_CLK_PCLKA_DIV_16 or PDL_MTU3_CLK_PCLKA_DIV_64 or	The internal clock signal PCLKA ÷ 1, 4, 16 or 64.
PDL_MTU3_CLK_PCLKA_DIV_256 or	PCLKA ÷ 256. Valid for n = 1, 3, 4, 6 and 7.
PDL_MTU3_CLK_PCLKA_DIV_1024 or	PCLKA ÷ 1024. Valid for n = 2, 3, 4, 6 and 7.
PDL_MTU3_CLK_MTCLKA or	MTCLKA pin input. Valid for n = 0 to 4.
PDL_MTU3_CLK_MTCLKB or	MTCLKB pin input. Valid for n = 0 to 4.
PDL_MTU3_CLK_MTCLKC or	MTCLKC pin input. Valid for n = 0 and 2.
PDL_MTU3_CLK_MTCLKD or	MTCLKD pin input. Valid for n = 0.
PDL_MTU3_CLK_CASCADE	The overflow / underflow signal from channel 2. Valid for n = 1.

- TCNT counter clock edge selection. Valid for n ≠ 5.

PDL_MTU3_CLK_RISING or PDL_MTU3_CLK_FALLING or PDL_MTU3_CLK_BOTH	The TCNT counter clock signal shall be counted on rising, falling or both edges.

- TCNT counter clearing. Valid for n ≠ 5, unless stated otherwise.

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

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

PDL_MTU3_CLKU_PCLKA_DIV_1 or PDL_MTU3_CLKU_PCLKA_DIV_4 or PDL_MTU3_CLKU_PCLKA_DIV_16 or PDL_MTU3_CLKU_PCLKA_DIV_64	Counter TCNTU is supplied by the internal clock signal PCLKA ÷ 1, 4, 16 or 64.
PDL_MTU3_CLKV_PCLKA_DIV_1 or PDL_MTU3_CLKV_PCLKA_DIV_4 or PDL_MTU3_CLKV_PCLKA_DIV_16 or PDL_MTU3_CLKV_PCLKA_DIV_64	Counter TCNTV is supplied by the internal clock signal PCLKA ÷ 1, 4, 16 or 64.
PDL_MTU3_CLKW_PCLKA_DIV_1 or PDL_MTU3_CLKW_PCLKA_DIV_4 or PDL_MTU3_CLKW_PCLKA_DIV_16 or PDL_MTU3_CLKW_PCLKA_DIV_64	Counter TCNTW is supplied by the internal clock signal PCLKA ÷ 1, 4, 16 or 64.

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

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

Description (4/9)

[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 ≠ 5, unless stated otherwise.

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

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

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

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

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

Description (5/9)

[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.

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

PDL_MTU3_BUFFER_DOUBLE_DISABLE or PDL_MTU3_BUFFER_DOUBLE_ENABLE	Disable or enable the double buffer function.

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

PDL_MTU3_CSB_DISABLE or PDL_MTU3_CSB_CREST or PDL_MTU3_CSB_TROUGH or PDL_MTU3_CSB_BOTH	Select no transfer, transfer on crest detection, transfer on trough detection or transfer on crest and trough detection.

PDL_MTU3_CSB_TROUGH and PDL_MTU3_CSB_BOTH are available only in complementary PWM mode.

- Buffer operation

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

- Buffer data transfer. Valid in PWM mode.

PDL_MTU3_BUFFER_AC_CM_A or PDL_MTU3_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, 4, 6 and 7.
PDL_MTU3_BUFFER_BD_CM_B or PDL_MTU3_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, 4, 6 and 7.
PDL_MTU3_BUFFER_EF_CM_E or PDL_MTU3_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 (6/9)

[TGR_A_B_operation]
Configure the operation for general registers TGRA and TGRB. Valid for n ≠ 5.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Input capture / output compare control for register TGRA

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

- Input capture / output compare control for register TGRB.

PDL_MTU3_B_OC_DISABLED or	MTIOCnB output disabled.
PDL_MTU3_B_OC_LOW or	MTIOCnB output low.
PDL_MTU3_B_OC_LOW_CM_HIGH or	MTIOCnB initial output low; goes high at compare match.
PDL_MTU3_B_OC_LOW_CM_INV or	MTIOCnB initial output low; toggles at compare match.
PDL_MTU3_B_OC_HIGH_CM_LOW or	MTIOCnB initial output high; goes low at compare match.
PDL_MTU3_B_OC_HIGH or	MTIOCnB output high.
PDL_MTU3_B_OC_HIGH_CM_INV or	MTIOCnB initial output high; toggles at compare match.
PDL_MTU3_B_IC_RISING_EDGE or	Input capture at MTIOCnB rising edge.
PDL_MTU3_B_IC_FALLING_EDGE or	Input capture at MTIOCnB falling edge.
PDL_MTU3_B_IC_BOTH_EDGES or	Input capture at MTIOCnB both edges.
PDL_MTU3_B_IC_COUNT or	Input capture at channel (n+1) up-count or down-count. Valid only for n = 0.
PDL_MTU3_B_IC_CM_IC	Input capture at channel (n-1) 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_MTU3_CASCADE_AL_IC_EXC_H or PDL_MTU3_CASCADE_AL_IC_INC_H	Exclude or include pin MTIOCnA in the TGRA input capture conditions for channel (n+1).
PDL_MTU3_CASCADE_BL_IC_EXC_H or PDL_MTU3_CASCADE_BL_IC_INC_H	Exclude or include pin MTIOCnB in the TGRB input capture conditions for channel (n+1).
PDL_MTU3_CASCADE_AH_IC_EXC_L or PDL_MTU3_CASCADE_AH_IC_INC_L	Exclude or include pin MTIOC(n+1)A in the TGRA input capture conditions for channel n.
PDL_MTU3_CASCADE_BH_IC_EXC_L or PDL_MTU3_CASCADE_BH_IC_INC_L	Exclude or include pin MTIOC(n+1)B in the TGRB input capture conditions for channel n.

Description (7/9)

[TGR_C_D_operation]
Configure the operation for general registers TGRC and TGRD. Valid for n = 0, 3, 4, 6 and 7.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Input capture / output compare control for register TGRC.

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

- Input capture / output compare control for register TGRD.

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

[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_MTU3_U_CM or	Compare match.
PDL_MTU3_U_IC_RISING_EDGE or	Input capture at MTICnU rising edge.
PDL_MTU3_U_IC_FALLING_EDGE or	Input capture at MTICnU falling edge.
PDL_MTU3_U_IC_BOTH_EDGES or	Input capture at MTICnU both edges.
PDL_MTU3_U_IC_PWM_LOW_TROUGH or PDL_MTU3_U_IC_PWM_LOW_CREST or PDL_MTU3_U_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU3_U_IC_PWM_HIGH_TROUGH or PDL_MTU3_U_IC_PWM_HIGH_CREST or PDL_MTU3_U_IC_PWM_HIGH_BOTH	Input capture at trough, crest or both for high pulse width measurement.

Description (8/9)

- Input capture / compare match control for register TGRV.

PDL_MTU3_V_CM or	Compare match.
PDL_MTU3_V_IC_RISING_EDGE or PDL_MTU3_V_IC_FALLING_EDGE or PDL_MTU3_V_IC_BOTH_EDGES or	Input capture at MTICnV rising edge. Input capture at MTICnV falling edge. Input capture at MTICnV both edges.
PDL_MTU3_V_IC_PWM_LOW_TROUGH or PDL_MTU3_V_IC_PWM_LOW_CREST or PDL_MTU3_V_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU3_V_IC_PWM_HIGH_TROUGH or PDL_MTU3_V_IC_PWM_HIGH_CREST or PDL_MTU3_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_MTU3_W_CM or	Compare match.
PDL_MTU3_W_IC_RISING_EDGE or PDL_MTU3_W_IC_FALLING_EDGE or PDL_MTU3_W_IC_BOTH_EDGES or	Input capture at MTICnW rising edge. Input capture at MTICnW falling edge. Input capture at MTICnW both edges.
PDL_MTU3_W_IC_PWM_LOW_TROUGH or PDL_MTU3_W_IC_PWM_LOW_CREST or PDL_MTU3_W_IC_PWM_LOW_BOTH or	Input capture at trough, crest or both for low pulse width measurement.
PDL_MTU3_W_IC_PWM_HIGH_TROUGH or PDL_MTU3_W_IC_PWM_HIGH_CREST or PDL_MTU3_W_IC_PWM_HIGH_BOTH	Input capture at trough, crest or both for high pulse width measurement.

[TCNT_TCNTU_value]
For n ≠ 5: The timer counter TCNT value.
For n = 5: The timer counter TCNTU value.

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

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

[TGRC_TGRU_value]
For n = 0, 3, 4, 6 and 7: The register TGRC value.
For n = 5: The register TGRU value.
Ignored for other channel.

[TGRD_TGRV_value]
For n = 0, 3, 4, 6 and 7: The register TGRD value.
For n = 5: The register TGRV value.
Ignored for other channel.

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

[TGRF_value]
For n = 0, 4 and 7: The register TGRF value.
Ignored for other channel.

[TADCORA_value]
For n = 4 and 7: The register TADCORA value.

[TADCORB_value]
For n = 4 and 7: The register TADCORB value.

[TADCOBRA_value]
For n = 4 and 7: The register TADCOBRA value.

Description (9/9)

[TADCOBRB_value]
For n = 4 and 7: The register TADCOBRB value.

[func1]
For n ≠ 5: The function to be called when a TGRA event occurs.
For n = 5: The function to be called when a TGRU event occurs.
Specify PDL_NO_FUNC if not required.

[func2]
For n ≠ 5: The function to be called when a TGRB event occurs.
For n = 5: The function to be called when a TGRV event occurs.
Specify PDL_NO_FUNC if not required.

[func3]
For n = 0, 3, 4, 6 and 7: The function to be called when a TGRC event occurs.
For n = 5: The function to be called when a TGRW event occurs.
Specify PDL_NO_FUNC if not required.

[func4]
For n = 0, 3, 4, 6 and 7: The function to be called when a TGRD event occurs.
Specify PDL_NO_FUNC if not required.

[interrupt_priority_1]
For n ≠ 7: The interrupt priority level for TGR(A to D or U to W) events.
For n = 7: The interrupt priority level for TGRA and TGRB events.
(See also [interrupt_priority_3].)
Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for the corresponding callback functions.

[func5]
For n = 0: The function to be called when a TGRE event occurs.
Specify PDL_NO_FUNC if not required.

[func6]
For n = 0: The function to be called when a TGRF event occurs.
Specify PDL_NO_FUNC if not required.

[func7]
For n = 0 to 3 or 6: The function to be called when an overflow occurs.
For n = 4 or 7: The function to be called when an overflow or underflow occurs.
Specify PDL_NO_FUNC if not required.

[func8]
For n = 1 and 2: The function to be called when an underflow occurs.
Specify PDL_NO_FUNC if not required.

[interrupt_priority_2]
The interrupt priority level for TGRE, TGRF, overflow and 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).

[interrupt_priority_3]
For n = 7: The interrupt priority level for TGRC and TGRD events.
Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for parameters func3 and func4.

Return value

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

Category

Multi-function Timer Pulse Unit

Reference

R_MTU3_Set, R_MTU3_ControlChannel, R_MTU3_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.
- Call R_MTU3_Set before calling this function to select the pins to be used.
- Either R_MTU3_ControlChannel or R_MTU3_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 reset-synchronised or complementary PWM mode is required then follow these steps in the order they are presented.
    1. Call this function for channel 3 or 6 and specify the required PWM mode. At this time the associated channel (4 or 7) will be set to normal mode.
    2. To configure the associated channel (4 or 7), call this function again and specify Normal operation (not a PWM mode).

  Call R_MTU3_ControlUnit with PDL_MTU3_PWM_RS_COMP_ENABLE.
- 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_MTU3_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_MTU3_ControlChannel or R_MTU3_ControlUnit).
- If using Complementary PWM mode with Synchronous Clearing and Waveform Retention enabled, then be aware of the cautions specified in the Usage Notes section of the hardware manual.

Program example

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

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

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

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

    /* Set the non-default options for channel 4 */
    ch4_parameters.channel_mode = PDL_MTU3_MODE_NORMAL |\
    PDL_MTU3_SYNC_ENABLE;
    ch4_parameters.event_trigger_operation =
    PDL_MTU3_TGRA_DTC_TRIGGER_ENABLE;
    ch4_parameters.counter_operation = PDL_MTU3_CLK_PCLKA_DIV_4;
    ch4_parameters.buffer_operation = PDL_MTU3_BUFFER_AC_CM_A;
    ch4_parameters.TGR_C_D_operation = PDL_MTU3_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_value = 0;

    R_MTU3_Create(
        4,
        &ch4_parameters
    );
}
```

3) R_MTU3_Destroy

Synopsis	Disable a Multi-function Timer Pulse Unit.
Prototype	**bool R_MTU3_Destroy(** **void** // No parameter is required **);**
Description	Shut down a timer pulse unit
Return value	True.
Category	Multi-function Timer Pulse Unit
Reference	None.
Remarks	• The unit is put into the stop state to reduce power consumption.
Program example	

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

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

void func(void)
{
    /* Shutdown all MTU channels */
    R_MTU3_Destroy(
    );
}
```

4) R_MTU3_ControlChannel

Synopsis

Control an MTU channel.

Prototype

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

R_MTU3_ControlChannel_structure members:

```
uint8_t 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 7).

[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 ≠ 5.

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

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

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

[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 ≠ 5.

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

Description (2/2)

For n = 5.

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

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

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

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

[TGRC_TGRU_value]
For n = 0, 3, 4, 6 and 7: The register TGRC value.
For n = 5: The register TGRU value.
This will be ignored if the register is not selected.

[TGRD_TGRV_value]
For n = 0, 3, 4, 6 and 7: The register TGRD value.
For n = 5: The register TGRV value.
This will be ignored if the register is not selected.

[TGRE_TGRW_value]
For n = 0, 3, 4, 6 and 7: The register TGRE value.
For n = 5: The register TGRW value.
This will be ignored if the register is not selected.

[TGRF_value]
For n = 0, 4 and 7: The general register TGRF value.
This will be ignored if the register is not selected.

[TADCOBRA_value]
For n = 4 and 7: ADC start request cycle set buffer A.
This will be ignored if the register is not selected.

[TADCOBRB_value]
For n = 4 and 7: ADC start request cycle set buffer B.
This will be ignored if the register is not selected.

Return value

True if the channel number is valid; otherwise false.

Category

Multi-function Timer Pulse Unit

Reference

R_MTU3_Create, R_MTU3_ControlUnit

Remarks

- Before calling this function, use R_MTU3_Create to configure the channel operation.
- Either this function or R_MTU3_ControlUnit must be used to start the timers.
- The Stop operation is executed at the start of this function.
  The Start operation is executed at the end.
  Therefore, both options can be selected together with other changes in one function call.

Program example

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

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

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

    /* Set the control options for channel 3 */
    ch3_parameters.control_setting = PDL_MTU3_START;
    ch3_parameters.register_selection = PDL_MTU3_REGISTER_COUNTER | \
                            PDL_MTU3_REGISTER_TGRB;
    ch3_parameters.TCNT_TCNTU_value = 0xFFDD;
    ch3_parameters.TGRB_TCNTW_value = 0x0020;

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

5) R_MTU3_ControlUnit

Synopsis

Control a Multi-function Timer Pulse Unit.

Prototype

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

R_MTU3_ControlUnit_structure members:

```
uint16_t simultaneous_control       // Control selection
uint16_t pair_control               // Control selection
uint32_t BDCM_sync_control          // Control selection
uint32_t output_control             // Control selection
uint32_t buffer_control             // Control selection
uint32_t int_skip_control           // Register selection
uint16_t DT_data_value              // Register value
uint16_t data_value                 // Register value
uint16_t buffer_value               // Register value
```

Description (1/6)

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_MTU3_STOP_CH_0	Stop the count operation for the selected channels.
PDL_MTU3_STOP_CH_1	
PDL_MTU3_STOP_CH_2	
PDL_MTU3_STOP_CH_3	
PDL_MTU3_STOP_CH_4	
PDL_MTU3_STOP_CH_6	
PDL_MTU3_STOP_CH_7	

- Counter start control

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

Description (2/6)

[pair_control]
Channel pair configuration control.
Select the pair of channels that will be configured by the controls specified below and in parameters data4 to data7.

- Channel pair selection

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

Select the controls. All selections are optional.
If identical controls are required for both pairs, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Register protection

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

- Dead time generation control.

PDL_MTU3_DEAD_TIME_DISABLE or PDL_MTU3_DEAD_TIME_ENABLE	Disable or enable dead time generation.

- Waveform retention control.

PDL_MTU3_WAVEFORM_RETAIN_DISABLE or PDL_MTU3_WAVEFORM_RETAIN_ENABLE	Disable or enable waveform output retention.

- Synchronous clearing control. Applies only to channel pair 6 and 7.

PDL_MTU3_SYNC_CLEAR_DISABLE or PDL_MTU3_SYNC_CLEAR_ENABLE	Disable or enable synchronous clearing.

- Compare match clearing control.

PDL_MTU3_CNT_CLEAR_CM_A_DISABLE or PDL_MTU3_CNT_CLEAR_CM_A_ENABLE	Disable or enable counter clearing on TGRA compare match. This must not be enabled if not in Complementary PWM Mode 1.

- Reset-synchronised or complementary PWM control

PDL_MTU3_PWM_RS_COMP_ENABLE	Enable reset-synchronised or complementary PWM mode.

- Registers to be modified.

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

Description (3/6)

[BDCM_sync_control]
The control settings which are specific to one pair of channels. All settings are optional.
If multiple selections are required, use "|" to separate each selection.
Applies only to reset-synchronised or complementary PWM modes.

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

PDL_MTU3_BDCM_ENABLE or PDL_MTU3_BDCM_DISABLE	Enable or disable brushless DC motor control
PDL_MTU3_BDCM_P_PHASE_ENABLE or PDL_MTU3_BDCM_P_PHASE_DISABLE	Enable or disable PWM outputs on the positive-phase output pins.
PDL_MTU3_BDCM_N_PHASE_ENABLE or PDL_MTU3_BDCM_N_PHASE_DISABLE	Enable or disable PWM outputs on the negative-phase output pins.
PDL_MTU3_BDCM_OPS_FB or	Use input capture signals for output switch control, or
PDL_MTU3_BDCM_OPS_000 or PDL_MTU3_BDCM_OPS_001 or PDL_MTU3_BDCM_OPS_010 or PDL_MTU3_BDCM_OPS_011 or PDL_MTU3_BDCM_OPS_100 or PDL_MTU3_BDCM_OPS_101 or PDL_MTU3_BDCM_OPS_110 or PDL_MTU3_BDCM_OPS_111	Set the outputs according to table 22.46 in the hardware manual.

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

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

Description (4/6)

[output_control]
The phase output control settings to be modified. All settings are optional.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Output enable control. To apply output control, make sure the operation of the corresponding channel is stopped.
  Select one option for each output.

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

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

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

- Output inversion control (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 inversion of the positive phases.

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

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

  Or independently by selecting one option for each required output.

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

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

PDL_MTU3_OUT_LOCK_ENABLE	Prevent further changes to the phase output control.

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

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

Description (5/6)

[buffer_control]
The phase output buffer control settings to be modified. All settings are optional.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Output level buffer control (applies only to reset-synchronised or complementary PWM modes).
  Set the output control to be transferred to the output:

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

- Set the transfer timing
  In complementary PWM mode:

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

  In Reset-synchronised PWM mode:

PDL_MTU3_OUT_BUFFER_TRANSFER_DISABLE or PDL_MTU3_OUT_BUFFER_TRANSFER_CLEAR	Disable or enable on counter clear.

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

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

Description (6/6)

[int_skip_control]
Interrupt skipping control settings. All settings are optional, but only one skipping function type may be selected.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Interrupt skipping control (type 1)

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

- Interrupt skipping control (type 2)

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

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

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

[buffer_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_MTU3_ControlChannel, R_MTU3_Create

Remarks

- Either this function or R_MTU3_ControlChannel must be used to start the timers.
- The Stop operation is executed at the start of this function.
  The Start operation is executed at the end.
  Therefore, both options can be selected together with other changes in one function call.
- The register access enable operation is executed at the start of this function.
  The register access disable operation is executed at the end.
  Therefore, both options can be selected together with other changes in one function call.
- The enabling of reset-synchronised or complementary PWM mode is made after all other configuration settings are made and before the timers are started. R_MTU3_Create must be used first to configure the timer channel (3 or 6).
- A companion function, R_MTU3_ControlUnit_load_defaults, can be used to load the default values into the structure.
- When generating PWM waveforms in complementary PWM mode 1 to complementary PWM mode 3, set the timer cycle data registers (TCDRA or TCDRB) and timer dead time data registers (TDDRA or TDDRB) to values that satisfy the following condition:
  Timer cycle data register value > Timer dead time data register value × 2 + 2
- Output protection function for complementary PWM mode is enabled at the initial state. To disable this function, call API R_POE_Set without options settings to MTU3, MTU4, MTU6, and MTU7 in data4.

Program example

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

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

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

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

    /* Set the control options for unit 0 */
    unit_parameters.simultaneous_control = PDL_MTU3_START_CH_0 |
                                           PDL_MTU3_START_CH_1;
    unit_parameters.pair_control = PDL_MTU3_DEAD_TIME_ENABLE |
                                   PDL_MTU3_REGISTER_DEAD_TIME |
                                   PDL_MTU3_REGISTER_CYCLE_DATA;
    unit_parameters.output_control =
PDL_MTU3_OUT_P_PHASE_ALL_HIGH_LOW;
    unit_parameters.DT_data_value = 0xFFDD;
    unit_parameters.data_value = 0x0100;

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

6) R_MTU3_ReadChannel

Synopsis

Read from MTU channel registers.

Prototype

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

Description (1/2)

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

[data1]
The channel number n (where n = 0 to 7).

[data2]
The status flags shall be stored in the format below.
The input capture / compare match flags will be set to 1 if the condition has been detected.
Specify PDL_NO_PTR if the flags are not to be read.

For n = 0

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

For n = 1 and 2

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

For n = 3 and 6

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

For n = 4 and 7

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

For n = 5

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

[data3]
For n ≠ 5: A pointer to where the TCNT register value shall be stored.
For n = 5: A pointer to where the TCNTU register value shall be stored.
Specify PDL_NO_PTR if it is not required.

Description (2/2)

[data4]
For n ≠ 5: A pointer to where the TGRA register value shall be stored.
For n = 5: A pointer to where the TCNTV register value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data5]
For n ≠ 5: A pointer to where the TGRB register value shall be stored.
For n = 5: A pointer to where the TCNTW register value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data6]
For n = 0, 3, 4, 6 and 7: A pointer to where the TGRC register value shall be stored.
For n = 5: A pointer to where the TGRU register value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data7]
For n = 0, 3, 4, 6 and 7: A pointer to where the TGRD register value shall be stored.
For n = 5: A pointer to where the TGRV register value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data8]
For n = 0, 3, 4, 6 and 7: A pointer to where the TGRE register value shall be stored.
For n = 5: A pointer to where the TGRW register value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data9]
For n = 0, 4 and 7: A pointer to where the TGRF register value shall be stored.
Specify PDL_NO_PTR if it is not required.

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.
- If corresponding interrupt source is enabled, the input capture / compare match status flag will be cleared automatically in interrupt service routine.

Program example

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

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

uint8_t Flags;
uint16_t General_A;
uint16_t General_D;

void func(void)
{
    /* Read the status flags and registers of channel 3 */
    R_MTU3_ReadChannel(
         3,
         &Flags,
         PDL_NO_PTR,
         &General_A,
         PDL_NO_PTR,
         PDL_NO_PTR,
         &General_D,
         PDL_NO_PTR,
         PDL_NO_PTR
    );
}
```

7) R_MTU3_ReadUnit

Synopsis

Read from MTU registers.

Prototype

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

Description

Read any of the timer units's counter registers

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

[data2]
- Channel pair selection

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

[data3]
A pointer to where the Timer subcounter register (TCNTS) value shall be stored.
Specify PDL_NO_PTR if it is not required.

[data4]
Where the Timer Interrupt Skipping Counters (TITCNT1 or TITCNT2) register value shall be stored. The choice of register depends on the type on interrupt skipping control selected by function R_MTU3_ControlUnit.
Specify PDL_NO_PTR if it is not required.

Return value

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

Category

Multi-function Timer Pulse Unit

Reference

R_MTU3_ControlUnit

Remarks

- None.

Program example

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

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

uint16_t Sub_count;
uint8_t Skip_count;

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

4.2.14. 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
        uint32_t data2,   // High impedance control
        uint16_t data3,   // High impedance control for MTU67 and GPT67
        uint16_t data4,   // Output pin selection
        uint32_t data5    // Output short detection and response
);
```

Description (1/4) Initialise the POE pins.

[data1]
Configure the input detection for pins of POE0, POE4, POE8, POE10, POE11 and POE12.
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 of POE0, POE4, POE8, POE10, POE11 and POE12, select falling edge or low level for 16 samples at PCLK ÷ 8, 16 or 128.
PDL_POE_4_MODE_EDGE or PDL_POE_4_MODE_LOW_8 or PDL_POE_4_MODE_LOW_16 or PDL_POE_4_MODE_LOW_128	
PDL_POE_8_MODE_EDGE or PDL_POE_8_MODE_LOW_8 or PDL_POE_8_MODE_LOW_16 or PDL_POE_8_MODE_LOW_128	
PDL_POE_10_MODE_EDGE or PDL_POE_10_MODE_LOW_8 or PDL_POE_10_MODE_LOW_16 or PDL_POE_10_MODE_LOW_128	
PDL_POE_11_MODE_EDGE or PDL_POE_11_MODE_LOW_8 or PDL_POE_11_MODE_LOW_16 or PDL_POE_11_MODE_LOW_128	
PDL_POE_12_MODE_EDGE or PDL_POE_12_MODE_LOW_8 or PDL_POE_12_MODE_LOW_16 or PDL_POE_12_MODE_LOW_128	

- Pin selection.

PDL_POE_PIN_POE10_PE2 or PDL_POE_PIN_POE10_PE4	Select the PE2 or PE4 pin for POE10. Not required if input POE10 is disabled.

[data2]
High impedance control selections.
If multiple selections are required, use "|" to separate each selection.
All selections are optional. Specify PDL_NO_DATA if none are required.

- High impedance request detection

PDL_POE_HI_Z_REQ_8_ENABLE	Enable high-impedance requests on pin POE8.
PDL_POE_HI_Z_REQ_10_ENABLE	Enable high-impedance requests on pin POE10.
PDL_POE_HI_Z_REQ_11_ENABLE	Enable high-impedance requests on pin POE11.
PDL_POE_HI_Z_REQ_12_ENABLE	Enable high-impedance requests on pin POE12.
PDL_POE_HI_Z_OSTST_ENABLE	If stopped oscillation is detected, place the MTU or GPT pins in the high impedance state.

Description (2/4)

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

PDL_POE_HI_Z_MT34_ADD_CFLAG	Comparator detection		MTU channel 3/4 and GPT channel 0 to 2 outputs.
PDL_POE_HI_Z_MT34_ADD_POE4	A valid edge on the pin:	POE4	
PDL_POE_HI_Z_MT34_ADD_POE8		POE8	
PDL_POE_HI_Z_MT34_ADD_POE10		POE10	
PDL_POE_HI_Z_MT34_ADD_POE11		POE11	
PDL_POE_HI_Z_MT34_ADD_POE12		POE12	

PDL_POE_HI_Z_MT0_ADD_CFLAG	Comparator detection		MTU channel 0 outputs.
PDL_POE_HI_Z_MT0_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_MT0_ADD_POE4		POE4	
PDL_POE_HI_Z_MT0_ADD_POE10		POE10	
PDL_POE_HI_Z_MT0_ADD_POE11		POE11	
PDL_POE_HI_Z_MT0_ADD_POE12		POE12	

PDL_POE_HI_Z_GPT01_ADD_CFLAG	Comparator detection		GPT channel 0 and 1 outputs.
PDL_POE_HI_Z_GPT01_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_GPT01_ADD_POE4		POE4	
PDL_POE_HI_Z_GPT01_ADD_POE8		POE8	
PDL_POE_HI_Z_GPT01_ADD_POE11		POE11	
PDL_POE_HI_Z_GPT01_ADD_POE12		POE12	

PDL_POE_HI_Z_GPT23_ADD_CFLAG	Comparator detection		GPT channel 2 and 3 outputs.
PDL_POE_HI_Z_GPT23_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_GPT23_ADD_POE4		POE4	
PDL_POE_HI_Z_GPT23_ADD_POE8		POE8	
PDL_POE_HI_Z_GPT23_ADD_POE10		POE10	
PDL_POE_HI_Z_GPT23_ADD_POE12		POE12	

[data3]
High impedance control selections for MTU67 and GPT67
If multiple selections are required, use “|” to separate each selection.
All selections are optional. Specify PDL_NO_DATA if none are required.

- Select any event flags to be added to the high-impedance control for MTU67 and GPT67.

PDL_POE_HI_Z_GPT67_ADD_CFLAG	Comparator detection		GPT channel 6 and 7 outputs.
PDL_POE_HI_Z_GPT67_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_GPT67_ADD_POE4		POE4	
PDL_POE_HI_Z_GPT67_ADD_POE8		POE8	
PDL_POE_HI_Z_GPT67_ADD_POE10		POE10	
PDL_POE_HI_Z_GPT67_ADD_POE11		POE11	

PDL_POE_HI_Z_MT67_ADD_CFLAG	Comparator detection		MTU channel 6/7 and GPT channel 4 to 6 outputs.
PDL_POE_HI_Z_MT67_ADD_POE0	A valid edge on the pin:	POE0	
PDL_POE_HI_Z_MT67_ADD_POE8		POE8	
PDL_POE_HI_Z_MT67_ADD_POE10		POE10	
PDL_POE_HI_Z_MT67_ADD_POE11		POE11	
PDL_POE_HI_Z_MT67_ADD_POE12		POE12	

[data4]
Select the output pins to be controlled.
If multiple selections are required, use “|” to separate each selection.
All selections are optional. Specify PDL_NO_DATA if none are required.

PDL_POE_HI_Z_ENABLE_MTIOC0A	MTU channel 0. Input pin or event high impedance request, software control or the oscillation stop detection flag.
PDL_POE_HI_Z_ENABLE_MTIOC0B	
PDL_POE_HI_Z_ENABLE_MTIOC0C	
PDL_POE_HI_Z_ENABLE_MTIOC0D	

Description (3/4)

PDL_POE_HI_Z_ENABLE_MTIOC7BD PDL_POE_HI_Z_ENABLE_MTIOC7AC PDL_POE_HI_Z_ENABLE_MTIOC6BD	MTU channel 6 and 7, GPT channel 4, 5 and 6. Input pin or event high impedance request, output short detection, software control or the oscillation stop detection flag.

PDL_POE_HI_Z_ENABLE_MTIOC4BD PDL_POE_HI_Z_ENABLE_MTIOC4AC PDL_POE_HI_Z_ENABLE_MTIOC3BD	MTU channel 3 and 4, GPT channel 0, 1 and 2. Input pin or event high impedance request, output short detection, software control or the oscillation stop detection flag.

PDL_POE_HI_Z_ENABLE_GTIOC0 PDL_POE_HI_Z_ENABLE_GTIOC1 PDL_POE_HI_Z_ENABLE_GTIOC2 PDL_POE_HI_Z_ENABLE_GTIOC3 PDL_POE_HI_Z_ENABLE_GTIOC6 PDL_POE_HI_Z_ENABLE_GTIOC7	GPT channel pins A and B. Input pin or event high impedance request, software control or the oscillation stop detection flag.

[data5]
Output short detection and response. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Output short detection channel

PDL_POE_SHORT_MTU34 or PDL_POE_SHORT_MTU67	Selects whether MTU3/4 or MTU6/7 are used as the MTU complementary PWM output channel

- Output short detection level for MTU channel 3 and 4

PDL_POE_SHORT_USE_MTU or PDL_POE_SHORT_SPECIFY	Use the active levels specified in the MTU3 functions, or enable the settings below.
PDL_POE_SHORT_P71_LOW or PDL_POE_SHORT_P71_HIGH	Select the port pin active level for detection of short circuits. Ignored if the MTU settings for channel 3/4 are used.
PDL_POE_SHORT_P74_LOW or PDL_POE_SHORT_P74_HIGH	
PDL_POE_SHORT_P72_LOW or PDL_POE_SHORT_P72_HIGH	
PDL_POE_SHORT_P75_LOW or PDL_POE_SHORT_P75_HIGH	
PDL_POE_SHORT_P73_LOW or PDL_POE_SHORT_P73_HIGH	
PDL_POE_SHORT_P76_LOW or PDL_POE_SHORT_P76_HIGH	

- Output short detection level for MTU channel 6 and 7

PDL_POE_SHORT_67_USE_MTU or PDL_POE_SHORT_67_SPECIFY	Use the active levels specified in the MTU3 (channel 6/7) functions, or enable the settings below.
PDL_POE_SHORT_67_P95_LOW or PDL_POE_SHORT_67_P95_HIGH	Select the port pin active level for detection of short circuits. Ignored if the MTU settings for channel 6/7 are used.
PDL_POE_SHORT_67_P92_LOW or PDL_POE_SHORT_67_P92_HIGH	
PDL_POE_SHORT_67_P94_LOW or PDL_POE_SHORT_67_P94_HIGH	
PDL_POE_SHORT_67_P91_LOW or PDL_POE_SHORT_67_P91_HIGH	
PDL_POE_SHORT_67_P93_LOW or PDL_POE_SHORT_67_P93_HIGH	
PDL_POE_SHORT_67_P90_LOW or PDL_POE_SHORT_67_P90_HIGH	

Description (4/4)

- Output short response

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

Return value

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

Category

Port Output Enable

Reference

R_POE_Control, R_POE_GetStatus, R_MTU3_Set

Remarks

- Pins POE4, POE10-E4 and POE12 are not available on the 64- and 48-pin packages.
  Pin POE12 is not available on the 100-pin package.
  The pin selection for POE10 is available on the package larger than 64-pin.
- Do not select MTU pins that are not used.
- Using R_POE_GetStatus to get the oscillation stop detection flag.
- All settings related to POE4 of data2 are not available on the 64- and 48-pin packages.
- All settings related to POE12 of data2 are not available on the 100-, 64- and 48-pin packages.
- All settings related to GPT6/7, MTU6/7, MTIOC7n (m: AC, BD), GTIOCn (n: 6, 7) are not available on the 64- and 48-pin packages.
- The settings PDL_POE_HI_Z_MT34_ADD_CFLAG and PDL_POE_HI_Z_MT34_ADD_POEn (n: 8, 10, 11) also support to control Hi-z for MTU6/7 on the 64- and 48-pin packages.
- The setting PDL_POE_SHORT_MTUm (m: 34, 67) is available on 64- and 48-pin packages.
- The settings PDL_POE_SHORT_USE_MTU, PDL_POE_SHORT_SPECIFY and PDL_POE_SHORT_Pm_HIGH/LOW (m: 71 to 76) also support to set active level for MTU6/7 on the 64- and 48-pin packages.

Program example

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

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

void func(void)
{
    /* Configure POE pins*/
    R_POE_Set(
        PDL_POE_0_MODE_EDGE   | \
        PDL_POE_8_MODE_LOW_16 | PDL_POE_10_MODE_LOW_128,
        PDL_POE_HI_Z_REQ_8_ENABLE | \
        PDL_POE_HI_Z_MT34_ADD_CFLAG | PDL_POE_HI_Z_GPT01_ADD_POE11,
        PDL_NO_DATA,
        PDL_POE_HI_Z_ENABLE_MTIOC0A | \
        PDL_POE_HI_Z_ENABLE_MTIOC3BD | PDL_POE_HI_Z_ENABLE_GTIOC3,
        PDL_POE_SHORT_SPECIFY | PDL_POE_SHORT_P71_HIGH | \
        PDL_POE_SHORT_P7X_HI_Z
    );
}
```

2) R_POE_Create

Synopsis

Configure the Port Output Enable event handling.

Prototype

```
bool R_POE_Create(
        uint16_t data1,     // Input configuration selection
        void * func1,       // Callback function
        void * func2,       // Callback function
        void * func3,       // Callback function
        void * func4,       // Callback function
        void * func5,       // Callback function
        uint8_t data2       // Interrupt priority level
);
```

Description (1/2)

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_DISABLE or PDL_POE_IRQ_HI_Z_0_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE0.
PDL_POE_IRQ_HI_Z_4_DISABLE or PDL_POE_IRQ_HI_Z_4_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE4.
PDL_POE_IRQ_HI_Z_8_DISABLE or PDL_POE_IRQ_HI_Z_8_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE8.
PDL_POE_IRQ_HI_Z_10_DISABLE or PDL_POE_IRQ_HI_Z_10_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE10.
PDL_POE_IRQ_HI_Z_11_DISABLE or PDL_POE_IRQ_HI_Z_11_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE11.
PDL_POE_IRQ_HI_Z_12_DISABLE or PDL_POE_IRQ_HI_Z_12_ENABLE	Disable or enable an interrupt on detection of any high impedance request on pin POE12.

- Output short detection response

PDL_POE_IRQ_SHORT_34_DISABLE or PDL_POE_IRQ_SHORT_34_ENABLE	Disable or enable an interrupt on detection of a short on any MTU channel 3/4 two-phase output pair.
PDL_POE_IRQ_SHORT_67_DISABLE or PDL_POE_IRQ_SHORT_67_ENABLE	Disable or enable an interrupt on detection of a short on any MTU channel 6/7 two-phase output pair.

[func1]
The function to be called when an enabled request on pins POE0 or an output short on MTU channels 3 or 4 occurs.
Specify PDL_NO_FUNC if not required.

[func2]
The function to be called when an enabled request on pin POE4 or an output short on MTU channels 6 or 7 occurs.
Specify PDL_NO_FUNC if not required.

[func3]
The function to be called when an enabled request on pin POE8 occurs.
Specify PDL_NO_FUNC if not required.

[func4]
The function to be called when an enabled request on pin POE10 or POE11 occurs.
Specify PDL_NO_FUNC if not required.

[func5]
The function to be called when an enabled request on pin POE4 or POE12 occurs.
Specify PDL_NO_FUNC if not required.

Description (2/2)

[data2]
The interrupt priority level. Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for all parameters func1 to func4.

Return value

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

Category

Port Output Enable

Reference

R_POE_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.
- All settings related to POE4 and MTU6/7 are not available on the 64- and 48-pin packages.
- All settings related to POE12 are not available on the 100-, 64- and 48-pin packages.
- The settings PDL_POE_IRQ_SHORT_34_ENABLE/DISABLE also support to control an interrupt on detection of a short on any MTU channel 6/7 two-phase output pair on the 64- and 48-pin packages.

Program example

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

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

void POE0_handler(void){}

void func(void)
{
    /* Assign the callback function for pin POE0 */
    R_POE_Create(
        PDL_POE_IRQ_HI_Z_0_ENABLE,
        POE0_handler,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        PDL_NO_FUNC,
        6
    );
}
```

3) R_POE_Control

Synopsis

Control the Port Output Enable module.

Prototype

```
bool R_POE_Control (
        uint16_t data1,    // Control options
        uint16_t data2,    // Control options
        uint16_t data3     // Control options
);
```

Description (1/2)

Change the state of output pins, status flags and interrupt control.

[data1]
Manual high impedance control.
If multiple selections are required, use "|" to separate each selection.
All settings are optional. Specify PDL_NO_DATA if no control is required.

- MTU / GPT channel output high impedance control

PDL_POE_MTU34_HI_Z_ON or PDL_POE_MTU34_HI_Z_OFF	Control the high impedance state of the MTU channel 3/4 outputs.
PDL_POE_MTU67_HI_Z_ON or PDL_POE_MTU67_HI_Z_OFF	Control the high impedance state of the MTU channel 6/7 outputs.
PDL_POE_MTU0_HI_Z_ON or PDL_POE_MTU0_HI_Z_OFF	Control the high impedance state of the MTU channel 0 outputs.
PDL_POE_GPT01_HI_Z_ON or PDL_POE_GPT01_HI_Z_OFF	Control the high impedance state of the GPT channel 0 and 1 outputs.
PDL_POE_GPT23_HI_Z_ON or PDL_POE_GPT23_HI_Z_OFF	Control the high impedance state of the GPT channel 2 and 3 outputs.
PDL_POE_GPT67_HI_Z_ON or PDL_POE_GPT67_HI_Z_OFF	Control the high impedance state of the GPT channel 6 and 7 outputs.

[data2]
Event flag control.
If multiple selections are required, use "|" to separate each selection.
All settings are optional. Specify PDL_NO_DATA if no control is required.

PDL_POE_FLAG_POE0_CLEAR	Select the flags to be cleared.
PDL_POE_FLAG_POE4_CLEAR	
PDL_POE_FLAG_POE8_CLEAR	
PDL_POE_FLAG_POE10_CLEAR	
PDL_POE_FLAG_POE11_CLEAR	
PDL_POE_FLAG_POE12_CLEAR	
PDL_POE_FLAG_SHORT_34_CLEAR	
PDL_POE_FLAG_SHORT_67_CLEAR	
PDL_POE_FLAG_OSTST_CLEAR	

[data3]
Interrupt control.
If multiple selections are required, use "|" to separate each selection.
All settings are optional. Specify PDL_NO_DATA if no control is required.

- High impedance request response

PDL_POE_IRQ_HI_Z_0_DISABLE	Control interrupts on detection of a high impedance request on pin POE0.
PDL_POE_IRQ_HI_Z_0_ENABLE	
PDL_POE_IRQ_HI_Z_4_DISABLE	Control interrupts on detection of a high impedance request on pin POE4.
PDL_POE_IRQ_HI_Z_4_ENABLE	
PDL_POE_IRQ_HI_Z_8_DISABLE	Control interrupts on detection of a high impedance request on pin POE8.
PDL_POE_IRQ_HI_Z_8_ENABLE	
PDL_POE_IRQ_HI_Z_10_DISABLE	Control interrupts on detection of a high impedance request on pin POE10.
PDL_POE_IRQ_HI_Z_10_ENABLE	
PDL_POE_IRQ_HI_Z_11_DISABLE	Control interrupts on detection of a high impedance request on pin POE11.
PDL_POE_IRQ_HI_Z_11_ENABLE	
PDL_POE_IRQ_HI_Z_12_DISABLE	Control interrupts on detection of a high impedance request on pin POE12.
PDL_POE_IRQ_HI_Z_12_ENABLE	

Description (2/2)

- Output short detection response

PDL_POE_IRQ_SHORT_34_DISABLE	Control interrupts on detection of a short on any MTU channel 3/4 two-phase output pair.
PDL_POE_IRQ_SHORT_34_ENABLE	
PDL_POE_IRQ_SHORT_67_DISABLE	Control interrupts on detection of a short on any MTU channel 6/7 two-phase output pair.
PDL_POE_IRQ_SHORT_67_ENABLE	

Return value

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

Category

Port Output Enable

Reference

R_POE_Create

Remark

- 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.
- All settings related to POE4, GPT6/7 and MTU6/7 are not available on the 64- and 48-pin packages.
- All settings related to POE12 are not available on the 100-, 64- and 48-pin packages.
- The settings PDL_POE_MTU34_HI_Z_ON/OFF, PDL_POE_IRQ_SHORT_34_ENABLE/DISABLE, PDL_POE_FLAG_SHORT_34_CLEAR also support to control the outputs of MTU channel 6/7 on the 64- and 48-pin packages.

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 Port Output Enable module status.

Prototype

```
bool R_POE_GetStatus(
        uint16_t * data     // Status flags pointer
);
```

Description

Return the status flags.

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

b15	b14	B13	b12	b11	b10	b9	b8
Output short detection	Output short detection	0 High impedance request detection (more)					
MTU6 or MTU7	MTU3 or MTU4	OSTSTF	POE12	POE11	POE10	-	POE8
0: Not detected 1: Detected	0: Not detected 1: Detected	0: No request 1: Requested					

b7	b6	b5	b4	b3	b2	b1	b0
High impedance request detection on pin POEn							
-	-	-	POE4	-	-	-	POE0
0: No request 1: Requested							

Return value

True.

Category

Port Output Enable

Reference

R_POE_Control

Remarks

- Use R_POE_Control to clear the flags.
- The status flags of POE4 and MTU6/7 are not available on the 64- and 48-pin packages.
- The status flag of POE12 is not available on the 100-, 64- and 48-pin packages.

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
    );
}
```

4.2.15. General PWM Timer

1) R_GPT_Set

Synopsis

Select the I/O pins for the GPT unit.

Prototype

```
bool R_GPT_Set (
    uint8_t data1,      // Channel selection
    uint16_t data2      // Configuration options
);
```

Description (1/2)

Set up the global GPT options.

[data1]
The channel number n (where n = 0 to 3 for 64 and 48 pin packages; n = 0 to 7 for 144, 120, 112, 100 pin packages).

[data2]
Pin configuration for the channel. To set multiple options at the same time, use "|" to separate each value.

- Valid when n = 0

PDL_GPT_PIN_GTIOC0A_P71 or PDL_GPT_PIN_GTIOC0A_PD7	Select P71 or PD7 for GTIOC0A
PDL_GPT_PIN_GTIOC0B_P74 or PDL_GPT_PIN_GTIOC0B_PD6	Select P74 or PD6 for GTIOC0B

- Valid when n = 1

PDL_GPT_PIN_GTIOC1A_P72 or PDL_GPT_PIN_GTIOC1A_PD5	Select P72 or PD5 for GTIOC1A
PDL_GPT_PIN_GTIOC1B_P75 or PDL_GPT_PIN_GTIOC1B_PD4	Select P75 or PD4 for GTIOC1B

- Valid when n = 2

PDL_GPT_PIN_GTIOC2A_P73 or PDL_GPT_PIN_GTIOC2A_PD3	Select P73 or PD3 for GTIOC2A
PDL_GPT_PIN_GTIOC2B_P76 or PDL_GPT_PIN_GTIOC2B_PB7 or PDL_GPT_PIN_GTIOC2B_PB6 or PDL_GPT_PIN_GTIOC2B_PD2	Select P76, PB7, PB6 or PD2 for GTIOC2B

- Valid when n = 3

PDL_GPT_PIN_GTIOC3A_P00 or PDL_GPT_PIN_GTIOC3A_PD1	Select P00 or PD1 for GTIOC3A
PDL_GPT_PIN_GTIOC3B_P01 or PDL_GPT_PIN_GTIOC3B_PD0	Select P01 of PD0 for GTIOC3B

- Valid when n = 4

PDL_GPT_PIN_GTIOC4A_P95	Select P95 for GTIOC4A
PDL_GPT_PIN_GTIOC4B_P92	Select P92 for GTIOC4B

- Valid when n = 5

PDL_GPT_PIN_GTIOC5A_P94	Select P94 for GTIOC5A
PDL_GPT_PIN_GTIOC5B_P91	Select P91 for GTIOC5B

- Valid when n = 6

PDL_GPT_PIN_GTIOC6A_P93 or PDL_GPT_PIN_GTIOC6A_PG3	Select P93 or PG3 for GTIOC6A
PDL_GPT_PIN_GTIOC6B_P90 or PDL_GPT_PIN_GTIOC6B_PG4	Select P90 of PG4 for GTIOC6B

- Valid when n = 7

PDL_GPT_PIN_GTIOC7A_PG0	Select PG0 for GTIOC7A
PDL_GPT_PIN_GTIOC7B_PG1	Select PG1 for GTIOC7B

Description (2/2)

- Valid always

PDL_GPT_PIN_GTETRG0_PB4	Select PB4 for GTETRG0
PDL_GPT_PIN_GTETRG1_P34	Select PB4 for GTETRG1

Return value

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

Category

General PWM Timer unit

Reference

R_GPT_CreateUnit

Remarks

- If there are I/O pins to be used, call this function before calling R_GPT_CreateUnit.
- Not all device packages have all of the pin options.

Program example

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

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

void func(void)
{
    R_GPT_Set(
        0,
        PDL_GPT_PIN_GTIOC0A_P71
    );
}
```

2) R_GPT_CreateUnit

Synopsis

Configure the GPT unit.

Prototype

```
bool R_GPT_CreateUnit (
    uint8_t data1,      // Unit selection
    uint32_t data2      // Configuration options
);
```

Description

Set up the global GPT options.

[data1]
The unit number n (where n = 0 for 64 and 48 pin packages; n = 0, 1 for 144, 120, 112, 100 pin packages).

[data2]
Configure the global options. Use "|" to separate each selection.

- External trigger interrupt control

PDL_GPT_EXT_TRIGGER_INT_DISABLE or PDL_GPT_EXT_TRIGGER_INT_RISING or PDL_GPT_EXT_TRIGGER_INT_FALLING or PDL_GPT_EXT_TRIGGER_INT_BOTH	Disable or enable an interrupt request for rising edge, falling edge or falling and rising edge on pin GTETRG.

- IWDTCLK counter control

PDL_GPT_IWDTCLK_COUNT_CLK_PCLK_DIV_1 or PDL_GPT_IWDTCLK_COUNT_CLK_PCLK_DIV_2 or PDL_GPT_IWDTCLK_COUNT_CLK_PCLK_DIV_4 or PDL_GPT_IWDTCLK_COUNT_CLK_PCLK_DIV_8	Select the clock (PCLKA ÷ 1, 2, 4 or 8) for counting frequency-divided IWDTCLK clock.
PDL_GPT_IWDTCLK_CLK_DIV_1 or PDL_GPT_IWDTCLK_CLK_DIV_16 or PDL_GPT_IWDTCLK_CLK_DIV_128 or PDL_GPT_IWDTCLK_CLK_DIV_256	Select the frequency division ratio (÷ 1, 16, 128 or 256) of the frequency-divided IWDTCLK clock.

- IWDTCLK-derived rising edge skipping control.

PDL_GPT_IWDTCLK_SKIP_NONE or PDL_GPT_IWDTCLK_SKIP_8 or PDL_GPT_IWDTCLK_SKIP_16 or PDL_GPT_IWDTCLK_SKIP_128 or PDL_GPT_IWDTCLK_SKIP_256	Select an interrupt request on every, every 8th, every 16th, every 128th or every 256th rising edge.
PDL_GPT_IWDTCLK_RESULT_SKIP_DISABLE or PDL_GPT_IWDTCLK_RESULT_SKIP_ENABLE	Disable or enable skipping of count results transfers at the same interval as the interrupt request skipping.

- IWDTCLK event interrupt request selection. Each event is disabled by default.

PDL_GPT_IWDTCLK_INT_RISING_ENABLE	IWDTCLK-derived rising edge (using the selected skipping interval).
PDL_GPT_IWDTCLK_INT_DEVIATION_ENABLE	The IWDTCLK frequency deviation has exceeded a permissible limit.
PDL_GPT_IWDTCLK_INT_OVERFLOW_ENABLE	IWDTCLK counter overflow.

- DTC/DMAC event trigger control.

PDL_GPT_EXT_IWDTCLK_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_EXT_IWDTCLK_DMAC_TRIGGER_ENABLE or PDL_GPT_EXT_IWDTCLK_DTC_TRIGGER_ENABLE	Activate the DTC or DMAC on a valid external trigger or enabled IWDTCLK event interrupt.

If a DTC or DMAC trigger is enabled, remember to enable the appropriate interrupt request.

Return value

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

Category

General PWM Timer unit

Reference R_GPT_CreateChannel

Remarks

- The callback function for external trigger and IWDTCLK event interrupts is specified using function func6 when R_GPT_CreateChannel is used to configure GPT channel 0 or 4.

Program example

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

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

void func(void)
{
    R_GPT_CreateUnit(
        0,
        PDL_GPT_IWDTCLK_INT_RISING_ENABLE
    );
}
```

3) R_GPT_CreateChannel

Synopsis

Configure a GPT channel.

Prototype

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

R_GPT_Create_structure members:

```
uint32_t data2      // Control and DTC/DMAC options
uint32_t data3      // Interrupt options
uint32_t data4      // Automatic clearing and hardware control options
uint32_t data5      // Hardware control selections
uint32_t data6      // I/O pin control options
uint32_t data7      // I/O pin control options
uint32_t data8      // ADC trigger and skipping control options
uint32_t data9      // Buffer control options
uint32_t data10     // Negate and Dead-time control options
void * func1        // Callback function
void * func2        // Callback function
void * func3        // Callback function
uint8_t data11      // Interrupt priority level
void * func4        // Callback function
void * func5        // Callback function
void * func6        // Callback function
uint8_t data12      // Interrupt priority level
```

Description (1/7)

Set up a GPT channel.

[data1]
The channel number n (where n = 0 to 3 for 64 and 48 pin packages; n = 0 to 7 for 144, 120, 112, 100 pin packages).

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

- Operation mode.

PDL_GPT_MODE_SAW or	Saw-wave mode.
PDL_GPT_MODE_SAW_ONE_SHOT or	Saw-wave one-shot pulse mode.
PDL_GPT_MODE_TRIANGLE_1 or PDL_GPT_MODE_TRIANGLE_2 or PDL_GPT_MODE_TRIANGLE_3	Triangle-wave PWM mode 1, 2 or 3.

- Counter clock source selection.

PDL_GPT_CLK_PCLK_DIV_1 or PDL_GPT_CLK_PCLK_DIV_2 or PDL_GPT_CLK_PCLK_DIV_4 or PDL_GPT_CLK_PCLK_DIV_8	The internal clock signal PCLKA ÷ 1, 2, 4 or 8.

Description (2/7)

- DTC / DMAC event trigger control.

PDL_GPT_CMICA_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_CMICA_DMAC_TRIGGER_ENABLE or PDL_GPT_CMICA_DTC_TRIGGER_ENABLE	GTCCRA compare match or input capture.
PDL_GPT_CMICB_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_CMICB_DMAC_TRIGGER_ENABLE or PDL_GPT_CMICB_DTC_TRIGGER_ENABLE	GTCCRB compare match or input capture.
PDL_GPT_CMCDDTE_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_CMCDDTE_DMAC_TRIGGER_ENABLE or PDL_GPT_CMCDDTE_DTC_TRIGGER_ENABLE	GTCCRC or GTCCRD input capture or Dead time error.
PDL_GPT_CMEF_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_CMEF_DMAC_TRIGGER_ENABLE or PDL_GPT_CMEF_DTC_TRIGGER_ENABLE	GTCCRE or GTCCRF input capture.
PDL_GPT_OU_DMAC_DTC_TRIGGER_DISABLE or PDL_GPT_OU_DMAC_TRIGGER_ENABLE or PDL_GPT_OU_DTC_TRIGGER_ENABLE	Counter overflow or underflow.

If a DTC or DMAC trigger is enabled, remember to enable the appropriate interrupt request.

[data3]
Interrupt and data transfer options. If multiple selections are required, use “|” to separate each selection.
The default settings are shown in **bold**.

- Event interrupt request selection.

PDL_GPT_IRQ_A_DISABLE or PDL_GPT_IRQ_A_ENABLE	GTCCRA input capture or compare match.
PDL_GPT_IRQ_B_DISABLE or PDL_GPT_IRQ_B_ENABLE	GTCCRB input capture or compare match.
PDL_GPT_IRQ_C_DISABLE or PDL_GPT_IRQ_C_ENABLE	GTCCRC compare match.
PDL_GPT_IRQ_D_DISABLE or PDL_GPT_IRQ_D_ENABLE	GTCCRD compare match.
PDL_GPT_IRQ_E_DISABLE or PDL_GPT_IRQ_E_ENABLE	GTCCRE compare match.
PDL_GPT_IRQ_F_DISABLE or PDL_GPT_IRQ_F_ENABLE	GTCCRF compare match.
PDL_GPT_IRQ_DEADTIME_DISABLE or PDL_GPT_IRQ_DEADTIME_ENABLE	Dead time error.
PDL_GPT_IRQ_OU_DISABLE or PDL_GPT_IRQ_OU_OVER or PDL_GPT_IRQ_OU_UNDER or PDL_GPT_IRQ_OU_BOTH	Select the Overflow / underflow detection.

Description (3/7)

[data4]
Automatic clearing and hardware control options.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Automatic counter clearing.

PDL_GPT_CLEAR_DISABLE or	Automatic clearing is disabled.
PDL_GPT_CLEAR_A or PDL_GPT_CLEAR_B or	Cleared by GTCCRA or GTCCRB input capture.
PDL_GPT_CLEAR_SYNC_CH_0 or PDL_GPT_CLEAR_SYNC_CH_1 or PDL_GPT_CLEAR_SYNC_CH_2 or PDL_GPT_CLEAR_SYNC_CH_3	Clearing at the same time as another channel m (where m ≠ n) in unit 0.
PDL_GPT_CLEAR_SYNC_CH_4 or PDL_GPT_CLEAR_SYNC_CH_5 or PDL_GPT_CLEAR_SYNC_CH_6 or PDL_GPT_CLEAR_SYNC_CH_7	Clearing at the same time as another channel m (where m ≠ n) in unit 1.

- Hardware counter start, stop and clearing.

PDL_GPT_HW_START_DISABLE or PDL_GPT_HW_START_RISING or PDL_GPT_HW_START_FALLING or PDL_GPT_HW_START_BOTH	Disable or enable start control from another peripheral. If enabled, select a start source using parameter data5.
PDL_GPT_HW_STOP_DISABLE or PDL_GPT_HW_STOP_RISING or PDL_GPT_HW_STOP_FALLING or PDL_GPT_HW_STOP_BOTH	Disable or enable stop control from another peripheral. If enabled, select a stop source using parameter data5.
PDL_GPT_HW_CLEAR_DISABLE or PDL_GPT_HW_CLEAR_RISING or PDL_GPT_HW_CLEAR_FALLING or PDL_GPT_HW_CLEAR_BOTH	Disable or enable counter clearing control from another peripheral. If enabled, select a clear source using parameter data5.

[data5]
Hardware control selections.
If multiple selections are required, use "|" to separate each selection.

- Hardware start source selection. Ignored if hardware start control is disabled.

PDL_GPT_HW_START_AN000 or PDL_GPT_HW_START_AN001 or PDL_GPT_HW_START_AN002 or PDL_GPT_HW_START_AN100 or PDL_GPT_HW_START_AN101 or PDL_GPT_HW_START_AN102 or	Comparator detection on 12-bit ADC input ANxxx.
PDL_GPT_HW_START_GTIOC3A_IN or PDL_GPT_HW_START_GTIOC3B_IN or	A valid edge on the GPT pin.
PDL_GPT_HW_START_GTIOC3A_OUT or PDL_GPT_HW_START_GTIOC3B_OUT or	A valid edge on the GPT output compare. Not valid for channel 3.
PDL_GPT_HW_START_GTETRG0 or PDL_GPT_HW_START_GTETRG1	A valid edge on the GTETRGn pin.

- Hardware stop / clear selection. Ignored if both hardware stop and clear control are disabled.

PDL_GPT_HW_STOP_CLEAR_AN000 or PDL_GPT_HW_STOP_CLEAR_AN001 or PDL_GPT_HW_STOP_CLEAR_AN002 or PDL_GPT_HW_STOP_CLEAR_AN100 or PDL_GPT_HW_STOP_CLEAR_AN101 or PDL_GPT_HW_STOP_CLEAR_AN102 or	Comparator detection on 12-bit ADC input ANxxx.
PDL_GPT_HW_STOP_CLEAR_GTIOC3A_IN or PDL_GPT_HW_STOP_CLEAR_GTIOC3B_IN or	A valid edge on the GPT pin.
PDL_GPT_HW_STOP_CLEAR_GTIOC3A_OUT or PDL_GPT_HW_STOP_CLEAR_GTIOC3B_OUT or	A valid edge on the GPT output compare. Not valid for channel 3.
PDL_GPT_HW_STOP_CLEAR_GTETRG0 or PDL_GPT_HW_STOP_CLEAR_GTETRG1	A valid edge on the GTETRGn pin.

Description (4/7)

[data6]
I/O pin control options. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Compare match / Input capture selection for pin GTIOCnA

PDL_GPT_A_DISABLED or	Not used, or
PDL_GPT_A_CM_RETAIN or PDL_GPT_A_CM_LOW or PDL_GPT_A_CM_HIGH or PDL_GPT_A_CM_INVERT or	At a compare match the output is retained, set low, set high or toggled, or
PDL_GPT_A_IC_RISING_EDGE or PDL_GPT_A_IC_FALLING_EDGE or PDL_GPT_A_IC_BOTH_EDGES	Input capture at rising edge, falling edge or both edges.

- Additional output settings for pin GTIOCnA. Required only if Compare match is enabled.

PDL_GPT_A_LOW_LOW or PDL_GPT_A_LOW_HIGH or PDL_GPT_A_HIGH_LOW or PDL_GPT_A_HIGH_HIGH or PDL_GPT_A_RETAIN	Set the output states at counter start and stop.
PDL_GPT_A_CYCLE_RETAIN or PDL_GPT_A_CYCLE_LOW or PDL_GPT_A_CYCLE_HIGH or PDL_GPT_A_CYCLE_INVERT	At the timer cycle end the output is retained, set low, set high or toggled.

[data7]
I/O pin control options. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Compare match / Input capture selection for pin GTIOCnB

PDL_GPT_B_DISABLED or	Not used, or
PDL_GPT_B_CM_RETAIN or PDL_GPT_B_CM_LOW or PDL_GPT_B_CM_HIGH or PDL_GPT_B_CM_INVERT or	At a compare match the output is retained, set low, set high or toggled, or
PDL_GPT_B_IC_RISING_EDGE or PDL_GPT_B_IC_FALLING_EDGE or PDL_GPT_B_IC_BOTH_EDGES	Input capture at rising edge, falling edge or both edges.

- Additional output settings for pin GTIOCnB. Required only if Compare match is enabled.

PDL_GPT_B_LOW_LOW or PDL_GPT_B_LOW_HIGH or PDL_GPT_B_HIGH_LOW or PDL_GPT_B_HIGH_HIGH or PDL_GPT_B_RETAIN	Set the output states at counter start and stop.
PDL_GPT_B_CYCLE_RETAIN or PDL_GPT_B_CYCLE_LOW or PDL_GPT_B_CYCLE_HIGH or PDL_GPT_B_CYCLE_INVERT	At the timer cycle end the output is retained, set low, set high or toggled.

Description (5/7)

[data8]
ADC trigger and skipping control options. If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- ADC conversion trigger selection.

PDL_GPT_ADC_TRIG_A_UP_DISABLE or PDL_GPT_ADC_TRIG_A_UP_ENABLE	Disable or enable ADC start requests on a GTADTRA compare match during up and / or down counting.
PDL_GPT_ADC_TRIG_A_DOWN_DISABLE or PDL_GPT_ADC_TRIG_A_DOWN_ENABLE	
PDL_GPT_ADC_TRIG_B_UP_DISABLE or PDL_GPT_ADC_TRIG_B_UP_ENABLE	Disable or enable ADC start requests on a GTADTRB compare match during up and / or down counting.
PDL_GPT_ADC_TRIG_B_DOWN_DISABLE or PDL_GPT_ADC_TRIG_B_DOWN_ENABLE	

- Interrupt and ADC trigger skipping control.

PDL_GPT_INT_SKIP_OU_DISABLE or PDL_GPT_INT_SKIP_OU_OVER or PDL_GPT_INT_SKIP_OU_UNDER or PDL_GPT_INT_SKIP_OU_BOTH	Disable skipping, or select skipping for overflow (crest), underflow (trough) or both overflow (crest) and underflow (trough).	
PDL_GPT_INT_SKIP_1 or PDL_GPT_INT_SKIP_2 or PDL_GPT_INT_SKIP_3 or PDL_GPT_INT_SKIP_4 or PDL_GPT_INT_SKIP_5 or PDL_GPT_INT_SKIP_6 or PDL_GPT_INT_SKIP_7	If skipping is enabled, select the skipping count.	
PDL_GPT_INT_SKIP_A	If skipping is enabled, select other events to be skipped.	GTCCRA Compare Match or Input Capture
PDL_GPT_INT_SKIP_B		GTCCRB Compare Match or Input Capture
PDL_GPT_INT_SKIP_C		GTCCRC Compare Match
PDL_GPT_INT_SKIP_D		GTCCRD Compare Match
PDL_GPT_INT_SKIP_E		GTCCRE Compare Match
PDL_GPT_INT_SKIP_F		GTCCRF Compare Match
PDL_GPT_ADC_TRIG_SKIP_A		GTADTRA ADC converter start request
PDL_GPT_ADC_TRIG_SKIP_B		GTADTRB ADC converter start request

Description (6/7)

[data9]
Buffer control options. If multiple selections are required, use "|" to separate each selection. The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- GTCCRA buffer operation

PDL_GPT_BUFFER_CMIC_A_DISABLE or PDL_GPT_BUFFER_CMIC_A_SINGLE or PDL_GPT_BUFFER_CMIC_A_DOUBLE	Disable or select single or double buffer operation for register GTCCRA.

- GTCCRB buffer operation

PDL_GPT_BUFFER_CMIC_B_DISABLE or PDL_GPT_BUFFER_CMIC_B_SINGLE or PDL_GPT_BUFFER_CMIC_B_DOUBLE	Disable or select single or double buffer operation for register GTCCRB.

- GTPR buffer operation

PDL_GPT_BUFFER_CYCLE_DISABLE or PDL_GPT_BUFFER_CYCLE_SINGLE or PDL_GPT_BUFFER_CYCLE_DOUBLE	Disable or select single or double buffer operation for register GTPR.

- GTADTRA buffer operation

PDL_GPT_BUFFER_ADC_TRIG_A_DISABLE or	Disable buffer operation for register GTADTRA or select one of the following:
PDL_GPT_BUFFER_ADC_TRIG_A_CREST or PDL_GPT_BUFFER_ADC_TRIG_A_TROUGH or PDL_GPT_BUFFER_ADC_TRIG_A_BOTH or	Triangle waves: Select transfer at crest, trough or crest and trough.
PDL_GPT_BUFFER_ADC_TRIG_A_SAW	Saw waves: Select transfer at underflow or overflow.

PDL_GPT_BUFFER_ADC_TRIG_A_SINGLE or PDL_GPT_BUFFER_ADC_TRIG_A_DOUBLE	If GTADTRA buffer operation is enabled, select single or double buffer operation.

- GTADTRB buffer operation

PDL_GPT_BUFFER_ADC_TRIG_B_DISABLE or	Disable buffer operation for register GTADTRB or select one of the following:
PDL_GPT_BUFFER_ADC_TRIG_B_CREST or PDL_GPT_BUFFER_ADC_TRIG_B_TROUGH or PDL_GPT_BUFFER_ADC_TRIG_B_BOTH or	Triangle waves: Select transfer at crest, trough or crest and trough.
PDL_GPT_BUFFER_ADC_TRIG_B_SAW	Saw waves: Select transfer at underflow or overflow.

PDL_GPT_BUFFER_ADC_TRIG_B_SINGLE or PDL_GPT_BUFFER_ADC_TRIG_B_DOUBLE	If GTADTRB buffer operation is enabled, select single or double buffer operation.

Description (7/7)

[data10]
Negate and Dead-time control options.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**. Specify PDL_NO_DATA to use the defaults.

- Negate control selection

PDL_GPT_NEGATE_A_DISABLE or PDL_GPT_NEGATE_A_LOW or PDL_GPT_NEGATE_A_HIGH	Disable negate operation or set the negate state for pin GTIOCnA.
PDL_GPT_NEGATE_B_DISABLE or PDL_GPT_NEGATE_B_LOW or PDL_GPT_NEGATE_B_HIGH	Disable negate operation or set the negate state for pin GTIOCnB.
PDL_GPT_NEGATE_AN000 or PDL_GPT_NEGATE_AN001 or PDL_GPT_NEGATE_AN002 or PDL_GPT_NEGATE_AN100 or PDL_GPT_NEGATE_AN101 or PDL_GPT_NEGATE_AN102 or PDL_GPT_NEGATE_GTETRG0 or PDL_GPT_NEGATE_GTETRG1 or PDL_GPT_NEGATE_SOFTWARE	If negate operation is enabled, select the negate source.
PDL_GPT_NEGATE_0 or PDL_GPT_NEGATE_1	If negate operation is enabled, select operation when the source becomes 0 or 1.

- Dead time control selection. Ignored in saw-wave PWM mode.

PDL_GPT_DEAD_TIME_DISABLE or PDL_GPT_DEAD_TIME_ENABLE	Disable or enable dead time control.
PDL_GPT_DEAD_TIME_UP_BUFFER_DISABLE or PDL_GPT_DEAD_TIME_UP_BUFFER_ENABLE	Disable or enable buffer operation for up-counting.
PDL_GPT_DEAD_TIME_DOWN_BUFFER_DISABLE or PDL_GPT_DEAD_TIME_DOWN_BUFFER_ENABLE or PDL_GPT_DEAD_TIME_DOWN_MATCH_UP	Disable or enable buffer operation for down-counting, or select loading of the up-counting value.

If any callback function is not required, specify PDL_NO_FUNC.

[func1]
The function to be called when a Compare match / Input capture A event occurs.

[func2]
The function to be called when a Compare match / Input capture B event occurs.

[func3]
The function to be called when a Compare match C, Compare match D or Dead Time error event occurs.

[data11]
The interrupt priority level for the above events.
Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for all parameters func(1 to 3).

[func4]
The function to be called when a Compare match E or Compare match F event occurs.

[func5]
The function to be called when an overflow and / or underflow occurs.

[func6]
The function to be called when an enabled external trigger or IWDTCLK counter interrupt request occurs. Ignored for $n \neq 0, 4$.

[data12]
The interrupt priority level for the above events.
Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for all parameters func(4 to 6).

Return value

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

Category

General PWM Timer unit

Reference

R_GPT_CreateUnit, R_GPT_ControlChannel, R_GPT_ControlUnit

Remarks

- R_GPT_CreateUnit should be called in advanced of this API.
- Use R_GPT_ControlChannel to load the registers and start the timer.
- If hardware start or stop control is enabled on a channel, starting or stopping any channel using R_GPT_ControlChannel or R_GPT_ControlUnit may cause the channels that use hardware start or stop control to change state in error.
- 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.
- A companion function, R_GPT_Create_load_defaults, can be used to load the default values into the structure.
- Setting for channel 4, 5, 6, 7 and unit 1 only available in 144, 120, 112, 100 pin packages.

Program example

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

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

void func(void)
{
    /* Allocate a copy of the structure */
    R_GPT_Create_structure ch_parameters;

    /* Load the defaults */
    R_GPT_Create_load_defaults(&ch_parameters);

    /* Set the non-default options for channel 4 */
    ch_parameters.data2 = PDL_GPT_MODE_SAW | PDL_GPT_CLK_PCLK_DIV_1;

    R_GPT_CreateChannel(
        2,
        &ch_parameters
    );
}
```

4) R_GPT_Destroy

Synopsis

Disable the GPT unit.

Prototype

```
bool R_GPT_Destroy(
    uint8_t data    // Unit selection
);
```

Description

Shut down the GPT unit and reduce the power consumption.

[data]
The unit number n (where n = 0 for 64 and 48 pin packages; n= 0, 1 for 144, 120, 112, 100 pin packages).

Return value

True.

Category

General PWM Timer unit

Reference

None.

Remarks

- The unit is put into the stop state.

Program example

```
#include “r_pdl_gpt.h”

void func(void)
{
    /* Shutdown all GPT channels */
    R_GPT_Destroy(
         0
    );
}
```

5) R_GPT_ControlChannel

Synopsis

Control a GPT channel.

Prototype

```
bool R_GPT_ControlChannel(
    uint8_t data1,                          // Channel selection
    uint32_t data2,                         // Control options
    uint32_t data3,                         // Register selection
    R_GPT_ControlChannel_structure ptr      // A pointer to the register structure
);
```

R_GPT_ControlChannel_structure members:

```
    uint16_t data4      // Timer counter register value
    uint16_t data5      // General register A value
    uint16_t data6      // General register B value
    uint16_t data7      // General register C value
    uint16_t data8      // General register D value
    uint16_t data9      // General register E value
    uint16_t data10     // General register F value
    uint16_t data11     // Cycle setting register value
    uint16_t data12     // Cycle setting buffer register value
    uint16_t data13     // Cycle setting double buffer register value
    uint16_t data14     // ADC start request A register value
    uint16_t data15     // ADC start request A buffer register value
    uint16_t data16     // ADC start request A double buffer register value
    uint16_t data17     // ADC start request B register value
    uint16_t data18     // ADC start request B buffer register value
    uint16_t data19     // ADC start request B double buffer register value
    uint16_t data20     // Dead-time up-counting register value
    uint16_t data21     // Dead-time up-counting buffer register value
    uint16_t data22     // Dead-time down-counting register value
    uint16_t data23     // Dead-time down-counting buffer register value
```

Description (1/3)

Modify a timer channel's registers.

[data1]
The channel number n (where n = 0 to 3 for 64 and 48 pin packages; n = 0 to 7 for 144, 120, 112, 100 pin packages).

[data2]
The channel settings to be modified. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Counter stop / start

PDL_GPT_STOP	Stop the count operation.
PDL_GPT_START	Start the count operation.

- Counter clearing

PDL_GPT_COUNTER_CLEAR	Clear the counter.

- Buffer operation control

PDL_GPT_BUFFER_CMIC_STOP	Disable and/or enable buffer operation using Compare match or Input capture.
PDL_GPT_BUFFER_CMIC_START	
PDL_GPT_BUFFER_CYCLE_STOP	Disable and/or enable buffer operation using over/under flow.
PDL_GPT_BUFFER_CYCLE_START	
PDL_GPT_BUFFER_ADC_TRIG_STOP	Disable and/or enable buffer operation using ADC start triggers.
PDL_GPT_BUFFER_ADC_TRIG_START	
PDL_GPT_BUFFER_DEAD_TIME_STOP	Disable and/or enable buffer operation using dead time up/down compare match.
PDL_GPT_BUFFER_DEAD_TIME_START	

- Forced buffer operation

PDL_GPT_BUFFER_CMIC_FORCE	Force a buffer transfer of registers GTCCRA and GTCCRB.

Description (2/3)

- Write protection

PDL_GPT_WRITE_ENABLE	Allow writing to the channel registers.
PDL_GPT_WRITE_DISABLE	Prevent writing to the channel registers.

- Count direction

PDL_GPT_COUNT_DIRECTION_DOWN or PDL_GPT_COUNT_DIRECTION_UP	Set the count direction.
PDL_GPT_COUNT_DIRECTION_FORCE	Forcibly set the count direction.

- Software negate control

PDL_GPT_NEGATE_ON or PDL_GPT_NEGATE_OFF	If software negate control has been enabled (in parameter data10 of R_GPT_CreateChannel), control the negate state.

- Output protection temporary release control

PDL_GPT_OUTPUT_PROTECTION_RELEASE or PDL_GPT_OUTPUT_PROTECTION_RESTORE	Release or restore the protected state of the GTIOCnB pin

[data3]
The registers to be modified. All selections are optional.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- The registers to be modified.

PDL_GPT_REGISTER_COUNTER	Timer counter.
PDL_GPT_REGISTER_A	General register A.
PDL_GPT_REGISTER_B	General register B.
PDL_GPT_REGISTER_C	General register C.
PDL_GPT_REGISTER_D	General register D.
PDL_GPT_REGISTER_E	General register E.
PDL_GPT_REGISTER_F	General register F.
PDL_GPT_REGISTER_CYCLE	Cycle setting register.
PDL_GPT_REGISTER_CYCLE_BUFFER	Cycle setting buffer register.
PDL_GPT_REGISTER_CYCLE_DOUBLE	Cycle setting double buffer register.
PDL_GPT_REGISTER_ADC_TRIG_A	ADC start request register.
PDL_GPT_REGISTER_ADC_TRIG_A_BUFFER	ADC start request buffer register.
PDL_GPT_REGISTER_ADC_TRIG_A_DOUBLE	ADC start request double buffer register.
PDL_GPT_REGISTER_ADC_TRIG_B	ADC start request register.
PDL_GPT_REGISTER_ADC_TRIG_B_BUFFER	ADC start request buffer register.
PDL_GPT_REGISTER_ADC_TRIG_B_DOUBLE	ADC start request double buffer register.
PDL_GPT_REGISTER_DEAD_TIME_UP	Dead-time up-counting register.
PDL_GPT_REGISTER_DEAD_TIME_UP_BUFFER	Dead-time up-counting buffer register.
PDL_GPT_REGISTER_DEAD_TIME_DOWN	Dead-time down-counting register.
PDL_GPT_REGISTER_DEAD_TIME_DOWN_BUFFER	Dead-time down-counting buffer register.

The following register values will be ignored if they have not been selected above.

[data4]
The timer counter (GTCNT) value.

[data5]
The general register A (GTCCRA) value.

[data6]
The general register B (GTCCRB) value.

[data7]
The general register C (GTCCRC) value.

[data8]
The general register D (GTCCRD) value.

Description (3/3)

[data9]
The general register E (GTCCRE) value.

[data10]
The general register F (GTCCRF) value.

[data11]
The cycle setting register (GTPR) value.

[data12]
The cycle setting buffer register (GTPBR) value.

[data13]
The cycle setting double buffer register (GTPDBR) value.

[data14]
The ADC start request register (GTADTRA) value.

[data15]
The ADC start request buffer register (GTADTBRA) value.

[data16]
The ADC start request double buffer register (GTADTDBRA) value.

[data17]
The ADC start request register (GTADTRB) value.

[data18]
The ADC start request buffer register (GTADTBRB) value.

[data19]
The ADC start request double buffer register (GTADTDBRB) value.

[data20]
The dead-time up-counting register (GTDVU) value.

[data21]
The dead-time up-counting buffer register (GTDBU) value.

[data22]
The dead-time down-counting register (GTDVD) value.

[data23]
The dead-time down-counting buffer register (GTDBD) value.

Return value

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

Category

General PWM Timer unit

Reference

R_GPT_CreateChannel

Remarks

- The Stop operation is executed at the start of this function.
  The Start operation is executed at the end.
  Both options can be selected together with other changes in one function call.
- The buffer operation disable control is executed at the start of this function.
  The buffer operation enable control is executed at the end.
  Both options can be selected together with buffer register access in one function call.
- If Stop or Start control is selected, any channel that has hardware start or stop control enabled may change state in error.
- The IWDTCLK counter Stop operation is executed at the start of this function.
  The Start operation is executed at the end.
  Both options can be selected together with other changes in one function call.

Program example

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

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

void func(void)
{
    /* Allocate a copy of the structure for the selected channel */
    R_GPT_ControlChannel_structure gpt_1_control_parameters;

    /* Set the register values for channel 1 */
    gpt_1_control_parameters.data4 = 0xFFDD;
    gpt_1_control_parameters.data6 = 0x0020;

    /* Load the register values and start channel 1 */
    R_GPT_ControlChannel(
         1,
         PDL_GPT_STOP | PDL_GPT_START,
         PDL_GPT_REGISTER_COUNTER | PDL_GPT_REGISTER_B,
         &gpt_1_control_parameters
    );
}
```

6) R_GPT_ControlUnit

Synopsis

Control the GPT unit.

Prototype

```
bool R_GPT_ControlUnit(
    uint8_t data1,     // Unit selection
    uint32_t data2,    // Control options
    uint16_t data3,    // Clock positive tolerance
    uint16_t data4     // Clock negative tolerance
);
```

Description (1/2)

Modify the timer unit registers.

[data1]
The unit number n (where n = 0 for 64 and 48 pin packages; n= 0, 1 for 144, 120, 112, 100 pin packages).

[data2]
The unit settings to be modified.
If multiple selections are required, use "|" to separate each selection.
Specify PDL_NO_DATA if no change is required.

- Counter stop

PDL_GPT_STOP_CH_0	Stop the count operation for the selected channels in unit 0. The stop operation will be simultaneous.
PDL_GPT_STOP_CH_1	
PDL_GPT_STOP_CH_2	
PDL_GPT_STOP_CH_3	
PDL_GPT_STOP_CH_4	Stop the count operation for the selected channels in unit 1. The stop operation will be simultaneous
PDL_GPT_STOP_CH_5	
PDL_GPT_STOP_CH_6	
PDL_GPT_STOP_CH_7	

- Counter start

PDL_GPT_START_CH_0	Start the count operation for the selected channels in unit 0. The start operation will be simultaneous.
PDL_GPT_START_CH_1	
PDL_GPT_START_CH_2	
PDL_GPT_START_CH_3	
PDL_GPT_START_CH_4	Start the count operation for the selected channels in unit 1. The start operation will be simultaneous.
PDL_GPT_START_CH_5	
PDL_GPT_START_CH_6	
PDL_GPT_START_CH_7	

- Counter clearing

PDL_GPT_CLEAR_CH_0	Clear the counters for the selected channels in unit 0. The clear operation will be simultaneous.
PDL_GPT_CLEAR_CH_1	
PDL_GPT_CLEAR_CH_2	
PDL_GPT_CLEAR_CH_3	
PDL_GPT_CLEAR_CH_4	Clear the counters for the selected channels in unit 1. The clear operation will be simultaneous.
PDL_GPT_CLEAR_CH_5	
PDL_GPT_CLEAR_CH_6	
PDL_GPT_CLEAR_CH_7	

- IWDTCLK counter stop / start

PDL_GPT_IWDTCLK_COUNT_DISABLE	Stop the IWDTCLK count operation.
PDL_GPT_IWDTCLK_COUNT_ENABLE	Start the IWDTCLK count operation.

- IWDTCLK counter clearing

PDL_GPT_IWDTCLK_COUNT_CLEAR	Clear the IWDTCLK counter (LCNT) to 0.

- IWDTCLK count result initialisation

PDL_GPT_IWDTCLK_RESULTS_INIT	Initialise the IWDTCLK count result registers (LCNT01 to LCNT15) using the first count (LCNT00) value.

Description (2/2)

- IWDTCLK count deviation update

PDL_GPT_IWDTCLK_DEVIATION_TOLERANCE or	Parameters data3 and data4 contain the positive and negative tolerances.
PDL_GPT_IWDTCLK_DEVIATION_REGISTER	Parameters data3 and data4 contain the register values.

[data3]
If required, either the maximum positive deviation for the main clock (ICLK) as a percentage, or the LCNTDU register value.
If the IWDTCLK count deviation update is not required, specify PDL_NO_DATA.

[data4]
If required, either the maximum negative deviation for the main clock (ICLK) as a percentage, or the LCNTDL register value.
If the IWDTCLK count deviation update is not required, specify PDL_NO_DATA.

Return value

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

Category

General PWM Timer unit

Reference

R_GPT_CreateUnit, R_IWDT_Set, R_IWDT_Control

Remarks

- The Stop operations are executed at the start of this function.
  The Start operations are executed at the end.
  Both options can be selected together with other changes in one function call.
- If Stop or Start control is selected, any channel that has hardware start or stop control enabled may change state in error.
- If the IWDTCLK counter is enabled, the Independent Watchdog Timer (IWDT) should also be enabled.
  The following sequence is recommended:
  a) Use R_GPT_CreateUnit to configure the IWDTCLK counter.
  b) Use R_IWDT_Set and R_IWDT_Control to configure and start the IWDT. If the IWDTCLK frequency division is 16 or more, avoid selecting PDL_IWDT_CLOCK_OCO_1.
  c) Use this function to initialise and start the IWDTCLK counter.
- Setting for channel 4, 5, 6, 7 only available in 144, 120, 112, 100 pin packages.

Program example

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

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

void func(void)
{
    /* Stop, clear and re-start channels 0 and 1 simultaneously */
    R_GPT_ControlUnit(
         0,
         PDL_GPT_STOP_CH_0  | PDL_GPT_STOP_CH_1 | \
         PDL_GPT_CLEAR_CH_0 | PDL_GPT_CLEAR_CH_1 | \
         PDL_GPT_START_CH_0 | PDL_GPT_START_CH_1,
         PDL_NO_DATA,
         PDL_NO_DATA
    );

    /* Update the IWDTCLK count deviation limits to +10% & -20% */
    R_GPT_ControlUnit(
         0,
         PDL_GPT_IWDTCLK_DEVIATION_TOLERANCE,
         10,
         20
    );
}
```

7) R_GPT_ReadChannel

Synopsis

Read from GPT channel registers.

Prototype

```
bool R_GPT_ReadChannel(
    uint8_t data1,      // Channel selection
    uint16_t * data2,   // Flag storage location
    uint16_t * data3,   // Timer counter register storage location
    uint16_t * data4,   // General register A storage location
    uint16_t * data5,   // General register B storage location
    uint16_t * data6,   // General register C storage location
    uint16_t * data7,   // General register D storage location
    uint16_t * data8,   // General register E storage location
    uint16_t * data9,   // General register F storage location
    uint16_t * data10,  // Cycle setting register storage location
    uint16_t * data11,  // Cycle setting buffer register storage location
    uint16_t * data12,  // Cycle setting double buffer register storage location
    uint16_t * data13,  // ADC start request A register storage location
    uint16_t * data14,  // ADC start request A buffer register storage location
    uint16_t * data15,  // ADC start request A double buffer register storage location
    uint16_t * data16,  // ADC start request B register storage location
    uint16_t * data17,  // ADC start request B buffer register storage location
    uint16_t * data18,  // ADC start request B double buffer register storage location
    uint16_t * data19,  // Dead-time up-counting register storage location
    uint16_t * data20,  // Dead-time up-counting buffer register storage location
    uint16_t * data21,  // Dead-time down-counting register storage location
    uint16_t * data22   // Dead-time down-counting buffer register storage location
);
```

Description (1/2)

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

[data1]
The channel number n (where n = 0 to 3 for 64 and 48 pin packages; n = 0 to 7 for 144, 120, 112, 100 pin packages).

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

b15	b14	b13 - b12	b11	b10 - b8
Timer counter status		Output protected state	Dead-time error detection	
0: Down 1: Up	0: Stopped 1: Counting	00b: Normal Other: Protected, due to the GTCCRA value at buffer transfer: 01b: GTCCRA = 0 10b: GTCCRA ≥ GTPR at trough 11b: GTCCRA ≥ GTPR at crest	0: Not detected 1: Detected	Interrupt skipping counter

b7	b6	b5	b4	b3	b2	b1	b0
Event detection							
Underflow	Overflow	Compare match				Input capture or compare match	
		F	E	D	C	B	A
0: Not detected 1: Detected							

If any register value is not required, specify PDL_NO_PTR.

[data3]
A pointer to where the Timer counter register value shall be stored.

[data4]
A pointer to where the General register A value shall be stored.

[data5]
A pointer to where the General register B value shall be stored.

Description (2/2)

[data6]
A pointer to where the General register C value shall be stored.

[data7]
A pointer to where the General register D value shall be stored.

[data8]
A pointer to where the General register E value shall be stored.

[data9]
A pointer to where the General register F value shall be stored.

[data10]
A pointer to where the Cycle setting register value shall be stored.

[data11]
A pointer to where the Cycle setting buffer register value shall be stored.

[data12]
A pointer to where the Cycle setting double buffer register value shall be stored.

[data13]
A pointer to where the ADC start request A register value shall be stored.

[data14]
A pointer to where the ADC start request A buffer register value shall be stored.

[data15]
A pointer to where the ADC start request A double buffer register value shall be stored.

[data16]
A pointer to where the ADC start request B register value shall be stored.

[data17]
A pointer to where the ADC start request B buffer register value shall be stored.

[data18]
A pointer to where the ADC start request B double buffer register value shall be stored.

[data19]
A pointer to where the Dead-time up-counting register value shall be stored.

[data20]
A pointer to where the Dead-time up-counting buffer register value shall be stored.

[data21]
A pointer to where the Dead-time down-counting register value shall be stored.

[data22]
A pointer to where the Dead-time down-counting buffer register value shall be stored.

Return value

True.

Category

General PWM Timer 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_gpt.h"

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

uint16_t Flags;
uint16_t General_A;
uint16_t General_D;

void func(void)
{
    /* Read the status flags and general registers A and D of channel 3
*/
    R_GPT_ReadChannel(
         3,
         &Flags,
         &General_A,
         PDL_NO_PTR,
         PDL_NO_PTR,
         &General_D,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR,
         PDL_NO_PTR
    );
}
```

8) R_GPT_ReadUnit

Synopsis

Read the GPT unit flags.

Prototype

```
bool R_GPT_ReadUnit(
    uint8_t data1,      // Unit selection
    uint8_t * data2,    // The flag storage location
    uint16_t * data3,   // The IWDTCLK counter storage location
    uint16_t * data4,   // The IWDTCLK count result average storage location
    uint16_t * data5,   // The IWDTCLK count upper permissible deviation location
    uint16_t * data6,   // The IWDTCLK count lower permissible deviation location
    uint16_t * data7    // The IWDTCLK count result storage location
);
```

Description

Read the GPT unit status and data

[data1]
The unit number n (where n = 0 for 64 and 48 pin packages; n= 0, 1 for 144, 120, 112, 100 pin packages).

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

b7 – b5	b4	b3	b2	b1	b0
0	External trigger edge		IWDTCLK counter		
	Falling	Rising	Overflow	Deviation limit	IWDTCLK-divided rising edge
	0: Not detected 1: Detected				

If any register value is not required, specify PDL_NO_PTR.

[data3]
A pointer to where the IWDTCLK counter (LCNT) register value shall be stored.

[data4]
A pointer to where the IWDTCLK count result average (LCNTA) register value shall be stored.

[data5]
A pointer to where the IWDTCLK count upper permissible deviation (LCNTDU) register value shall be stored.

[data6]
A pointer to where the IWDTCLK count lower permissible deviation (LCNTDL) register value shall be stored.

[data7]
A pointer to where the IWDTCLK count result (LCNTxx) registers value shall be stored.
Provide space for 16 sixteen-bit values.

Return value

True.

Category

General PWM Timer unit

Reference

None.

Remarks

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

Program example

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

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

uint8_t Flags;
uint16_t iwdtclk_counter;
uint16_t iwdtclk_counter_result_average;
uint16_t iwdtclk_counter_upper_limit;
uint16_t iwdtclk_counter_lower_limit;
uint16_t iwdtclk_counter_results[16];

void func(void)
{
    /* Read the status flags and IWDTCLK count registers */
    R_GPT_ReadUnit(
         0,
         &Flags,
         &iwdtclk_counter,
         &iwdtclk_counter_result_average,
         &iwdtclk_counter_upper_limit,
         &iwdtclk_counter_lower_limit,
         iwdtclk_counter_results
    );
}
```

9) R_GPT_EdgeDelay_Create

Synopsis

Enable the PWM Edge Delay circuit.

Prototype

```
bool R_GPT_EdgeDelay_Create(
    uint8 data1,    // Channel 0 configuration
    uint8 data2,    // Channel 1 configuration
    uint8 data3,    // Channel 2 configuration
    uint8 data4     // Channel 3 configuration
);
```

Description

Enable the PWM Edge Delay circuit.

[data1]
Channel 0 configuration. Specify PDL_NO_DATA to use the default.

- Configuration

PDL_GPT_PWM_DELAY_LEAVE	Leave the current configuration.
PDL_GPT_PWM_DELAY_ENABLE	Enable and activate the delay circuit.
PDL_GPT_PWM_DELAY_DISABLE	Disable the delay circuit.

[data2]
Channel 1 configuration. See [data1] for format.

[data3]
Channel 2 configuration. See [data1] for format.

[data4]
Channel 3 configuration. See [data1] for format.

Return value

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

Category

General PWM Timer unit

Reference

None.

Remarks

- During execution of this function the ICLK frequency is temporarily halved in value. Therefore, it is recommended to call this function before enabling other modules that rely on ICLK.
- If the GPT module is in the stopped state, the stop state will be cancelled.
- If a delay is going to be enabled for a channel then the counter for that channel will be stopped.
- Function R_CGC_Set must be called before any use of this function.
- ICLK must be ≥ 80MHz to enable the Delay circuit. This function will return false if this is not the case.
- The initialisation process of the delay circuit includes a waiting time that this function automatically provides. If this function is used to enable all of the required delay channels at the same time then this delay period is shared, thus minimising the overall initialisation time.
- When a channel is first enabled this function will set the delay time to zero (no delay). Use function R_GPT_EdgeDelay_Control to set the required delay time.
- It is recommended to use this function before using R_GPT_CreateChannel.

Program example

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

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

void func(void)
{
    /* Enable the PWM delay circuit for channels 0 and 2. */
    R_GPT_EdgeDelay_Create(
         PDL_GPT_PWM_DELAY_ENABLE,
         PDL_GPT_PWM_DELAY_DISABLE,
         PDL_GPT_PWM_DELAY_ENABLE,
         PDL_GPT_PWM_DELAY_DISABLE
    );
}
```

10) R_GPT_EdgeDelay_Control

Synopsis

Control the PWM Edge Delay circuit.

Prototype

```
bool R_GPT_EdgeDelay_Control(
      uint8_t data1,
      uint8_t data2,
      R_GPT_EdgeDelay_Times_structure *          // A pointer to a structure
);
```

R_GPT_EdgeDelay_Times_structure members:

```
      uint8_t  GTIOCA_Rising_Delay      // GTIOCA Rising Edge Delay Time
      uint8_t  GTIOCA_Falling_Delay     // GTIOCA Falling Edge Delay Time
      uint8_t  GTIOCB_Rising_Delay      // GTIOCB Rising Edge Delay Time
      uint8_t  GTIOCB_Falling_Delay     // GTIOCB Falling Edge Delay Time
```

Description

Control the PWM Edge Delay circuit including setting / adjusting the delay times.

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

[data2]
Control option. To set multiple options at the same time, use “|” to separate each value.

- Time adjustment.

PDL_GPT_PWM_DELAY_TIME	Update the delay times with those specified in [data3] to [data6].

- Activate (this is not required for normal use but provides control of GTDLYCR.DLYEN).

PDL_GPT_PWM_DELAY_ACTIVE or	Activate the delay circuit.
PDL_GPT_PWM_DELAY_INACTIVE	Inactivate (bypass) the delay circuit.

- Reset (This is not required for normal use but provides control of GTDLYCR.DLYRST)

PDL_GPT_PWM_DELAY_RESET	Perform a reset by setting the DLYRST bit high then low.

[GTIOCA_Rising_Delay]
The delay to be applied to the rising edge of GTIOCA.

The delay is specified in fractions of the ICLK period:
0 = no delay.
1 = 1/32 of ICLK.
2 = 2/32 of ICLK.
…
30 = 30/32 of ICLK
31 = 31/32 of ICLK.

[GTIOCA_Falling_Delay]
The delay to be applied to the falling edge of GTIOCA (See [data3] for format).

[GTIOCB_Rising_Delay]
The delay to be applied to the rising edge of GTIOCB (See [data3] for format).

[GTIOCB_Falling_Delay]
The delay to be applied to the falling edge of GTIOCB (See [data3] for format).

Return value

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

Category

General PWM Timer unit

Reference

None.

Remarks

- Call R_GPT_EdgeDelay_Create to enable a channel before using this function.
- If a delay time adjustment is made the actual adjustment will take place on overflows (in up-counting) or underflows (in down-counting) for saw waves, and in the troughs of triangle waves.
- Adjustments should not be made to the delay time when the relevant compare match value falls in the range in the following table.

Mode	Counting Direction	Compare Match Value
Saw	Counting up	"GTPR-2" or more
	Counting down	"2" or less
Triangle	Counting down	"2" or less

The GTPR register is loaded from the "Cycle setting register value" set in data11 of R_GPT_ControlChannel.

Program example

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

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

R_GPT_EdgeDelay_Times_structure Delay2;

void func(void)
{
    /* Adjust channel 2 delay times */
    Delay2.GTIOCA_Rising_Delay = 1;
    Delay2.GTIOCA_Falling_Delay = 2;
    Delay2.GTIOCB_Rising_Delay = 3;
    Delay2.GTIOCB_Falling_Delay = 4;

    R_GPT_EdgeDelay_Control(
         2,
         PDL_GPT_PWM_DELAY_TIME,
         &Delay2
    );
}
```

11) R_GPT_EdgeDelay_Destroy

Synopsis

Disable the Edge Delay circuit.

Prototype

```
bool R_GPT_EdgeDelay_Destroy(
    uint8_t data    // Unit selection
);
```

Description

Disable the edge delay circuit on all channels.

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

Return value

True.

Category

General PWM Timer unit

Reference

None.

Remarks

Program example

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

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

void func(void)
{
    /* Disable the Edge Delay circuits */
    R_GPT_EdgeDelay_Destroy(
        0
    );
}
```

4.2.16. 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.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use in §6.
- A callback function is executed by the interrupt processing function. This means that no other interrupt can be processed until the callback function has completed.
- Ensure that the timer channel is stopped before calling this function.
- The timing limits depend on the frequency of the peripheral module clock, PCLKB.

	Equation	f_{PCLKB} (MHz)					
		50	48	12.5	12	32	8
$Period_{MIN}$	$\frac{8}{f_{PCLKB}}$	160ns	167ns	640ns	667ns	250ns	1.0µs
$Period_{MAX}$	$\frac{2^{25}}{f_{PCLKB}}$	671ms	699ms	2.68s	2.79s	1.05s	4.19s
f_{MAX}	$\frac{f_{PCLKB}}{8}$	6.25 MHz	6.0 MHz	1.56 MHz	1.5 MHz	4.0 MHz	1.0 MHz
f_{MIN}	$\frac{f_{PCLKB}}{2^{25}}$	1.49 Hz	1.43 Hz	0.37 Hz	0.357 Hz	0.95 Hz	0.24 Hz

- If the requested period is not a multiple of the minimum period, the actual time period will be more than the requested time period.

Program example

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

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

void func(void)
{
    /* Configure CMT channel 0 for 10µs operation */
    R_CMT_Create(
         0,
         PDL_CMT_PERIOD,
         10E-6,
         PDL_NO_FUNC,
         0
    );

    /* Configure CMT channel 1 for 1kHz operation */
    R_CMT_Create(
         1,
         PDL_CMT_FREQUENCY,
         1E3,
         PDL_NO_FUNC,
         0
    );

    /* Configure CMT channel 2 using register values */
    R_CMT_Create(
         2,
         PDL_CMT_PCLK_DIV_32,
         0x55AA,
         PDL_NO_FUNC,
         0
    );
}
```

2) R_CMT_CreateOneShot

Synopsis

Configure a CMT channel as a one-shot event.

Prototype

```
bool R_CMT_CreateOneShot(
    uint8_t data1,      // Timer channel selection
    uint16_t data2,     // Configuration selection
    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

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)					
		50	48	12.5	12	32	8
T_{MIN}	$\frac{8}{f_{PCLK}}$	160ns	166.67ns	640ns	666.67ns	250ns	1µs
T_{MAX}	$\frac{2^{25}}{f_{PCLK}}$	671ms	699ms	2.68s	2.79s	1.05s	4.19s

- If the requested period is not a multiple of the minimum period, the actual time period will be more than the requested time period.

Program example

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

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

void func(void)
{
    /* Use CMT channel 0 for a 1ms pause */
    R_CMT_CreateOneShot(
         0,
         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
    );
}
```

4.2.17. Watchdog Timer

1) R_WDT_Set

Synopsis

Configure the Watchdog timer.

Prototype

```
bool R_WDT_Set(
    uint32_t data       // Configuration selection
);
```

Description

Set up and start the Watchdog timer.

[data]
Configure the timer. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Time-out selection

PDL_WDT_TIMEOUT_1024 or PDL_WDT_TIMEOUT_4096 or PDL_WDT_TIMEOUT_8192 or **PDL_WDT_TIMEOUT_16384**	Time out period specified in cycles of the divided clock as specified in the Clock Selection below.

- Clock selection

PDL_WDT_PCLK_DIV_4 or PDL_WDT_PCLK_DIV_64 or **PDL_WDT_PCLK_DIV_128** or PDL_WDT_PCLK_DIV_512 or PDL_WDT_PCLK_DIV_2048 or PDL_WDT_PCLK_DIV_8192	The division ratio for the internal clock signal PCLKB.

- MCU reset control

PDL_WDT_TIMEOUT_RESET or PDL_WDT_TIMEOUT_NMI	When the WDT times out, select if either a Reset or an NMI interrupt will be generated.

- Window Start Position

PDL_WDT_WIN_START_25 or PDL_WDT_WIN_START_50 or PDL_WDT_WIN_START_75 or **PDL_WDT_WIN_START_100**	The window start position specified as a percent of the down counter. 0% is when the downcounter would underflow. Selecting 100% is equivalent to no window start position.

- Window End Position

PDL_WDT_WIN_END_0 or PDL_WDT_WIN_END_25 or PDL_WDT_WIN_END_50 or PDL_WDT_WIN_END_75	The window end position specified as a percent of the down counter. 0% is when the downcounter would underflow. Hence specifying 0% is equivalent to no window end position.

Return value

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

Category

Watchdog Timer

Reference

R_INTC_CreateExtInterrupt, R_MCU_OFS

Remarks

- If using the Initial Setting Memory (using R_MCU_OFS) to enable the WDT from reset this
- function will have no effect.
- If configuring to use a NMI handler then R_INTC_CreateExtInterrupt must be used to enable the NMI for WDT.
- The timing limits depend on the frequency of the peripheral module clock, PCLKB.

$$Period = \frac{n \times cycles}{f_{PCLKB}} \text{ or } Frequency = \frac{f_{PCLKB}}{n \times cycles}$$

Where:
n = 4, 64, 128, 512, 2048, or 8192.
cycles = 1024, 4096, 8192, 16384.
Example periods are given below for f_{PCLKB} = 50MHz.

	Time out cycles			
	1024	**4096**	**8192**	**16384**
Period $_{PCLK÷4}$	81.9 µs	328 µs	735 µs	1.31 ms
Period $_{PCLK÷64}$	1.31 ms	5.24 ms	11.8 ms	21.0 ms
Period $_{PCLK÷128}$	2.62 ms	10.5 ms	23.5 ms	41.9 ms
Period $_{PCLK÷512}$	10.5 ms	41.9 ms	94.1 ms	168 ms
Period $_{PCLK÷2048}$	41.9 ms	168 ms	377 ms	671 ms
Period $_{PCLK÷8192}$	168 ms	671 ms	1.51s	2.68s

Program example

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

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

void func(void)
{
    /* Configure the watchdog timer for PCLKB/4,
       Timeout cycles = 4096, no windowing and reset operation.*/
    R_WDT_Set(
         PDL_WDT_PCLK_DIV_4 | PDL_WDT_TIMEOUT_4096 | \
         PDL_WDT_TIMEOUT_RESET
    );

    /* Configure the watchdog timer for PCLKB/128,
       Timeout cycles = 8192, windowing (50% to 25%) and reset
       operation.*/
    R_WDT_Set(
         PDL_WDT_PCLK_DIV_128 | PDL_WDT_TIMEOUT_8192 | \
         PDL_WDT_TIMEOUT_RESET | PDL_WDT_WIN_START_50 | \
         PDL_WDT_WIN_END_25
    );
}
```

2) R_WDT_Control

Synopsis

Control the Watchdog operation.

Prototype

```
bool R_WDT_Control(
    uint8_t data    // Control selection
);
```

Description

Modify the operation of the Watchdog timer.

[data]
Control the Watchdog timer.

- Counter update

PDL_WDT_RESET_COUNTER	Refresh the counter.

Return value

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

Category

Watchdog Timer

Reference

R_WDT_Set

Remarks

- R_WDT_Set must be called first to configure the timer unless using Initial Setting Memory (using R_MCU_OFS) to enable the WDT from reset.

Program example

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

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

void func(void)
{
    /* Prevent the watchdog timer from overflowing */
    R_WDT_Control(
         PDL_WDT_RESET_COUNTER
    );
}
```

3) R_WDT_Read

Synopsis

Read the Watchdog timer status.

Prototype

```
bool R_WDT_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 Flag	Underflow Flag	Down Counter Value
1: Refresh error 0: No refresh error	1: Underflow 0: No underflow	

Return value

True.

Category

Watchdog Timer

Reference

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_wdt.h"

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

uint16_t WDT_Status;

void func(void)
{
    /* Read the timer values */
    R_WDT_Read(
         &WDT_Status
    );
}
```

4.2.18. 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 specifiying 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, deep 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 PCLKB / 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	
0: No refresh error 1: Refresh error	0: No underflow 1: Underflow	Down Counter Value

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
    );
}
```

4.2.19. 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
    uint32_t data2      // I/O configuration for channels
);
```

Description (1/4)

Configure I/O pins for all SCI channels. There is no default option.

[data1]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[data2]
Configure the I/O pins (required only if the pins are used for the SCI function). Use "|" to separate each selection.

- Valid when n = 0

PDL_SCI_PIN_SCI0_RXD0_PB1 or PDL_SCI_PIN_SCI0_RXD0_P24 or PDL_SCI_PIN_SCI0_RXD0_P22 or PDL_SCI_PIN_SCI0_RXD0_PA5	SCI0	RXD0
PDL_SCI_PIN_SCI0_SMISO0_PB1 or PDL_SCI_PIN_SCI0_SMISO0_P24 or PDL_SCI_PIN_SCI0_SMISO0_P22 or PDL_SCI_PIN_SCI0_SMISO0_PA5		SMISO0
PDL_SCI_PIN_SCI0_SSCL0_PB1 or PDL_SCI_PIN_SCI0_SSCL0_P24 or PDL_SCI_PIN_SCI0_SSCL0_P22 or PDL_SCI_PIN_SCI0_SSCL0_PA5		SSCL0
PDL_SCI_PIN_SCI0_TXD0_PB2 or PDL_SCI_PIN_SCI0_TXD0_P30 or PDL_SCI_PIN_SCI0_TXD0_P23 or PDL_SCI_PIN_SCI0_TXD0_PA4		TXD0
PDL_SCI_PIN_SCI0_SMOSI0_PB2 or PDL_SCI_PIN_SCI0_SMOSI0_P30 or PDL_SCI_PIN_SCI0_SMOSI0_P23 or PDL_SCI_PIN_SCI0_SMOSI0_PA4		SMOSI0
PDL_SCI_PIN_SCI0_SSDA0_PB2 or PDL_SCI_PIN_SCI0_SSDA0_P30 or PDL_SCI_PIN_SCI0_SSDA0_P23 or PDL_SCI_PIN_SCI0_SSDA0_PA4		SSDA0
PDL_SCI_PIN_SCI0_SCK0_PB3 or PDL_SCI_PIN_SCI0_SCK0_P23 or PDL_SCI_PIN_SCI0_SCK0_P30 or PDL_SCI_PIN_SCI0_SCK0_PA3		SCK0
PDL_SCI_PIN_SCI0_CTS0_PD7 or PDL_SCI_PIN_SCI0_CTS0_P22 or PDL_SCI_PIN_SCI0_CTS0_P00 or PDL_SCI_PIN_SCI0_CTS0_P01 or PDL_SCI_PIN_SCI0_CTS0_P24		CTS0
PDL_SCI_PIN_SCI0_RTS0_PD7 or PDL_SCI_PIN_SCI0_RTS0_P22 or PDL_SCI_PIN_SCI0_RTS0_P00 or PDL_SCI_PIN_SCI0_RTS0_P01 or PDL_SCI_PIN_SCI0_RTS0_P24		RTS0
PDL_SCI_PIN_SCI0_SS0_PD7 or PDL_SCI_PIN_SCI0_SS0_P22 or PDL_SCI_PIN_SCI0_SS0_P00 or PDL_SCI_PIN_SCI0_SS0_P01 or PDL_SCI_PIN_SCI0_SS0_P24		SS0

Description (2/4)

- Valid when n = 1

PDL_SCI_PIN_SCI1_RXD1_PD5 or PDL_SCI_PIN_SCI1_RXD1_P93 or PDL_SCI_PIN_SCI1_RXD1_P96 or PDL_SCI_PIN_SCI1_RXD1_PF2 or PDL_SCI_PIN_SCI1_RXD1_PF4 or	SCI1	RXD1
PDL_SCI_PIN_SCI1_SMISO1_PD5 or PDL_SCI_PIN_SCI1_SMISO1_P93 or PDL_SCI_PIN_SCI1_SMISO1_P96 or PDL_SCI_PIN_SCI1_SMISO1_PF2		SMISO1
PDL_SCI_PIN_SCI1_SSCL1_PD5 or PDL_SCI_PIN_SCI1_SSCL1_P93 or PDL_SCI_PIN_SCI1_SMISO1_P96 or PDL_SCI_PIN_SCI1_SMISO1_PF2		SSCL1
PDL_SCI_PIN_SCI1_TXD1_PD3 or PDL_SCI_PIN_SCI1_TXD1_P94 or PDL_SCI_PIN_SCI1_TXD1_PF3 or PDL_SCI_PIN_SCI1_TXD1_P95 or PDL_SCI_PIN_SCI1_TXD1_P26		TXD1
PDL_SCI_PIN_SCI1_SMOSI1_PD3 or PDL_SCI_PIN_SCI1_SMOSI1_P94 or PDL_SCI_PIN_SCI1_SMOSI1_PF3 or PDL_SCI_PIN_SCI1_SMOSI1_P95 or PDL_SCI_PIN_SCI1_SMOSI1_P26		SMOSI1
PDL_SCI_PIN_SCI1_SSDA1_PD3 or PDL_SCI_PIN_SCI1_SSDA1_P94 or PDL_SCI_PIN_SCI1_SSDA1_PF3 or PDL_SCI_PIN_SCI1_SSDA1_P95 or PDL_SCI_PIN_SCI1_SSDA1_P26		SSDA1
PDL_SCI_PIN_SCI1_SCK1_PD4 or PDL_SCI_PIN_SCI1_SCK1_P92 or PDL_SCI_PIN_SCI1_SCK1_PG6 or PDL_SCI_PIN_SCI1_SCK1_P25		SCK1
PDL_SCI_PIN_SCI1_CTS1_P70 or PDL_SCI_PIN_SCI1_CTS1_P91 or PDL_SCI_PIN_SCI1_CTS1_P94		CTS1
PDL_SCI_PIN_SCI1_RTS1_P70 or PDL_SCI_PIN_SCI1_RTS1_P91 or PDL_SCI_PIN_SCI1_RTS1_P94		RTS1
PDL_SCI_PIN_SCI1_SS1_P70 or PDL_SCI_PIN_SCI1_SS1_P91 or PDL_SCI_PIN_SCI1_SS1_P94		SS1

Description (3/4)

- Valid when n = 2

PDL_SCI_PIN_SCI2_RXD2_P03 or PDL_SCI_PIN_SCI2_RXD2_PG1 or PDL_SCI_PIN_SCI2_RXD2_PA2	SCI2	RXD2
PDL_SCI_PIN_SCI2_SMISO2_P03 or PDL_SCI_PIN_SCI2_SMISO2_PG1 or PDL_SCI_PIN_SCI2_SMISO2_PA2		SMISO2
PDL_SCI_PIN_SCI2_SSCL2_P03 or PDL_SCI_PIN_SCI2_SSCL2_PG1 or PDL_SCI_PIN_SCI2_SSCL2_PA2		SSCL2
PDL_SCI_PIN_SCI2_TXD2_P02 or PDL_SCI_PIN_SCI2_TXD2_PG0 or PDL_SCI_PIN_SCI2_TXD2_PA1		TXD2
PDL_SCI_PIN_SCI2_SMOSI2_P02 or PDL_SCI_PIN_SCI2_SMOSI2_PG0 or PDL_SCI_PIN_SCI2_SMOSI2_PA1		SMOSI2
PDL_SCI_PIN_SCI2_SSDA2_P02 or PDL_SCI_PIN_SCI2_SSDA2_PG0 or PDL_SCI_PIN_SCI2_SSDA2_PA1		SSDA2
PDL_SCI_PIN_SCI2_SCK2_P14 or PDL_SCI_PIN_SCI2_SCK2_PG2 or PDL_SCI_PIN_SCI2_SCK2_PA0		SCK2
PDL_SCI_PIN_SCI2_CTS2_P13 or PDL_SCI_PIN_SCI2_CTS2_P93		CTS2
PDL_SCI_PIN_SCI2_RTS2_P13 or PDL_SCI_PIN_SCI2_RTS2_P93 or		RTS2
PDL_SCI_PIN_SCI2_SS2_P13 or PDL_SCI_PIN_SCI2_SS2_P93		SS2

- Valid when n = 3

PDL_SCI_PIN_SCI3_RXD3_P34 or PDL_SCI_PIN_SCI3_RXD3_PG4	SCI3	RXD3
PDL_SCI_PIN_SCI3_SMISO3_P34 or PDL_SCI_PIN_SCI3_SMISO3_PG4		SMISO3
PDL_SCI_PIN_SCI3_SSCL3_P34 or PDL_SCI_PIN_SCI3_SSCL3_PG4		SSCL3
PDL_SCI_PIN_SCI3_TXD3_P35 or PDL_SCI_PIN_SCI3_TXD3_PG3		TXD3
PDL_SCI_PIN_SCI3_SMOSI3_P35 or PDL_SCI_PIN_SCI3_SMOSI3_PG3		SMOSI3
PDL_SCI_PIN_SCI3_SSDA3_P35 or PDL_SCI_PIN_SCI3_SSDA3_PG3		SSDA3
PDL_SCI_PIN_SCI3_SCK3_PG5		SCK3
PDL_SCI_PIN_SCI3_CTS3_PA6		CTS3
PDL_SCI_PIN_SCI3_RTS3_PA6		RTS3
PDL_SCI_PIN_SCI3_SS3_PA6		SS3

Description (4/4)

- Valid when n = 12

PDL_SCI_PIN_SCI12_RXD12_PB6 PDL_SCI_PIN_SCI12_RXD12_P80	SCI12	RXD12
PDL_SCI_PIN_SCI12_SMISO12_PB6 PDL_SCI_PIN_SCI12_SMISO12_P80		SMISO12
PDL_SCI_PIN_SCI12_SSCL12_PB6 PDL_SCI_PIN_SCI12_SSCL12_P80		SSCL12
PDL_SCI_PIN_SCI12_TXD12_PB5 PDL_SCI_PIN_SCI12_TXD12_P81		TXD12
PDL_SCI_PIN_SCI12_SMOSI12_PB5		SMOSI12
PDL_SCI_PIN_SCI12_SSDA12_PB5 PDL_SCI_PIN_SCI12_SSDA12_P81		SSDA12
PDL_SCI_PIN_SCI12_SCK12_PB7 PDL_SCI_PIN_SCI12_SCK12_P82		SCK12
PDL_SCI_PIN_SCI12_CTS12_PB4 PDL_SCI_PIN_SCI12_CTS12_PE1		CTS12
PDL_SCI_PIN_SCI12_RTS12_PB4 PDL_SCI_PIN_SCI12_RTS12_PE1		RTS12
PDL_SCI_PIN_SCI12_SS12_PB4 PDL_SCI_PIN_SCI12_SS12_PE1		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, call this function to configure the relevant pins.
- Please refer to the “Multifunction Pin Controller (MPC)” section in the RX63T Hardware Manual for details of SCI pin selection.
- 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.
- Not all device packages have all of the pin options.

Program example

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

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

void func(void)
{
    /* Configure RXD1 and TXD1 pins*/
    R_SCI_Set(
        1,
        PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_TXD1_PD3
    );
}
```

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
    uint8_t data5      // Interrupt priority level
);
```

Description (1/4)

Set up the selected SCI channel.

[data1]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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 can not 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	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_MTU3	The base clock to be input from MTU3 must be set to a frequency no greater than 1/4 that of PCLK. Not valid in 64 and 48 pin packages.	

- 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.

- 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 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 of output on the SDA Pin relative to the falling edge on the 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 for data transmission. 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.

[data5]
The interrupt priority level for receive error detection. Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for error callback function in R_SCI_Receive or R_SCI_SPI_Transfer.
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

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.
- This function configures each SCI pin that is required for operation. It also disables the alternative modes on those pins.
- 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 can not 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.
- The range of achievable bit rates (bps) is listed below.
- MTU3 clock can not be used on some device pin packages

Mode	Data clock source	Limit	f_{PCLKB}				
			50 MHz	32 MHz	12.5 MHz	12 MHz	8 MHz
Asynchronous	Internal	Minimum	96	62	24	23	16
		Maximum	3,125,000	2,000,000	781,250	750,000	500,000
	External		1,562,500	1,000,000	390,625	375,000	250,000
Synchronous	Internal	Minimum	763	489	191	184	123
		Maximum	6,250,000	4,000,000	1,562,500	1,500,000	1,000,000
	External		8,333,333	5,333,333	2,083,333	2,000,000	1,333,333
Smart card	Internal	Minimum	3	2	1	1	1
		Maximum	781,250	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 SCI0 for asynchronous, 8N1, 38400 baud */
    R_SCI_Create(
        0,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1,
        0
    );

    /* 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,
        0
    );
}
```

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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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 the usage example in Section 5.17.7.
- 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_t data1,      // 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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 the continuous receive when 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, R_SCI_Create, R_SCI_Send
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 the usage example in Section 5.17.6. • 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 Section 5.17.5. • 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 func 1 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

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

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

volatile uint8_t SCI1RxBuffer [9];

/* 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 0 */
    R_SCI_Receive(
        0,
        PDL_NO_DATA,
        &temp,
        1,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );

    /* Start the reception of 9 characters on channel 1 */
    R_SCI_Receive(
        1,
        PDL_NO_DATA,
        SCI1RxBuffer,
        9,
        SCI1RxFunc,
        SCI1ErrFunc
    );
}
```

6) R_SCI_SPI_Transfer

Synopsis

Perform an SPI transfer on an SCI channel.

Prototype

```
bool R_SCI_SPI_Transfer(
    uint8_t data1,      // 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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

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

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

volatile uint8_t SCI1RxBuffer[9];

/* 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 0 */
    R_SCI_SPI_Transfer (
        0,
        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_t data1,      // 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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

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

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

#define CHANNEL_SCI_IIC 1
#define SLAVE_ADDRESS   0xA0

/* Buffer for IIC data */
volatile 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_t data1,      // 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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

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

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

#define CHANNEL_SCI_IIC 1
#define SLAVE_ADDRESS   0xA0

/* Buffer for IIC data */
volatile 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[data2]
The address of the buffer that will receive the byte.

Return value

True.

Category

SCI

Reference

R_SCI_IIC_Read

Remarks

Program example

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

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

#define CHANNEL_SCI_IIC 1

/* Buffer for IIC data */
volatile 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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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

Remarks

Program example

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

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

void func(void)
{
    /* Terminate SCI reception on channel 0 */
    R_SCI_Control(
        0,
        PDL_SCI_STOP_RX
    );
}
```

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]
The channel number n (where n = 0, 1 or 12 for 64 and 48 pin packages; n = 0, 1, 2, 3 or 12 for 144, 120, 112 and 100 pin packages).

[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.

Program example

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

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

uint8_t StatusValue;
uint16_t TxChars;
uint16_t RxChars;

void func(void)
{
    /* Read the status of SCI channel 0 */
    R_SCI_GetStatus(
        0,
        &StatusValue,
        PDL_NO_PTR,
        &TxChars,
        &RxChars
    );
}
```

4.2.20. I²C Bus Interface

1) 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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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 or	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

Remarks

- Function R_CGC_Set must be called (with the current clock source selected) before using 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	-

- Channel 1 is supported on 120-pin, 144-pin packages only.
- 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.

When the digital noise filter circuit is enabled, the ICBRL, ICBRH register value should be equal or greater than <the number of noise filter steps + 1>.

The absolute limits (with zero rise and fall times) are:

f_{IRC}	f_{PCLKB} (MHz)					
	50	**48**	**12.5**	**12**	**32**	**8**
$f_{PCLKB} \div 1$	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
$f_{PCLKB} \div 2$	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
$f_{PCLKB} \div 4$	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
$f_{PCLKB} \div 8$	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
$f_{PCLKB} \div 16$	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
$f_{PCLKB} \div 32$	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
$f_{PCLKB} \div 64$	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
$f_{PCLKB} \div 128$	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:

IRC	f_{PCLKB} (MHz)					
	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 I²C mode at 100kHz, 100ns rise and fall times */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC | PDL_IIC_INT_PCLK_DIV_8,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        100E3,
        (100 << 16) | 100
    );

    /* Select I²C mode with two slave addresses */
    R_IIC_Create(
        0,
        PDL_IIC_MODE_IIC,
        PDL_IIC_SLAVE_0_ENABLE_7 | PDL_IIC_SLAVE_1_ENABLE_7,
        0x0020,
        0x0056,
        PDL_NO_DATA,
        100E3,
        (300 << 16) | 200
    );
}
```

2) R_IIC_Destroy

Synopsis

Disable an I²C channel.

Prototype

```
bool R_IIC_Destroy(
    uint8_t data    // Channel selection
);
```

Description

Shut down the selected I²C channel.

[data]
Select channel IICn (where n = 0 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

Return value

True if the parameter is valid; otherwise false.

Category

I²C

Reference

R_IIC_Create

Remarks

- The I²C module is put into the power-down state.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

void func(void)
{
    /* Shutdown IIC channel 0 */
    R_IIC_Destroy(
        0
    );
}
```

3) R_IIC_MasterSend

Synopsis

Write data to a slave device.

Prototype

```
bool R_IIC_MasterSend(
    uint8_t data1,      // Channel selection
    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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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.

- 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.

- 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 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.

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_Create, R_IIC_GetStatus

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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

const uint8_t data_array[5] = {0x23, 0x48, 0x59, 0x60, 0xFE};

void func(void)
{
    /* Send 5 bytes to device 0x0A0 on channel 0, using polling */
    R_IIC_MasterSend(
        0,
        PDL_NO_DATA,
        0x0A0,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}
```

4) R_IIC_MasterReceive

Synopsis

Read data from a slave device.

Prototype

```
bool R_IIC_MasterReceive(
    uint8_t data1,      // Channel selection
    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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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_Create, R_IIC_GetStatus, R_IIC_MasterReceiveLast

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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

volatile uint8_t data_array[5];

void func(void)
{
    /* Read 5 bytes from device 0xAA on channel 0, using polling */
    R_IIC_MasterReceive(
        0,
        PDL_NO_DATA,
        0xAA,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}
```

5) R_IIC_MasterReceiveLast

Synopsis

Complete a 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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

volatile uint8_t data_array[5];

void func(void)
{
    /* Read 1 byte on channel 0 and stop */
    R_IIC_MasterReceiveLast(
        0,
        &data_array[4]
    );
}
```

6) R_IIC_SlaveMonitor

Synopsis

Monitor the bus.

Prototype

```
bool R_IIC_SlaveMonitor(
    uint8_t data1,      // Channel selection
    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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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_Create, R_IIC_GetStatus, R_IIC_SlaveSend

Remarks

- If a callback function is specified, interrupts are used.
  Use R_IIC_GetStatus in the callback function to identify the activity that has occurred.
  Please see the notes on callback function usage in §6.
- If 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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

volatile uint8_t data_array[5];

void func(void)
{
    /* Monitor channel 0, using polling */
    R_IIC_SlaveMonitor(
        0,
        PDL_NO_DATA,
        data_array,
        5,
        PDL_NO_FUNC,
        0
    );
}
```

7) R_IIC_SlaveSend

Synopsis

Write data to a master device.

Prototype

```
bool R_IIC_SlaveSend(
    uint8_t data1,      // Channel selection
    uint8_t * data2,    // Data start address
    uint16_t data3      // Data count
);
```

Description

Transmit data on the specified channel.

[data1]
Select channel IICn (where n = 0 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

const uint8_t data_array[5] = {0x23, 0x48, 0x59, 0x60, 0xFE};

void func(void)
{
    /* Assign 5 bytes to be read by a master on channel 0 */
    R_IIC_SlaveSend(
        0,
        data_array,
        5
    );
}
```

8) R_IIC_Control

Synopsis

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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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

R_IIC_Create

Remarks

- Channel 1 is supported on 120-pin, 144-pin packages only.

Program example

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

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

void func(void)
{
    /* Issue a Stop condition on channel 0 */
    R_IIC_Control(
        0,
        PDL_IIC_STOP
    );
}
```

9) R_IIC_GetStatus

Synopsis

Read the status for an 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 or n = 1).
Channel 1 is not available for device packages with 112 pins or less.

[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
	Buffer status	
0	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_IIC_Create

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 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.
- Channel 1 is supported on 120-pin, 144-pin packages only.

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
    );
}
```

4.2.21. Serial Peripheral Interface

1) R_SPI_Set

Synopsis

Configure the SPI pin selection.

Prototype

```
bool R_SPI_Set(
    uint8_t data1,     // Channel selection
    uint32_t data2,    // Channel 0 pin selection
    uint32_t data3     // Channel 1 pin selection
);
```

Description (1/2)

Set up the global SPI options.

[data1]
Select channel SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[data2]
Pin configuration for the channel 0. Use “|” to separate each selection. Specify PDL_NO_DATA if not required.

- Pin selection for channel 0

PDL_SPI_RSPCKA_P24 or PDL_SPI_RSPCKA_PA4 or PDL_SPI_RSPCKA_PD0	Select the RSPCKA pin.
PDL_SPI_MOSIA_P23 or PDL_SPI_MOSIA_PB0 or PDL_SPI_MOSIA_PD2	Select the MOSIA pin.
PDL_SPI_MISOA_P22 or PDL_SPI_MISOA_PA5 or PDL_SPI_MISOA_PD1	Select the MISOA pin.
PDL_SPI_SSLA0_P30 or PDL_SPI_SSLA0_PA3 or PDL_SPI_SSLA0_PD6	Select the SSLA0 pin (optional).
PDL_SPI_SSLA1_P31 or PDL_SPI_SSLA1_PA2 or PDL_SPI_SSLA1_PD7	Select the SSLA1 pin (optional).
PDL_SPI_SSLA2_P32 or PDL_SPI_SSLA2_PE0 or PDL_SPI_SSLA2_PA1	Select the SSLA2 pin (optional).
PDL_SPI_SSLA3_P33 or PDL_SPI_SSLA3_PE1 or PDL_SPI_SSLA3_PA0	Select the SSLA3 pin (optional).

Description (2/2)

[data3]
Pin configuration for the channel 1. Use “|” to separate each selection. Specify PDL_NO_DATA if not required.
Valid on packages with 100 pins or more.

- Pin selection for channel 1

PDL_SPI_RSPCKB_P24 or PDL_SPI_RSPCKB_PA4 or PDL_SPI_RSPCKB_PD0	Select the RSPCKB pin.
PDL_SPI_MOSIB_P23 or PDL_SPI_MOSIB_PB0 or PDL_SPI_MOSIB_PD2	Select the MOSIB pin.
PDL_SPI_MISOB_P22 or PDL_SPI_MISOB_PA5 or PDL_SPI_MISOB_PD1	Select the MISOB pin.
PDL_SPI_SSLB0_P30 or PDL_SPI_SSLB0_PA3 or PDL_SPI_SSLB0_PD6	Select the SSLB0 pin (optional).
PDL_SPI_SSLB1_P31 or PDL_SPI_SSLB1_PA2 or PDL_SPI_SSLB1_PD7	Select the SSLB1 pin (optional).
PDL_SPI_SSLB2_P32 or PDL_SPI_SSLB2_PE0 or PDL_SPI_SSLB2_PA1	Select the SSLB2 pin (optional).
PDL_SPI_SSLB3_P33 or PDL_SPI_SSLB3_PE1 or PDL_SPI_SSLB3_PA0	Select the SSLB3 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.
- Please refer to the “Multifunction Pin Controller (MPC)” section in the RX63T Hardware Manual for details of SPI pin selection.
- Pins which are not used for the SPI functions may be omitted.
- Same pin cannot be used for different pin functions.
- Not all device packages have all of the pin options.

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(
        0,
        PDL_SPI_RSPCKA_P24 | PDL_SPI_MOSIA_P23 | PDL_SPI_MISOA_P22 | \
        PDL_SPI_SSLA0_P30 | PDL_SPI_SSLA1_P31 | \
        PDL_SPI_SSLA2_P32 | PDL_SPI_SSLA3_P33,
        PDL_NO_DATA
    );
}
```

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 SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[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 32.4 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.
- R_IO_PORT_Set can be used to select between CMOS and Open-drain output.
- 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, 1 only).
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

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 SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[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 * func1,       // Callback function
    uint8_t data6,      // Interrupt priority level
    void * func2,       // Callback function
    uint8_t data7       // Interrupt priority level
);
```

Description (1/2)

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 SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[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.
If the DMAC or DTC shall be used to handle the transfer, 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. R_SPI_Transfer will handle the data transfer until completion.
Interrupts	The function to be called when the transfer has completed.
DMAC or DTC	The function to be called when the DMAC or DTC passes on the transfer interrupt.

[data6]
The interrupt priority level for data transmission. Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for parameter func1.

Description (2/2)

[func2]
The function to be called if an error occurs. Specify PDL_NO_FUNC to ignore errors.

[data7]
The interrupt priority level for error detection. Select between 1 (lowest priority) and 15 (highest priority).
This parameter will be ignored if PDL_NO_FUNC is specified for parameter func2.
Use the same error interrupt priority level as R_SCI_Create parameter data5.

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.
- If an error interrupt function is specified for parameter func 2 while PDL_NO_FUNC is specified for parameter func 1, the error flag is polled without using an interrupt, and the error interrupt function will be called when an error occurs.
- 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,
        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 SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[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 SPI.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[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
    );
}
```

4.2.22. 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
    );
}
```

4.2.23. 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(
    uint16_t data    // ADC pin 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
- Please refer to Table 21.1 at the “Multifunction Pin Controller (MPC)” section in the RX63T Hardware Manual for details of pin-package.

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_AN100_P44	Select P44 for AN100.
PDL_ADC_12_PIN_AN101_P45	Select P45 for AN101.
PDL_ADC_12_PIN_AN102_P46	Select P46 for AN102.
PDL_ADC_12_PIN_AN103_P47	Select P47 for AN103.
PDL_ADC_12_PIN_CVREFL_P43	Select P43 for CVREFL.
PDL_ADC_12_PIN_CVREFH_P47	Select P47 for CVREFH.
PDL_ADC_12_PIN_ADTRG0_P20	Select P20 for ADTRG0.
PDL_ADC_12_PIN_ADTRG0_PA4	Select PA4 for ADTRG0.
PDL_ADC_12_PIN_ADTRG1_P21	Select P21 for ADTRG1.
PDL_ADC_12_PIN_ADTRG1_PA5	Select PA5 for ADTRG1.

Return value

False if an invalid pin selection is made, otherwise True.

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.
- Not all device packages have all of the pin options. Do not specify an option that does not exist for the device package being used.

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,     // Additional options for Group A
    uint32_t data5,     // Options for Group B
    uint32_t data6,     // Additional options for Group B
    uint32_t data7,     // Options for Camparator
    double data8,       // Sampling time for self-diagnosis
    double data9,       // Sampling and hold time
    void * func1,       // Callback function for Group A
    uint8_t data10,     // Interrupt priority level for Group A
    void * func2,       // Callback function for Group B
    uint8_t data11      // Interrupt priority level for Group B
);
```

Description (1/6)

Set the ADC mode and operating condition.

[data1]
Select the ADC unit to be configured.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[data2]
Conversion options. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- 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 source 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 synchronous or asynchronous trigger source.

- ADC value addition selection

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	No addition Addition once Addition twice 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).

- Data accuracy

PDL_ADC_12_PRECISION_12 or PDL_ADC_12_PRECISION_10 or PDL_ADC_12_PRECISION_8	Set the date accuracy of the 12-bit ADC conversion result to 12-bit, 10-bit, or 8-bit.

- Analog pin discharging

PDL_ADC_12_DISCHARGE_DISABLE or PDL_ADC_12_DISCHARGE_ENABLE	Disable or enable analog pin discharging on completion of A/D conversion.

Description (2/6)

- 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.

- Scan priority control for Group A (valid on group scan mode)

PDL_ADC_12_GPA_PRIORITY_DISABLE or PDL_ADC_12_GPA_PRIORITY_ENABLE	Disable or enable Group A priority control.

- Sampling time

PDL_ADC_12_ADSSTR_CALCULATE or PDL_ADC_12_ADSSTR_SPECIFY	Select whether parameter data7 is used to calculate the ADSSTR0 value, or contains the value to be stored in register ADSSTR0.

- Sampling time calculation for sample-and-hold circuit

PDL_ADC_12_ADSHCR_CALCULATE or PDL_ADC_12_ADSHCR_SPECIFY	Select whether parameter data6 is used to calculate the ADSHCR value, or contains the value to be stored in register ADSHCR.

- 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.

Description (3/6)

[data3] / [data5]

Options for two ADC groups in group scan mode. In other operating modes, only data3 is valid, applying to all the working ADC channels. To set multiple options at the same time, use "|" to separate each value.

- MTU trigger source selection
  Valid only if PDL_ADC_12_SYNC_TRIGGER_ENABLE is selected.

PDL_ADC_12_GP_TRIGGER_MTU3_TRGA0N or	Input capture or compare match with MTU0.TGRA
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA1N or	nput capture or compare match with MTU1.TGRA
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA2N or	Input capture or compare match with MTU2.TGRA
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA3N or	Input capture or compare match with MTU3.TGRA
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA4N or	Input capture or compare match with MTU4.TGRA or, in complementary PWM mode, an underflow of MTU4.TCNT (in the trough)
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA6N or	Input capture or compare match with MTU6.TGRA
PDL_ADC_12_GP_TRIGGER_MTU3_TRGA7N or	Input capture or compare match with MTU7.TGRA, or in complementary PWM mode, an underflow of MTU7.TCNT (in the trough)
PDL_ADC_12_GP_TRIGGER_MTU3_TRG0N or	Compare match with MTU0.TGRE
PDL_ADC_12_GP_TRIGGER_MTU3_TRG4AN or	Compare match between MTU4.TADCORA and MTU4.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG4BN or	Compare match between MTU4.TADCORB and MTU4.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG4AN_4BN or	Compare match between MTU4.TADCORA and MTU4.TCNT or between MTU4.TADCORB and MTU4.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG4ABN or	Compare match between MTU4.TADCORA and MTU4.TCNT or between MTU4.TADCORB and MTU4.TCNT (when interrupt skipping function 2 is in use)
PDL_ADC_12_GP_TRIGGER_MTU3_TRG7AN or	Compare match between MTU7.TADCORA and MTU7.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG7BN or	Compare match between MTU7.TADCORB and MTU7.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG7AN_7BN or	Compare match between MTU7.TADCORA and MTU7.TCNT or between MTU7.TADCORB and MTU7.TCNT
PDL_ADC_12_GP_TRIGGER_MTU3_TRG7ABN	Compare match between MTU7.TADCORA and MTU7.TCNT and between MTU7.TADCORB and MTU7.TCNT (when interrupt skipping function 2 is in use)

Description (4/6)

- 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.

[data4] / [data6]

Additional options for two ADC groups in group scan mode. In other operating modes, only data4 is valid, applying to all the working ADC channels.

- GPT trigger source selection
  Valid only if PDL_ADC_12_SYNC_TRIGGER_ENABLE is selected and no MTU trigger source is specified in data3 / data5, otherwise the following options are ignored.

- Trigger source for the GPT0 - GPT3

PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA0N or	Compare match with GPT0.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB0N or	Compare match with GPT0.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA1N or	Compare match with GPT1.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB1N or	Compare match with GPT1.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA2N or	Compare match with GPT2.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB2N or	Compare match with GPT2.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA3N or	Compare match with GPT3.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB3N or	Compare match with GPT3.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA0N_B0N or	Compare match with GPT0.GTADTRA or with GPT0.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA1N_B1N or	Compare match with GPT1.GTADTRA or with GPT1.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA2N_B2N or	Compare match with GPT2.GTADTRA or with GPT2.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA3N_B3N or	Compare match with GPT3.GTADTRA or with GPT3.GTADTRB

Description (5/6)

- Trigger source selection for the GPT4 - GPT7. valid on packages with 100 pins or more

PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA4N or	Compare match with GPT4.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB4N or	Compare match with GPT4.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA5N or	Compare match with GPT5.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB5N or	Compare match with GPT5.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA6N or	Compare match with GPT6.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB6N or	Compare match with GPT6.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA7N or	Compare match with GPT7.GTADTRA
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRB7N or	Compare match with GPT7.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA4N_B4N or	Compare match with GPT4.GTADTRA or with GPT4.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA5N_B5N or	Compare match with GPT5.GTADTRA or with GPT5.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA6N_B6N or	Compare match with GPT6.GTADTRA or with GPT6.GTADTRB
PDL_ADC_12_GP_TRIGGER_GPT_GTADTRA7N_B7N	Compare match with GPT7.GTADTRA or with GPT7.GTADTRB

[data7]
Camparator control selection. If multiple selections are required, use "|" to separate each selection. Specify PDL_NO_DATA if not required.

- Comparator internal REFL selection

PDL_ADC_12_CMP_VSELL0_AN003 or PDL_ADC_12_CMP_VSELL0_INTERNAL	Set the voltage on AN003 or the selected internal voltage as the REFL for comparator

- Comparator REFH selection (valid on device packages with 48 or 64 pin)

PDL_ADC_12_CMP_VSELH0_AN007 or PDL_ADC_12_CMP_VSELH0_INTERNAL	Set the voltage on AN007 or the selected internal voltage as the REFL for comparator.

- Comparator REFH selection (valid on device packages with 100, 112, 120 and 144 pin)

PDL_ADC_12_CMP_VSELH0_AN103 or PDL_ADC_12_CMP_VSELH0_INTERNAL	Set the voltage on AN103 or the selected internal voltage as the REFL for comparator.

- Comparator input selection (valid on device packages with 100, 112, 120 and 144 pin)

PDL_ADC_12_CMP_ANX0X_BEFORE_AMPLIFIER or PDL_ADC_12_CMP_ANX0X_AFTER_AMPLIFIER	Before or after amplified by the programmable gain amplifier.

Description (6/6)

- Comparator internal REFL selection

PDL_ADC_12_CMP_REFL_DISABLE or PDL_ADC_12_CMP_REFL_AVCC0_1_8 or PDL_ADC_12_CMP_REFL_AVCC0_2_8 or PDL_ADC_12_CMP_REFL_AVCC0_3_8 or PDL_ADC_12_CMP_REFL_AVCC0_4_8 or PDL_ADC_12_CMP_REFL_AVCC0_5_8 or PDL_ADC_12_CMP_REFL_AVCC0_6_8 or PDL_ADC_12_CMP_REFL_AVCC0_7_8	Disable the internal REFL for comparator or set it as AVCC0 * 1/8, 2/8, 3/8, 4/8, 5/8, 6/8 or 7/8.

- Comparator internal REFH selection

PDL_ADC_12_CMP_REFH_DISABLE or PDL_ADC_12_CMP_REFH_AVCC0_1_8 or PDL_ADC_12_CMP_REFH_AVCC0_2_8 or PDL_ADC_12_CMP_REFH_AVCC0_3_8 or PDL_ADC_12_CMP_REFH_AVCC0_4_8 or PDL_ADC_12_CMP_REFH_AVCC0_5_8 or PDL_ADC_12_CMP_REFH_AVCC0_6_8 or PDL_ADC_12_CMP_REFH_AVCC0_7_8	Disable the internal REFH for comparator or set it as AVCC0 x 1/8, 2/8, 3/8, 4/8, 5/8, 6/8 or 7/8.

[data8]
The data to be used for the sampling state register value calculations for self-diagnosis. If PDL_ADC_12_ADSSTR_SPECIFY is selected for parameter data2, the value should not be less than 13 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

[data9]
The data to be used for the sample and hold circuit control register value calculations. If PDL_ADC_12_ADSHCR_SPECIFY is selected for data2, the value should not be less than 4 or more then 255.

Data use	Parameter type
The timer period in seconds or	double
The value to be put in register ADSHCR	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.

[data10]
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.

[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 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 PCLK 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, the two ADC groups should not share the same trigger source. For more details of the MTU or GPT trigger options, please refer to the RX63T hardware manual.
- This function will return false if an invalid / unachievable sampling time is specified.
- 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"

/* 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_NO_DATA,
          PDL_NO_DATA,
          PDL_NO_DATA,
          PDL_NO_DATA,
          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
    uint32_t data3,     // Channel configuration
    uint16_t data4,     // Comparator configuration
    double data5,       // Sampling time
    void * func,        // Callback function for comparator on the channel
    uint8_t data6       // Interrupt priority level for comparator on the channel
);
```

Description (1/2)

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.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[data2]
Select the analog input channel.
This must be from 0 to 7 for device packages with 48 or 64 pin, otherwise 0 to 3.

[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

- Sample and hold circuit control

PDL_ADC_12_CH_SAMPLE_AND_HOLD_DISABLE or PDL_ADC_12_CH_SAMPLE_AND_HOLD_ENABLE	Enable or disable sample and hold circuit. For channels 0, 1 and 2 only.

- 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.

- Programmale Gain Amplifer configuration. Valid for Channel 0, 1 and 2 only
  Valid for packages with 100 pins or more/

PDL_ADC_12_CH_GAIN_DISABLE or PDL_ADC_12_CH_GAIN_2_000 or PDL_ADC_12_CH_GAIN_2_500 or PDL_ADC_12_CH_GAIN_3_077 or PDL_ADC_12_CH_GAIN_3_636 or PDL_ADC_12_CH_GAIN_4_000 or PDL_ADC_12_CH_GAIN_4_444 or PDL_ADC_12_CH_GAIN_5_000 or PDL_ADC_12_CH_GAIN_5_714 or PDL_ADC_12_CH_GAIN_6_667 or PDL_ADC_12_CH_GAIN_10_000 or PDL_ADC_12_CH_GAIN_13_333	Specify the amplifer gain required.

Description (2/2)

[data4]
Comparator options. Valid for Channel 0, 1 and 2 only. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Comparator mode

PDL_ADC_12_CMP_OFF or PDL_ADC_12_CMP_LOW or PDL_ADC_12_CMP_HIGH or PDL_ADC_12_CMP_WINDOW	Set the comparator on the channel to be inactive or working in low, high or window mode.

- Sampling frequency for Comparator

PDL_ADC_12_CMP_SAMPLE_DISABLE or PDL_ADC_12_CMP_SAMPLE_PCLK or PDL_ADC_12_CMP_SAMPLE_PCLK_1_2 or PDL_ADC_12_CMP_SAMPLE_PCLK_1_4 or PDL_ADC_12_CMP_SAMPLE_PCLK_1_8 or PDL_ADC_12_CMP_SAMPLE_PCLK_1_16 or PDL_ADC_12_CMP_SAMPLE_PCLK_1_128	Diable sampling of comparator detection result, or set the sampling frequency to be PCLK , PCLK / 2, PCLK /4, PCLK / 8, PCLK / 16, or PCLK / 128.

- POE trigger by Comparator

PDL_ADC_12_CMP_POE_DISABLE or PDL_ADC_12_CMP_POE_ENABLE	Disable or enable activation of POE by Comparator.

- DTC/DMAC trigger by Comparator

PDL_ADC_12_CMP_DMAC_DTC_TRIGGER_DISABLE or PDL_ADC_12_CMP_DMAC_TRIGGER_ENABLE or PDL_ADC_12_CMP_DTC_TRIGGER_ENABLE	Disable or enable activation of DMAC or DTC.

[data5]
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 13 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

[func]
The function to be called when Comparator detects an event.
Specify PDL_NO_FUNC if no callback function is required.

[data6]
The interrupt priority level for Comparator. 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 a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit, R_CGC_Set

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.
- Group A and group B cannot use the same channels.
- Channels with PDL_ADC_12_CH_SAMPLE_AND_HOLD_ENABLE and with programmable gain amplifiers are not selectable for group B.
- 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,
         PDL_NO_DATA,
         5E-6,
         PDL_NO_FUNC,
         PDL_NO_DATA
    );
}
```

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 including the Comparator into the Power-down state, with minimal power consumption.

[data]
Select the ADC unit to be shut down.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

Return value

True if a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit

Remarks

- This function includes a 1 ms 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 0 unit */
    R_ADC_12_Destroy(
         0
    );
}
```

5) R_ADC_12_Control

Synopsis

Start or stop an ADC unit.

Prototype

```
bool R_ADC_12_Control(
    uint32_t data    // Conversion unit control
);
```

Description (1/2)

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 for Comparator on AN000 (Unit 0)

PDL_ADC_12_CMP_000_ON or	Start the comparator on AN000
PDL_ADC_12_CMP_000_OFF	Stop the comparator on AN000

- On / off control for Comparator on AN001 (Unit 0)

PDL_ADC_12_CMP_001_ON or	Start the comparator on AN001
PDL_ADC_12_CMP_001_OFF	Stop the comparator on AN001

- On / off control for Comparator on AN002 (Unit 0)

PDL_ADC_12_CMP_002_ON or	Start the comparator on AN002
PDL_ADC_12_CMP_002_OFF	Stop the comparator on AN002

- On / off control for Comparator on AN100 (Unit 1)

PDL_ADC_12_CMP_100_ON or	Start the comparator on AN100
PDL_ADC_12_CMP_100_OFF	Stop the comparator on AN100

- On / off control for Comparator on AN101 (Unit 1)

PDL_ADC_12_CMP_101_ON or	Start the comparator on AN101
PDL_ADC_12_CMP_101_OFF	Stop the comparator on AN101

- On / off control for Comparator on AN102 (Unit 1)

PDL_ADC_12_CMP_102_ON or	Start the comparator on AN102
PDL_ADC_12_CMP_102_OFF	Stop the comparator on AN102

- On / off control for ADC unit 0

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).

- On / off control for ADC unit 1

PDL_ADC_12_1_ON or	Start a software-triggered conversion or re-enable the trigger.
PDL_ADC_12_1_OFF	Stop the conversion (and disable all triggers).

- Scan priority control for Group B for ADC unit 0

PDL_ADC_12_0_GPB_RESTART_CONTINUOUS_OFF or	disable restarted and scan continuous in Single-cycle for Group B
PDL_ADC_12_0_GPB_RESTART_ON or	Set Group B to be restarted after getting discontinued due to Group A priority control
PDL_ADC_12_0_GPB_CONTINUOUS_ON	Set Group B to be continuously activated for single-cycle scan

Description (2/2)

- Scan priority control for Group B for ADC unit 1

PDL_ADC_12_1_GPB_RESTART_CONTINUOUS_OFF or	disable restarted and scan continuous in Single-cycle for Group B
PDL_ADC_12_1_GPB_RESTART_ON or	Set Group B to be restarted after getting discontinued due to Group A priority control
PDL_ADC_12_1_GPB_CONTINUOUS_ON	Set Group B to be continuously activated for single-cycle scan

- 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

R_ADC_12_CreateUnit and R_ADC_12_CreateChannel

Remarks

- For single scan mode, the ADC will stop automatically when the conversion is complete.
- Set Group B to be restarted after getting discontinued due to Group A priority control must be work on the group scan with option PDL_ADC_12_GPA_PRIORITY_ENABLE.
- Single-cycle scans through group B are continuously activated must be work on the group scan with option PDL_ADC_12_GPA_PRIORITY_ENABLE.
- The Unit 1 is valid on device packages with 100 pins or more.
- 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 additional results are to be stored
    uint8_t * data4     // Pointer to the address where the comparator status is to be stored
);
```

Description

Reads the conversion values for an ADC unit.

[data1]
Select the ADC unit to be configured.
For device packages with 48 or 64 pin this must be 0; for other packages 0 or 1.

[data2]
Specify a pointer to an array where the converted values for analog input channels are to be stored. The array length must >= the number of channels: 8 for device packages with 48 or 64 pin, otherwise 3.
Specify PDL_NO_PTR if this information is not required.

[data3]
Specify a pointer to an array with 1 member or 3 members (in double trigger mode only), where the diagnostic result or double trigger results are to be stored, depending on the ADC mode.
Specify PDL_NO_PTR if this information is not required.
Refer to hardware manual Section 34.2.2 for the format of self-diagnosis result.

[data4]
Specify a pointer to the address where the comparator status is to be stored in the format below.
Specify PDL_NO_PTR if this information is not required.

For ADC unit 0

b7 – b3	b2	b1	b0
0	Specified condition detection (0 = Not detected, 1= Detected)		
	AN002	AN001	AN000

For ADC unit 1. Valid on device packages with 100 pins or more.

b7 – b3	b2	b1	b0
0	Specified condition detection (0 = Not detected, 1= Detected)		
	AN102	AN101	AN100

Return value

True if a valid unit is selected; otherwise false.

Category

12-bit ADC

Reference

R_ADC_12_CreateUnit and R_ADC_12_CreateChannel

Remarks

- From 1 to 8 valid conversion results will be read and stored to the array specified by data2. The number depends on the parameters supplied to R_ADC_12_CreateUnit and R_ADC_12_CreateChannel for configuration.
- The values stored to the array specified by data3 depend on the ADC mode. In self-diagnostic mode, the diagnostic result is stored into the first place. In double trigger mode, the double trigger result is store into the first place. In extended double trigger mode, the result from synchronous trigger A is stored into the second place, and that from synchronous trigger B is stored into the third place.
- The data alignment is controlled using the R_ADC_12_CreateUnit.
- If no callback function is used, this function waits for the IR flag to indicate that conversion is complete before reading the results. If the ADC unit's control registers are directly modified by the user, this function may lock up.

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[8];

    /* Read the ADC */
    R_ADC_12_Read(
        0,
        ADCresult,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}
```

4.2.24. 10-bit Analog to Digital Converter

1) R_ADC_10_Set

Synopsis

Select the I/O pins for the 10-bit ADC.

Prototype

```
bool R_ADC_10_Set(
    uint16_t data    // ADC pin selection
);
```

Description

Select the I/O pins for the10-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_10_PIN_AN0_P60	Select P60 for AN0.
PDL_ADC_10_PIN_AN1_P61	Select P61 for AN1.
PDL_ADC_10_PIN_AN2_P62	Select P62 for AN2.
PDL_ADC_10_PIN_AN3_P63	Select P63 for AN3.
PDL_ADC_10_PIN_AN4_P64	Select P64 for AN4.
PDL_ADC_10_PIN_AN5_P65	Select P65 for AN5.
PDL_ADC_10_PIN_AN6_P50	Select P50 for AN6.
PDL_ADC_10_PIN_AN7_P51	Select P51 for AN7.
PDL_ADC_10_PIN_AN8_P52	Select P52 for AN8.
PDL_ADC_10_PIN_AN9_P53	Select P53 for AN9.
PDL_ADC_10_PIN_AN10_P54	Select P54 for AN10.
PDL_ADC_10_PIN_AN11_P55	Select P55 for AN11.
PDL_ADC_10_PIN_AN12_P56	Select P56 for AN12.
PDL_ADC_10_PIN_AN13_P57	Select P57 for AN13.
PDL_ADC_10_PIN_AN14_PC0	Select PC0 for AN14.
PDL_ADC_10_PIN_AN15_PC1	Select PC1 for AN15.
PDL_ADC_10_PIN_AN16_PC2	Select PC2 for AN16.
PDL_ADC_10_PIN_AN17_PC3	Select PC3 for AN17.
PDL_ADC_10_PIN_AN18_PC4	Select PC4 for AN18.
PDL_ADC_10_PIN_AN19_PC5	Select PC5 for AN19.
PDL_ADC_10_PIN_ADTRG_P22 or PDL_ADC_10_PIN_ADTRG_PG5	Select P22 or PG5 for ADTRG#.

Return value

False if an invalid pin selection is made, otherwise True.

Category

10-bit ADC

Reference

R_ADC_10_CreateUnit

Remarks

- If there are I/O pins to be used, call this function before calling R_ADC_10_CreateUnit.
- Not all device packages have all of the pin options. Do not specify an option that does not exist for the device package being used.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

void func( void )
{
    /* Set analog channel AN0 */
    R_ADC_10_Set(
         PDL_ADC_10_PIN_AN0_P60
    );
}
```

2) R_ADC_10_CreateUnit

Synopsis

Configure the 10-bit ADC unit.

Prototype

```
bool R_ADC_10_CreateUnit(
    uint8_t data1,      // Unit selection
    uint32_t data2,     // Unit specific options
    uint32_t data3,     // Options for trigger
    double data4,       // Sampling time for self-diagnosis
    void * func,        // Callback function
    uint8_t data5,      // Interrupt priority level
);
```

Description (1/3)

Set the ADC mode and operating condition.

[data1]
Select the ADC unit to be configured.
For all device packages this must 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**.

- Scan mode

PDL_ADC_10_SCAN_SINGLE or PDL_ADC_10_SCAN_CONTINUOUS	Select Single scan, Continuous scan mode.

- ADC value addition count selection

PDL_ADC_10_VALUE_ADD_TIME_1 or PDL_ADC_10_VALUE_ADD_TIME_2 or PDL_ADC_10_VALUE_ADD_TIME_3 or PDL_ADC_10_VALUE_ADD_TIME_4	No addition Addition once Addition twice Addition three times

- Data alignment

PDL_ADC_10_DATA_ALIGNMENT_RIGHT or PDL_ADC_10_DATA_ALIGNMENT_LEFT	The alignment of the 10-bit ADC conversion result within the 16-bit register. Ignored for channels using value addition mode (the 12-bit result is always left-aligned).

- Data-Register Bit-Precision

PDL_ADC_10_DATA_ACCURACY_10_BIT or PDL_ADC_10_DATA_ACCURACY_8_BIT	Set the data accuracy of the 10-bit ADC conversion result to 10-bit or 8-bit.

- Automatic clearing control

PDL_ADC_10_RETAIN_RESULT or PDL_ADC_10_CLEAR_RESULT	Disable or enable automatic clearing of ADDRy and ADRD after the register has been read.

- Self-diagnostic control

PDL_ADC_10_DIAG_DISABLE or PDL_ADC_10_DIAG_VREFH0_ZERO or PDL_ADC_10_DIAG_VREFH0_HALF or PDL_ADC_10_DIAG_VREFH0_FULL or PDL_ADC_10_DIAG_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.

- Sampling time

PDL_ADC_10_ADSSTR_CALCULATE_UNIT or PDL_ADC_10_ADSSTR_SPECIFY_UNIT	Select whether parameter data4 is used to calculate the ADSSTR0 value, or contains the value to be stored in register ADSSTR0.

Description (2/3)

- DMAC or DTC trigger control

PDL_ADC_10_DMAC_DTC_TRIGGER_DISABLE or PDL_ADC_10_DTC_TRIGGER_ENABLE or PDL_ADC_10_DMAC_TRIGGER_ENABLE	Enable or disable activation of the DMAC or DTC.

[data3]
Condition to start ADC conversion.

PDL_ADC_10_TRIGGER_SOFTWARE or	Software trigger
PDL_ADC_10_TRIGGER_ADTRG or	A/D conversion start trigger pin
PDL_ADC_10_TRIGGER_MTU0_CMIC or	TRGA compare match/input capture from MTU0
PDL_ADC_10_TRIGGER_MTU1_CMIC or	TRGA compare match/input capture from MTU1
PDL_ADC_10_TRIGGER_MTU2_CMIC or	TRGA compare match/input capture from MTU2
PDL_ADC_10_TRIGGER_MTU3_CMIC or	TRGA compare match/input capture from MTU3
PDL_ADC_10_TRIGGER_MTU4_CMIC or	MTU4.TRGA compare match/input capture or MTU4.TCNT underflow (trough) in complementary PWM mode
PDL_ADC_10_TRIGGER_MTU6_CMIC or	TRGA compare match/input capture from MTU6
PDL_ADC_10_TRIGGER_MTU7_CMIC or	MTU7.TRGA compare match/input capture or MTU7.TCNT underflow (trough) in complementary PWM mode
PDL_ADC_10_TRIGGER_MTU0_CM_E or	TRGE compare match from MTU0
PDL_ADC_10_TRIGGER_MTU4_CM_A or	MTU4.TADCORA and MTU4.TCNT compare match
PDL_ADC_10_TRIGGER_MTU4_CM_B or	MTU4.TADCORB and MTU4.TCNT compare match
PDL_ADC_10_TRIGGER_MTU4_CM_AB or	MTU4.TADCORA and MTU4.TCNT compare match or MTU4.TADCORB and MTU4.TCNT compare match
PDL_ADC_10_TRIGGER_MTU4_CM_AB_IS or	MTU4.TADCORA and MTU4.TCNT compare match and MTU4.TADCORB and MTU4.TCNT compare match (interrupt skipping function 2)
PDL_ADC_10_TRIGGER_MTU7_CM_A or	MTU7.TADCORA and MTU7.TCNT compare match
PDL_ADC_10_TRIGGER_MTU7_CM_B or	MTU7.TADCORB and MTU7.TCNT compare match
PDL_ADC_10_TRIGGER_MTU7_CM_AB or	MTU7.TADCORA and MTU7.TCNT compare match or MTU7.TADCORB and MTU7.TCNT compare match
PDL_ADC_10_TRIGGER_MTU7_CM_AB_IS or	MTU7.TADCORA and MTU7.TCNT compare match and MTU7.TADCORB and MTU7.TCNT compare match (interrupt skipping function 2)
PDL_ADC_10_TRIGGER_GPT0_CM_A or	GPT0.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT0_CM_B or	GPT0.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT1_CM_A or	GPT1.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT1_CM_B or	GPT1.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT2_CM_A or	GPT2.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT2_CM_B or	GPT2.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT3_CM_A or	GPT3.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT3_CM_B or	GPT3.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT0_CM_AB or	GPT0.GTADTRA compare match or GPT0.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT1_CM_AB or	GPT1.GTADTRA compare match or GPT1.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT2_CM_AB or	GPT2.GTADTRA compare match or GPT2.GTADTRB compare match

PDL_ADC_10_TRIGGER_GPT3_CM_AB or	GPT3.GTADTRA compare match or GPT3.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT4_CM_A or	GTADTRA4 GPT4.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT4_CM_B or	GTADTRB4 GPT4.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT5_CM_A or	GTADTRA5 GPT5.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT5_CM_B or	GTADTRB5 GPT5.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT6_CM_A or	GTADTRA6 GPT6.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT6_CM_B or	GTADTRB6 GPT6.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT7_CM_A or	GTADTRA7 GPT7.GTADTRA compare match
PDL_ADC_10_TRIGGER_GPT7_CM_B or	GTADTRB7 GPT7.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT4_CM_AB or	GPT4.GTADTRA compare match or GPT4.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT5_CM_AB or	GPT4.GTADTRA compare match or GPT4.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT6_CM_AB or	GPT6.GTADTRA compare match or GPT6.GTADTRB compare match
PDL_ADC_10_TRIGGER_GPT7_CM_AB	GPT7.GTADTRA compare match or GPT7.GTADTRB compare match

[data4]
The data to be used for the sampling state register value calculations for self-diagnosis. If PDL_ADC_10_ADSSTR_SPECIFY_UNIT is selected for parameter data2, the value should not be less than 7 or more then 255.

Data use	Parameter type
The timer period in seconds or	double
The value to be put in register ADSSTR0	uint8_t

[func]
The function to be called when the ADC conversion scan cycle is complete in single scan mode and continuous scan mode.
Specify PDL_NO_FUNC if no callback function is 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 parameter func1.

Return value

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

Category

10-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.
- 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.
- Pin PG5 does not exist in 100 pin device package.
  Function R_CGC_Set must be called to set the frequency.
- The frequency division ratio between PCLK and ADCLK should be one of the following:
  1:1, 1:2, 1:4, 1:8, 2:1, 4:1
- Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

/* ADC callback function */
void ADC_callback(void);

void func(void)
{
    /* Set up the ADC in single mode, using software trigger */
    R_ADC_10_CreateUnit(
          0,
          PDL_ADC_10_SCAN_SINGLE,
          PDL_ADC_10_TRIGGER_SOFTWARE,
          PDL_NO_DATA,
          ADC_callback,
          2
    );
}
```

3) R_ADC_10_CreateChannel

Synopsis

Configure 10-bit ADC analog channels.

Prototype

```
bool R_ADC_10_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_10_CreateUnit to configure 10-bit ADC analog channels, if analog channels are selected as the input source.

[data1]
Select the ADC unit.
This must be 0 for All device packages.

[data2]
Select the analog input channel.
This must be 0 to 19 for 144-pin package, and 0 to 12 for others.

[data3]
Channel options. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Value addition control

PDL_ADC_10_CH_VALUE_ADDITION_DISABLE or PDL_ADC_10_CH_VALUE_ADDITION_ENABLE	Enable or disable value addition

- Sampling time calculation

PDL_ADC_10_ADSSTR_CALCULATE or PDL_ADC_10_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_10_ADSSTR_SPECIFY is selected for data3, specify the value to be written to the sampling state register. The value should not be less than 7 or more then 255.
If PDL_ADC_10_ADSSTR_CALCULATE is selected for data3, specify the required sampling time in seconds. This value must be more than 0.25 µs for channel 0 to 7, and more than 0.5 µs for others.

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

10-bit ADC

Reference

R_ADC_10_CreateUnit

Remarks

- If analog channels are used as the input sources, call this function after calling R_ADC_10_CreateUnit.
- Function R_CGC_Set must be called to set the frequency.
- The frequency division ratio between PCLK and ADCLK should be one of the following: 1:1, 1:2, 1:4, 1:8, 2:1, 4:1.
- Channel 8 to channel 19 shares the same sampling state register, so please use same sampling time for channel 8 onwards.
- Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

void func( void )
{
    /* Configure AN0 */
    R_ADC_10_CreateChannel(
         0,
         0,
         PDL_ADC_10_CH_VALUE_ADDITION_DISABLE,
         6E-6
    );
}
```

4) R_ADC_10_Destroy

Synopsis

Shut down the ADC unit.

Prototype

```
bool R_ADC_10_Destroy(
    uint8_t data    // ADC unit selection
);
```

Description

Put the ADC into the Power-down state, with minimal power consumption.

[data]
Select the ADC unit to be shut down.
This must be 0 for all device packages.

Return value

True if a valid unit is selected; otherwise false.

Category

10-bit ADC

Reference

Remarks

- This function includes a 1 ms delay to allow the ADC to stop any current scan cycle.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

void func( void )
{
    /* Shut down the ADC 0 unit */
    R_ADC_10_Destroy(
         0
    );
}
```

5) R_ADC_10_Control

Synopsis

Start or stop an ADC unit.

Prototype

```
bool R_ADC_10_Control(
    uint8_t data      // Conversion unit control
);
```

Description

Start / stop operation of the specified ADC.

[data]
To select multiple options at the same time, use "|" to separate each value.

- On / off control for unit 0

PDL_ADC_10_0_ON or	Start a software-triggered conversion.
PDL_ADC_10_0_OFF	Stop the undergoing conversion.

- Control the CPU during the ADC conversion.

PDL_ADC_10_CPU_OFF	Stop the CPU when the conversion process starts. The CPU will re-start when any valid interrupt occurs.

Return value

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

Category

10-bit ADC

Reference

R_ADC_10_CreateUnit, R_ADC_10_CreateChannel

Remarks

- For single scan mode, the ADC will stop automatically when the conversion is complete.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.
- Do not select CPU Off unless there is any interrupt to wake up the CPU.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

void func(void)
{
    /* Start the ADC conversion process */
    R_ADC_10_Control(
         PDL_ADC_10_0_ON
    );
}
```

6) R_ADC_10_Read

Synopsis

Read the ADC conversion results.

Prototype

```
bool R_ADC_10_Read(
    uint8_t data1,      // ADC unit selection
    uint16_t * data2,   // Pointer to the address where the results are to be stored
    uint16_t * data3,   // Pointer to the buffer where the self-diagnostic result is to be stored
);
```

Description

Reads the conversion values for an ADC unit.

[data1]
Select the ADC unit.
This must be 0 for all device packages.

[data2]
Specify a pointer to an array where the converted values for analog input channels are to be stored. The array length must >= the number of channels: 20 for device packages with 144 pin, otherwise 12.
Specify PDL_NO_PTR if this information is not required.

[data3]
Specify a pointer to a variable where the self-diagnostic result shall be stored.
Specify PDL_NO_PTR if this information is not required.
Refer to hardware manual Section 36.2.2 for the format of self-diagnosis result.

Return value

True if a valid unit is selected; otherwise false.

Category

10-bit ADC

Reference

R_ADC_10_CreateUnit, R_ADC_10_CreateChannel

Remarks

- Depends on the parameters supplied to R_ADC_10_CreateUnit and R_ADC_10_CreateChannel for configuration.
- If analog channels are selected as the input source, valid array pointer should be supplied to data2. If self-diagnostic is enabled, the respective result will be stored to data3. Ensure that the storage area is big enough for the requested number of results.
- The format of values stored to the array is specified using R_ADC_10_CreateUnit.
- If no callback function is used, this function waits for the IR flag to indicate that conversion is complete before reading the results.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_adc_10.h"

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

void func(void)
{
    uint16_t adc_results[20];
    uint16_t diag_result;

    /* Read the ADC */
    R_ADC_10_Read(
        0,
        adc_results,
        &diag_result
    );
}
```

4.2.25. 10-bit Digital to Analog Converter

1) R_DAC_10_Create

Synopsis

Configure the 10-bit DAC module.

Prototype

```
bool R_DAC_10_Create(
    uint8_t data1,     // Configuration
    uint16_t data2,    // Output value
    uint16_t data3     // Output value
);
```

Description

Enable the DAC module and set the operating conditions.

[data1]
Configuration options. To set multiple options at the same time, use "|" to separate each value. The default settings are shown in **bold**.

- Channel enable

PDL_DAC_10_CHANNEL_0	Enable channel 0
PDL_DAC_10_CHANNEL_1	Enable channel 1

- Data alignment selection

PDL_DAC_10_ALIGN_LEFT or **PDL_DAC_10_ALIGN_RIGHT**	The alignment of the 10-bit output data within the 16-bit parameters data2 and data3. Left: padded at the MSB end. Right: padded at the LSB end.

- D/A A/D Synchronous Start Control

PDL_DAC_10_ADC_SYNC_CONV_DISABLE or PDL_DAC_10_ADC_SYNC_CONV_ENABLE	Disable or enable the D/A A/D synchronous conversion.

[data2]
The value to be written to the channel 0 output register. Ignored if the channel is not enabled.

[data3]
The value to be written to the channel 1 output register. Ignored if the channel is not enabled.

Return value

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

Category

DAC

References

None.

Remarks

- This function configures the relevant pin of selected channel for DAC operation.
- This function brings the converter module out of the power-down state.
- If the D/A A/D synchronous conversion is enabled, the 10-bit ADC should not be shut down, as it will halt the D/A conversion too.
- User should ensure that the 10-bit A/D converter remains stopped, when setting the D/A A/D synchronous conversion.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_dac_10.h"

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

void func(void)
{
    /* Set up DAC channel 1 with default operation, mid voltage */
    R_DAC_10_Create(
         PDL_DAC_10_CHANNEL_1,
         0,
         1024 / 2
    );
}
```

2) R_DAC_10_Destroy

Synopsis

Disable a DAC channel.

Prototype

```
bool R_DAC_10_Destroy(
     uint8_t data     // Channel selection
);
```

Description

Disable the channel output.

[data1]
Disable selection. To set multiple options at the same time, use "|" to separate each value.

PDL_DAC_10_CHANNEL_0	Disable channel 0
PDL_DAC_10_CHANNEL_1	Disable channel 1

Return value

True if the parameter is valid; otherwise false.

Category

DAC

Reference

None.

Remarks

- Once both channels are disabled, the module is put into the power-down state.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_dac_10.h"

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

void func( void )
{
    /* Shut down DAC channel 1 */
    R_DAC_10_Destroy(
         PDL_DAC_10_CHANNEL_1
    );
}
```

3) R_DAC_10_Write

Synopsis

Write data to a DAC channel.

Prototype

```
bool R_DAC_10_Write(
    uint8_t data1,     // Channel selection
    uint16_t data2,    // Output value
    uint16_t data3     // Output value
);
```

Description

Write data to the selected DAC channel(s).

[data1]
Select the DAC channel output to be modified.

PDL_DAC_10_CHANNEL_0	Select channel 0
PDL_DAC_10_CHANNEL_1	Select channel 1

[data2]
The value to be written to the channel 0 output register. Ignored if the channel is not selected.

[data3]
The value to be written to the channel 1 output register. Ignored if the channel is not selected.

Return value

True if all parameters are valid; otherwise false.

Category

DAC

Reference

R_DAC_10_Create

Remarks

- Refer to the data alignment that was selected when R_DAC_10_Create was called.
- This function is supported on 100-pin, 112-pin, 120-pin, 144-pin packages only.

Program example

```
/* RPDL definitions */
#include "r_pdl_dac_10.h"

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

void func(void)
{
    /* Write new data to DAC channel 1 */
    R_DAC_10_Write(
         PDL_DAC_10_CHANNEL_1,
         0,
         100
    );
}
```

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	Put the DOC module into the power-down state.
Return value	True
Category	DOC
Reference	
Remarks	
Program example	

```
/* RPDL definitions */
#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 5.2 shows an example of configuring the clock generation circuit.

After a power-on reset, both the PLL and the main clock oscillator (which drives the PLL circuit) are 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 and the PLL circuit.
After an appropriate time to allow for the crystal-based main clock oscillator and the PLL circuit to stabilise, a call to R_CGC_Control is used to select the PLL circuit 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)
{

#if defined (DEVICE_PACKAGE_64_PIN) || defined (DEVICE_PACKAGE_48_PIN)
    /* Set the LOCO clock settings (the clock source used after a power-on reset) */
    /* ICLK = 125 kHz, PCLKA = 125 kHz, PCLKB = 125 kHz */
    /* PCLKD = 125 kHz, FCLK = 125 kHz */
    R_CGC_Set(
        PDL_CGC_CLK_LOCO,
        PDL_NO_DATA,
        125E3,
        125E3,
        125E3,
        125E3,
        PDL_NO_DATA,
        125E3,
        125E3,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Configure main clock operation using a 16.0 MHz crystal */
    /* ICLK = 4 MHz, PCLKA = 4 MHz, PCLKB = 4 MHz */
    /* PCLKD = 4 MHz, FCLK = 4 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,
        16E6,
        4E6,
        4E6,
        4E6,
        PDL_NO_DATA,
        4E6,
        4E6,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

    /* Configure PLL operation.  The PLL will be set to 200 MHz */
    /* ICLK = 100 MHz, PCLKA = 100 MHz, PCLKB = 50 MHz */
    /* PCLKD = 50 MHz, FCLK = 50 MHz */
    R_CGC_Set(
        PDL_CGC_CLK_PLL,
        PDL_NO_DATA,
        200E6,
        100E6,
        100E6,
        50E6,
        PDL_NO_DATA,
        50E6,
        50E6,
```

```
            PDL_NO_DATA,
            PDL_NO_DATA
        );
    #else
        /* Set the LOCO clock settings (the clock source used after a power-on reset) */
        /* ICLK = 125 kHz, PCLKA = 125 kHz, PCLKB = 125 kHz */
        /* PCLKD = 125 kHz, FCLK = 125 kHz */
        R_CGC_Set(
            PDL_CGC_CLK_LOCO,
            PDL_CGC_BCLK_DISABLE,
            125E3,
            125E3,
            125E3,
            125E3,
            125E3,
            125E3,
            125E3,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Configure main clock operation using a 12.0 MHz crystal */
        /* ICLK = 3 MHz, PCLKA = 3 MHz, PCLKB = 3 MHz, PCLKC = 3 MHz */
        /* PCLKD = 3 MHz, FCLK = 3 MHz */
        R_CGC_Set(
            PDL_CGC_CLK_MAIN,
            PDL_CGC_BCLK_DISABLE,
            12E6,
            3E6,
            3E6,
            3E6,
            3E6,
            3E6,
            3E6,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Configure PLL operation.  The PLL will be set to 150 MHz */
        /* ICLK = 75 MHz, PCLKA = 75 MHz, PCLKB = 37.5 MHz, PCLKC = 37.5 MHz */
        /* PCLKD = 37.5E6 MHz, FCLK = 37.5E6 MHz */
        R_CGC_Set(
            PDL_CGC_CLK_PLL,
            PDL_CGC_BCLK_DISABLE,
            150E6,
            75E6,
            75E6,
            37.5E6,
            37.5E6,
            37.5E6,
            37.5E6,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
    #endif

        /* Allow time for the main clock and PLL oscillator to stabilise.  This example */
        /* uses the CMT timer (running from the LOCO) to generate a 100us delay */

        /* Generate the 100us delay */
        R_CMT_CreateOneShot(
            0,
            PDL_NO_DATA,
            100E-6,
            PDL_NO_FUNC,
            0
        );
```

```
        /* Select the PLL as the clock source */
        R_CGC_Control(
            PDL_CGC_CLK_PLL,
            PDL_NO_DATA,
            PDL_NO_DATA
        );
    }
```

Figure 5.1: Example of Clock configuration and control

5.2. Interrupt control

Figure 5.2 shows an example of external interrupt use.

Pin IRQ1-DS is used to detect a falling edge 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 */
void SW1_handler (void);
void SW2_handler (void);
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 */
#if defined (DEVICE_PACKAGE_64_PIN) || defined (DEVICE_PACKAGE_48_PIN)
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ0_P10 | PDL_INTC_IRQ1_P11 | PDL_INTC_IRQ2_P00
    );
#else
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ0_P10 | PDL_INTC_IRQ1_P11 | PDL_INTC_IRQ2_PE3
    );
#endif

    /* Configure the SW1 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ0,
        PDL_INTC_FALLING,
        SW1_handler,
        7
    );

    /* Configure the SW2 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ1,
        PDL_INTC_FILTER_DIV_32 | PDL_INTC_FALLING,
        SW2_handler,
        7
    );

    /* Configure the SW3 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ2,
        PDL_INTC_FALLING,
        SW3_handler,
        7
    );
}

void SW1_handler(void)
{
    volatile uint8_t irq_status = 0u;

    R_INTC_GetExtInterruptStatus(
        PDL_INTC_IRQ0,
```

```
            &irq_status
        );

        /* Falling edge detected? */
        if ((irq_status & 0x0C) == 0x04)
        {
            /* Disable and invert the edge interrupt */
            R_INTC_ControlExtInterrupt(
                PDL_INTC_IRQ0,
                PDL_INTC_RISING | PDL_INTC_DISABLE
            );
        }
        else if ((irq_status & 0x0C) == 0x08)
        {
            /* Disable and invert the edge interrupt */
            R_INTC_ControlExtInterrupt(
                PDL_INTC_IRQ0,
                PDL_INTC_FALLING | PDL_INTC_DISABLE
            );
        }
    }

    void SW2_handler(void)
    {
        /* Handle the interrupt */
    }

    void SW3_handler(void)
    {
        /* Handle the interrupt */
    }
```

Figure 5.2: Example of External Interrupt

5.3. I/O Port

Figure 5.3 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 control_register;

    /* 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,
        PDL_IO_PORT_INPUT
    );

    /* Configure port pin P22 as an open-drain output */
    R_IO_PORT_Set(
        PDL_IO_PORT_2_2,
        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 P22 to output high */
    R_IO_PORT_Write(
        PDL_IO_PORT_2_2,
        1
    );

    /* Invert pin P22 */
    R_IO_PORT_Modify(
        PDL_IO_PORT_2_2,
        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 direction for pin P22 */
    R_IO_PORT_ReadControl(
        PDL_IO_PORT_2_2,
        PDL_IO_PORT_DIRECTION,
        &control_register
    );
```

```
        /* Set the lower 4 bits on port P3 to output */
        R_IO_PORT_ModifyControl(
            PDL_IO_PORT_3,
            PDL_IO_PORT_DIRECTION | PDL_IO_PORT_OR,
            0x0F
        );
    }
```

Figure 5.3: Example of I/O Port Operations

5.4. MCU Operation

This sample sets the Option Setting Memory and then detects if a Cold start has occurred.

```
/* Peripheral driver function prototypes */
#include "r_pdl_mcu.h"
#include "r_pdl_mcu_ofs.h"

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

/* Set the Option Setting Memory:
Enable the IWDT auto-start mode.
Leave the WDT disabled.
Enable reset at Vdet0. */
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_WDT_HALTED,
    PDL_MCU_OFS_LVD_0_ENABLE
);

void main(void)
{
    uint16_t mode_status;
    uint16_t reset_status;
    uint32_t ofs0_copy;
    uint32_t ofs1_copy;

    /* Read the MCU status registers */
    R_MCU_GetStatus(
           &mode_status,
           &reset_status,
           &ofs0_copy,
           &ofs1_copy
    );

    /* Is this a Cold start? */
    if ((reset_status & BIT_8) == 0)
    {
           /* Set the warm start indicator */
    R_MCU_Control(
           PDL_MCU_WARM_START
           );

           /* USER: Handle cold start */
    }
}
```

Figure 5.4: Example of MCU Operation

5.5. Voltage Detection Circuit

Figure 5.5 shows an example of Voltage detection circuit usage.

An NMI is generated if the supply voltage drops below 2.95V.

```
/* 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 2.95V */
    R_LVD_Create(
        PDL_LVD_INTERRUPT_NMI_DETECT_FALL | PDL_LVD_FILTER_DISABLE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA
    );
}

/* NMI Callback function */
static void Callback_NMI(void)
{
    uint8_t status = 0;

    /* 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 5.5: Example of Voltage Detection Circuit use

5.6. Clock Frequency Accuracy Measurement Circuit

The main clock is used as the reference to measure the IWDTLOCO 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_overflow( void );


/* Expected Clock values */
#define FREQ_IWDTLOCO 125E3
#define FREQ_MAIN     12E6


void main(void)
{

    /* Configure the IWDTLOCO clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_IWDTLOCO,
        PDL_NO_DATA,          /* Reserved */
        FREQ_IWDTLOCO,        /* IWDTLOCO*/
        PDL_NO_DATA,          /* ICLK */
        PDL_NO_DATA,          /* PCLKA */
        PDL_NO_DATA,          /* PCLKB */
        PDL_NO_DATA,          /* Reserved */
        PDL_NO_DATA,          /* PCLKD */
        PDL_NO_DATA,          /* FCLK */
        PDL_NO_DATA,          /* Reserved */
        PDL_NO_DATA           /* Reserved */
    );


    /* Configure the main clock settings */
    R_CGC_Set(
        PDL_CGC_CLK_MAIN,
        PDL_NO_DATA,  /* Reserved */
        FREQ_MAIN,    /* Main */
        FREQ_MAIN/4,  /* ICLK */
        FREQ_MAIN/4,  /* PCLKA */
        FREQ_MAIN/4,  /* PCLKB */
        PDL_NO_DATA,  /* Reserved */
        FREQ_MAIN/4,  /* PCLKD */
        FREQ_MAIN/4,  /* FCLK */
        PDL_NO_DATA,  /* Reserved */
        PDL_NO_DATA   /* Reserved */
    );


    /* 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%). */
        /* Register functions that will be called if an error is detected. */
        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% tolerance */
            10,           /* 10% tolerance */
            CAC_frequency_error,
            PDL_NO_FUNC,
            CAC_overflow,
            10
        );

        while(1);
    }

    void CAC_frequency_error(void)
    {
        /* Handle the frequency error */
    }

    void CAC_overflow(void)
    {
        /* Handle the overflow error */
    }
```

Figure 5.6: Example of Clock Frequency measurement use

5.7. Low Power Consumption

5.7.1. Software Standby Mode

Figure 5.7 shows an example of entering Software Standby mode through Low Power Consumption control.

```
/* Peripheral driver function prototypes */
#include "r_pdl_lpc.h"
#include "r_pdl_intc.h"

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

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
    #define PDL_INTC_IRQ2_PIN PDL_INTC_IRQ2_PE3
#else
    #define PDL_INTC_IRQ2_PIN PDL_INTC_IRQ2_P00
#endif

static void SW3_handler(void);

void main(void)
{
    /* Set IRQ2 : interrupt */
    R_INTC_SetExtInterrupt(
        PDL_INTC_IRQ2_PIN
    );

    /* Enable the IRQ2 interrupt */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_IRQ2,
        PDL_INTC_FALLING,
        SW3_handler,
        7
    );

    /* Select the default options */
    R_LPC_Create(
        PDL_NO_DATA,
        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 IRQ2 falling signal */
    while(1);

}

/* IRQ2 : Falling signal */
static void SW3_handler(void)
{
}
```

Figure 5.7: Example of Software Standby Mode

5.7.2. Deep Software Standby Mode

Figure 5.8 shows an example of entering Deep Software Standby mode through Low Power Consumption control.

```
/* PDL functions */
#include "r_pdl_lpc.h"
#include "r_pdl_intc.h"

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

void NMI_handler_lpc(void);

void main(void)
{
    const uint8_t data_to_save[] = "Hello_World_1234567890_abcdefghi";
    uint8_t data_to_restore[R_PDL_LPC_BACKUP_AREA_SIZE];
    uint32_t status_flags;

    /* Read the LPC status */
    R_LPC_GetStatus(&status_flags);

    /* Check if this is an exit from deep software standby (BIT_23 = 1) */
    if( (status_flags & 0x00800000) != 0)
    {
        /* Read data from the backup registers */
        R_LPC_ReadBackup(
            data_to_restore,
            R_PDL_LPC_BACKUP_AREA_SIZE
        );

        /* Have exited deep standby, sample finishes here. */
        while(1);
    }

    /* Configure the NMI pin interrupt*/
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_FALLING,
        NMI_handler_lpc,
        7
    );
    /* Allow a falling edge on NMI to cancel deep software standby */
    R_LPC_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_LPC_CANCEL_NMI_FALLING,
        PDL_NO_DATA,
        PDL_NO_DATA
    );

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

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

    /* An internal reset will occur when exiting from deep software standby */
    /* The program counter will not return to here */
    while(1);
}

void NMI_handler_lpc(void)
{
    nop();
}
```

Figure 5.8: Example of Deep Software Standby Mode

5.8. Bus Controller

5.8.1. Bus controller for the 64-pin and 48-pin package.

Figure 5.9 shows an example of bus controller usage

```
/* Peripheral driver function prototypes */
#include "r_pdl_bsc.h"
#include "r_pdl_cgc.h"
#include "r_pdl_intc.h"

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

/* Callback function prototype */
void BSC_error_handler(void);

void main(void)
{
    /* Set the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(
        PDL_INTC_REG_IPL,
        0
    );

    /* Configure the bus controller */
    R_BSC_Create(
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE,
        BSC_error_handler,
        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
}

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

Figure 5.9: Example of Bus Controller use

5.8.2. Bus controller for the 100-pin, 112-pin, 120-pin and 144-pin package.

Figure 5.10 shows an example of external bus controller usage.

```
/* Peripheral driver function prototypes */
#include “r_pdl_bsc.h”
#include “r_pdl_cgc.h”
#include “r_pdl_intc.h”

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

/* Callback function prototype */
void BSC_error_handler(void);

void main(void)
{
    volatile uint8_t  * cs0_location_8;
    volatile uint8_t  * cs1_location_8;
    volatile uint16_t * cs2_location_16;
    volatile uint16_t * cs3_location_16;

    /* Point to respective external memory areas */
    cs3_location_16 = (uint16_t *)0x05000000ul;
    cs2_location_16 = (uint16_t *)0x06000000ul;
    cs1_location_8  = ( uint8_t *)0x07000000ul;
    cs0_location_8  = ( uint8_t *)0xFF000000ul;

    /* Initialise the system clocks including the external bus clock */
    NOTE: The code to initialise the system clocks using R_CGC_Set is omitted here.

    /* Configure the bus priority */
    R_BSC_Set(
        PDL_BSC_PRIORITY_PB1_MB1
    );

    /* Configure area 0 */
    R_BSC_CreateArea(
        0,
        PDL_BSC_WIDTH_8,
        15,
        15,
        7,
        7,
        31,
        31,
        7,
        7,
        7,
        3,
        7,
        7,
        7,
        7
    );

    /* Configure area 1 */
    R_BSC_CreateArea(
        1,
        PDL_BSC_WIDTH_8 | PDL_BSC_WRITE_BYTE,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
```

```
            0,
            0,
            0,
            0,
            0,
            0,
            0
        );

        /* Configure area 2 */
        R_BSC_CreateArea(
            2,
            PDL_BSC_WIDTH_16 | PDL_BSC_WRITE_SINGLE,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0,
            0
        );

        /* Configure area 3 */
        R_BSC_CreateArea(
            3,
            PDL_BSC_WIDTH_16,
            15,
            15,
            7,
            7,
            31,
            31,
            7,
            7,
            7,
            3,
            7,
            7,
            7,
            7
        );


        /* Configure the bus controller */
        R_BSC_Create(
            PDL_BSC_CS0_P26 | PDL_BSC_CS1_PF2 | PDL_BSC_CS2_PD2 | PDL_BSC_CS3_P12 | \
            PDL_BSC_WAIT_P82 | PDL_BSC_ALE_ENABLE,
            PDL_BSC_A9_DISABLE,
            PDL_BSC_RCV_SRRS_ENABLE,
            PDL_BSC_ERROR_ILLEGAL_ADDRESS_ENABLE | PDL_BSC_ERROR_TIME_OUT_ENABLE,
            BSC_error_handler,
            5
        );


        /* Enable the bus controller */
        R_BSC_Control(
             PDL_BSC_ENABLE
        );

        /* Write to external areas */
```

```
        *cs0_location_8 = 0x23u;
        *cs1_location_8 = 0xAAu;
        *cs2_location_16 = 0x3344u;
        *cs3_location_16 = 0xAA55u;

        /* Disable area CS1 */
        R_BSC_Destroy(
            1
        );

        /* This should generate an illegal address error */
        *cs1_location_8 = 0xAAu;
    }

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

Figure 5.10: Example of Bus Controller use for the external bus

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 RX63T” into the destination area when a falling edge occurs on pin IRQ0.
Channel 1 will copy the string “Hello, World” into the destination area as soon as it is enabled.
Notes: IRQ0 is connected to SW1 in the RSK, press SW1 to generate IRQ0 trigger.
When the transfer has completed, LED0’s state is inverted.

```
/* 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 RX63T";
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 LED: using LED0 to monitor */
        R_IO_PORT_Set(
                PDL_IO_PORT_7_1,
                PDL_IO_PORT_OUTPUT
        );

        /* 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_IRQ0,
                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 IRQ0 pin */
        R_INTC_SetExtInterrupt(PDL_INTC_IRQ2_P10);

        /* Enable the SW1(IRQ0) interrupt */
        R_INTC_CreateExtInterrupt(
                PDL_INTC_IRQ0,
                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 LED port pin */
        R_IO_PORT_Modify(
                PDL_IO_PORT_7_1,
                PDL_IO_PORT_XOR,
                1
        );

        /* Stop channel 0 */
        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 5.11: Two examples of DMAC use

5.10. Data Transfer Controller

5.10.1. Block transfer mode

Figure 5.12 shows an example of Data Transfer Controller usage with a single block transfer.

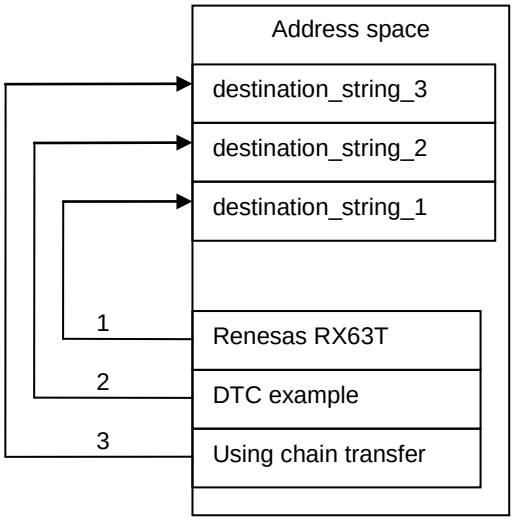
```
/* Peripheral driver function prototypes */
#include "r_pdl_dtc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_intc.h"

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

/* Required for this example */
#include <string.h>

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        #define PDL_INTC_IRQ2_PIN PDL_INTC_IRQ2_PE3
#else
        #define PDL_INTC_IRQ2_PIN PDL_INTC_IRQ2_P00
#endif

/* 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 IRQ2-triggered transfer data area */
uint32_t dtc_irq_transfer_data[4];

/* Data source and destination declarations */
const char source_string_1[]="Renesas RX63T";
volatile uint8_t destination_string_1[]=".....................";

/* Callback function prototype */
void IRQ2_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 LED */
        R_IO_PORT_Set(
                PDL_IO_PORT_7_1,
                PDL_IO_PORT_OUTPUT
        );

        /* Set the DTC options */
        R_DTC_Set(
                PDL_NO_DATA,
                dtc_vector_table
        );

        /* Configure the DTC for IRQ2 */
        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_IRQ2,
                dtc_irq_transfer_data,
                source_string_1,
```

```
                destination_string_1,
                1,
                (uint8_t)(strlen((char *)source_string_1))
        );

        /* Set IRQ2 pin */
        R_INTC_SetExtInterrupt(PDL_INTC_IRQ2_PIN);

        /* Enable the (IRQ2) interrupt */
        R_INTC_CreateExtInterrupt(
                PDL_INTC_IRQ2,
                PDL_INTC_FALLING | PDL_INTC_DTC_TRIGGER_ENABLE,
                IRQ2_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 IRQ2 falling signal*/

        while (1);
}

void IRQ2_handler(void)
{
        uint16_t StatusValue;
        uint32_t SourceAddr;
        uint32_t DestAddr;
        uint16_t TransferCount;

        /* Read the status and current source address for the IRQ2 transfer */
        R_DTC_GetStatus(
                dtc_irq_transfer_data,
                &StatusValue,
                &SourceAddr,
                &DestAddr,
                &TransferCount,
                PDL_NO_DATA
        );

        /* Invert the LED port pin */
        R_IO_PORT_Modify(
                PDL_IO_PORT_7_1,
                PDL_IO_PORT_XOR,
                1
        );

        /* Re-enable IRQ2 as a DTC trigger */
        R_DTC_Control(
                PDL_DTC_TRIGGER_IRQ2,
                PDL_NO_PTR,
                PDL_NO_PTR,
                PDL_NO_PTR,
                PDL_NO_DATA,
                PDL_NO_DATA
        );
}
```

Figure 5.12: Example of DTC use

5.10.2. Chain transfer operation

Figure 5.13 shows an example of Data Transfer Controller operation, using chain transfer of blocks.



Transfer 1 is triggered by a software interrupt and copies data from ROM into RAM.
On completion of transfer 1, transfer 2 is started.
On completion of transfer 2, transfer 3 is started.

```
/* Peripheral driver function prototypes */
#include "r_pdl_dtc.h"
#include "r_pdl_intc.h"

/* 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 RX63T";
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[] = ".....................";

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
        );
}
```

Figure 5.13: Example of DTC chain transfer

5.11. Port Output Enable

Figure 5.14 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 POE8_handler(void);

void main(void)
{
     /* Configure POE mode */
     R_POE_Set(
         PDL_POE_8_MODE_LOW_16,
         PDL_POE_HI_Z_REQ_8_ENABLE,
         PDL_NO_DATA,
         PDL_NO_DATA,
         PDL_POE_SHORT_USE_MTU
     );

     /* Configure POE event handling */
     R_POE_Create(
         PDL_POE_IRQ_HI_Z_8_ENABLE,
         PDL_NO_FUNC,
         PDL_NO_FUNC,
         POE8_handler,
         PDL_NO_FUNC,
         PDL_NO_FUNC,
         15
     );
}

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 5.14: Example of Port Output Enable function

5.12. General PWM Timer Driver

Figure 5.15 shows a usage example of General PWM Timer Driver.

```
/* Peripheral driver function prototypes */
#include "r_pdl_gpt.h"
#include "r_pdl_cgc.h"

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

void main(void)
{
    R_GPT_ControlChannel_structure gpt_ch_control_parameters;
    R_GPT_Create_structure ch_create_parameters;

    /* Initialise the system clocks */
    NOTE: The code to initialise the system clock using R_CGC_Set is omitted here.
    Please refer to 5.1 Clock Generation Circuit

    /* Configure GPT pins*/
    R_GPT_Set(
        0,
        PDL_GPT_PIN_GTIOC0A_P71 | PDL_GPT_PIN_GTIOC0B_P74
    );

    /* Power on the GPT unit*/
    R_GPT_CreateUnit(
        0,
        PDL_NO_DATA
    );

    /* Load the defaults */
    R_GPT_Create_load_defaults(&ch_create_parameters);

    /* Set the non-default options */
    ch_create_parameters.data2 = PDL_GPT_MODE_SAW | PDL_GPT_CLK_PCLK_DIV_1;
    ch_create_parameters.data6 = PDL_GPT_A_CM_INVERT | PDL_GPT_A_LOW_LOW | \
                                 PDL_GPT_A_CYCLE_RETAIN;
    ch_create_parameters.data7 = PDL_GPT_B_CM_INVERT | PDL_GPT_B_LOW_LOW | \
                                 PDL_GPT_B_CYCLE_RETAIN;

    /* Configure channel 0 for dual-waveform (A and B) output */
    R_GPT_CreateChannel(
        0,
        &ch_create_parameters
    );

    /* Set the register values for channel 0 */
    gpt_ch_control_parameters.data4 = 0x0000;
    gpt_ch_control_parameters.data5 = 0x2222;
    gpt_ch_control_parameters.data6 = 0xBBBB;
    gpt_ch_control_parameters.data11 = 0xFFDD;

    /* Load the register values and start channel 1 */
    R_GPT_ControlChannel(
        0,
        PDL_GPT_START | PDL_GPT_COUNT_DIRECTION_UP,
        PDL_GPT_REGISTER_CYCLE | PDL_GPT_REGISTER_COUNTER | \
        PDL_GPT_REGISTER_A | PDL_GPT_REGISTER_B,
        &gpt_ch_control_parameters
    );

}
```

Figure 5.15: Example of General PWM Timer Driver

5.13. Register Write Protection

Figure 5.16 shows a usage example of Register Write Protection.

```
/* Peripheral driver function prototypes */
#include " r_pdl_rwp.h"

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

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

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

    /* Enable access to the LVD registers */
    R_RWP_Control(
        PDL_RWP_ENABLE_LVD_WRITE
    );

}
```

Figure 5.16: Example of Register Write Protection

5.14. Watchdog Timer

Here the watchdog is configured to generate an NMI interrupt when the counter underflows. Notice how the NMI is enabled for WDT interrupts.

```
/* Peripheral driver function prototypes */
#include "r_pdl_intc.h"
#include "r_pdl_wdt.h"

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

static void NMI_handler(void);

void main(void)
{
    /* Enable the NMI interrupt for WDT */
    R_INTC_CreateExtInterrupt(
        PDL_INTC_NMI,
        PDL_INTC_WDT_ENABLE,
        NMI_handler,
        7
    );

    /* Configure WDT with a 25% to 75% window, no reset - hence generate NMI.*/
    R_WDT_Set(
        PDL_WDT_TIMEOUT_1024 | PDL_WDT_PCLK_DIV_2048 |
        PDL_WDT_WIN_START_75 | PDL_WDT_WIN_END_25 |
        PDL_WDT_TIMEOUT_NMI
    );

    /* Main program loop */
    while(1)
    {
        /* Refresh the watchdog */
        R_WDT_Control(
                PDL_WDT_RESET_COUNTER
        );

        /* User code is omitted here. */
    }
}

static void NMI_handler(void)
{
    uint16_t Status;

    /* Read the WDT status */
    R_WDT_Read(
        &Status
    );

    /* Has an underflow occurred? */
    if ((Status & BIT_14) != 0x0u)
    {
        /* Handle the watchdog underflow here */

    }

    /* Has a refresh error occurred? */
    if ((Status & BIT_15) != 0x0u)
    {
        /* Handle the watchdog refesh error here */
    }
}
```

Figure 5.17: Example of Watchdog Timer use

5.15. Compare Match Timer

Figure 5.18 shows an example of Compare Match Timer usage. One channel is used to generate interrupts at regular intervals.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cmt.h"
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"
#include "r_pdl_intc.h"

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

void main(void)
{
    uint8_t  Flags;
    uint16_t Counter;

    /* 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 the CPU's Interrupt Priority Level to 0 */
    R_INTC_Write(PDL_INTC_REG_IPL,0);

    /* Configure a port pin for output */
    R_IO_PORT_Set(PDL_IO_PORT_7_1, PDL_IO_PORT_OUTPUT);
    R_IO_PORT_Set(PDL_IO_PORT_7_2, PDL_IO_PORT_OUTPUT);

    R_IO_PORT_Write(PDL_IO_PORT_7_1, 1);      /* off LED0 */
    R_IO_PORT_Write(PDL_IO_PORT_7_2, 0);      /* on LED1 */

    /* 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,
        15
    );

    /* Configure CMT channel 1 in 0.1sec period and start CMT*/
    R_CMT_Create(
        1,
        PDL_CMT_PERIOD,
        1E-1,
        CMT1_handler,
        15
    );

    /* Change the frequency to 10kHz */
    R_CMT_Control(0,PDL_CMT_FREQUENCY,10E3);

    R_CMT_Read(0, PDL_NO_PTR, PDL_NO_PTR);
    R_CMT_Read(1, &Flags, &Counter);

    /* Wait for 0.5sec */
    R_CMT_CreateOneShot(
        2,
        PDL_NO_DATA,
        0.5,
        PDL_NO_FUNC,
        0
    );
```

```
        R_CMT_Control(0, PDL_CMT_START, 0); /* now start CMT0 */

        R_CMT_Control(1, PDL_CMT_STOP, 0);  /* now stop CMT1 */

        while(1);
    }

    void CMT0_handler(void)
    {
        /* Toggle the LED0 state */
        R_IO_PORT_Modify(PDL_IO_PORT_7_1, PDL_IO_PORT_XOR, 1);
    }
    void CMT1_handler(void)
    {
        /* Toggle the LED1 state */
        R_IO_PORT_Modify(PDL_IO_PORT_7_2, PDL_IO_PORT_XOR, 1)
    }
```

Figure 5.18: Example of Compare Match Timer use

5.16. Independent Watchdog Timer

Figure 5.19 shows an example of Independent Watchdog timer usage.

At start-up the underflow is checked to identify if the the reset was caused by the Independent Watchdog timer.
The watchdog timer is then configured for a 1024-count timeout period and started.
Because the watchdog timer is not refreshed, after two seconds (this depends on the frequency of the on-chip oscillator) the MCU is reset and the underflow condition is detected.

```
/* Peripheral driver function prototypes */
#include "r_pdl_iwdt.h"

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

void main(void)
{
    uint16_t Status;

    /* Read the timer status */
    R_IWDT_Read(
        &Status
    );

    /* Has an underflow occurred? */
    if ((Status & BIT_14) != 0x0u)
    {
        /* Handle the watchdog-induced reset here */
    }

    /* Configure the IWDT */
    R_IWDT_Set(
        PDL_IWDT_TIMEOUT_1024 | PDL_IWDT_CLOCK_OCO_256
    );

    /* Start the IWDT */
    R_IWDT_Control(
        PDL_IWDT_REFRESH
    );
}
```

Figure 5.19: Example of Independent Watchdog Timer use

5.17. Serial Communication Interface

5.17.1. SCI Asynchronous Using Polling.

This shows the setting of a SCI channel and the transmission and reception of data using polling.

```
/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"

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

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
    #define CHANNEL_SCI 1
#else
    #define CHANNEL_SCI 0
#endif

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 pin options */
    R_SCI_Set(
        CHANNEL_SCI,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_TXD1_PD3
    #else
        PDL_SCI_PIN_SCI0_RXD0_P24 | PDL_SCI_PIN_SCI0_TXD0_P30
    #endif
    );

    /* Set up SCI channel : Async, 8N1, 38400 baud */
    R_SCI_Create(CHANNEL_SCI,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1,
        0
    );

    /* Wait while send message */
    R_SCI_Send(CHANNEL_SCI,
        PDL_NO_DATA,
        "\r\nHello. Type 5 characters and I will echo them back.\r\n",
        0,
        PDL_NO_FUNC
    );

    /* Wait for 5 characters to be read. */
    R_SCI_Receive(CHANNEL_SCI,
        PDL_NO_DATA,
        rx_buffer,
        5,
        PDL_NO_FUNC,
        PDL_NO_FUNC
    );
```

```
        /* Echo the 5 characters back. */
        R_SCI_Send(CHANNEL_SCI,
            PDL_NO_DATA,
            rx_buffer,
            5,
            PDL_NO_FUNC
        );
    }
```

Figure 5.20: Example of SCI asynchronous operation using polling.

5.17.2. SCI Asynchronous Using Interrupts.

This shows the setting of a SCI channel 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"

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
    #define CHANNEL_SCI 1
#else
    #define CHANNEL_SCI 0
#endif

void SCIrx(void);
void SCItx(void);

volatile bool data_received;
volatile bool data_sent;

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.

    /* Initialise flags */
    data_sent = false;
    data_received = false;

    /* Set pin options */
    R_SCI_Set(
        CHANNEL_SCI,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_TXD1_PD3
    #else
        PDL_SCI_PIN_SCI0_RXD0_P24 | PDL_SCI_PIN_SCI0_TXD0_P30
    #endif
    );

    /* Set up SCI channel 0: Async, 8N1, 38400 baud */
    R_SCI_Create(
        CHANNEL_SCI,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        38400,
        1,
        0
    );

    /* Send message - register callback to say when sent */
    R_SCI_Send(CHANNEL_SCI,
        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(CHANNEL_SCI,
            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(CHANNEL_SCI,
            PDL_NO_DATA,
            rx_buffer,
            5,
            PDL_NO_FUNC
        );
    }

    void SCIrx(void)
    {
        data_received = true;
    }

    void SCItx(void)
    {
        data_sent = true;
    }
```

Figure 5.21: Example of SCI Asynchronous operation using interrupts.

5.17.3. SCI Asynchronous Using DMAC.

This shows the setting of a SCI channel 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"

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

#include <stddef.h>
#include <string.h>

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
    #define CHANNEL_SCI 1
#else
    #define CHANNEL_SCI 0
#endif

const uint8_t* string = "Hello from Renesas RX63T SCI DMAC\r\n";

void main(void)
{
    uint8_t SCI_status;

    /* 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 up SCI : Async, 8N1, 19200 baud */
    R_SCI_Create
    (
        CHANNEL_SCI,
        PDL_SCI_ASYNC | PDL_SCI_8N1,
        19200,
        1,
        0
    );

    /* Configure channel 3 of DMAC to be triggered by SCI Tx */
    R_DMAC_Create(
        3,
        PDL_DMAC_REPEAT | PDL_DMAC_SOURCE_ADDRESS_PLUS |
                PDL_DMAC_DESTINATION_ADDRESS_FIXED | PDL_DMAC_SIZE_8,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_DMAC_TRIGGER_SCI1_TX,
        string, /* Source */
        (const char *)&SCI1.TDR, /* Destination */
    #else
        PDL_DMAC_TRIGGER_SCI0_TX,
        string, /* Source */
        (const char *)&SCI0.TDR, /* Destination */
    #endif
        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
        (
            CHANNEL_SCI,
            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
            (
                    CHANNEL_SCI,
                    &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
        (
            CHANNEL_SCI,
            PDL_SCI_STOP_TX
        );

        /* Send using polling mode */
        R_SCI_Send
        (
            CHANNEL_SCI 0,
            PDL_NO_DATA,
            "Hello from Renesas RX63T SCI Polling.\r\n",
            PDL_NO_DATA,
            PDL_NO_FUNC
        );

        while(1);
    }
```

Figure 5.22: Example of SCI Asynchronous operation using DMAC.

5.17.4. Synchronous Transmission and Reception

This shows the configuration of SCI channel 0 as the clock master and channel 1 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 0
#define SLAVE_CHANNEL  1

/* Rx complete flag */
volatile uint8_t data_received;

/* Callback function prototype */
static void SCI0RxFunc(void);

void main(void)
{
    volatile uint8_t rx_buffer[5] = {0, 0, 0, 0, 0};

    /* 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(
        0,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_SCI_PIN_SCI0_TXD0_PB2 | PDL_SCI_PIN_SCI0_SCK0_PB3
    #else
        PDL_SCI_PIN_SCI0_TXD0_P30 | PDL_SCI_PIN_SCI0_SCK0_P23
    #endif
    );

    R_SCI_Set(
        1,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_SCK1_PD4
    #else
        PDL_SCI_PIN_SCI1_RXD1_P93 | PDL_SCI_PIN_SCI1_SCK1_P92
    #endif
    );

    /* Create Master Channel */
    R_SCI_Create(
        MASTER_CHANNEL,
        PDL_SCI_SYNC | PDL_SCI_RX_DISCONNECTED |
        PDL_SCI_CLK_INT_OUT,
        19200,
        1,
        0
    );
```

```
        /* 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,
            0
        );

        /* 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,
            SCI0RxFunc,
            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 0 receive complete handler */
    static void SCI0RxFunc(void)
    {
        /* Set flag */
        data_received = true;
    }
```

Figure 5.23: Example of Synchronous Transmission and Reception code

5.17.5. Synchronous Full Duplex Operation

This shows the configuration of SCI channel 1 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"

#include <stddef.h>
#include <string.h>

/*For testing connect 2 channels to each other (Rx to Tx and clock to clock):
    1. SCI0:
    RXD0 = P24, TXD0 = P30, SCK0 = P23
    Generates clock - MASTER.

    2. SCI1
    RXD1 = P93, TXD1 = P94, SCK1 = P92
    Uses external clock - SLAVE.
    */

/* SCI channel selection */
#define MASTER_CHANNEL 1
#define SLAVE_CHANNEL  0

#define DATA_LENGTH 5

/* Rx complete flag */
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);

//NOTE Before adding to the API manual remove the slave side.
void main(void)
{
    volatile uint8_t rx_buffer[DATA_LENGTH] = {0,0,0,0,0};

    /* 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(
        0,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
        PDL_SCI_PIN_SCI0_RXD0_PB1 | PDL_SCI_PIN_SCI0_TXD0_PB2 | \
        PDL_SCI_PIN_SCI0_SCK0_PB3
    #else
        PDL_SCI_PIN_SCI0_RXD0_P24 | PDL_SCI_PIN_SCI0_TXD0_P30 | \
        PDL_SCI_PIN_SCI0_SCK0_P23
    #endif
    );
```

```
        R_SCI_Set(
            1,
        #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
            PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_TXD1_PD3 | \
            PDL_SCI_PIN_SCI1_SCK1_PD4
        #else
            PDL_SCI_PIN_SCI1_RXD1_P93 | PDL_SCI_PIN_SCI1_TXD1_P94 | \
            PDL_SCI_PIN_SCI1_SCK1_P92
        #endif
        );

        /* Create Clock master channel for Rx and Tx */
        R_SCI_Create(
            MASTER_CHANNEL,
            PDL_SCI_SYNC | PDL_SCI_CLK_INT_OUT,
            19200,
            1,
            0
        );

        /* 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,
            0
        );

        /* 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 the Rx to finish. */
        while(data_received == false);
```

```
        /* Check we got the data we expected */
        if(0 != strncmp((const char *)rx_buffer, "Slave", 5))
        {
            while(1);
        }

        /* 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 5.24: Example of Synchronous Full Duplex operation

5.17.6. SCI Reception in Asynchronous Multi-Processor mode

This shows the setting of a SCI channel and the Multi-Processor mode reception of data using interrupts and polling.

```
/* Peripheral driver function prototypes */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"
#include "r_pdl_io_port.h"

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

#if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
     #define CHANNEL_SCI 1
#else
     #define CHANNEL_SCI 0
#endif

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[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.

     for (i=0; i<NUM_DATA; i++)
     {
         receive_data[i] = 0;
     }

     /* Set pin options */
     R_SCI_Set(
         CHANNEL_SCI,
    #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
         PDL_SCI_PIN_SCI1_RXD1_PD5 | PDL_SCI_PIN_SCI1_TXD1_PD3
    #else
         PDL_SCI_PIN_SCI0_RXD0_P24 | PDL_SCI_PIN_SCI0_TXD0_P30
    #endif
     );

     /* Configure the RS232 port, specify Async MP mode */
     R_SCI_Create(
         CHANNEL_SCI,
         PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
         57600,
         15,
         0
     );

     /* ------------------------------------------ */
     /*   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(
```

```
            CHANNEL_SCI,
            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(
            CHANNEL_SCI,
            PDL_NO_DATA,
            receive_data,
            10,
            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(
            CHANNEL_SCI,
            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(
                   CHANNEL_SCI,
                   PDL_NO_DATA,
                   receive_data,
                   10,
                   PDL_NO_FUNC,
                   SCIEr
            );
        }
    }

    void SCIrx(void)
    {
        data_received = true;
    }

    void SCIEr(void)
    {
        error_happen = true;
    }
```

Figure 5.25: Example of SCI Reception code in Asynchronous Multi-Processor mode

5.17.7. SCI Transmission in Asynchronous Multi-Processor mode

This shows the setting of a SCI channel and the Multi-Processor mode transmission of data using interrupts and polling.

```
/* PDL functions */
#include "r_pdl_sci.h"
#include "r_pdl_cgc.h"

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

void SCItx(void);

uint8_t* send_data0 = "Welcome to the Renesas RX63T.";
uint8_t* send_data = "testing ASYNC MP mode";
bool tx_end;

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(
          0,
      #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
          PDL_SCI_PIN_SCI0_RXD0_PB1 | PDL_SCI_PIN_SCI0_TXD0_PB2
      #else
          PDL_SCI_PIN_SCI0_RXD0_P24 | PDL_SCI_PIN_SCI0_TXD0_P30
      #endif
      );

      /* Configure the RS232 port, specify Async MP mode */
      R_SCI_Create(
          0,
          PDL_SCI_8N1 | PDL_SCI_ASYNC_MP,
          57600,
          15,
          0
      );

      /* ------------------------------------------- */
      /* 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(
          0,
          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(
          0,
          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(
            0,
            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(
            0,
            PDL_NO_DATA,
            send_data,
            0,
            PDL_NO_FUNC
        );
        while(1);
    }

    void SCItx(void)
    {
        tx_end = true;
    }
```

Figure 5.26: Example of SCI Transmission code in Asynchronous Multi-Processor mode

5.17.8. SCI in SPI Mode

This shows the setting of SCI channel 0 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 0 pin options */
     R_SCI_Set(
         0,
     #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
         PDL_SCI_PIN_SCI0_SMISO0_PB1 | PDL_SCI_PIN_SCI0_SMOSI0_PB2 | \
         PDL_SCI_PIN_SCI0_SCK0_PB3 | PDL_SCI_PIN_SCI0_SS0_PD7
     #else
         PDL_SCI_PIN_SCI0_SMISO0_P24 | PDL_SCI_PIN_SCI0_SMOSI0_P30 | \
         PDL_SCI_PIN_SCI0_SCK0_P23 | PDL_SCI_PIN_SCI0_SS0_P22
     #endif
     );

     /* Create SPI master */
     R_SCI_Create(
         0,
         PDL_SCI_SYNC | PDL_SCI_SPI_MODE |
         PDL_SCI_RX_DISCONNECTED | PDL_SCI_CLK_INT_OUT,
         19200,
         1,
         0
     );

     /* Start sending data */
     R_SCI_SPI_Transfer(
         0,
         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(0);
}

static void SCItx(void)
{
     data_sent = true;
}
```

Figure 5.27: Example of SCI in SPI mode

5.17.9. SCI in IIC Mode

This shows the setting of SCI channel 1 in to 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 1
/* 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 1  pin options */
     R_SCI_Set(
         1,
     #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
         PDL_SCI_PIN_SCI1_SSCL1_PD5  | PDL_SCI_PIN_SCI1_SSDA1_PD3
     #else
         PDL_SCI_PIN_SCI1_SSCL1_P93  | PDL_SCI_PIN_SCI1_SSDA1_P94
     #endif
     );

     /* 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,
         0
     );

     /* 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 5.28: Example of SCI in IIC mode

5.17.10. SCI in IIC Mode using DMAC

This shows the setting of SCI channel 1 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 1
/* 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 1  pin options */
     R_SCI_Set(
         CHANNEL_SCI_IIC,
     #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
         PDL_SCI_PIN_SCI1_SSCL1_PD5  | PDL_SCI_PIN_SCI1_SSDA1_PD3
     #else
         PDL_SCI_PIN_SCI1_SSCL1_P93  | PDL_SCI_PIN_SCI1_SSDA1_P94
     #endif
     );

     /* 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,
         0
     );

     /* 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 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_IRQ_END,
            PDL_DMAC_TRIGGER_SCI1_TX,
            IIC_Buffer,           /* Source */
            (uint8_t *)&SCI1.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 5.29: Example of SCI in IIC mode using DMAC

5.17.11. SCI in IIC Mode using DTC

This shows the setting of SCI channel 1 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 1
/* 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 1  pin options */
     R_SCI_Set(
         1,
     #if !defined (DEVICE_PACKAGE_64_PIN) && !defined (DEVICE_PACKAGE_48_PIN)
         PDL_SCI_PIN_SCI1_SSCL1_PD5  | PDL_SCI_PIN_SCI1_SSDA1_PD3
     #else
         PDL_SCI_PIN_SCI1_SSCL1_P93  | PDL_SCI_PIN_SCI1_SSDA1_P94
     #endif
     );

     /* 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,
         0
     );

     /* 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_SCI1_RX,
            dtc_iic1_rx_transfer_data,
            (uint8_t *)&SCI1.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_SCI1_TX,
            dtc_iic1_tx_transfer_data,
            &IIC_Dummy_value,             /* Source */
            (uint8_t *)&SCI1.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 5.30: Example of SCI in IIC mode using DTC

5.18. I²C Bus Interface

In the following examples, the bus activity will be illustrated using the following format.

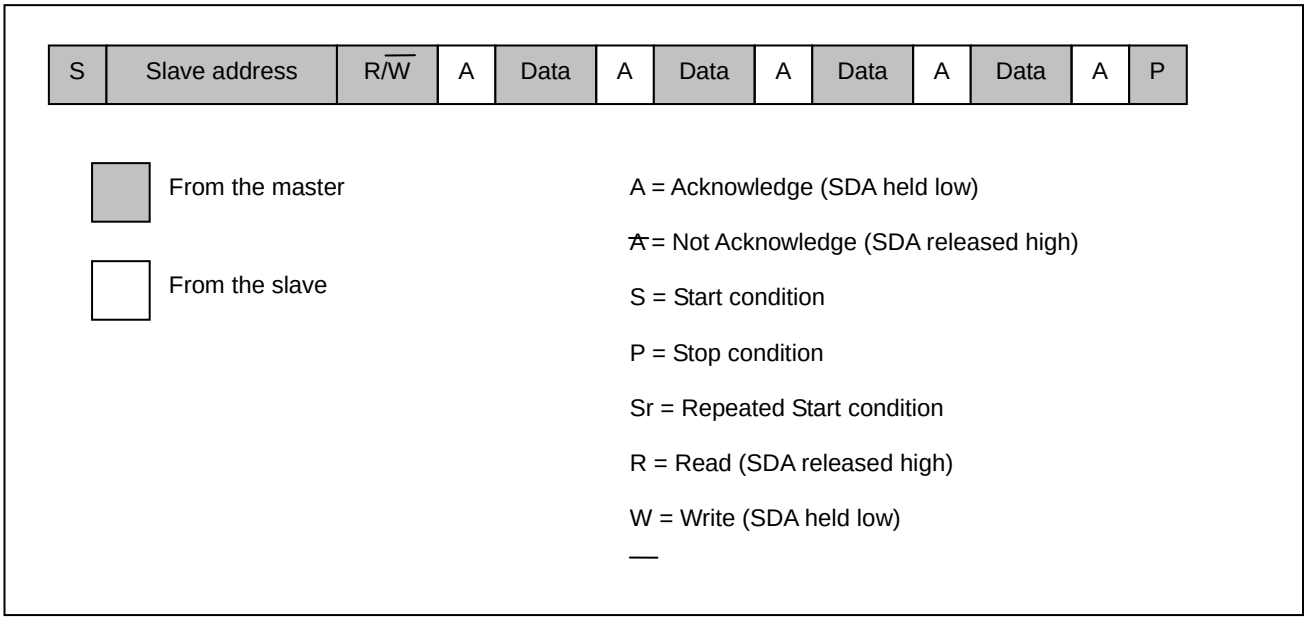


Figure 5.31: I²C bus activity notation

5.18.1. 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.

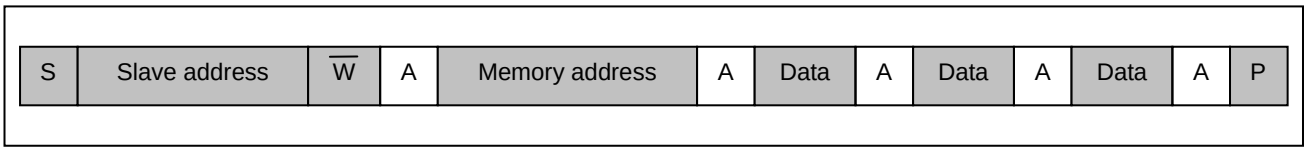


Figure 5.32: The bus activity, showing 4 bytes being transmitted to the EEPROM

```
/* 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[5] = {0x00, 0x01, 0x02, 0x03, 0x04};
    uint8_t data_storage[5];
    uint32_t status_flags = 0;
    uint16_t TxChars;
    uint16_t RxChars;

    /* 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.

    /* 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 5.33: Configure the I²C channel and write 3 data bytes to the first locations

2) Reception

Continuing from above; The I²C in master is now used to read 4 bytes from a slave device from the current memory address.

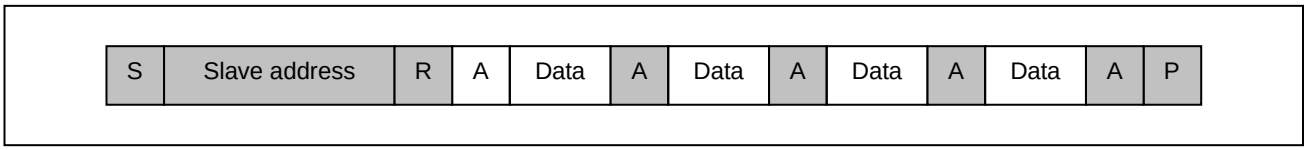


Figure 5.34: The bus activity, showing 4 bytes being transmitted by the EEPROM

```
        /* Read data from the EEPROM, using polling */
        if (R_IIC_MasterReceive(
            0,
            PDL_NO_DATA,
            EEPROM_ADDRESS,
            data_storage,
            4,
            PDL_NO_FUNC,
            0
        ) == false)
        {
            /* Read the channel and transfer status */
            R_IIC_GetStatus(
                0,
                &status_flags,
                PDL_NO_PTR,
                &RxChars
            );
            /* Review the flags and transmit count to decide on the next action */
        }
```

Figure 5.35: An example of reading data from the EEPROM

3) Repeated Start

Continuing from above; The memory address pointer of an EEPROM will be modified, and then a Repeat Start condition used to change to read the byte at that memory location in the EEPROM.

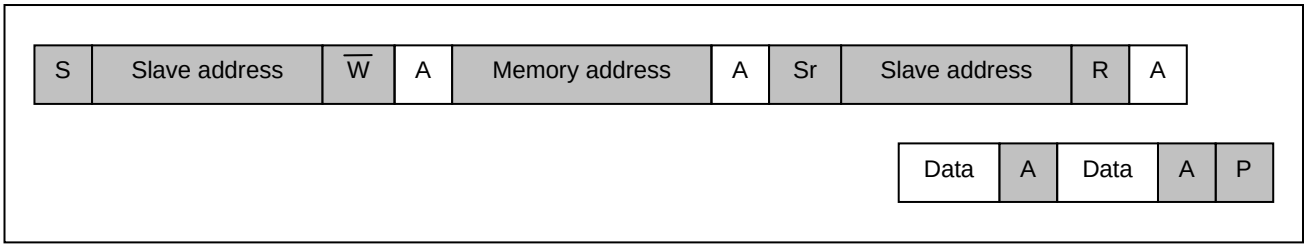


Figure 5.36: The bus activity, showing the Repeated Start condition when switching to the Read process

```
/* Send 1 byte to the EEPROM to update the 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,
    2,
    PDL_NO_FUNC,
    0
);
```

Figure 5.37: Set the EEPROM sub address and then read 2 bytes.

5.18.2. 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.
    Please refer to 5.1 Clock Generation Circuit.

    /* 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 */
        if(false == R_IIC_MasterSend(
            IIC_CHANNEL,
            PDL_IIC_DMAC_TRIGGER_ENABLE,
            EEPROM_ADDRESS,
            PDL_NO_PTR,
            0,
            PDL_NO_FUNC,
            0))
        {
            while(1);
        }
        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 */
        if(false == R_IIC_MasterReceive(
            IIC_CHANNEL,
            PDL_IIC_DMAC_TRIGGER_ENABLE,
            EEPROM_ADDRESS,
            PDL_NO_PTR,
            0,
            PDL_NO_FUNC,
            0
        ))
        {
            while(1);
        }
        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 5.38: An example of writing data to and reading data from an EEPROM, using two DMAC channels

5.18.3. 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

volatile uint8_t bus_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.
    Please refer to 5.1 Clock Generation Circuit.

    /* 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)
    {
        bus_busy = true;

        /* 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 (bus_busy == true)
        {
            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)
{
    bus_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 (bus_busy == true);
}

void iic_tx_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_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
    );

    bus_busy = false;
}
```

Figure 5.39: An example of writing data to and reading data from an EEPROM, using the DTC

5.18.4. 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"
#include "r_pdl_cmt.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 0xA0

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.
    Please refer to 5.1 Clock Generation Circuit.

    /* Select IIC mode at 100kHz, 300ns rise time, 200ns fall time */
    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,
        (300 << 16) | 200
    );

    /* 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 5.40: Virtual IIC Slave memory

5.19. Serial Peripheral Interface

5.19.1. Master operation with multiple slaves

This is an example of Serial Peripheral Interface usage where one SPI master communicates with four SPI slaves. Each slave requires different data bit lengths.

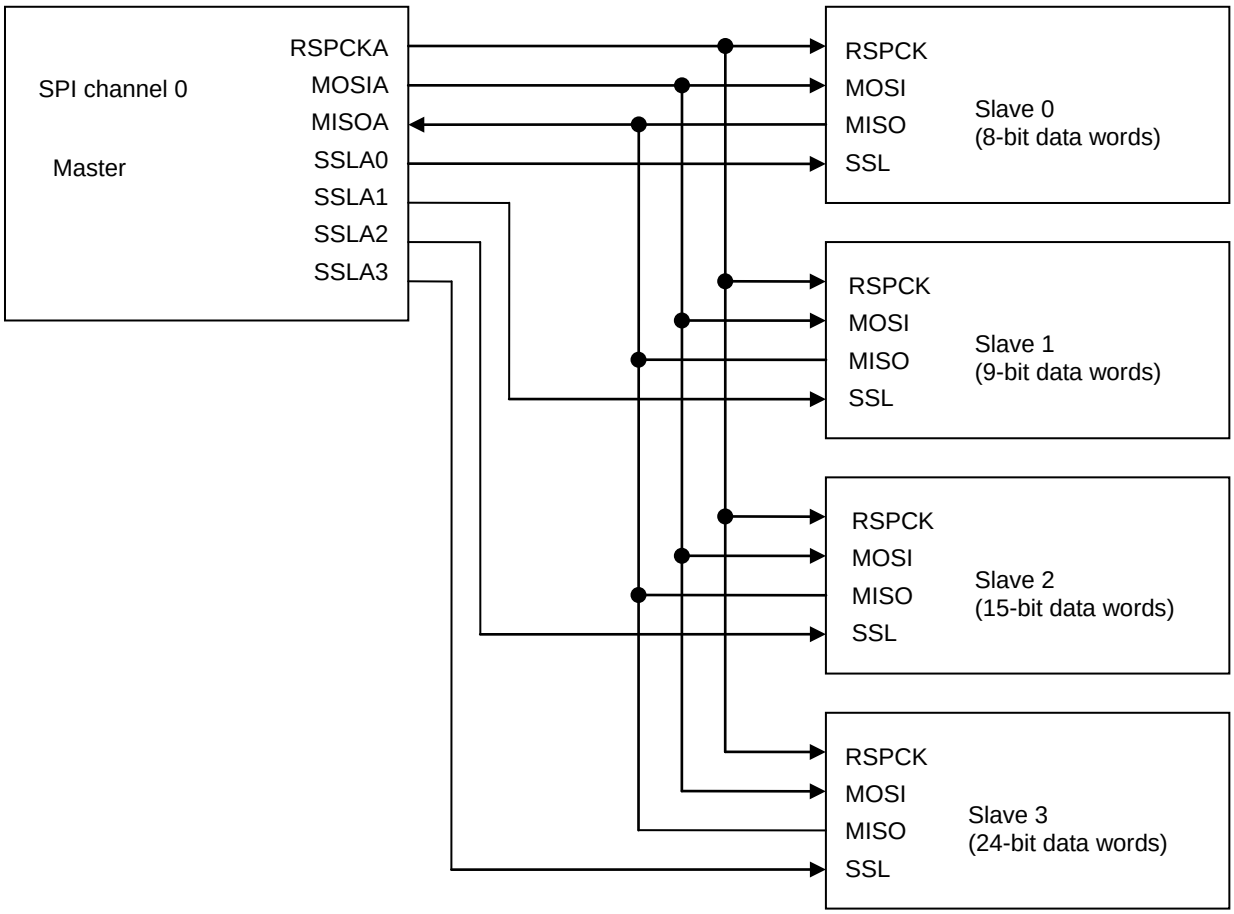


Figure 5.41 shows how data of appropriate bit lengths is transferred to each SPI slave.
Commands 0 to 3 are executed in sequence, with each command asserting the appropriate SSL pin.

```
/* Peripheral driver function prototypes */
#include "r_pdl_spi.h"
#include "r_pdl_cgc.h"

/* 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(
            MASTER_CHANNEL,
            PDL_SPI_RSPCKA_P24 | PDL_SPI_MOSIA_P23 | PDL_SPI_MISOA_P22 | \
            PDL_SPI_SSLA0_P30 | PDL_SPI_SSLA1_P31 | \
            PDL_SPI_SSLA2_P32 | PDL_SPI_SSLA3_P33,
            PDL_NO_DATA
        );

        /* 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,
            PDL_NO_FUNC,
            0
        );
    }
```

Figure 5.41: Example of multiple slave Serial Peripheral Interface use

5.20. CRC calculator

Figure 5.42 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 5.42: Example of CRC calculation

5.21. 12-bit Analog to Digital Converter

This example 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_cmt.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_results[8];

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 AN000 */
    R_ADC_12_Set(
        PDL_ADC_12_PIN_AN000_P40
    );

    /* Configure ADC in single scan mode. */
    R_ADC_12_CreateUnit(
        0,
        PDL_ADC_12_SCAN_SINGLE,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        PDL_NO_DATA,
        0,
        0,
        PDL_NO_FUNC,
        0,
        PDL_NO_FUNC,
        0
    );

    /* Configure ADC on AN000*/
    R_ADC_12_CreateChannel(
        0,
        0,
        PDL_ADC_12_CH_SAMPLE_AND_HOLD_ENABLE,
        PDL_NO_DATA,
        5E-6,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );

    /* Wait 10 ms for the ADC to stabilise */
    R_CMT_CreateOneShot(
        0,
        0,
        10E-3,
        PDL_NO_FUNC,
        PDL_NO_DATA
    );

    /* Start / Trigger the ADC */
    R_ADC_12_Control(
        PDL_ADC_12_0_ON
```

```
        );

        /* Fetch the results */
        R_ADC_12_Read(
            0,
            adc_results,
            PDL_NO_DATA,
            PDL_NO_DATA
        );

        /* Shut down ADC */
        R_ADC_12_Destroy(
            0
        );

        while(1);
    }
```

Figure 5.43: Example of ADC_12

5.22. 10-bit Analog to Digital Converter

Figure 5.44 shows ADC_10 used in single scan mode, with a software trigger and a specified sampling time.

```
/* Peripheral driver function prototypes */
#include "r_pdl_adc_10.h"
#include "r_pdl_cmt.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_results[20];
uint16_t diag_result;

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 AN0 */
    R_ADC_10_Set(
        PDL_ADC_10_PIN_AN0_P60
    );

    /* Configure ADC in single scan mode. */
    R_ADC_10_CreateUnit(
        0,
        PDL_ADC_10_SCAN_SINGLE,
        PDL_ADC_10_TRIGGER_SOFTWARE,
        0,
        PDL_NO_FUNC,
        0
    );

    /* Configure sampling time for AN0*/
    R_ADC_10_CreateChannel(
        0,
        0,
        PDL_ADC_10_CH_VALUE_ADDITION_DISABLE,
        5E-6
    );

    /* Wait 10 ms for the ADC to stabilise */
    R_CMT_CreateOneShot(
       0,
       0,
       10E-3,
       PDL_NO_FUNC,
       PDL_NO_DATA
    );

    /* Start conversion on the ADC */
    R_ADC_10_Control(
       PDL_ADC_10_0_ON
    );

    /* Fetch the results */
    R_ADC_10_Read(
       0,
       adc_results,
       &diag_result
    );

    /* Shut down ADC */
```

```
        R_ADC_10_Destroy(
            0
        );

        while(1);
    }
```

Figure 5.44: Example of ADC_10

5.23. 10-bit Digital to Analog Converter

Figure 5.45 shows an example of DAC_10 usage

```
/* Peripheral driver function prototypes */
#include "r_pdl_dac_10.h"

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

void main(void)
{
        /*-------------------------------------------------*/
        /* Test normal DAC_10 operation                    */
        /* VREF = 5.0V                                     */
        /* Expected output voltages are shown in comments  */
        /*-------------------------------------------------*/


        /* Test align right (default) */
        R_DAC_10_Create(
                PDL_DAC_10_CHANNEL_1,
                0x0,
                0x0     // 0.0V
        );

        /* Write new data to both DAC channels */
        R_DAC_10_Write(
                PDL_DAC_10_CHANNEL_1,
                0x0,
                0x200   // 2.5V
        );

        /* Shut down both DAC channels */
        R_DAC_10_Destroy(
                PDL_DAC_10_CHANNEL_1
        );

        /* Test align left */
        R_DAC_10_Create(
                PDL_DAC_10_CHANNEL_1 | PDL_DAC_10_ALIGN_LEFT,
                0x0,
                0xffc0  // 5.0V
        );

        /* Write new data to both DAC channels */
        R_DAC_10_Write(
                PDL_DAC_10_CHANNEL_1,
                0x0,
                0x8000  // 2.5V
        );

        /* Shut down both DAC channels */
        R_DAC_10_Destroy(
                PDL_DAC_10_CHANNEL_1
        );

        while(1);
}
```

Figure 5.45: Example of DAC_10

5.24. Data Operation Circuit

Figure 5.46: Example of DOC shows an example Data Operation Circuit usage.

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

static void SetClocks(void);
static 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 is omitted here.
    Please refer to 5.1 Clock Generation Circuit

    /* 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 */
        15                      /* 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 DMAC to complete */
        while(false == g_bCallbackDone);

        /* Read the result including checking for overflow */
        R_DOC_Read(
            &status,
            &result
        );

        while(1);
    }

    static void Callback_Done(void)
    {
        g_bCallbackDone = true;
    }
```

Figure 5.46: Example of DOC

5.25. Multifunction Pin Controller

Figure 5.47: Example of MPC show an example multifunction 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 P00PFS */
    R_MPC_Write(
        PDL_MPC_REG_P00PFS,
        0xC5
    );

    /* Set bit 3 in P00PFS to 1 */
    R_MPC_Modify(
        PDL_MPC_REG_P00PFS,
        PDL_MPC_OR,
        0x08
    );

    /* Get the value of register P00PFS */
    R_MPC_Read(
        PDL_MPC_REG_P00PFS,
        &data
    );

    while(1);
}
```

Figure 5.47: Example of MPC

5.26. Multi-Function Timer Pulse Unit

Figure 5.48: Example of MTU show an example multi-function timer pulse unit usage.

```
/* Peripheral driver function prototypes */
#include "r_pdl_cgc.h"
#include "r_pdl_mtu3.h"

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

static void Callback_Match_TGRA (void);
static void Callback_Match_TGRE (void);

/* Allocate structures */
R_MTU3_Create_structure create_parameters;
volatile int Counter_Callback_Match_TGRA;
volatile int Counter_Callback_Match_TGRE;

void main(void)
{
    /* Parameters that are structures. */
    R_MTU3_Create_structure create_parameters;
    R_MTU3_ControlChannel_structure control_parameters;

    /* Load the Create defaults */
    R_MTU3_Create_load_defaults(&create_parameters);

    NOTE: The code to initialise the system clock is omitted here.
    Please refer to 5.1 Clock Generation Circuit

    /* Set Create options */
    create_parameters.channel_mode = PDL_MTU3_MODE_NORMAL;
    /* Clock - slow down from defualt */
    create_parameters.counter_operation = PDL_MTU3_CLK_PCLKA_DIV_64;

    /* Compare match interrupts */
    create_parameters.func1 = Callback_Match_TGRA;
    create_parameters.interrupt_priority_1 = 5;
    create_parameters.func5 = Callback_Match_TGRE;
    create_parameters.interrupt_priority_2 = 7;
    /* Count match values */
    create_parameters.TGRA_TCNTV_value = 0x1111;
    create_parameters.TGRE_TGRW_value = 0xEEEE;

    /* Create Channel 0 */
    R_MTU3_Create(0, &create_parameters);

    /* Clear flags */
    Counter_Callback_Match_TGRA = 0;
    Counter_Callback_Match_TGRE = 0;

    /* Set Control options to start the timer */
    control_parameters.control_setting = PDL_MTU3_START;
    control_parameters.register_selection = PDL_NO_DATA;
    R_MTU3_ControlChannel(0, &control_parameters);

    /* Wait for match */
    while(0 == Counter_Callback_Match_TGRA);
    while(0 == Counter_Callback_Match_TGRE);

    /* Stop the counter */
    control_parameters.control_setting = PDL_MTU3_STOP;
    control_parameters.register_selection = PDL_NO_DATA;
    R_MTU3_ControlChannel(0, &control_parameters);

    R_MTU3_Destroy();
```

```
        while(1);
    }

    static void Callback_Match_TGRA (void)
    {
        uint8_t status;
        uint16_t TGRA_count_value;
        uint16_t TCNT_Counter;

        /* Read the counter value */
        R_MTU3_ReadChannel(0,
            &status,
            &TCNT_Counter,
            &TGRA_count_value,
            PDL_NO_PTR, PDL_NO_PTR, PDL_NO_PTR,
            PDL_NO_PTR, PDL_NO_PTR );

        /* Increment count */
        Counter_Callback_Match_TGRA++;
    }

    static void Callback_Match_TGRE (void)
    {
            Counter_Callback_Match_TGRE++;
    }
```

Figure 5.48: Example of MTU

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	RX63T Group User's Manual

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 09, 2012	—	First Edition issued. Developed using MCU hardware manual Rev.1.00.
1.01	Sep. 28, 2012	55	R_INTC_SetExtInterrupt: Amended the program example.
		71	R_INTC_Modify: Amended the program example.
		77	R_IO_PORT_Set: Modified the program example.
		82	R_IO_PORT_Read: Modified the program example.
		83	R_IO_PORT_Write: Modified the program example.
		84	R_IO_PORT_Compare: Modified the program example.
		85	R_IO_PORT_Modify: Modified the program example.
		86	R_IO_PORT_Wait: Modified the program example.
		89	R_MPC_Read: Amended the program example.
		90	R_MPC_Write: Amended the program example.
		91	R_MPC_Modify: Amended the program example.
		104	R_CAC_Create: Added remarks that callbacks must clear flags as interrupt is level based.
		107	R_CAC_Control: Added remark about avoiding lockup by not doing enable/disable from interrupts and main at same time.
		120	R_POE_Set: Updated the description for date3.
		129	R_DMAC_Create: Updated the program example.
		136	R_DTC_Set: Edit remark of data2.
		140	R_DTC_Create: Updated the program example.
		143	R_DTC_Control: Updated the second program example.
		147	R_MTU3_Set: Amended the program example.
		158	R_MTU3_Destroy: Amended the program example.
		173	R_BSC_Create: Updated the description for data1, data2 and data3.
		175	R_POE_Create: Updated the description for func2 and func5.
		181	R_GPT_Set: Amended the program example.
		182	R_GPT_CreateUnit: Added support for DMAC activation.
		183	R_GPT_CreateUnit: Amended the program example.
		184	R_GPT_CreateChannel: Added support for DMAC activation.
		190	R_GPT_CreateChannel: Updated the description for data11, data12, data13 and data14.
		196	R_GPT_ControlChannel: Amended the program example.
		198	R_GPT_ControlUnit: Amended the program example.
		221	R_SCI_Set: Amended the program example.
		225	R_SCI_Create: Modified the description of data3.
		288	R_ADC_12_CreateChannel: Corrected the program example.
		291	R_ADC_12_Read: Updated the description for data3.
		307	Modified the IO Port usage example.
		319	Updated the DTC usage example.
2.00	Oct 31, 2012	1	No RTOS Support sentence added.
		3	Updated screen shots of batch copy utility.
		49-51	R_CGC_Set: Added support for packages with 100 pins or more.

Rev.	Date	Description	
		Page	Summary
			Corrected the equation for achievable PLL frequencies.
		52	R_CGC_Control: Added support for packages with 100 pins or more.
		76	I/O Port: Added support for packages with 100 pins or more.
		77	R_IO_PORT_Set: Added support for packages with 100 pins or more.
		78	R_IO_PORT_ReadControl: Added support for packages with 100 pins or more.
		80	R_IO_PORT_ModifyControl: Added support for packages with 100 pins or more.
		98	R_LVD_Create: Added voltage level configuration for packages with 100 pins or more.
		179	R_POE_Set: Updated the description for data4.
		222	R_SCI_Set: Added extra pin definitions availble on larger pin packages.
		222-248	SCI: Added support for the extra channels availble on larger pin packages.
		280-294	ADC_12: Added unit for packages with 100 pins or more.
2.01	Feb 01, 2013	118	R_RWP_Control: Update description for Register write control option in data1.
		119	R_RWP_GetStatus: Update data1, data2 description: Change PDL_NO_DATA to PDL_NO_PTR.
		42	DAC_10: Added description for overview.
		47	DAC_10: Added description for R_DAC_10_Create, R_DAC_10_Destroy, R_DAC_10_Write.
		312 – 315	DAC_10: Added detail for R_DAC_10_Create, R_DAC_10_Destroy, R_DAC_10_Write.
		397	DAC_10: Added the program example.
		337 – 339	Update DMAC usage examples to be compatible with RSK.
		41	ADC_10: Added description for overview.
		47	ADC_10: Added description for R_ADC_10_ControlAll, R_ADC_10_CreateChannelAll, R_ADC_10_CreateUnitAll, R_ADC_10_DestroyAll, R_ADC_10_ReadAll, R_ADC_10_SetAll
		300 – 311	ADC_10: Added detail for R_ADC_10_ControlAll, R_ADC_10_CreateChannelAll, R_ADC_10_CreateUnitAll, R_ADC_10_DestroyAll, R_ADC_10_ReadAll, R_ADC_10_SetAll
		395 – 396	ADC_10: Added the program example.
		128 – 130	R_DMAC_Create: Added trigger source support larger pin packages.
		267 – 279	SPI: Add channel 1 support to RSPI, and new port allocation of RSPI0
		389	SPI: Correct pin name of RSPI channel 0
		110	R_LPC_Create: Added IRQ6-DS pin and IRQ7-DS pin interrupt selection. Added output port retention control function
		118	R_LPC_GetStatus: Added interrupt flags of IRQ6-DS and IRQ7-DS
		335	Revised software standby mode example
		61	R_CGC_Set: Added remark for UCLK and changed setting range for UCLK.
		62	R_CGC_Set: Update program example.
		78 – 82	IO_PORT: remove option “PDL_IO_PORT_PULL_UP_ON”, “PDL_IO_PORT_PULL_UP_OFF” and “PDL_IO_PORT_PULL_UP”
		122 – 130	BSC: modified the API R_BSC_Set,R_BSC_Create, R_BSC_CreateArea, R_BSC_Destroy, R_BSC_Control
		104	CAC: add option PDL_CAC_CACREF_PORT_0_0

Rev.	Date	Description	
		Page	Summary
		337	CGC: Update usage example.
		66	R_INTC_SetExtInterrupt: Add pins for high pin packages
		180 – 188	POE3: Added settings of POE4, POE10-E4, POE12, GPT67 and MTU67 into R_POE_Set, R_POE_Create, R_POE_Control, R_POE_GetStatus
		64	R_INTC_CreateFastInterrupt: Update interrupt vectors for high pin package.
		79-82	INTC: Update group interrupt.
		340	INTC: Update usage example.
		191 – 214	GPT: Added extra channels/unit support for bigger pin package.
		294 – 308	ADC_12 : add new option and unit support for bigger pin package
		97	R_MCU_GetStatus: update date1
		92	MPC: MPC register definitions
		156	R_MTU3_Set: Add new macro definition
		270 – 284	IIC: Add a new channel option for bigger pin package.
		233	SCI: Delete a wrong pin
		236	SCI: Add MTU3 clock source selection for bigger pin package
		365 – 388	SCI: Update usage example
		92	MPC: add note for Multifunction Pin Controller
		64	INTC: Update external interrupt
		232	R_IWDT_Set: Add a remark
		145 – 146	DTC: Add lack trigger for bigger pin package
		184	R_MTU3_ReadChannel: Add a remark
		239-240	R_SCI_Receive: Add continuous receive mode
2.02	Apr 05, 2013	178 – 179	R_POE_Set: Correct macro and remove redundant information
		79	R_IO_PORT_ReadControl: add description in data3 and remark
		81	R_IO_PORT_ModifyControl: add description in data3 and remark
		313	R_ADC_10_Control: remove redundant comment
		338	LPC: revise wrong comments of sample code
		48	R_CGC_Set: Change BCLK pin limit.
		66-69	INTC: Update parameter name.
		271-283	SPI: Update description in data1 of all APIs, and update description in data3 of R_SPI_Set
		310	ADC_10: Add type description for data4
		78	R_IO_PORT_Set: Add a remark
		30	GPT: Modify number of channel from four to eight
		186-209	GPT: Update packages description for all APIs
		151	MTU: Add MTU5 macro for bigger pin package
		254-271	IIC: add more comment for data1 of all APIs
		260, 262	IIC: Add PDL_IIC_10_BIT_SLAVE_ADDRESS for channel configuration parameter [data2]
		74, 76	INTC: Add comment for package support.
		306-309	ADC_10: Add sampling time parameter for Self-Dianostic
		403	Revise sample code of ADC_10
		308, 310	Add remark: Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same

Rev.	Date	Description	
		Page	Summary
		295,297	Add remark: Make sure sampling time calculated or specified for channel 0 and self-diagnosis are the same.
		302	R_ADC_12_Read: Add remark for Self-diagnosis result format.
		314	R_ADC_10_Read: Add remark for Self-diagnosis result format.
		144	R_DTC_Create: Update DTC vector name.
		254-270	IIC: Rephrase comment for available channels in different pin packages.
		271-283	SPI: Rephrase comment for available channels in different pin packages.
2.03	Sept 12,2013	2	Add the “1.2. Compiler options when you use this product”
		3	Change 1.3.2 content into "Using RPDL stand-alone"
		4	Revise picture: “litle” to “little”
		6	Add content “To use library with debug information, enter “RPDL\RX630_library_debug” as the File path.”
		10	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”
		13	Add the “11) Using library with debug information”
2.10	Nov 08, 2013	103	R_LVD_Create: User wants to use both LVD1 and LVD2; user must configure both LVD1 and LVD2 simultaneously.
		108	R_CAC_Create: Remove “extern” in sample code.
		153	R_DTC_Getstatus: Remove “extern” in sample code.
		155	R_MTU3_Set: Revise MTCLKA, MTCLKB: Valid when n = 0 to 4
		171	R_MTU3_ControlUnit: “Disable or enable counter clearing on TGRA compare match. This must not be enabled if not in Complementary PWM Mode 1.”
		176	R_MTU3_ControlUnit: Add remark for PWM waveforms generation and output protection function in complementary PWM mode.
		213-214	Add R_GPT_EdgeDelay_Create.
		215-216	Add R_GPT_EdgeDelay_Control.
		217	Add R_GPT_EdgeDelay_Destroy.
		247-257	SCI: Correct sample code.
		264	R_IIC_Create: Add remark: “When the digital noise filter circuit is enabled, the ICBRL, ICBRH register value should be equal or greater than <the number of noise filter steps + 1>.”
		309	R_ADC_12_Control: Add remark: Do not select CPU Off unless there is any interrupt to wake up the CPU.
		321	R_ADC_10_Control: Add remark: Do not select CPU Off unless there is any interrupt to wake up the CPU.
		361	Replace usage example of TPU by RWP.
		415-417	Add usage example for MTU and MPC.
2.11	Sept 12, 2014	96	R_MCU_Control: Remove On-chip RAM control.
		230	R_IWDT_Set: Add remark “The IWDT counter frequency must not be greater than the PCLKB / 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

Rev.	Date	Description	
		Page	Summary
			IWDTCLK clock frequency after division”

Renesas Peripheral Driver Library
User's Manual
RX63T Group

Publication Date: Rev.2.11 Sept 12, 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

RX63T Group

Renesas Electronics Corporation

R20UT2201EE0211