To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

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

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

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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

User's Manual

Renesas Starter Kit for SH7216

RPDL Manual

RENESAS SINGLE-CHIP MICROCOMPUTER
SuperH RISC Engine

Renesas Electronics
www.renesas.com

Rev.1.00 2010.02

Table of Contents

1. Introduction 1-1

1.1. Using the library within your project 1-2
1.1.1. Copy the files 1-2
1.1.2. Include the header files 1-2
1.1.3. Add the library file path 1-2
1.1.4. Build the project 1-2

1.2. Document structure 1-2

1.3. Acronyms and abbreviations 1-3

2. Driver 2-1

2.1. Overview 2-1

2.2. Control Function 2-1

2.3. Serial Communication Interface Driver 2-2

2.4. I/O Port Driver 2-3

2.5. Pin Function Control Driver 2-4

2.6. Analog to Digital Converter Driver 2-5

2.7. Clock Pulse Generator Driver 2-6

2.8. Compare Match Timer Driver 2-7

2.9. Bus State Controller Driver 2-8

2.10. DMA Controller Driver 2-9

2.11. Watchdog Timer Driver 2-10

3. Standard Types 3-1

4. Library Reference 4-1

4.1. API List by Peripheral Function 4-1

4.2. Description of Each API 4-2
4.2.1. Clock Pulse Generator 4-3
1) R_CPG_Set 4-3
4.2.2. I/O Port 4-6
1) R_IO_PORT_Set 4-6
2) R_IO_PORT_ReadData 4-9
3) R_IO_PORT_ReadControl 4-12
4) R_IO_PORT_Write 4-15
5) R_IO_PORT_Compare 4-18
6) R_IO_PORT_ModifyData 4-21
7) R_IO_PORT_ModifyControl 4-24
8) R_IO_PORT_Wait 4-27
4.2.3. Pin Function Control 4-30
1) R_PFC_Write 4-30
2) R_PFC_Read 4-32
3) R_PFC_Modify 4-34
4.2.4. Bus State Controller 4-37
1) R_BSC_Create 4-37
2) R_BSC_CreateArea 4-41
3) R_BSC_Destroy 4-46
4) R_BSC_Control 4-47
5) R_BSC_GetStatus 4-48

4.2.5. DMA Controller 4-49
1) R_DMAC_Create 4-49
2) R_DMAC_Destroy 4-52
3) R_DMAC_Control 4-53
4) R_DMAC_GetStatus 4-55
4.2.6. Compare Match Timer 4-57
1) R_CMT_Create 4-57
2) R_CMT_Destroy 4-59
3) R_CMT_Control 4-60
4) R_CMT_Read 4-61
5) R_CMT_CreateOneShot 4-62
4.2.7. Serial Communication Interface 4-64
1) R_SCI_Create 4-64
2) R_SCI_Destroy 4-68
3) R_SCI_Send 4-69
4) R_SCI_Receive 4-71
5) R_SCI_Stop 4-73
6) R_SCI_GetStatus 4-74
4.2.8. 12-bit Analog to Digital Converter 4-75
1) R_ADC_12_Create 4-75
2) R_ADC_12_Destroy 4-78
3) R_ADC_12_Control 4-79
4) R_ADC_12_Read 4-80
4.2.9. Watchdog Timer 4-81
1) R_WDT_Create 4-81
2) R_WDT_Destroy 4-83
3) R_WDT_Control 4-84
4) R_WDT_Read 4-85

1. Introduction

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

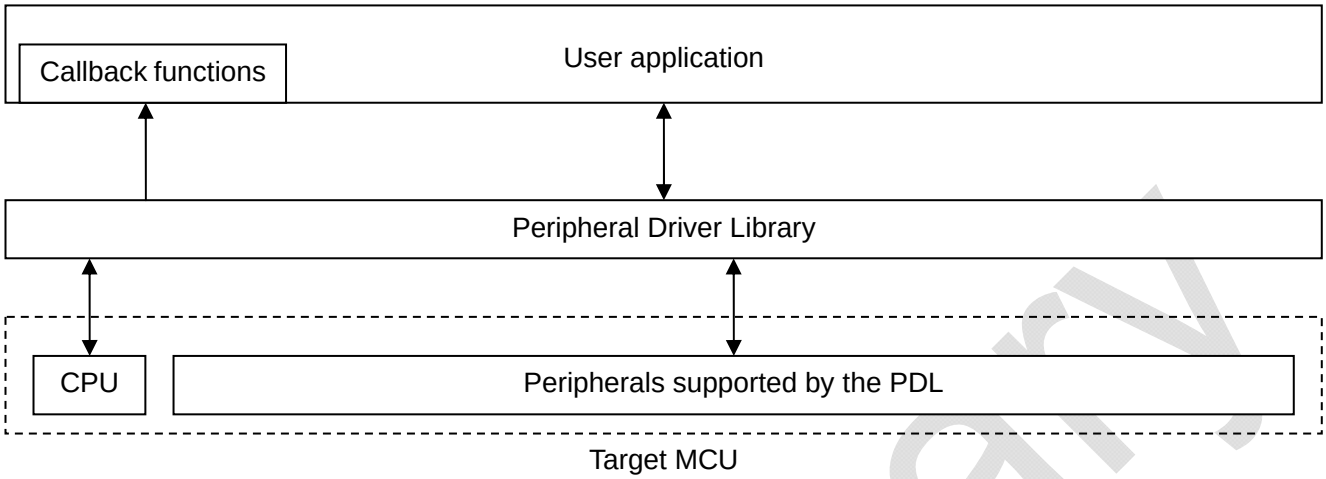

Figure 1-1: System configuration, with all peripherals supported by PDL

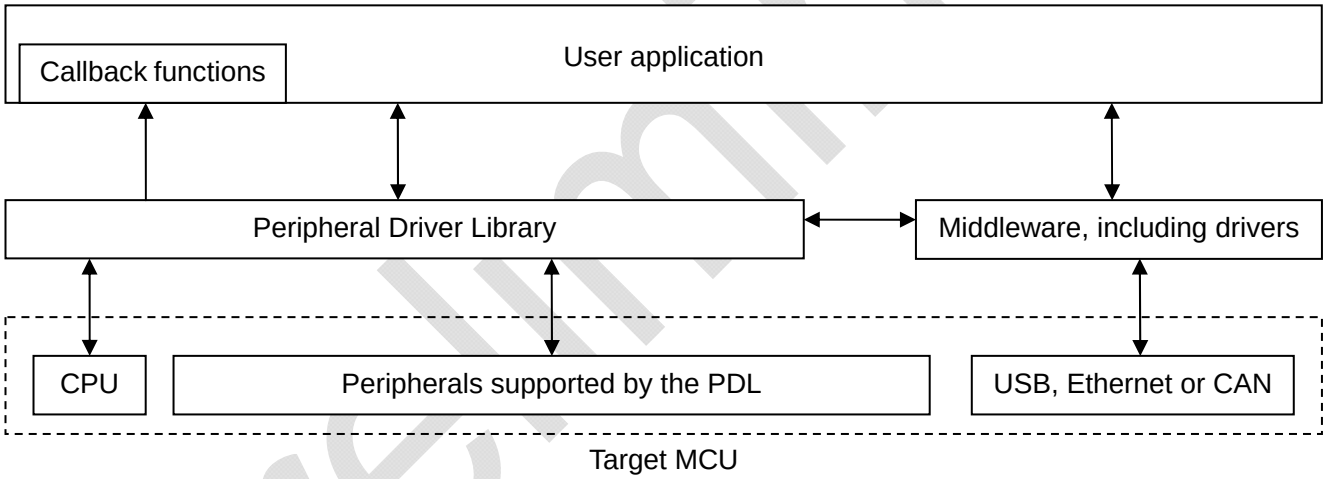

Figure 1-2: System configuration, with middleware taking direct control of some peripherals

b) Header files containing the information that the user needs to call any of the functions from their own application code.

The binary file is produced using the Renesas SuperH Standard Tool chain v9.3. It should be usable by another compiler that conforms to the Renesas Application Binary Interface.

The coding standards and naming conventions are specified by Renesas.

1.1. Using the library within your project

The driver library can be used:

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.

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

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

a) Copy the required header and library files to a suitable location.
b) Include the required header files.
c) Add the driver library file to the linked files list.

1.1.1. Copy the files

Copy the files to a suitable area using the Copy_PDL_SH7216_144_pin.bat utility.

1.1.2. Include the header files

The header files are made available by adding an entry to their location.

From the Build menu, select "SuperH Standard engine Standard tool chain".
From the C/C++ tab, open the "Show entries for :" drop-down menu and select "Include file directories".

Click on the "Add…" button. From the "Relative to" drop-down list, select "Custom directory".
Click on the "Browse…" button and navigate to the area where the files were copied.
Open the folder and click on "Select".
Click on "OK".

1.1.3. Add the library file path

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

From the Link/Library tab, open the "Show entries for :" drop-down menu and select "Library files".
Use the "Add…" button to add an entry for the PDL library.

1.1.4. Build the project

No further configuration is required
Simply add the required PDL function calls to your source code include the necessary header files and re-build the project.

1.2. Document structure

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

1.3. Acronyms and abbreviations

ADC	Analog to Digital Converter
API	Application Programming Interface
Bit	Binary digit
BSC	Bus State Controller
CMT	Compare Match Timer
CPG	Clock Pulse Generator
CPU	Central Processing Unit
DMA	Direct Memory Access
DMAC	DMA Controller
INTC	Interrupt Controller
I/O	Input / Output
MCU	Microcontroller Unit
MTU	Multi-Function Timer Pulse Unit
NMI	Non-Maskable Interrupt
PDG	Peripheral Driver Generator
PDL	Peripheral Driver Library
PFC	Port Function Control
SCI	Serial Communications Interface

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 Function

This library has the following control functions available as a peripheral driver.

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

(2) I/O Port
These driver functions are used to configure the I/O pins and provide data read, write, compare and modify operations.

(3) Pin Function controller
These driver functions are used for configuring the I/O pin optional functions.

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

(5) Clock Pulse Generator
These driver functions are used to configure the various internal/external clock signals.

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

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

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

(9) Watchdog Timer
These driver functions are used for configuring the watchdog timer to various modes.

2.3. Serial Communication Interface Driver

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

1. Configuration for use, including
    a. Automatic baud rate clock calculations
    b. Automatic interrupt control
    c. Automatic I/O pin configuration
2. Disabling channels that are no longer required and enabling low-power mode.
3. Transmitting data, with polling or interrupt mode automatically selected.
4. Receiving data, with polling or interrupt mode automatically selected.
5. Stopping the transmission and / or reception of data.
6. Reading the status flags.

2.4. I/O Port Driver

The driver functions support the use of the 110 I/O port pins, providing the following operations.

1. Configuration for use.
2. Reading a pin or 16-bit port value.
3. Writing to a pin or 16-bit port.
4. Comparing a pin or 16-bit port with a supplied value.
5. Modifying a pin or 16-bit port using a logical operation.
6. Waiting until a pin or 16-bit port matches a supplied value.

2.5. Pin Function Control Driver

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

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

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

2.6. Analog to Digital Converter Driver

The driver functions support the use of the 8 ADC channels, providing the following operations.

1. Configuration for use, including
   a. Automatic clock setting using sampling time as an input.
   b. Automatic interrupt control
   c. Automatic I/O pin configuration
2. Disabling units that are no longer required and enabling low-power mode.
3. Control of one or more channels (depending upon the selected mode)
4. Reading the conversion results of one of more units, with support for polling or interrupts.

2.7. Clock Pulse Generator Driver

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

1. Configuration of the multiple clock outputs for system, peripheral and external bus operation
2. Controlling the bus clock output pin.

2.8. 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
    a. Automatic clock setting using frequency or period as an input.
    b. Automatic interrupt control
2. Disabling channels that are no longer required and enabling low-power mode.
3. Control of a timer, including change of frequency.

2.9. Bus State Controller Driver

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

1. Configuration of the seven address space areas
2. Configuration of the error handling functions.
3. Disabling an area that is not required.

2.10. 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
   a. Access to all control bits.
   b. Automatic interrupt control
2. Disabling DMA channels that are no longer required and enabling low-power mode.
3. Control of one or more channels.
4. Reading the status and operation registers of a channel.

2.11. Watchdog Timer Driver

The driver functions support the control of the Watchdog Timer (WDT), providing the following operations.

1. Configuration for use, including
   a. Use as an interval timer.
   b. Use as a watchdog timer.
2. Enabling disabling timer interrupt.
3. Selecting whether to reset the system on watchdog timer interrupt or not.

3. Standard Types

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

Table 3-1: Data types

Type	Defined in	Description	Range
bool	r_pdl_common_defs.h	Boolean	0 (false) to 1 (true)
float	C	Floating point, 32 bits	-∞ to +∞
uint8_t	r_pdl_common_defs.h	Unsigned, 8 bits	0 to 255
uint16_t		Unsigned, 16 bits	0 to 2^{15}-1
uint32_t		Unsigned, 32 bits	0 to 2^{32}-1

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 Pulse Generator	1	R_CPG_Set	Configure the clock pulse generator
IO PORT	1	R_IO_PORT_Set	Write an I/O port's control registers.
	2	R_IO_PORT_ReadData	Read an I/O port's control registers.
	3	R_IO_PORT_ReadControl	Read control data corresponding to an I/O port.
	4	R_IO_PORT_Write	Write data to an I/O port.
	5	R_IO_PORT_Compare	Compare specified data with the data from an I/O port.
	6	R_IO_PORT_ModifyData	Modify the pin states on an I/O port.
	7	R_IO_PORT_ModifyControl	Modify the pin states on an I/O port
	8	R_IO_PORT_Wait	Wait for a match on an I/O port.
Port Function Control	1	R_PFC_Write	Write to a PFC register.
	2	R_PFC_Read	Read a PFC register.
	3	R_PFC_Modify	Modify a PFC register.
Bus Controller	1	R_BSC_Create	Configure the external bus controller.
	2	R_BSC_CreateArea	Configure an external bus area.
	3	R_BSC_Destroy	Stop the Bus Controller.
	4	R_BSC_Control	Modify the External Bus Controller operation.
	5	R_BSC_GetStatus	Read the External Bus Controller status flags.
DMA Controller	1	R_DMAC_Create	Configure the DMA controller.
	2	R_DMAC_Destroy	Disable a DMA channel.
	3	R_DMAC_Control	Control the DMA controller.
	4	R_DMAC_GetStatus	Check the status of the DMA channel.
Compare Match Timer	1	R_CMT_Create	Configure a CMT channel.
	2	R_CMT_Destroy	Disable a CMT unit.
	3	R_CMT_Control	Control CMT operation.
	4	R_CMT_Read	Read CMT channel status and registers.
	5	R_CMT_CreateOneShot	Supports one shot function using CMT.
Serial Communication Interface	1	R_SCI_Create	SCI channel setup.
	2	R_SCI_Destroy	Shut down a SCI channel.
	3	R_SCI_Send	Send a string of characters.
	4	R_SCI_Receive	Receive a string of characters.
	5	R_SCI_Stop	Terminate SCI transmission or reception.
	6	R_SCI_GetStatus	Check the status of a SCI channel.
12-bit Analog to Digital converter	1	R_ADC_12_Create	Configure an ADC unit.
	2	R_ADC_12_Destroy	Disable the specified ADC unit.
	3	R_ADC_12_Control	Start or stop an ADC unit.
	4	R_ADC_12_Read	Read the ADC conversion results.
Watchdog Timer	1	R_WDT_Create	Configure WDT module
	2	R_WDT_Destroy	Stop & disable WDT module
	3	R_WDT_Control	Start, stop WDT module or clear counter
	4	R_WDT_Read	Read WDT status flags

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.

```
/* PDL definitions */
#include “r_pdl_pfc.h”
#include “r_pdl_sci.h”

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

/* Application-specific definitions */
#include <stddef.h>

void func( void )
{
        bool result;

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

        /* Keep trying to send a string (if the channel is busy) */
        do
        {
                result = R_SCISendString(
                        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.

4.2.1. Clock Pulse Generator

1) R_CPG_Set

Synopsis

Configure the clock pulse generator

Prototype

```
bool R_CPG_Set (
    uint32_t data1,     // Input frequency
    uint32_t data2,     // System clock frequency
    uint32_t data3,     // Bus clock frequency
    uint32_t data4,     // Peripheral clock frequency
    uint32_t data5,     // MTU clock frequency
    uint32_t data6,     // ADC clock frequency
    uint8_t data7,      // USB clock configuration
    uint8_t data8       // Configuration options
);
```

Description

Set the clock output frequencies and options.

[data1]
The frequency of the main clock oscillator in Hertz.

[data2]
The desired frequency of the System clock (ICLK) in Hertz.

[data3]
The desired frequency of the Bus clock (BCLK) in Hertz.

[data4]
The desired frequency of the Peripheral clock (PCLK) in Hertz.

[data5]
The desired frequency of the MTU clock (MCLK) in Hertz.

[data6]
The desired frequency of the ADC clock (ADCLK) in Hertz.

[data7]

- pin output control

PDL_CPG_USB_PLL PDL_CPG_USB_EXT	Specifies USB clock configuration

[data8]
Select the clock source

PDL_CPG_CK_DISABLE PDL_CPG_CK_ENABLE	Specifies the clock configuration options

Note: The default setting is shown in **bold**.

Return value

True if all parameters are valid and exclusive; otherwise false.
For SH7216, the following rules shall be checked:

- Main clock oscillator frequency: 10 to 12.5 MHz.
- f_{ICLK}: 20 to 200 MHz
- f_{PCLK}: 20 to 50 MHz
- f_{BCLK}: 20 to 50 MHz
- f_{MCLK}:40 to 100MHz
- f_{ACLK}:40 to 50MHz
- $f_{ICLK} \geq f_{BCLK} \geq f_{PCLK}$ and $f_{MCLK} \geq f_{PCLK}$

Functionality

Clock pulse generator

References

None.

Remarks

- This function must be called before configuring clock-dependent modules.
- This function modifies the BCLK pin for input or output.

Program example

```
/* PDL definitions */
#include “r_pdl_cpg.h”

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

void func(void)
{
      /* Configure operation using a 12.5 MHz input clock */
      /* ICLK = 200 MHz, PCLK = 50 MHz, BCLK = 50 MHz, MCLK =  50 MHz, ACLK = 50MHz*/
      R_CPG_SetAll(12.5E6, 200E6, 50E6, 50E6, 50E6, 50E6, PDL_CPG_USB_EXT,
PDL_CPG_CK_ENABLE);
}
```

4.2.2. I/O Port

1) R_IO_PORT_Set

Synopsis

Set IO port control registers to desired value..

Prototype

```
bool R_IO_PORT_Set (
    uint32_t data1,     // Port or port pin number to select
    uint16_t data2,     // Value to be assigned to data direction (I/O) register
    uint16_t data3      // Value to be assigned to CMOS control register
);
```

Description (1/2)

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

[data1]

Use either one of the following definition values to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2

PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3
PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4
PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]

- The value will be between 0x0000 and 0xFFFF for a port. For a port pin values from table below should be used.

PDL_IO_PORT_INPUT or PDL_IO_PORT_OUTPUT	Port pin should function as either input or output pin.

[data3]

- The value will be between 0x0000 and 0xFFFF for a port. For a port pin values from table below should be used.

PDL_PULLED_HIGH or **PDL_NOT_PULLED_HIGH**	Port pin should be internally pulled high or not.

Return value

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

Functionality

I/O port

References

R_IO_PORT_ReadData, R_IO_PORT_ReadControl

Remarks

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

Program example

```
#include "  r_pdl_io_port_library_SH7216.h"

void func( void )
{
        uint16_t data = 0;

        /* Get the value of port E */
        R_IO_PORT_Set (PDL_IO_PORT_E,
                                PDL_IO_PORT_INPUT,
                                PDL_NOT_PULLED_HIGH
                                );

        /* Get the value of port F2 */
        R_IO_PORT_Set ( PDL_IO_PORT_F_2, 4, 4);
}
```

2) R_IO_PORT_ReadData

Synopsis

Read data from an I/O port.

Prototype

```
bool R_IO_PORT_ReadData (
    uint32_t data1,     // Port or port pin selection
    uint16_t * data2    // Pointer to the variable in which read value shall be stored.
);
```

Description (1/2)

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

[data1]

Use either one of the following definition values to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3
PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4

PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]
The value will be between 0x0000 and 0xFFFF 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.

Functionality

I/O port

Reference

R_IO_PORT_ReadControl, R_IO_PORT_Set

Remark

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

Program example

```
#include "  r_pdl_io_port_library_SH7216.h"

void func( void )
{
        uint16_t data = 0;

        /* Get the value of port E */
        R_IO_PORT_ReadData (PDL_IO_PORT_E, &data);

        /* Get the value of port F2 */
        R_IO_PORT_ReadData (PDL_IO_PORT_F_2, &data);
}
```

3) R_IO_PORT_ReadControl

Synopsis

Read control data corresponding to an I/O port.

Prototype

```
bool R_IO_PORT_ReadControl (
    uint32_t data1,     // Port or port pin selection
    uint16_t * data2    // Pointer to the variable in which CMOS control value shall be stored.
  | uint16_t * data3  | // Pointer to the variable in which IO control value shall be stored.
);
```

Description (1/2)

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

[data1]

Use either one of the following definition values to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3

PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4
PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]
If a port pin has been passed in argument 1 then following are the possible values:
PDL_NOT_PULLED_HIGH or
PDL_PULLED_HIGH
If a port has been passed in argument 1, then the variable holds the value of CMOS register contents

[data3]
If a port pin has been passed in argument 1 then following are the possible values:
PDL_IO_PORT_INPUT or
PDL_IO_PORT_OUTPUT
If a port has been passed in argument 1, then the variable holds the value of IO register contents

Return value	If the I/O port specification is incorrect, false is returned; otherwise, true is returned.
Functionality	I/O port
Reference	R_IO_PORT_Set, R_IO_PORT_ModifyControl
Remark	• If an invalid port or pin is specified, the operation of the function cannot be guaranteed.

Program example

```
#include "  r_pdl_io_port_library_SH7216.h"

void func( void )
{
        uint16_t io_data = 0;
        uint16_t cmos_data = 0;

        /* Get the value of port E */
        R_IO_PORT_ReadControl (PDL_IO_PORT_E, &cmos_data , &io_data);

        /* Get the value of port F2 */
        R_IO_PORT_ReadControl (PDL_IO_PORT_F_2, &cmos_data , &io_data);
}
```

4) R_IO_PORT_Write

Synopsis

Write data to an I/O port.

Prototype

```
bool R_IO_PORT_Write (
    uint32_t data1,     // Port or port pin selection
    uint16_t data2      // The data to be written to the I/O port or port pin.
);
```

Description (1/2)

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

[data1]

Use either one of the following definition values to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3
PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4

PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]
The value must be between 0x0000 and 0xFFFF for a port, 0 or 1 for a pin.

Return value

True if the parameters are valid; otherwise false.

Functionality

I/O port

References

R_IO_PORT_Set, R_IO_PORT_ReadData, R_IO_PORT_ModifyData

Remarks

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

Program example

```
#include "  r_pdl_io_port_library_SH7216.h"

void func( void )
{
      /* Write 0xFF01 to value of port E */
      R_IO_PORT_Write(PDL_IO_PORT_E, 0xFF01);

      /* Write 1 to the Port Pin D3 */
      R_IO_PORT_Write(PDL_IO_PORT_D_2, 0x1);
}
```

5) R_IO_PORT_Compare

Synopsis

Check the pin states on an I/O port.

Prototype

```
bool R_IO_PORT_Compare (
    uint32_t data1,     // Input port or port pin selection
    uint16_t data2,     // Target value to compare
    void * func         // Call_back function if a match occurs
);
```

Description (1/2)

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 to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3

PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4
PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]
The value to be compared with; Between 0x0000 and 0xFFFF 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.

Functionality

I/O port

References

R_IO_PORT_Set, R_IO_PORT_ReadData, R_IO_PORT_ModifyData

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_io_port_library_SH7216.h”

void IoHandler1{}
void IoHandler2{}

void func( void )
{

	/* Call function IoHandler1 if port pin PC5 is high */
	R_IO_PORT_ Compare(
		PDL_IO_PORT_C_5,
		1,
		IoHandler1
	);

	/* Call function IoHandler2 if port G reads as 0x5501 */
	R_IO_PORT_Compare(PDL_IO_PORT_DH,
		0x5501,
		IoHandler2
	);
}
```

6) R_IO_PORT_ModifyData

Synopsis

Modify the pin states on an I/O port.

Prototype

```
bool R_IO_PORT_ModifyData(
    uint32_t data1,     // Output port or port pin selection
    uint8_t data2,      // Modification value
    uint8_t data3       // Logical operation
);
```

Description (1/2)

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 to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3

PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4
PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[data2]
The value to be used for the modification; Between 0x00 and 0xFFFF for a port, 0 or 1 for a pin.

[data3]
- 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 (^).

Return value

True if the parameters are valid; otherwise false.

Functionality

I/O port

References

R_IO_PORT_ReadData, R_IO_PORT_Set, R_IO_PORT_Compare

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_io_port_library_SH7216.h”

void func( void )
{
        /* Invert port pin PC5 */
        R_IO_PORT_Modify (
                PDL_IO_PORT_C_5,
                1,
                PDL_IO_PORT_XOR
        );

        /* And the value port B with 0x0055 */
        R_IO_PORT_Modify (
                PDL_IO_PORT_B,
                0x55,
                PDL_IO_PORT_AND
        );
}
```

7) R_IO_PORT_ModifyControl

Synopsis

Modify the pin states on an I/O port.

Prototype

```
bool R_IO_PORT_ModifyControl(
    uint32_t data1,     // Output port or port pin selection
    uint8_t data2,      // Modification value
    uint8_t data3       // Control Register to be modified
    uint8_t data4       // Logical operation
);
```

Description (1/2)

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 to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2

PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3
PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4
PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

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

[data3]
- The control registers to be modified.

PDL_IO_PORT_DIRECTION or	Data direction (I/O) registers.
PDL_IO_PORT_PULL_UP	CMOS control register

[data4]
- 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 (^).

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

Functionality
I/O port

References
R_IO_PORT_ReadControl, R_IO_PORT_Set

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

Program example

```
/* PDL definitions */
#include “r_pdl_io_port_library_SH7216.h”

void func( void )
{
        /* Invert port pin PC5 */
        R_IO_PORT_ModifyControl (
                PDL_IO_PORT_C_5,
                1,
                PDL_IO_PORT_DIRECTION,
                PDL_IO_PORT_XOR
        );

        /* And the value port B with 0x0055 */
        R_IO_PORT_ModifyControl (
                PDL_IO_PORT_B,
                0x55,
                PDL_IO_PORT_PULL_UP,
               PDL_IO_PORT_AND
        );
}
```

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 (1/2)

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

[data1]

Use either one of the following definition values to select the I/O port.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_IO_PORT_AH	PORT A High Bit	PDL_IO_PORT_AL	PORT A Lower Bit
PDL_IO_PORT_B	PORT BL	PDL_IO_PORT_C	PORT C
PDL_IO_PORT_DH	PORT D High Bit	PDL_IO_PORT_DL	PORT D Lower bit
PDL_IO_PORT_F	PORT F	PDL_IO_PORT_E	PORT E

Or any of the following definition values to select the I/O port pin.

PDL_IO_PORT_A_0	PORT A0	PDL_IO_PORT_D_0	PORT D0
PDL_IO_PORT_A_1	PORT A1	PDL_IO_PORT_D_1	PORT D1
PDL_IO_PORT_A_2	PORT A2	PDL_IO_PORT_D_2	PORT D2
PDL_IO_PORT_A_3	PORT A3	PDL_IO_PORT_D_3	PORT D3
PDL_IO_PORT_A_4	PORT A4	PDL_IO_PORT_D_4	PORT D4
PDL_IO_PORT_A_5	PORT A5	PDL_IO_PORT_D_5	PORT D5
PDL_IO_PORT_A_6	PORT A6	PDL_IO_PORT_D_6	PORT D6
PDL_IO_PORT_A_7	PORT A7	PDL_IO_PORT_D_7	PORT D7
PDL_IO_PORT_A_8	PORT A8	PDL_IO_PORT_D_8	PORT D8
PDL_IO_PORT_A_9	PORT A9	PDL_IO_PORT_D_9	PORT D9
PDL_IO_PORT_A_10	PORT A10	PDL_IO_PORT_D_10	PORT D10
PDL_IO_PORT_A_11	PORT A11	PDL_IO_PORT_D_11	PORT D11
PDL_IO_PORT_A_12	PORT A12	PDL_IO_PORT_D_12	PORT D12
PDL_IO_PORT_A_13	PORT A13	PDL_IO_PORT_D_13	PORT D13
PDL_IO_PORT_A_14	PORT A14	PDL_IO_PORT_D_14	PORT D14
PDL_IO_PORT_A_15	PORT A15	PDL_IO_PORT_D_15	PORT D15
PDL_IO_PORT_A_16	PORT A16	PDL_IO_PORT_D_16	PORT D16
PDL_IO_PORT_A_17	PORT A17	PDL_IO_PORT_D_17	PORT D17
PDL_IO_PORT_A_18	PORT A18	PDL_IO_PORT_D_18	PORT D18
PDL_IO_PORT_A_19	PORT A19	PDL_IO_PORT_D_19	PORT D19
PDL_IO_PORT_A_20	PORT A20	PDL_IO_PORT_D_20	PORT D20
PDL_IO_PORT_A_21	PORT A21	PDL_IO_PORT_D_21	PORT D21
PDL_IO_PORT_A_22	PORT D7	PDL_IO_PORT_D_22	PORT D22
PDL_IO_PORT_B_0	PORT B0	PDL_IO_PORT_D_23	PORT D23
PDL_IO_PORT_B_1	PORT B1	PDL_IO_PORT_D_24	PORT D24
PDL_IO_PORT_B_2	PORT B2	PDL_IO_PORT_D_25	PORT D25
PDL_IO_PORT_B_3	PORT B3	PDL_IO_PORT_D_26	PORT D26
PDL_IO_PORT_B_4	PORT B4	PDL_IO_PORT_D_27	PORT D27
PDL_IO_PORT_B_5	PORT B5	PDL_IO_PORT_D_28	PORT D28
PDL_IO_PORT_B_6	PORT B6	PDL_IO_PORT_D_29	PORT D29
PDL_IO_PORT_B_7	PORT B7	PDL_IO_PORT_D_30	PORT D30
PDL_IO_PORT_B_8	PORT B8	PDL_IO_PORT_D_31	PORT D31
PDL_IO_PORT_B_9	PORT B9	PDL_IO_PORT_E_0	PORT E0
PDL_IO_PORT_B_10	PORT B10	PDL_IO_PORT_E_1	PORT E1
PDL_IO_PORT_B_11	PORT B11	PDL_IO_PORT_E_2	PORT E2
PDL_IO_PORT_B_12	PORT B12	PDL_IO_PORT_E_3	PORT E3
PDL_IO_PORT_B_13	PORT B13	PDL_IO_PORT_E_4	PORT E4

PDL_IO_PORT_B_14	PORT B14	PDL_IO_PORT_E_5	PORT E5
PDL_IO_PORT_B_15	PORT B15	PDL_IO_PORT_E_6	PORT E6
PDL_IO_PORT_C_0	PORT C0	PDL_IO_PORT_E_7	PORT E7
PDL_IO_PORT_C_1	PORT C1	PDL_IO_PORT_E_8	PORT E8
PDL_IO_PORT_C_2	PORT C2	PDL_IO_PORT_E_9	PORT E9
PDL_IO_PORT_C_3	PORT C3	PDL_IO_PORT_E_10	PORT E10
PDL_IO_PORT_C_4	PORT C4	PDL_IO_PORT_E_11	PORT E11
PDL_IO_PORT_C_5	PORT C5	PDL_IO_PORT_E_12	PORT E12
PDL_IO_PORT_C_6	PORT C6	PDL_IO_PORT_E_13	PORT E13
PDL_IO_PORT_C_7	PORT C7	PDL_IO_PORT_E_14	PORT E14
PDL_IO_PORT_C_8	PORT C8	PDL_IO_PORT_E_15	PORT E15
PDL_IO_PORT_C_9	PORT C9	PDL_IO_PORT_F_0	PORT F0
PDL_IO_PORT_C_10	PORT C10	PDL_IO_PORT_F_1	PORT F1
PDL_IO_PORT_C_11	PORT C11	PDL_IO_PORT_F_2	PORT F2
PDL_IO_PORT_C_12	PORT C12	PDL_IO_PORT_F_3	PORT F3
PDL_IO_PORT_C_13	PORT C13	PDL_IO_PORT_F_4	PORT F4
PDL_IO_PORT_C_14	PORT C14	PDL_IO_PORT_F_5	PORT F5
PDL_IO_PORT_C_15	PORT C15	PDL_IO_PORT_F_6	PORT F6
		PDL_IO_PORT_F_7	PORT F7

[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.

Functionality

I/O port

References

R_IO_PORT_Set, R_IO_PORT_ReadData, R_IO_PORT_Compare, R_IO_PORT_ModifyData

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_io_port_library_SH7216.h”

void func( void )
{
        /* Wait until pin PC1 reads as 0 */
        R_IO_PORT_Wait(
                PDL_IO_PORT_C_1,
                0
        );

        /* Wait until port E reads as 0x0015 */
        R_IO_PORT_Wait(
                PDL_IO_PORT_E,
                0x0015
        );
}
```

4.2.3. Pin Function Control

1) R_PFC_Write

Synopsis

Set the pin function for a port pin.

Prototype

```
bool R_PFC_Write (
    uint32_t data1,      // Port number or port control register
    uint8_t data2,       // Pin function to be assigned to the port pin
);
```

Description (1/2)

Set the pin function control value.

[data1]

Any of the following definition values to select the I/O port control registers

PDL_PFC_PACRH2	PDL_PFC_PBCRL4	PDL_PFC_PCCRL2	PDL_PFC_PDCRL3
PDL_PFC_PACRH1	PDL_PFC_PBCRL3	PDL_PFC_PCCRL1	PDL_PFC_PDCRL2
PDL_PFC_PACRL4	PDL_PFC_PBCRL2	PDL_PFC_PDCRH4	PDL_PFC_PDCRL1
PDL_PFC_PACRL3	PDL_PFC_PBCRL1	PDL_PFC_PDCRH3	PDL_PFC_PECRL4
PDL_PFC_PACRL2	PDL_PFC_PCCRL4	PDL_PFC_PDCRH2	PDL_PFC_PECRL3
PDL_PFC_PACRL1	PDL_PFC_PCCRL3	PDL_PFC_PDCRH1	PDL_PFC_PECRL2
		PDL_PFC_PDCRL4	PDL_PFC_PECRL1

Any of the following definition values to select the I/O port pin.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_PFC_A_0	PORT A0	PDL_PFC_D_0	PORT D0
PDL_PFC_A_1	PORT A1	PDL_PFC_D_1	PORT D1
PDL_PFC_A_2	PORT A2	PDL_PFC_D_2	PORT D2
PDL_PFC_A_3	PORT A3	PDL_PFC_D_3	PORT D3
PDL_PFC_A_4	PORT A4	PDL_PFC_D_4	PORT D4
PDL_PFC_A_5	PORT A5	PDL_PFC_D_5	PORT D5
PDL_PFC_A_6	PORT A6	PDL_PFC_D_6	PORT D6
PDL_PFC_A_7	PORT A7	PDL_PFC_D_7	PORT D7
PDL_PFC_A_8	PORT A8	PDL_PFC_D_8	PORT D8
PDL_PFC_A_9	PORT A9	PDL_PFC_D_9	PORT D9
PDL_PFC_A_10	PORT A10	PDL_PFC_D_10	PORT D10
PDL_PFC_A_11	PORT A11	PDL_PFC_D_11	PORT D11
PDL_PFC_A_12	PORT A12	PDL_PFC_D_12	PORT D12
PDL_PFC_A_13	PORT A13	PDL_PFC_D_13	PORT D13
PDL_PFC_A_14	PORT A14	PDL_PFC_D_14	PORT D14
PDL_PFC_A_15	PORT A15	PDL_PFC_D_15	PORT D15
PDL_PFC_A_16	PORT A16	PDL_PFC_D_16	PORT D16
PDL_PFC_A_17	PORT A17	PDL_PFC_D_17	PORT D17
PDL_PFC_A_18	PORT A18	PDL_PFC_D_18	PORT D18
PDL_PFC_A_19	PORT A19	PDL_PFC_D_19	PORT D19
PDL_PFC_A_20	PORT A20	PDL_PFC_D_20	PORT D20
PDL_PFC_A_21	PORT A21	PDL_PFC_D_21	PORT D21
PDL_PFC_A_22	PORT D7	PDL_PFC_D_22	PORT D22
PDL_PFC_B_0	PORT B0	PDL_PFC_D_23	PORT D23
PDL_PFC_B_1	PORT B1	PDL_PFC_D_24	PORT D24
PDL_PFC_B_2	PORT B2	PDL_PFC_D_25	PORT D25
PDL_PFC_B_3	PORT B3	PDL_PFC_D_26	PORT D26
PDL_PFC_B_4	PORT B4	PDL_PFC_D_27	PORT D27
PDL_PFC_B_5	PORT B5	PDL_PFC_D_28	PORT D28
PDL_PFC_B_6	PORT B6	PDL_PFC_D_29	PORT D29
PDL_PFC_B_7	PORT B7	PDL_PFC_D_30	PORT D30
PDL_PFC_B_8	PORT B8	PDL_PFC_D_31	PORT D31

PDL_PFC_B_9	PORT B9	PDL_PFC_E_0	PORT E0
PDL_PFC_B_10	PORT B10	PDL_PFC_E_1	PORT E1
PDL_PFC_B_11	PORT B11	PDL_PFC_E_2	PORT E2
PDL_PFC_B_12	PORT B12	PDL_PFC_E_3	PORT E3
PDL_PFC_B_13	PORT B13	PDL_PFC_E_4	PORT E4
PDL_PFC_B_14	PORT B14	PDL_PFC_E_5	PORT E5
PDL_PFC_B_15	PORT B15	PDL_PFC_E_6	PORT E6
PDL_PFC_C_0	PORT C0	PDL_PFC_E_7	PORT E7
PDL_PFC_C_1	PORT C1	PDL_PFC_E_8	PORT E8
PDL_PFC_C_2	PORT C2	PDL_PFC_E_9	PORT E9
PDL_PFC_C_3	PORT C3	PDL_PFC_E_10	PORT E10
PDL_PFC_C_4	PORT C4	PDL_PFC_E_11	PORT E11
PDL_PFC_C_5	PORT C5	PDL_PFC_E_12	PORT E12
PDL_PFC_C_6	PORT C6	PDL_PFC_E_13	PORT E13
PDL_PFC_C_7	PORT C7	PDL_PFC_E_14	PORT E14
PDL_PFC_C_8	PORT C8	PDL_PFC_E_15	PORT E15
PDL_PFC_C_9	PORT C9		
PDL_PFC_C_10	PORT C10		
PDL_PFC_C_11	PORT C11		
PDL_PFC_C_12	PORT C12		
PDL_PFC_C_13	PORT C13		
PDL_PFC_C_14	PORT C14		
PDL_PFC_C_15	PORT C15		

[data2]
The value to be written to control register; Between 0x0 and 0xFFFF for a port and 0 to 7 for a pin.

Return value

True if a valid register is specified; otherwise false.

Functionality

Modify value of entire PFC control register.

References

R_PFC_Modify

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_pfc_library_SH7216.h”

void func( void )
{
  /*  write to control bits corresponding to Port C pin 15 */
  R_PFC_Write (
        PDL_PFC_C_15,
        6
         );

  /* Write data to PFC register control bits corresponding to Port A control register 3 */
  R_PFC_Write (
        PDL_PFC_PACRL3,
        0x42
         );
}
```

2) R_PFC_Read

Synopsis

Read the pin function control bits for a port pin or an entire register.

Prototype

```
bool R_PFC_Read (
    uint32_t data1,      // Port number or port control register
    uint16_t* data2,     // Pointer to store read data
);
```

Description (1/2)

Set the pin function control value.

[data1]

Any of the following definition values to select the I/O port control registers

PDL_PFC_PACRH2	PDL_PFC_PBCRL4	PDL_PFC_PCCRL2	PDL_PFC_PDCRL3
PDL_PFC_PACRH1	PDL_PFC_PBCRL3	PDL_PFC_PCCRL1	PDL_PFC_PDCRL2
PDL_PFC_PACRL4	PDL_PFC_PBCRL2	PDL_PFC_PDCRH4	PDL_PFC_PDCRL1
PDL_PFC_PACRL3	PDL_PFC_PBCRL1	PDL_PFC_PDCRH3	PDL_PFC_PECRL4
PDL_PFC_PACRL2	PDL_PFC_PCCRL4	PDL_PFC_PDCRH2	PDL_PFC_PECRL3
PDL_PFC_PACRL1	PDL_PFC_PCCRL3	PDL_PFC_PDCRH1	PDL_PFC_PECRL2
		PDL_PFC_PDCRL4	PDL_PFC_PECRL1

Any of the following definition values to select the I/O port pin.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_PFC_A_0	PORT A0	PDL_PFC_D_0	PORT D0
PDL_PFC_A_1	PORT A1	PDL_PFC_D_1	PORT D1
PDL_PFC_A_2	PORT A2	PDL_PFC_D_2	PORT D2
PDL_PFC_A_3	PORT A3	PDL_PFC_D_3	PORT D3
PDL_PFC_A_4	PORT A4	PDL_PFC_D_4	PORT D4
PDL_PFC_A_5	PORT A5	PDL_PFC_D_5	PORT D5
PDL_PFC_A_6	PORT A6	PDL_PFC_D_6	PORT D6
PDL_PFC_A_7	PORT A7	PDL_PFC_D_7	PORT D7
PDL_PFC_A_8	PORT A8	PDL_PFC_D_8	PORT D8
PDL_PFC_A_9	PORT A9	PDL_PFC_D_9	PORT D9
PDL_PFC_A_10	PORT A10	PDL_PFC_D_10	PORT D10
PDL_PFC_A_11	PORT A11	PDL_PFC_D_11	PORT D11
PDL_PFC_A_12	PORT A12	PDL_PFC_D_12	PORT D12
PDL_PFC_A_13	PORT A13	PDL_PFC_D_13	PORT D13
PDL_PFC_A_14	PORT A14	PDL_PFC_D_14	PORT D14
PDL_PFC_A_15	PORT A15	PDL_PFC_D_15	PORT D15
PDL_PFC_A_16	PORT A16	PDL_PFC_D_16	PORT D16
PDL_PFC_A_17	PORT A17	PDL_PFC_D_17	PORT D17
PDL_PFC_A_18	PORT A18	PDL_PFC_D_18	PORT D18
PDL_PFC_A_19	PORT A19	PDL_PFC_D_19	PORT D19
PDL_PFC_A_20	PORT A20	PDL_PFC_D_20	PORT D20
PDL_PFC_A_21	PORT A21	PDL_PFC_D_21	PORT D21
PDL_PFC_A_22	PORT D7	PDL_PFC_D_22	PORT D22
PDL_PFC_B_0	PORT B0	PDL_PFC_D_23	PORT D23
PDL_PFC_B_1	PORT B1	PDL_PFC_D_24	PORT D24
PDL_PFC_B_2	PORT B2	PDL_PFC_D_25	PORT D25
PDL_PFC_B_3	PORT B3	PDL_PFC_D_26	PORT D26
PDL_PFC_B_4	PORT B4	PDL_PFC_D_27	PORT D27
PDL_PFC_B_5	PORT B5	PDL_PFC_D_28	PORT D28
PDL_PFC_B_6	PORT B6	PDL_PFC_D_29	PORT D29
PDL_PFC_B_7	PORT B7	PDL_PFC_D_30	PORT D30
PDL_PFC_B_8	PORT B8	PDL_PFC_D_31	PORT D31
PDL_PFC_B_9	PORT B9	PDL_PFC_E_0	PORT E0
PDL_PFC_B_10	PORT B10	PDL_PFC_E_1	PORT E1

PDL_PFC_B_11	PORT B11	PDL_PFC_E_2	PORT E2
PDL_PFC_B_12	PORT B12	PDL_PFC_E_3	PORT E3
PDL_PFC_B_13	PORT B13	PDL_PFC_E_4	PORT E4
PDL_PFC_B_14	PORT B14	PDL_PFC_E_5	PORT E5
PDL_PFC_B_15	PORT B15	PDL_PFC_E_6	PORT E6
PDL_PFC_C_0	PORT C0	PDL_PFC_E_7	PORT E7
PDL_PFC_C_1	PORT C1	PDL_PFC_E_8	PORT E8
PDL_PFC_C_2	PORT C2	PDL_PFC_E_9	PORT E9
PDL_PFC_C_3	PORT C3	PDL_PFC_E_10	PORT E10
PDL_PFC_C_4	PORT C4	PDL_PFC_E_11	PORT E11
PDL_PFC_C_5	PORT C5	PDL_PFC_E_12	PORT E12
PDL_PFC_C_6	PORT C6	PDL_PFC_E_13	PORT E13
PDL_PFC_C_7	PORT C7	PDL_PFC_E_14	PORT E14
PDL_PFC_C_8	PORT C8	PDL_PFC_E_15	PORT E15
PDL_PFC_C_9	PORT C9		
PDL_PFC_C_10	PORT C10		
PDL_PFC_C_11	PORT C11		
PDL_PFC_C_12	PORT C12		
PDL_PFC_C_13	PORT C13		
PDL_PFC_C_14	PORT C14		
PDL_PFC_C_15	PORT C15		

[data2]
Pointer to location where value read from pin function control register is to be stored.
The value read from control register is between 0x0 and 0xFFFF for a port and 0 to 7 for a pin.

Return value

True if a valid register is specified; otherwise false.

Functionality

Modify value of entire PFC control register.

References

R_PFC_Write

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_pfc_library_SH7216.h”

void func( void )
{
  uint16_t data = 0;

  /* Read Port D control Register L1 */
  R_PFC_Read(
        PDL_PFC_PDCRL1,
        &data
         );

  /* Read data from PFC register control bits corresponding to Port D pin number 4 */
  R_PFC_Read(
        PDL_PFC_D_4,
        &data
         );

}
```

3) R_PFC_Modify

Synopsis

Read the pin function control bits for a port pin or an entire register.

Prototype

```
bool R_PFC_Modify (
    uint32_t data1,     // Port number or port control register
    uint8_t data2,      // Operation to be performed
    uint8_t data3       // Modifier value
);
```

Description (1/2)

Set the pin function control value.

[data1]

Any of the following definition values to select the I/O port control registers

PDL_PFC_PACRH2	PDL_PFC_PBCRL4	PDL_PFC_PCCRL2	PDL_PFC_PDCRL3
PDL_PFC_PACRH1	PDL_PFC_PBCRL3	PDL_PFC_PCCRL1	PDL_PFC_PDCRL2
PDL_PFC_PACRL4	PDL_PFC_PBCRL2	PDL_PFC_PDCRH4	PDL_PFC_PDCRL1
PDL_PFC_PACRL3	PDL_PFC_PBCRL1	PDL_PFC_PDCRH3	PDL_PFC_PECRL4
PDL_PFC_PACRL2	PDL_PFC_PCCRL4	PDL_PFC_PDCRH2	PDL_PFC_PECRL3
PDL_PFC_PACRL1	PDL_PFC_PCCRL3	PDL_PFC_PDCRH1	PDL_PFC_PECRL2
		PDL_PFC_PDCRL4	PDL_PFC_PECRL1

Any of the following definition values to select the I/O port pin.

Format of table: Definition used, Register referenced, Definition used, Register referenced

PDL_PFC_A_0	PORT A0	PDL_PFC_D_0	PORT D0
PDL_PFC_A_1	PORT A1	PDL_PFC_D_1	PORT D1
PDL_PFC_A_2	PORT A2	PDL_PFC_D_2	PORT D2
PDL_PFC_A_3	PORT A3	PDL_PFC_D_3	PORT D3
PDL_PFC_A_4	PORT A4	PDL_PFC_D_4	PORT D4
PDL_PFC_A_5	PORT A5	PDL_PFC_D_5	PORT D5
PDL_PFC_A_6	PORT A6	PDL_PFC_D_6	PORT D6
PDL_PFC_A_7	PORT A7	PDL_PFC_D_7	PORT D7
PDL_PFC_A_8	PORT A8	PDL_PFC_D_8	PORT D8
PDL_PFC_A_9	PORT A9	PDL_PFC_D_9	PORT D9
PDL_PFC_A_10	PORT A10	PDL_PFC_D_10	PORT D10
PDL_PFC_A_11	PORT A11	PDL_PFC_D_11	PORT D11
PDL_PFC_A_12	PORT A12	PDL_PFC_D_12	PORT D12
PDL_PFC_A_13	PORT A13	PDL_PFC_D_13	PORT D13
PDL_PFC_A_14	PORT A14	PDL_PFC_D_14	PORT D14
PDL_PFC_A_15	PORT A15	PDL_PFC_D_15	PORT D15
PDL_PFC_A_16	PORT A16	PDL_PFC_D_16	PORT D16
PDL_PFC_A_17	PORT A17	PDL_PFC_D_17	PORT D17
PDL_PFC_A_18	PORT A18	PDL_PFC_D_18	PORT D18
PDL_PFC_A_19	PORT A19	PDL_PFC_D_19	PORT D19
PDL_PFC_A_20	PORT A20	PDL_PFC_D_20	PORT D20
PDL_PFC_A_21	PORT A21	PDL_PFC_D_21	PORT D21
PDL_PFC_A_22	PORT D7	PDL_PFC_D_22	PORT D22
PDL_PFC_B_0	PORT B0	PDL_PFC_D_23	PORT D23
PDL_PFC_B_1	PORT B1	PDL_PFC_D_24	PORT D24
PDL_PFC_B_2	PORT B2	PDL_PFC_D_25	PORT D25
PDL_PFC_B_3	PORT B3	PDL_PFC_D_26	PORT D26
PDL_PFC_B_4	PORT B4	PDL_PFC_D_27	PORT D27
PDL_PFC_B_5	PORT B5	PDL_PFC_D_28	PORT D28
PDL_PFC_B_6	PORT B6	PDL_PFC_D_29	PORT D29
PDL_PFC_B_7	PORT B7	PDL_PFC_D_30	PORT D30
PDL_PFC_B_8	PORT B8	PDL_PFC_D_31	PORT D31
PDL_PFC_B_9	PORT B9	PDL_PFC_E_0	PORT E0

PDL_PFC_B_10	PORT B10	PDL_PFC_E_1	PORT E1
PDL_PFC_B_11	PORT B11	PDL_PFC_E_2	PORT E2
PDL_PFC_B_12	PORT B12	PDL_PFC_E_3	PORT E3
PDL_PFC_B_13	PORT B13	PDL_PFC_E_4	PORT E4
PDL_PFC_B_14	PORT B14	PDL_PFC_E_5	PORT E5
PDL_PFC_B_15	PORT B15	PDL_PFC_E_6	PORT E6
PDL_PFC_C_0	PORT C0	PDL_PFC_E_7	PORT E7
PDL_PFC_C_1	PORT C1	PDL_PFC_E_8	PORT E8
PDL_PFC_C_2	PORT C2	PDL_PFC_E_9	PORT E9
PDL_PFC_C_3	PORT C3	PDL_PFC_E_10	PORT E10
PDL_PFC_C_4	PORT C4	PDL_PFC_E_11	PORT E11
PDL_PFC_C_5	PORT C5	PDL_PFC_E_12	PORT E12
PDL_PFC_C_6	PORT C6	PDL_PFC_E_13	PORT E13
PDL_PFC_C_7	PORT C7	PDL_PFC_E_14	PORT E14
PDL_PFC_C_8	PORT C8	PDL_PFC_E_15	PORT E15
PDL_PFC_C_9	PORT C9		
PDL_PFC_C_10	PORT C10		
PDL_PFC_C_11	PORT C11		
PDL_PFC_C_12	PORT C12		
PDL_PFC_C_13	PORT C13		
PDL_PFC_C_14	PORT C14		
PDL_PFC_C_15	PORT C15		

[data2]

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

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

[data3]

The value to be used for the modification; Between 0x0 and 0xFFFF for a port register and from 0 to 7 for a pin.

Return value

True if a valid register is specified; otherwise false.

Functionality

Modify value of entire PFC control register.

References

R_PFC_Write

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_pfc_library_SH7216.h”

void func( void )
{
  /* Modify PFC register control bits corresponding to Port C pin number 3 */
  R_PFC_Modify (
        PDL_PFC_PBCRL4,
        0x2300,
        PDL_PFC_AND
         );

  /* Modify PFC register control bits corresponding to Port C pin number 3 */
  R_PFC_Modify (
        PDL_PFC_C_3,
        0x3,
        PDL_PFC_OR
         );
}
```

4.2.4. Bus State Controller

1) R_BSC_Create

Synopsis

Configure the external bus controller.

Prototype

```
bool R_BSC_Create(
        uint32_t data1,     // Chip select Configuration
        uint32_t data2,     // Address Output Configuration
        uint32_t data3,     // Signals Configuration
        Unit32_t data4,     // Error Detection options
        Unit8_t data5,      // Error Detection timeout counter value
        void * func,        // Callback function to be called when a bus error occurs
        uint8_t data6       // Interrupt priority level
);
```

Description (1/2)

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

Control the external bus controller.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

[data1]

- Chip select selection (only required for each external memory area that is enabled).

PDL_BSC_CS0_A0 PDL_BSC_CS0_A12 PDL_BSC_CS0_B10 PDL_BSC_CS1_A11 PDL_BSC_CS1_A1 PDL_BSC_CS1_B11 PDL_BSC_CS2_A10 PDL_BSC_CS2_A2 PDL_BSC_CS2_B10 PDL_BSC_CS2_B8 PDL_BSC_CS3_A9 PDL_BSC_CS3_A3 PDL_BSC_CS3_B11 PDL_BSC_CS3_B9 PDL_BSC_CS4_A8 PDL_BSC_CS4_A4 PDL_BSC_CS5_A7 PDL_BSC_CS5_A5 PDL_BSC_CS6_A6 PDL_BSC_CS6_B10 PDL_BSC_CS7_B11	Select the Chip Select that need to be configured.

[data2]

- Address output control. The signals are **enabled** by default. Specify 0 for no change.
- In a cell with multiple options specified, the options are listed in the order of priority. That is, if a disable option at the top is not selected then that option is configured and options below it in the cell are ignored.
- So in order to configure an option in row 3 of a cell, the disable options in rows 1 & 2 of the cell have to be selected with a bitwise OR operator separator.

PDL_BSC_A0_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A1_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A2_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A3_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A4_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A5_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A6_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A7_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A8_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A9_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A10_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A11_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A12_DISABLE	Disable the output of the BSC Address signal.

PDL_BSC_A13_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A14_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A15_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A16_DISABLE PDL_BSC_RD_WR_DISABLE	Disable the output of the BSC Address signal Or Disable the Read/Write output of the BSC SDRAM signal.
PDL_BSC_A17_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A18_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A19_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A20_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A21_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A22_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A23_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A24_DISABLE	Disable the output of the BSC Address signal.
PDL_BSC_A25_DISABLE	Disable the output of the A23 signal.

[data3]

- Signals control. The signals are **enabled** by default. Specify 0 for no change.
- In a cell with multiple options specified, the options are listed in the order of priority. That is, if a disable option at the top is not selected then that option is configured and options below it in the cell are ignored.
- So in order to configure an option in row 3 of a cell, the disable options in rows 1 & 2 of the cell have to be selected with a bitwise OR operator separator.

PDL_BSC_BS_DISABLE PDL_BSC_WAIT_DISABLE PDL_BSC_RASU_DISABLE	Disable the bus Select signal or Disable the External Wait input pin or Disable the RASU signals Pin for SDRAM..
PDL_BSC_RD_DISABLE PDL_BSC_BACK_DISABLE PDL_BSC_CKE_DISABLE	Disable the memory Read strobe signals for PCMCIA. or Disable the Bus enable output pin or Disable the CKE signals Pin for SDRAM.
PDL_BSC_WRHH_DQMUU_WE_DISABLE PDL_BSC_RASL_DISABLE	Disable the WRHH / DQMUU /WE signals or Disable the RASL signals Pin for SDRAM
PDL_BSC_WRHL_DQMUL_DISABLE PDL_BSC_CASL_DISABLE	Disable the WRHL / DQMUL /WE signals or Disable the CASL signals Pin for SDRAM.
PDL_BSC_CASU_DISABLE PDL_BSC_BREQ_DISABLE	Disable the CASU signals Pin for SDRAM Or Disable the Bus request output pin.
PDL_BSC_ICIORD_DISABLE	Disable the IO read cycle's strobe signals for PCMCIA.
PDL_BSC_ICIOWR_DISABLE	Disable the IO Write cycles strobe signals for PCMCIA.
PDL_BSC_WRH_DQMLU_DISABLE	Disable the WRH / DQMLU signals.
PDL_BSC_WRL_DQMLL_DISABLE	Disable the WRL/ DQMLL signals.

[data4]

- Select the SDRAM Refresh clock and Refresh Time

PDL_BSC_REFRESH_1_TIME or PDL_BSC_REFRESH_2_TIME or PDL_BSC_REFRESH_4_TIME or PDL_BSC_REFRESH_6_TIME or PDL_BSC_REFRESH_8_TIME	Select the Number of continuous refresh cycle for SDRAM
PDL_BSC_REFRESH_INT_ENABLE or PDL_BSC_REFRESH_INT_DISABLE	Enable or Disable the Compare match interrupt for SDRAM Refresh.

PDL_BSC_REFRESH_STOP or PDL_BSC_REFRESH_DIV_4_CLOCK or PDL_BSC_REFRESH_DIV_16_CLOCK or PDL_BSC_REFRESH_DIV_64_CLOCK or PDL_BSC_REFRESH_DIV_256_CLOCK or PDL_BSC_REFRESH_DIV_1024_CLOCK or PDL_BSC_REFRESH_DIV_2048_CLOCK or PDL_BSC_REFRESH_DIV_4096_CLOCK	Select the clock divider for the SDRAM Refresh control.

[data5]
- Select the Refresh counter timeout.

[func]
The function to be called when a bus error occurs. Specify PDL_NA if not required.

[data6]
The interrupt priority level. Select between 0 (interrupt disabled) and 15 (highest priority).

Return value

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

Category

Bus Controller

Reference

R_BSC_Destroy

Remarks

- The external bus is enabled by this function.
- Call this function before using function R_BSC_Control
- 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

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

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

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

void func(void)
{
	/* create a space with SDRAM control register and SDRAM refresh rate control */
	R_BSC_CreateAll(0,0,0,
		PDL_BSC_REFRESH_DIV_4_CLOCK | PDL_BSC_REFRESH_COUNTER
		|PDL_BSC_REFRESH_1_TIME,
			0xa55a0046ul,
			BusErrorFunc,
			2
			);
}
```

2) R_BSC_CreateArea

Synopsis

Configure an external bus area.

Prototype

```
bool R_BSC_CreateArea(
        uint8_t data1,      // Type of memory to be configured for the area
        uint8_t data2,      // External memory area CSn
        unit32_t data3,     // SDRAM configuration
        unit32_t data4,     // BUS configuration
        uint8_t data5,      // BUS Write control register configuration
        uint8_t data6,      // SW cycles
        uint8_t data7,      // WR cycles
        uint8_t data8,      // HW cycles
        uint8_t data9,      // BW cycles
        uint8_t data10,     // W cycles
        uint8_t data11,     // WW cycles
        uint8_t data12,     // A2CL cycles
        uint8_t data13,     // WTRP cycles
        uint8_t data14,     // WTRCD cycles
        uint8_t data15,     // A3CL cycles
        uint8_t data16,     // TRWL cycles
        uint8_t data17,     // WTRC cycles
        uint8_t data18,     // IWW cycles
        uint8_t data19,     // WRWD cycles
        uint8_t data20,     // IWRWS cycles
        uint8_t data21,     // IWRRD cycles
        uint8_t data22,     // IWRRS cycles
        uint8_t data23,     // DMAIW cycles
);
```

Description (1/2)

Set up an external bus area.

[data1]

- Select type of memory that need to be configured for the given chip select

PDL_BSC_TYPE_NORMAL or PDL_BSC_TYPE_BROMA PDL_BSC_TYPE_MPXIO PDL_BSC_TYPE_SRAM PDL_BSC_TYPE_SDRAM PDL_BSC_TYPE_PCMCIA PDL_BSC_TYPE_BROMS	Select type of memory that need to be configured

[data2]

The address area n (where n = 0 to 7).

[data2]

Configure the data for SDRAM memory controls.
If multiple selections are required, use "|" to separate each selection.
The default settings are shown in **bold**.

- Select the number of bits of ROW Address for Area 2

PDL_BSC_A2ROW_11 or PDL_BSC_A2ROW_12 or PDL_BSC_A2ROW_13	Select the Number of Bit use to select ROW Address for Area 2

- Select the number of bits of COL Address for Area 2

PDL_BSC_A2COL_8 or PDL_BSC_A2COL_9 or PDL_BSC_A2COL_10	Select the Number of Bit use to select COL Address for Area 2

- Select the number of bits of ROW Address for Area 3

PDL_BSC_A3ROW_11 or PDL_BSC_A3ROW_12 or PDL_BSC_A3ROW_13	Select the Number of Bit use to select ROW Address for Area 3

- Select the number of bits of COL Address for Area 3

PDL_BSC_A3COL_8 or PDL_BSC_A3COL_9 or PDL_BSC_A3COL_10	Select the Number of Bit use to select COL Address for Area 3

- Deep Power Down mode

PDL_BSC_DEEP_ENABLE or PDL_BSC_DEEP_DISABLE	Enable or disable the Deep Power Down mode

- Low Frequency Mode

PDL_BSC_SLOW_DISABLE or PDL_BSC_SLOW_ENABLE	Enable or disable the LOW Frequency Mode

- Refresh Mode

PDL_BSC_RFSH_DISABLE or PDL_BSC_RFSH_ENABLE	Enable or disable the Refresh mode

- Refresh Type

PDL_BSC_RMODE_AUTO or PDL_BSC_RMODE_SELF	Select the Type of refresh operation to be performed

- Power Down mode

PDL_BSC_PDOWN_DISBLE or PDL_BSC_PDOWN_ENABLE	Enable or disable the Power Down mode

- Bank Active mode

PDL_BSC_BACTV_DISBLE or PDL_BSC_BACTV_ENABLE	Enable or disable the BANK Active Mode

[data4]
Configure the BUS options

- ENDIANESS Select

PDL_BSC_BIG_ENDIAN or PDL_BSC_LITTLE_ENDIAN	Select the type of Endianess for the BUS

- IWW Cycles

data21 values	Configure the data21 value for IWW cycles

- IWRWD Cycles

data22 values	Configure the data22 value for IWRWD cycles

- IWRWS Cycles

data23 values	Configure the data23 value for IWRWS cycles

- IWRRD Cycles

data24 values	Configure the data24 value for IWRRD cycles

- IWRRS Cycles

data25 values	Configure the data25 value for IWRRS cycles

- DMAIW Cycles

data26 values	Configure the data26 value for DMAIW cycles

[data5]
Select the parameter that needs to be configured for the WCR register.

- Wait Cycles configuration

PDL_BSC_SW	configure sw cycles
PDL_BSC_WR	configure wr cycles
PDL_BSC_HW	configure hw cycles
PDL_BSC_WW	configure ww cycles
PDL_BSC_A2CL	configure a2cl cycles
PDL_BSC_WTRP	configure wtrp cycles
PDL_BSC_WTRCD	configure wtrcd cycles
PDL_BSC_A3CL	configure a3cl cycles
PDL_BSC_TRWL	configure trwl cycles
PDL_BSC_WTRC	configure wtrc cycles
PDL_BSC_TED	configure ted cycles
PDL_BSC_PCW	configure pcw cycles
PDL_BSC_TEH	configure teh cycles

- Write Mode Enable

PDL_BSC_WAIT_DISABLE or PDL_BSC_WAIT_ENABLE	Enable or Disable Write Mode

- Bus With Specification

PDL_BSC_SZSEL_A14 or PDL_BSC_SZSEL_A15	Set the BUS Width Specification to either 14 or 15 bits

- Burst Count Specification

PDL_BSC_BST_16X1 or PDL_BSC_BST_4X4	Set Burst count specification to either 16x1 type or 4x4 type if bus width is 168 bits
PDL_BSC_BST_8X1 or PDL_BSC_BST_2x4 or PDL_BSC_BST_242	Set Burst count specification to either 8x1 type or 22*4 or 2-4-2 type if bus width is 8 bits

- Space attribute Specification

PDL_BSC_SA1_MEM or PDL_BSC_SA1_IO	Select the space attribute specification for A25=1
PDL_BSC_SA0_MEM or PDL_BSC_SA0_IO	Select the space attribute specification for A25=0

[data6]
The Number of Delay Cycles from Address, CS0 Assertion to RD, WEn Assertionn (SW). Valid between 0 and 3.

[data7]
The number of cycles that are necessary for read/write access (WR). Valid between 0 and 12.

[data8]
The number of delay cycles from RD and WEn negation to address and CS0 negation (HW). Valid between 0 and 3.

[data9]
The number of wait cycles to be inserted between the second or subsequent access cycles in burst access (BW). Valid between 0 and 3.

[data10]
The number of wait cycles to be inserted in the first access cycle (W). Valid between 0 and 12.

[data11]
The number of cycles that are necessary for write access (WW). Valid between 0 and 6.

[data12]
Specify the CAS latency for area 2 (A2CL). Valid between 0 and 3.

[data13]
The number of minimum precharge completion wait cycles (WTRP). Valid between 0 and 3.

[data14]

The minimum number of wait cycles from issuing the ACTV command to issuing the READ (A)/WRIT (A) command (WTRCD). Valid between 0 and 3.

[data15]
Specify the CAS latency for area 3 (A3CL). Valid between 0 and 3.

[data16]
The number of minimum auto-precharge startup wait cycles (TRWL). Valid between 0 and 3.

[data17]
Number of Idle Cycles from REF Command/Self-Refresh Release to ACTV/REF/MRS Command (WTRC). Valid between 0 and 3.

[data18]
The number of idle cycles to be inserted after the access to a memory that is connected to the space. (IWW). Valid between 0 and 7.

[data19]
The number of idle cycles to be inserted after the access to a memory that is connected to the space (IWRWD). Valid between 0 and 7.

[data20]
The number of idle cycles to be inserted after the access to a memory that is connected to the space (IWRWS). Valid between 0 and 7.

[data21]
The number of idle cycles to be inserted after the access to a memory that is connected to the space (IWRRD). Valid between 0 and 7.

[data22]
The number of idle cycles to be inserted after the access to a memory that is connected to the space (IWRRS). Valid between 0 and 7.

[data23]
The number of idle cycles to be inserted after an access to an external device with DACK when DMA single address transfer is performed (IDMAIW). Valid between 0 and 7.

Return value	True if all parameters are valid and exclusive; otherwise false.
Category	Bus Controller
Reference	R_BSC_Destroy, R_BSC_GetStatus
Remarks	• Ensure that function R_BSC_Create is called once before using this function. • The endian mode of the CPU is selected by the MDE pin (low = little endian; high = big endian). • The cycle count parameters are not checked for validity. • Port Function Control registers PFCR0 and PFCR5 are modified by this function.

Program example

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

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

void func(void)
{
    /* Configure the CS0 Space for NORMAL module. 16 bit bus width, Big
    Endian, and delay and wait timings */

    R_BSC_CreateArea(
                PDL_BSC_TYPE_NORMAL,0,0,
        PDL_BSC_WIDTH_16|PDL_BSC_BIG_ENDIAN|PDL_BSC_IWW,
        PDL_BSC_BAS_DISABLE|PDL_BSC_SW|PDL_BSC_WR| PDL_BSC_HW|
        PDL_BSC_WAIT_ENABLE,
        1,6,1,0,0,0,0,0,0,0,0,0,
                1,0,0,0,0,0);
}
```

3) R_BSC_Destroy

Synopsis	Stop the External Bus Controller.
Prototype	**bool R_BSC_Destroy(** void **);**
Description	Disable an external bus area.
Return value	True.
Category	Bus Controller
Reference	R_BSC_Create
Remarks	• None.

Program example	#include “r_pdl_bsc.h”
	
	void func(void)
	{
	/* Disable an external bus area */
	R_BSC_Destroy();
	}

4) R_BSC_Control

Synopsis	Modify the External Bus Controller operation.
Prototype	**bool R_BSC_Control(** **void** **);**
Description	Provide interrupt flag clearing
Return value	True if all parameters are valid; otherwise false.
Category	Bus Controller
Reference	R_BSC_Create, R_BSC_CreateArea
Remarks	• This function can be called from the error handling function
Program example	

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

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

void func(void)
{
        /* Clear the timer flag */
        R_BSC_Control();
}
```

5) R_BSC_GetStatus

Synopsis

Read the External Bus Controller status flags.

Prototype

```
bool R_BSC_GetStatus(
        uint8_t * data      // Flag store pointer
);
```

Description

Read the interrupt status

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

b7 – b1	b0
-	0: Idle 1: Bus error condition detected

Return value

True.

Category

Bus Controller

Reference

R_BSC_CreateR_BSC_CreateArea

Remarks

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

Program example

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

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

void func(void)
{
        uint8_t status;

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

4.2.5. DMA Controller

1) R_DMAC_Create

Synopsis

Configure the DMA controller.

Prototype

```
bool R_DMAC_Create(
        uint8_t data1,      // The DMA channel to be configured
        uint32_t data2,     // Configure the operation of channel DMAn.
        uint32_t data3,     // Configure the start trigger for channel DMAn.
        uint16_t data4,     // Trigger selection
        void * data5,       // Source start address
        void * data6,       // Destination start address
        uint32_t data7,     // Transfer byte count
        void * data8,       // Source reload address
        void * data9,       // Destination reload address
        uint32_t data10,    // Transfer byte count reload value
        void * func,        // Function called when a transfer completes
        uint8_t data11      // Interrupt priority level
);
```

Description (1/3)

Set up a DMA channel.

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

[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 system selection

PDL_DMAC_SINGLE_REQUEST or PDL_DMAC_CONSECUTIVE_REQUESTS	Single-operand, consecutive-operand

- Transfer Bus Mode

PDL_DMAC_CYCLE_STEAL_MODE or PDL_DMAC_BURST_MODE	Bus operates in cycle steal mode, Bus operates in burst mode

- Address addition direction selection

PDL_DMAC_SOURCE_ADDRESS_FIXED or	Leave the address unchanged.
PDL_DMAC_SOURCE_ADDRESS_PLUS or	Increment the address by the data size.
PDL_DMAC_SOURCE_ADDRESS_MINUS or	Decrement the address by the data size.
PDL_DMAC_DESTINATION_ADDRESS_FIXED or	Leave the address unchanged.
PDL_DMAC_DESTINATION_ADDRESS_PLUS or	Increment the address by the data size.
PDL_DMAC_DESTINATION_ADDRESS_MINUS or	Decrement the address by the data size.

- Transfer data size

PDL_DMAC_SIZE_8 or PDL_DMAC_SIZE_16 or PDL_DMAC_SIZE_32 or PDL_DMAC_SIZE_64	Select 8, 16, 32 or 64 bits for the data to be transferred.

- Half End Interrupt control

PDL_DMAC_HALF_END_INTERRUPT_DISABLE or PDL_DMAC_HALF_END_INTERRUPT_ENABLE	Disable or enable generation of interrupt on transfer of half of total data.

- end-of-transfer reload control

PDL_DMAC_RELOAD_DISABLE or PDL_DMAC_RELOAD_ENABLE	Disable or enable reloading of source & destination addresses.

- DREQ signal type

PDL_DMAC_DREQ_LOW_LEVEL or PDL_DMAC_DREQ_FALLING_EDGE or PDL_DMAC_DREQ_HIGH_LEVEL or PDL_DMAC_DREQ_RISING_EDGE	Select signal type at which DMA request is detected: low level, high level, rising edge or falling edge.

[data3]
Configure the start trigger for channel DMAn.

- Start trigger

PDL_DMAC_REQUEST_EXT_DUAL	External request, dual address mode.
PDL_DMAC_REQUEST_EXT_ADD_TO_DEV	External request, single address mode, External address space to External device with DACK
PDL_DMAC_REQUEST_EXT_DEV_TO_ADD	External request, single address mode, External device with DACK to External address space
PDL_DMAC_REQUEST_AUTO	Auto request
PDL_DMAC_REQUEST_DMA_EXTENSION	DMA extension resource selector
PDL_DMAC_REQUEST_CAN0	Controller area network, channel 0
PDL_DMAC_REQUEST_CAN1	Controller area network, channel 1

[data4]
The start address of the source.

Description (2/3)

[data5]
The start address of destination

[data6]
The number of bytes to be transferred.

[data7]
The reload address value of source. This value is ignored if the reload function is disabled.

[data8]
The reload address value of destination. This value is ignored if the reload function is disabled.

[data9]
The reload value for number of bytes. This value is ignored if the reload function is disabled.

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

[data10]
The interrupt priority level. Select between 0 (interrupt disabled) and F (highest priority).

Return value

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

Category

DMA controller

Reference

R_DMAC_Destroy, R_DMAC_Control, R_DMAC_GetStatus

Remarks

- If another peripheral will be used to trigger a DMA transfer, call this function before calling the Create function for the peripheral.
- 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

```
/* PDL definitions */
#include “r_pdl_dmac_library_SH7216.h”

void func(void)
{
      /* Configure DMA channel 2 */
      R_DMAC_Create(
                  PDL_DMAC_7,
                  (PDL_DMAC_CYCLE_STEAL_MODE | \
                  PDL_DMAC_SINGLE_REQUEST | \
                  PDL_DMAC_SOURCE_ADDRESS_FIXED | \
                  PDL_DMAC_DESTINATION_ADDRESS_FIXED | PDL_DMAC_SIZE_8 | \
                  PDL_DMAC_HALF_END_INTERRUPT_ENABLE | \
                  PDL_DMAC_RELOAD_DISABLE | \
                  PDL_DMAC_DREQ_HIGH_LEVEL),
                  (PDL_DMAC_REQUEST_AUTO | PDL_DMAC_TEND_ACTIVE_LOW | \
                  PDL_DMAC_ACK_RD_CYCLE | PDL_DMAC_ACK_ACTIVE_LOW),
                  PDL_DMAC_REQUEST_USB0_TX,
                  &source_addr,
                  &destination_addr,
                  transfer_bytes,
                   (void *)0,
                   (void *)0,
                  0,
                  PDL_NO_FUNC,
                  priority
                  );
}
```

2) R_DMAC_Destroy

Synopsis

Shutdown the DMA controller.

Prototype

```
bool R_DMAC_Destroy(
        uint8_t data1        // channel to be disabled
);
```

Description

Disable the DMAC module.

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

Return value

True if the disable operation succeeded; otherwise false.

Category

DMA controller

Reference

R_DMAC_Create, R_DMAC_Control

Remarks

- If all channels have been suspended, the DMAC module will be disabled.
- 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

```
/* PDL definitions */
#include “r_pdl_dmac_library_SH7216.h”

void func(void)
{
        /* Shutdown the DMA controller for channel 7 */
        R_DMAC_Destroy(PDL_DMAC_7);
}
```

3) R_DMAC_Control

Synopsis

Control the DMA controller.

Prototype

```
bool R_DMAC_Control (
        uint8_t data1,      // The DMA channel to be configured
        uint8_t data2,      // The changes to be made
        uint32_t data3,     // Source reload address
        uint32_t data4,     // Destination reload address
        uint32_t data5      // Transfer byte reload count
);
```

Description

Change the state of a DMA controller channel.

[data1]
Channel selection.
If multiple selections are required, use "|" to separate each selection.

PDL_DMAC_0 PDL_DMAC_1 PDL_DMAC_2 PDL_DMAC_3 … PDL_DMAC_7	The channel to be controlled.

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

- Enable / suspend control

PDL_DMAC_ENABLE or PDL_DMAC_SUSPEND or	Enable / re-enable or suspend DMA transfers on the selected channel(s).

- Software trigger control

PDL_DMAC_START	Start a DMA transfer.

- The reload registers to be modified.

PDL_DMAC_SOURCE	Update the Transfer Source Address reload register (DMRSA).
PDL_DMAC_DESTINATION	Update the Transfer Destination Address reload register (DMRDA).
PDL_DMAC_BYTES	Update the Byte Count reload register (DMRBC).

[data3]
The source-reload address value. This value is ignored if the reload function is disabled.

[data4]
The destination-reload address value. This value is ignored if the reload function is disabled.

[data5]
The number of bytes-reload value. This value is ignored if the reload function is disabled.

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.

Program example

```
/* PDL definitions */
#include “r_pdl_dmac_library_SH7216.h”

void func(void)
{
	/* Enable transfers on channel 2 */
	R_DMAC_Control(
		PDL_DMAC_0,
		(PDL_DMAC_ENABLE | PDL_DMAC_RELOAD_ENABLE),
		source_addr,
		destination_addr,
		transfer_bytes
		);

}
```

4) R_DMAC_GetStatus

Synopsis

Check the status of the DMA channel.

Prototype

```
bool R_DMAC_GetStatus(
        uint8_t data1,      // The channel selected
        uint16_t * data2,   // Status flags pointer
        uint32_t * data3,   // Current source address pointer
        uint32_t * data4,   // Current destination address pointer
        uint32_t * data5    // Current transfer byte count
);
```

Description

Return status flags and current channel registers.

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

[data2]
The status flags shall be stored in the following format:

b15 – b14	b13 – b12	b11 – b10	b9 – b8
-	Cycle Steal Mode Select	-	Priority Mode

b7 – b3	b2	b1	b0
-	AE	NMIF	DME
	0: No Address error occurred 1: Address error occurred	0 : No NMI interrupt. 1 : NMI interrupt occurred.	0: DMA transfer is disabled 1: DMA transfer is enabled

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

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

[data5]
Where the current transfer byte count shall be stored. Specify PDL_NA 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 a Transfer End Interrupt request flag is set to 1, the flag will be cleared to 0 by this function.

Program example

```
/* PDL definitions */
#include “r_pdl_dmac.h”

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

void func(void)
{
        uint32_t source_addr;
        uint32_t destination_addr;
        uint32_t transfer_bytes;
        uint16_t statusvalue;

        /* Read the status and current source address for channel 2 */
        R_DMAC_GetStatus(
                PDL_DMAC_4,
                &statusvalue,
                &source_addr,
                &destination_addr,
                transfer_bytes
                );
}
```

4.2.6. 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
    void * data3,       // Period, frequency or register data
    void * func,        // Callback function to be called at the periodic interval
    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 or 1).

[data2]
Configure the timer. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Clock calculation

PDL_CMT_PERIOD or	The parameter data3 will specify the timer period. The counter clock source and compare match value will be calculated by this function.
PDL_CMT_FREQUENCY or	The parameter data3 will specify the timer frequency. The counter clock source and compare match value will be calculated by this function.
PDL_CMT_PCLK_DIV_8 or PDL_CMT_PCLK_DIV_32 or PDL_CMT_PCLK_DIV_128 or PDL_CMT_PCLK_DIV_512	Select the internal clock signal PCLK ÷ 8, 32, 128 or 512 as the counter clock source. The parameter data3 will be the register CMCOR value.

[data3]
The data to be used for the register value calculations.

Data use	Parameter type
The timer period in seconds or	float
The timer frequency in Hz or	float
The value to be put in register CMCOR	uint16_t

[func]
The function to be called at the periodic interval. Specify PDL_NA if not required.

[data4]
The interrupt priority level. Select between 0 (interrupt disabled) and 15 (highest priority).

Return value

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

Category

Compare Match Timer

Reference

R_CMT_Destroy

Remarks (1/2)

- Function R_CPG_Set must be called before any use of this function.
- If a callback function is specified, this function will enable the relevant interrupt.
  Please see the notes on callback function use

Remarks (2/2)

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

Program example

```
/* PDL definitions */
#include “r_pdl_cmt.h”

/* PDL 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_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).
Return value	True if the unit selection is valid; otherwise false.
Category	Compare Match Timer
Reference	Compare Match Timer R_CMT_Create
Remarks	• The timer unit is put into the stop state to reduce power consumption.
Program example	

```
/* PDL definitions */
#include “r_pdl_cmt.h”

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

void func(void)
{
        /* Shutdown channels 0 and 1 */
        R_CMT_Destroy(
                0
        );
}
```

3) R_CMT_Control

Synopsis

Control CMT operation.

Prototype

```
bool R_CMT_Control(
        uint8_t data1       // Channel selection
        uint16_t data2      // Configuration selection
        void * 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.

- Counter stop / re-start

PDL_CMT_STOP or PDL_CMT_START	Disable or re-enable the counter clock source. Omit this option to leave the timer state unchanged.

- Period or frequency calculation

PDL_CMT_PERIOD or PDL_CMT_FREQUENCY or PDL_CMT_CONSTANT	The parameter data3 will contain the new period, frequency or new constant register (CMCOR) value. Omit this option to leave the timing unchanged.

[data3]
The new period, frequency or register value. This will be ignored if a timing change is not requested.

Data use	Parameter type
The timer period in seconds or	float
The timer frequency in Hz or	float
The value to be put in register CMCOR	uint16_t

Return value

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

Category

Compare Match Timer

Reference

Compare Match Timer
R_CMT_Create

Remarks

- Compare Match Timer
  R_CMT_Create must be first be used to configure the channel.

Program example

```
/* PDL definitions */
#include “r_pdl_cmt.h”

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

void func(void)
{
        /* Change channel 2 to 1ms period */
        R_CMT_Control(
                2,
                PDL_PDL_CMT_PERIOD,
                1E-3
        );
}
```

4) 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 for read status of CMT
    uint16_t * data3    // A pointer to the data storage for CMT counter value
);
```

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.

b7 – b1	b0
-	0: Idle 1: Compare match condition detected

[data3]
A pointer to where the counter value shall be stored. Specify PDL_NA if it is not required.

Return value

True if all parameters are valid; otherwise false.

Category

Compare Match Timer

Reference

Compare Match Timer
R_CMT_Create

Remarks

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

Program example

```
/* PDL definitions */
#include “r_pdl_cmt.h”

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

uint8_t Flags;
uint16_t Counter;

void func(void)
{
    /* Change the channel 2 values */
    R_CMT_Read(
        2,
        &Flags,
        &Counter
    );
}
```

5) R_CMT_CreateOneShot

Synopsis

Supports one shot function using CMT.

Prototype

```
bool R_CMT_CreateOneShot (
    uint8_t data1,      // Timer channel selection
    float data3,        // Time Period
    void * func,        // Callback function when compare match interrupt triggers
    uint8_t data4       // Interrupt priority level
);
```

Description

Set up a Compare Match Timer channel and use it in one shot mode.

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

[data2]
The timer period in seconds. This data is to be used for the register value calculations.

[func]
The function to be called at compare match interrupt.

[data3]
The interrupt priority level. Select between 0 (interrupt disabled) and 15 (highest priority).

Return value

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

Category

Compare Match Timer

Reference

R_CMT_Destroy

Remarks (1/2)

- Function R_CPG_Set must be called before any use of this function.
- If a callback function is specified, this function will enable the relevant interrupt. Please see the notes on callback function use

Remarks (2/2)

- 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 interrupt service routine clears interrupt request flag & stops the CMT timer.

Program example

```
/* PDL definitions */
#include “r_pdl_cmt.h”

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

Callback_func()
{
   nop();
}
void func(void)
{
        /* Configure CMT channel 0 for 1ms operation */
        R_CMT_CreateOneShot(
                0,
                0.001,
                Callback_func,
                2
        );
}
```

4.2.7. Serial Communication Interface

1) R_SCI_Create

Synopsis

SCI channel setup.

Prototype

```
bool R_SCI_Create(
    uint8_t data1,      // Channel selection
    uint32_t data2,     // Channel pin select
    uint32_t data3,     // Channel configuration
    uint8_t data4,      // Bit rate or register value
    uint8_t data5,      // Interrupt priority level

);
```

Description (1/3)

Set up the selected SCI channel.

[data1]
Select channel SCIn (where n = 0,1,2,4).

[data2]
Configure the channel pin.

- Operation mode

PDL_SCI_TXD0_A10 PDL_SCI_TXD0_C9 PDL_SCI_TXD0_C11 PDL_SCI_TXD0_A1 PDL_SCI_TXD0_B6 PDL_SCI_TXD1_A20 PDL_SCI_TXD1_A7 PDL_SCI_TXD1_A4 PDL_SCI_TXD2_E10 PDL_SCI_TXD2_D3 PDL_SCI_TXD2_B11 PDL_SCI_TXD4_B9 PDL_SCI_RXD0_A11 PDL_SCI_RXD0_C8 PDL_SCI_RXD0_C10 PDL_SCI_RXD0_A0 PDL_SCI_RXD0_B5 PDL_SCI_RXD1_A3 PDL_SCI_RXD1_A8 PDL_SCI_RXD1_A19 PDL_SCI_RXD2_B10 PDL_SCI_RXD2_D2 PDL_SCI_RXD2_E7 PDL_SCI_RXD4_B8 PDL_SCI_SCK0_A9 PDL_SCI_SCK0_A2 PDL_SCI_SCK1_A21 PDL_SCI_SCK1_A6 PDL_SCI_SCK1_A5 PDL_SCI_SCK2_E8 PDL_SCI_SCK2_D4 PDL_SCI_SCK4_B7	Choose pin for TXD, RXD, SCK signal.

[data3]
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	Choose between Asynchronous, Clock synchronous operation.

- Data transfer format

PDL_SCI_LSB_FIRST or PDL_SCI_MSB_FIRST	Select least- or most-significant bit first. In 7-bit mode the format is fixed to LSB first.

Options which only apply to Asynchronous mode

- Data clock source selection

PDL_SCI_CLK_INT_IO or	Select the on-chip baud rate generator.	The SCKn pin functions as an I/O pin.
PDL_SCI_CLK_INT_OUT or		The SCKn pin outputs the bit clock.
PDL_SCI_CLK_EXT_DIV_16	Input a clock of 16 times the desired bit rate to the SCKn pin.	
PDL_SCI_CLK_TMR	Select Timer output TMO0.	

- Data length

PDL_SCI_8_BIT_LENGTH or PDL_SCI_7_BIT_LENGTH	8- or 7-bit data length.

Description (2/3)

- 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.

- 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 only apply to Clock Synchronous 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.
PDL_SCI_CLK_EXT	Input the clock to the SCKn pin.

[data4]
This parameter is ignored if the on-chip baud rate generator is not selected.
The format can be either:

- The desired bit rate in bits per second.
  The clock source and division values will be calculated and set by this function.

Or one selection from each of the following, using “|” to separate each selection.

- CKS selection

PDL_SCI_PCLK_DIV_1 or PDL_SCI_PCLK_DIV_4 or PDL_SCI_PCLK_DIV_16 or PDL_SCI_PCLK_DIV_64	Select the internal clock signal PCLK ÷ 1, 4, 16 or 64 as the baud rate generator clock source.

- The BRR register value.

[data5]
The interrupt priority level. Select between 0 (interrupt disabled) and 15 (highest priority).

Return value

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

Category

SCI

Reference

R_SCI_Destroy, R_SCI_Stop

Remarks

- Function R_CPG_Set must be called before any use of this function.
- This function configures each SCI pin that is required for operation. It also disables the alternative modes on those pins.
- The wait time of 1 data bit period that is required during configuration is handled within this function.

Program example

```
/* PDL definitions */
#include “r_pdl_sci.h”

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

void func(void)
{
	/* configure SCI_0 in synchronous serial Mode. Configured it as a
	transmitter with  BAUD rate of 10,000 */

	R_SCI_Create(3, PDL_SCI_SCK0_A9 | PDL_SCI_TXD0_A10 | PDL_SCI_RXD0_A11,
	 PDL_SCI_SYNC | PDL_SCI_CLK_INT_OUT |PDL_SCI_TX_CONNECTED ,
	                10000,2);

	/* Configure SCI1 for asynchronous,  register values supplied */
	R_SCI_Create(
		1,
		PDL_SCI_SCK1_A5 | PDL_SCI_TXD1_A4 | PDL_SCI_RXD1_A19,
		PDL_SCI_ASYNC,
		PDL_SCI_PCLK_DIV_1 | PDL_SCI_CYCLE_BIT_16 | 0x50,
		1,
		0
	);
}
```

2) R_SCI_Destroy

Synopsis	Shut down a SCI channel.
Prototype	**bool R_SCI_Destroy(** **uint8_t data1,** // Channel selection **);**
Description	Stop data flow and shutdown the selected SCI channel. **[data1]** Select channel SCIn (where n = 0,1,2,4).
Return value	True if all parameters are valid; otherwise false.
Category	SCI
Reference	R_SCI_Create
Remarks	• The SCI channel is put into the power-down state.
Program example	

```
/* PDL definitions */
#include “r_pdl_sci.h”

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

void func(void)
{
    /* Shutdown SCI channel 1 */
    R_SCI_Destroy(1);
}
```

3) R_SCI_Send

Synopsis

Send a string of characters.

Prototype

```
bool R_SCI_Send(
    uint8_t data1,      // Channel selection
    uint8_t * data2,    // Pointer to transmit data
    uint8_t data3,      // Channel configuration
    void * func         // Callback function when the last byte has been sent
);
```

Description

Transmit characters on the specified serial channel.

[data1]
Select channel SCIn (where n = 0,1,2,4).

[data2]
A string of 8-bit values, ending in 0 (null).

[data3]
- Control the generation of a break signal at the end of transmission.

PDL_SCI_IDLE or PDL_SCI_BREAK	Select an Idle (normal) or Break signal to be output when all data has been sent.

[func]
The function to be called when the last byte has been sent.
Use R_SCI_Stop to terminate this operation early.
R_SCI_GetStatus can be used to find out how many characters have been transmitted.

Specify PDL_NO_FUNC for this function to wait until the last byte has been sent.

Return value

True if all parameters are valid and the operation completed;
False if a parameter was out of range or if the channel was already transmitting.

Category

SCI

Reference

R_SCI_Create, R_SCI_Receive, R_SCI_Stop, 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
- If no callback function is specified, this function will monitor the TXI and TEND flags to manage the data transmission. If the SCI channel's control registers are directly modified by the user, this function may lock up.
- If a Break signal is selected, it will be cleared when more data is transmitted
- The count of transmitted characters will loop back to 0 after 255 have been sent.
- 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

```
/* PDL definitions */
#include “r_pdl_sci.h”

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

void func( void )
{
    /* Send the Data from the SCI_0 with polling base transmission */
    R_SCI_Send(
        3,
        PDL_SCI_TXD0_A10,
        c_data,
        PDL_SCI_IDLE,
        PDL_NA
    );
}
```

4) R_SCI_Receive

Synopsis

Receive a string of characters.

Prototype

```
bool R_SCI_Receive(
    uint8_t data1,              // Channel selection
    uint8_t * data2,            // Receive buffer pointer
    uint8_t data3,              // Receive threshold
    void * func1,               // Callback function when the threshold is reached
    void * func2( uint8_t ),    // Callback function if an error occurs
    void * func3                // Callback function if a break is detected
);
```

Description

Enable SCI reception and acquire any incoming data.

[data1]
Select channel SCIn (where n = 0,1,2,4).

[data2]
The storage area for the expected data.
A null character shall be appended to the received data if more than one character is expected.

[data3]
The number of characters that must be received before the function completes or the callback function is called.

[func1]
The function to be called when the number of received characters reaches the threshold number.
While the receive operation is in progress:
R_SCI_GetStatus can be used to find out how many characters have been received so far.
R_SCI_Stop can be used to terminate this operation early.

Specify PDL_NO_FUNC for this function to continue until the required number of characters has been received.

[func2]
The function to be called if a receive error occurs. The error flags shall be supplied to the function in the format of the Serial Status Register:

B7	b6	b5	b4	b3	b2	b1	b0
0	0	ORER	FRE	PER	0	0	0

ORER: Overrun error
FRE: Framing error
PER: Parity error
Specify PDL_NO_FUNC to ignore errors.

[func3]
The function to be called if the Break signal is detected.
Specify PDL_NO_FUNC to ignore Break signals.

Return value

True if all parameters are valid and the operation completed;
False if a parameter was out of range or if the channel was already receiving.

Category

SCI

Reference

R_SCI_Create, R_SCI_Send, R_SCI_GetStatus

Remarks (1/2)

- The maximum number of characters to be received is 255.
- This function will wait until a transmission is complete before enabling the reception.
- If callback function func1 is specified, reception interrupts are used.
  Please see the notes on callback function usage
- If no callback function func1 is specified, this function will monitor the RXI flag to manage the data reception. If the SCI channel's control registers are directly modified by the user, this function may lock up.
- Error flags are cleared automatically.

Remarks (2/2)

- If callback function func1 is specified, callback functions are executed by the interrupt processing

function. This means that no other interrupt can be processed until a callback function has completed.

Program example

```
/* PDL functions */
#include “r_pdl_sci.h”

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

volatile uint8_t gSCI1ReceiveBuffer[10];

/* SCI channel 1 receive data handler */
void SCI1RxFunc(void){}

/* SCI channel 1 error handler */
void SCI1ErrFunc(uint8_t error_flags){}

/* SCI channel 1 break signal handler */
void SCI1BreakFunc(void){}

void func( void )
{
	uint8_t temp;

	/* Put a Null character at the end of the string */
	gSCI1ReceiveBuffer[10] = NULL;

	R_SCI_Receive(
		0,
		PDL_SCI_TXD0_A1,
		&c_ gSCI1ReceiveBuffer,
		 8,
		SCI1RxFunc,
		SCI1ErrFunc,
		SCI1BreakFunc
	);
}
```

5) R_SCI_Stop

Synopsis

Terminate SCI transmission or reception.

Prototype

```
bool R_SCI_Stop(
        uint8_t data1,      // Channel selection
        uint8_t data2       // Which processes are to be stopped
);
```

Description

Stops SCI transmission or reception.

[data1]
Select channel SCIn (where n = 0,1,2,4).

[data2]
- Select the process to be stopped.
  If multiple selections are required, use "|" to separate each selection.

PDL_SCI_TX	Stop the transmission process.
PDL_SCI_RX	Stop the reception process.

Return value

True if the channel is valid; otherwise false.

Category

SCI

Reference

R_SCI_Create, R_SCI_Send

Remarks

None.

Program example

```
/* PDL definitions */
#include "r_pdl_sci.h"

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

void func(void)
{
   /* Terminate SCI reception on channel 0 */
   R_SCI_Stop(
            0,
            PDL_SCI_RX
      );
}
```

6) R_SCI_GetStatus

Synopsis

Check the status of a SCI channel.

Prototype

```
bool R_SCI_GetStatus(
        uint8_t data1,          // Channel selection
        uint8_t * data2,        // Status flags store pointer
        uint8_t * data3,        // Character transmit counter pointer
        uint8_t * data4         // Character receive counter pointer
);
```

Description

Acquires status of SCI transmission / reception.

[data1]
Select channel SCIn (where n = 0,1,2,4).

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

b7 to b6	b5	b4	b3 – b0
-	0: Transmit idle 1: Transmission in progress	0: Receive idle 1: Reception in progress	-

[data3]
The storage locations for the number of characters that are have been transmitted in the current transmission. Specify PDL_NA if this information is not required.

[data4]
The storage locations for the number of characters that are have been received in the current reception process. Specify PDL_NA if this information is not required.

Return value

True if all parameters are valid and the operation completed; false if a parameter was out of range.

Category

SCI

Reference

R_SCI_Create, R_SCI_Send

Remarks

None.

Program example

```
/* PDL definitions */
#include “r_pdl_sci.h”

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

uint8_t StatusValue;
uint32_t TxChars;
uint32_t RxChars;

void func(void)
{
        /* Read the status of SCI channel 0 */
        R_SCI_GetStatus(
                0,
                &StatusValue,
                &TxChars,
                &RxChars
        );
}
```

4.2.8. 12-bit Analog to Digital Converter

1) R_ADC_12_Create

Synopsis

Configure an ADC unit.

Prototype

```
bool R_ADC_12_Create(
    uint8_t data1,      // ADC unit (module)
    uint32_t data2,     // ADC configuration
    void * func,        // Callback function
    uint8_t data3       // Interrupt priority level
);
```

Description (1/2)

[data1]
The ADC unit on which channel to be created resides.

[data2]
Set the ADC's mode and operating condition.
Conversion options. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Input channel selection

PDL_ADC_CHANNELS_OPTION_0 or	Unit 0: AN0 Unit 1: AN4
PDL_ADC_CHANNELS_OPTION_1 or	Unit 0: AN1 Unit 1: AN5
PDL_ADC_CHANNELS_OPTION_2 or	Unit 0: AN2 Unit 1: AN6
PDL_ADC_CHANNELS_OPTION_3	Unit 0: AN3 Unit 1: AN7

Description (2/2)

- Scan mode

PDL_ADC_SINGLE_SCAN or PDL_ADC_CONTINUOUS_SCAN	Select single mode or continuous scan mode.

- Trigger selection

PDL_ADC_TRIGGER_SOFTWARE or	Software
PDL_ADC_TRIGGER_ADTRG or	Trigger on external ADC trigger pin
PDL_ADC_TRIGGER_MTU_TRGAN and	MTU Trigger additional select one of the following
PDL_ADC_TRIGGER_MTU_STR0_ENABLE or	Trigger on MTU2 TRG4BN trigger
PDL_ADC_TRIGGER_MTU_STR1_ENABLE or	Trigger on MTU2 TRG4AN trigger
PDL_ADC_TRIGGER_MTU_STR2_ENABLE or	Trigger on MTU2 TRGAN trigger
PDL_ADC_TRIGGER_MTU_STR3_ENABLE or	Trigger on MTU2 TRG0N trigger
PDL_ADC_TRIGGER_MTU_STR4_ENABLE or	Trigger on MTU2S TRG4BN trigger
PDL_ADC_TRIGGER_MTU_STR5_ENABLE or	Trigger on MTU2S TRG4AN trigger
PDL_ADC_TRIGGER_MTU_STR6_ENABLE	Trigger on MTU2S TRGAN trigger

[data3]
Pointer to a callback function.

[data4]
The interrupt priority for ADC conversion end interrupt.

Bit 0 - 3	Interrupt Priority for ADC unit 0
Bit 4 – 7	Interrupt Priority for ADC unit 1

Return value

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

Functionality

ADC

References

R_ADC_12_Read

Remarks (1/2)

- This function configures the selected pin(s) for ADC operation by setting the direction to input and turning off the input buffer.
  The port control settings for any ADC pins that subsequently become inactive are not modified.
- This function brings the selected converter unit out of the power-down state.

Remarks (2/2)

- Interrupts are enabled automatically if a callback function is specified.
- 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

```
/* PDL definitions */
#include “r_pdl_adc_12_library_SH7216.h”

void func(void)
{
   /* Configure ADC channel AN0 in One shot mode with interrupt enabled */
      R_ADC_12_Create (
                  PDL_ADC_0,
                  PDL_ADC_SINGLE_SCAN | PDL_ADC_CHANNELS_OPTION_0 | \
                  PDL_ADC_TRIGGER_ADTRG | \
                  PDL_ADC_TRIGGER_ADC_ENABLE | PDL_ADC_TRIGGER_ADTRGB_PB1 | \
                  PDL_ADC_SAMPLEHOLD_ENABLE | PDL_ADC_ACE | \
                  PDL_ADC_INTERRUPT_ENABLE,
                  PDL_NA,
                  0u
                   );

}
```

2) R_ADC_12_Destroy

Synopsis	Shut down an ADC unit.
Prototype	**bool R_ADC_12_Destroy (** **uint8_t data1** // ADC unit to be destroyed **);**
Description	Put the ADC into the Power-down state, with minimal power consumption.
Return value	True ADC powered down successfully.
Category	ADC
Reference	R_ADC_12_Create
Remarks	• If the ADC unit's control registers are directly modified by the user, this function may lock up.
Program example	

```
/* PDL definitions */
#include "r_pdl_adc_12_library_SH7216.h"

void func( void )
{
    /* Shut down ADC unit */
    R_ADC_12_Destroy(PDL_ADC_0);
}
```

3) R_ADC_12_Control

Synopsis

Start or stop an ADC unit.

Prototype

```
bool R_ADC_12_Control(
    uint8_t  data          // ADC ON/OFF control
);
```

Description

Controls start / stop operation of the ADC.

[data]
Start or stop ADC operation.

- On / off control

PDL_ADC_ON or PDL_ADC_OFF	Start or stop ADC conversion for ADC units..

Return value

True if all parameters are valid; otherwise false.

Category

ADC

Reference

R_ADC_12_Create, R_ADC_12_Read

Remarks

- Use this API function only when the software trigger option is selected.
- For single mode, the ADC will stop automatically when the conversion is complete.
- The time delay between starting conversions on multiple units is minimised, but has to use separate instructions. This function minimises the delay between starts.
  For true simultaneous starting of ADC units, select hardware trigger e.g. MTU2 trigger.

Program example

```
/* PDL definitions */
#include “r_pdl_adc_12_library_SH7216.h”

void func(void)
{
    /* Stop ADC unit 1 and start ADC unit 0 */
    R_ADC_12_Control(
        PDL_ADC_0_ON | \
        PDL_ADC_1_OFF
        );
}
```

4) R_ADC_12_Read

Synopsis

Read the ADC conversion results.

Prototype

```
bool R_ADC_12_Read(
    uint8_t data1       // ADC unit on which channel resides
    uint16_t * data2    // Pointer to the buffer where the converted values are to be stored
);
```

Description

Reads the conversion values for an ADC unit.

[data1]
The ADC unit on which channel resides.

[data2]
Specify a pointer to a variable or array where the results shall be stored.

Return value

True if a valid unit is selected; otherwise false.

Category

ADC

Reference

R_ADC_12_Create, R_ADC_12_Control

Remarks

- Four conversion results will be read and stored.
- The 12-bit data alignment is controlled using the R_ADC_12_Create function.
- Ensure that the buffer is big enough for the requested number of values.
- If no callback function is used, this function waits for the ADI flag 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

```
/* PDL definitions */
#include "r_pdl_adc_12_library_SH7216.h"

void func(void)
{
    uint16_t ADCresult[4];

    /* Read the ADC values for channel 0 */
    R_ADC_12_ReadAll (
        PDL_ADC_0,
        &ADCresult
        );
}
```

4.2.9. Watchdog Timer

1) R_WDT_Create

Synopsis

Configure an WDT module.

Prototype

```
bool R_WDT_Create(
    uint32_t data1,     // WDT configuration
    uint8_t data2       // Initial value of WDT counter
    void * func,        // Callback function
    uint8_t data3       // Interrupt priority level
);
```

Description (1/2)

[data1]
The WDT configuration.
Conversion options. To set multiple options at the same time, use "|" to separate each value.
The default settings are shown in **bold**.

- Watchdog Timer Operating Mode

PDL_WDT_INTERVAL_TIMER_MODE	WDT module works as an interval timer.
PDL_WDT_WATCHDOG_TIMER_MODE	WDT module works as a watchdog timer..

- Enable / Disable watchdog timer.

PDL_WDT_TIMER_DISABLE	Disable watchdog Timer
PDL_WDT_TIMER_ENABLE	Enable watchdog Timer

- Select the clock to be used for the WTCNT count

PDL_WDT_PCLK or	Peripheral clock is input clock for WDT.
PDL_WDT_PCLK_BY_64 or	Peripheral clock divided by 64 is input clock for WDT.
PDL_WDT_PCLK_BY_128 or	Peripheral clock divided by 128 is input clock for WDT.
PDL_WDT_PCLK_BY_256 or	Peripheral clock divided by 256 is input clock for WDT.
PDL_WDT_PCLK_BY_512 or	Peripheral clock divided by 512 is input clock for WDT.
PDL_WDT_PCLK_BY_1024 or	Peripheral clock divided by 1024 is input clock for WDT.
PDL_WDT_PCLK_BY_4096 or	Peripheral clock divided by 4096 is input clock for WDT.
PDL_WDT_PCLK_BY_16384	Peripheral clock divided by 16384 is input clock for WDT.

- Enable / Disable reset on WDT overflow.

PDL_WDT_NO_RESET_ON_WDT_OVF	Don't reset on WDT overflow.
PDL_WDT_RESET_ON_WDT_OVF	Reset on WDT overflow.

- Select type of reset on WDT overflow

PDL_WDT_SELECT_POWER_ON_RESETor	A Power ON RESET is performed on WDT overflow.
PDL_WDT_SELECT_MANUAL_RESET or	A Manual RESET is performed on WDT overflow.

[data2]
Counter value to be initialised to WDT counter.

[func]
Pointer to a callback function.

[data4]
The interrupt priority for WDT interrupt.

Bit 0 - 3	Interrupt Priority for WDT

Return value

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

Functionality

WDT

References

R_ADC_12_Read

Remarks (1/2)

- This function configures WDT module to desired configuration.
- If the WDT unit's control registers are directly modified by the user, this function may lock up

Program example

```
/* PDL definitions */
#include "r_pdl_wdt_library_SH7216.h"

Callback_func()
{
   nop();
}

void func(void)
{
      /* Configure WDT to interval timer mode */
      R_WDT_Create(
            (PDL_WDT_INTERVAL_TIMER_MODE | PDL_WDT_TIMER_ENABLE | \
            PDL_WDT_PCLK),
            128,
            Callback_func,
            3
            );
}
```

2) R_WDT_Destroy

Synopsis	Stop WDT timer. & disable the unit
Prototype	**bool R_WDT_Destroy(** **void** **);**
Description	Stop the watchdog timer & disable the module.
Return value	True WDT disabled successfully.
Category	WDT
Reference	R_WDT_Create
Remarks	• If the WDT unit's control registers are directly modified by the user, this function may lock up.
Program example	

```
/* PDL definitions */
#include "r_pdl_wdt_library_SH7216.h"

void func(void)
{
    /* Stop WDT */
    R_WDT_Destroy();
}
```

3) R_WDT_Control

Synopsis

Start or stop an Watchdog (WDT) unit.

Prototype

```
bool R_WDT_Control (
    uint8_t  data          // WDT Control
);
```

Description

Controls start / stop operation of the WDT or clear the count.

[data]
Control WDT operation.

- Control Options available

PDL_WDT_START or PDL_WDT_STOP or PDL_WDT_RESET_COUNTER	Start or stop WDT or clear the count..

Return value

True if parameter is valid; otherwise false.

Category

WDT

Reference

1)R_WDT_Create, R_WDT_Read

Remarks

- This function configures WDT module to desired configuration.
- Clears timer counter & starts or stops the timer as per user's configuration

Program example

```
/* PDL definitions */
#include “r_pdl_wdt_library_SH7216.h”

void func(void)
{
    /* Stop WDT  */
     R_WDT_Control(
          PDL_WDT_STOP
           );
}
```

4) R_WDT_Read

Synopsis

Read the WDT status flags.

Prototype

```
bool R_WDT_Read(
    uint8_t * data1        // Buffer pointer to store WDT status flags
);
```

Description

Reads the status flags of WDT module.
[data1]
Specify a pointer to a variable or array where the status shall be stored.
Bit0: Interval Timer Overflow Flag Bit1: Watchdog Timer Overflow Flag

Return value

True if a valid unit is selected; otherwise false.

Category

WDT

Reference

R_WDT_Create, R_WDT_Destroy

Remarks

- After reading, this API clears the WDT flags from hardware.

Program example

```
/* PDL definitions */
#include “r_pdl_wdt_library_SH7216.h”

void func(void)
{
    uint16_t  status_flags;

    R_WDT_Read(&status_flags);
}
```

Revision History	Renesas Starter Kit2+ for SH7216 RPDL Manual

Rev.	Date	Description	
		Page	Summary
1.0	Feb.26, 2010	-	Preliminary version