RX Family

Parallel Data Capture Unit (PDC) Module Using Firmware Integration Technology

Introduction

This application note describes the parallel data capture unit (PDC) using firmware integration technology (FIT). This module controls the PDC to capture parallel data output by an image sensor such as a camera module. The module is referred to below as the PDC FIT module.

It should be noted that this application note is not compatible with application note “RX Family Parallel Data Capture Unit (PDC) Module Using Firmware Integration Technology” (R01AN2220).

Operation Confirmation Devices

The following is a list of devices that are currently supported by this API:

- RX64M
- RX71M
- RX651, RX65N
- RX72M

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Operation Confirmation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family DMA Controller DMACA Control Module Using Firmware Integration Technology (R01AN2063)
- RX Family DTC Module Using Firmware Integration Technology (R01AN1819).

(THE latest version can be downloaded from the Renesas Electronics website.)

Contents

1. Overview

The PDC provides functionality for communicating with an external I/O device such as an image sensor and transferring parallel data, such as image data, output by the external I/O device to the on-chip RAM or an external address space (CS area or SDRAM area), via the DTC or DMAC. Figure 1.1 shows an overview of the PDC.

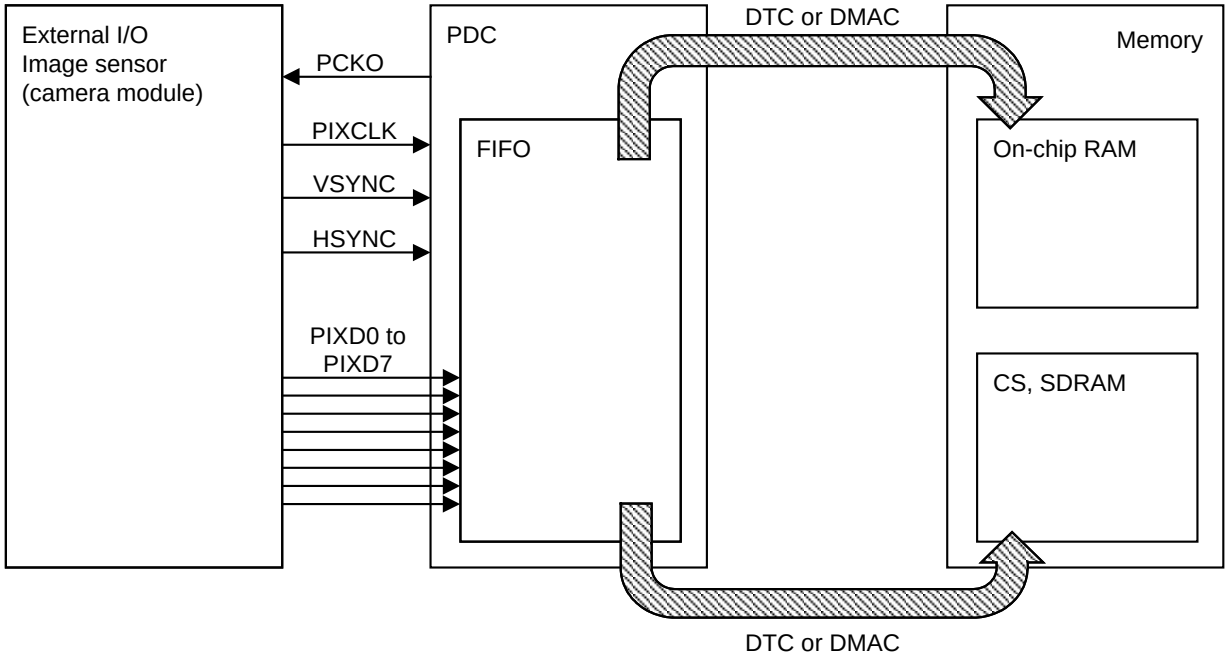

Figure 1.1 Overview of PDC

Limitations

This module utilizes the hardware locking function of the r_bsp.

1.1 About the PDC FIT Module

This module is used by embedding it in a project as an API. For information on how to embed the module, see 2.11, Adding the FIT Module to Your Project.

Notes

The endianness of the PDC FIT module switches automatically to match the endian setting of the compiler.

It is not possible to acquire image data from an image sensor using this module alone. The DMAC or DTC is used to transfer data to the memory, so refer to the manual of the corresponding FIT module and embed the FIT module in your project. You must prepare an initialization program for the image sensor and make settings yourself. For information on image sensor settings, contact the sensor manufacturer.

For information on the hardware lock function of r_bsp, see 2.17, Atomic Locking, in application note “RX Family Board Support Package Module Using Firmware Integration Technology” (R01AN1685).

1.2 API Overview

Table 1.1 lists the API functions included in the PDC FIT module.

Table 1.1 API Functions

Function	Description
R_PDC_Open	This function initializes the PDC FIT module.
R_PDC_Close	This function ends PDC operation and puts the PDC into the module stop state.
R_PDC_Control	This function performs processing according to control codes.
R_PDC_GetFifoAddr	This function gets the FIFO address of the PDC.
R_PDC_GetVersion	This function returns the API version number.

2. API Information

The API function of the PDC FIT module adhere to the Renesas API naming standards.

2.1 Hardware Requirements

The microcontroller used must support the following functions:

- PDC
- DTC
- DMAC

2.2 Software Requirements

This FIT module is dependent upon the following package:

Renesas Board Support Package (r_bsp) Rev.5.20 or higher

2.3 Supported Toolchain

This FIT module is tested and working with toolchains listed in 6.1 Operation Confirmation Environment.

2.4 Interrupt Vector

When the R_PDC_Open function is executed, the PCDFI, PCFEI, and PCERI interrupts are enabled according to the parameter values.

Table 2.1 lists the interrupt vector used in the PDC FIT Module.

Table 2.1 Interrupt Vector Used in the PDC FIT Module

Device	Interrupt Vector
RX64M	PCDFI interrupt (vector no.: 97)
RX65N	GROUPBL0 interrupt (vector no.: 110)
RX71M	● PCFEI interrupt (group interrupt source no.: 30)
RX72M	● PCERI interrupt (group interrupt source no.: 31)

2.5 Header Files

All API calls and their supporting interface definitions are located in r_pdc_rx_if.h.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Compile Settings

The configuration option settings of this module are located in r_pdc_rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_pdc_rx_config.h	
PDC_CFG_PCKO_DIV Note: The default value is “2”.	Set the PCKO frequency division ratio select bits in PDC control register 0 (PCCR0) according to the specified frequency division ratio. The parallel data transfer clock output (PCKO) operating frequency is the clock source, peripheral module clock B (PCLKB), divided by this setting value. The available setting values are 2, 4, 6, 8, 10, 12, 14, and 16. Specifying a value other than the preceding will result in an error at compile time. Note: The operating frequency range is 1 to 30 MHz, but the optimum value at which operation is possible under the specifications of the image sensor (camera module) used should be specified.

2.8 Code Size

The code size estimates for the supported toolchains (listed in section 2.3) assume optimization level 2 and optimization prioritizing code size. The ROM size (code and constants) and RAM size (global data) are determined by the configuration options specified in the module’s configuration header file at build time.

The values in the table below are confirmed under the following conditions.

Module Revision: r_pdc_rx rev2.04

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of “-lang = c99” is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201902

(The option of “-std=gnu99” is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.12.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX72M	ROM	1867 bytes	3760 bytes	3291 bytes
	RAM	17 bytes	20 bytes	21 bytes
	STACK *1	136 bytes	-	108 bytes

Note1. The sizes of maxmum usage stack of Interrupts functions is included.

2.9 Arguments

The structures and enumerated types used as arguments for the API functions are listed below. The API functions and their prototype declarations are located in r_pdc_rx_if.h.

```
/* Interrupt priority level control */
typedef struct st_pdc_int_priority_data_cfg
{
    uint8_t   pcdfi_level;              /* PCDFI interrupt priority level */
    uint8_t   groupbl0_level;           /* GROUPBL0 interrupt priority level */
} pdc_ipr_dcfg_t;
```

```
/* Interrupt controller (ICUA) PDC interrupt enable/disable */
typedef struct st_pdc_inticu_data_cfg
{
    bool   pcfei_ien;                   /* Frame-end interrupt request enabled */
    bool   pceri_ien;                   /* Error interrupt request enabled */
    bool   pcdfi_ien;                   /* Receive data-ready interrupt request enabled */
} pdc_inticu_dcfg_t;
```

```
/* PDC interrupt enable/disable */
typedef struct st_pdc_intpdc_data_cfg
{
    bool   dfie_ien;                    /* Receive data-ready interrupt request enabled */
    bool   feie_ien;                    /* Frame-end interrupt request enabled */
    bool   ovie_ien;                    /* Overrun interrupt request enabled */
    bool   udrie_ien;                   /* Underrun interrupt request enabled */
    bool   verie_ien;                   /* Vertical line count setting error interrupt request enabled */
    bool   herie_ien;                   /* Horizontal byte count setting error interrupt request enabled */
} pdc_intpdc_dcfg_t;
```

```
/* Capture position specification */
typedef struct st_pdc_position_data_cfg
{
    uint16_t   vst_position;            /* Vertical capture start line position */
    uint16_t   hst_position;            /* Horizontal capture start byte position */
} pdc_pos_dcfg_t;
```

```
/* Capture size specification */
typedef struct st_pdc_size_data_cfg
{
    uint16_t   vsz_size;                /* Vertical capture size */
    uint16_t   hsz_size;                /* Horizontal capture size */
} pdc_size_dcfg_t;
```

```
/* PDC settings */
typedef struct st_pdc_data_cfg
{
    uint16_t            iupd_select;      /* Interrupt setting update select */
    pdc_ipr_dcfg_t      priority;         /* Interrupt priority level */
    pdc_inticu_dcfg_t   inticu_req;       /* ICU interrupt setting */
    pdc_intpdc_dcfg_t   intpdc_req;       /* PDC interrupt setting */
    bool                vps_select;       /* VSYNC signal polarity select */
    bool                hps_select;       /* HSYNC signal polarity select */
    pdc_pos_dcfg_t      capture_pos;      /* Capture position setting */
    pdc_size_dcfg_t     capture_size;     /* Capture size setting */
    pdc_cb_t            p_callback;       /* Pointer to callback function */
} pdc_data_cfg_t;
```

```
/* Copy of PDC status register (PCSR) */
typedef struct st_pdc_data_cfg
{
    bool        frame_busy;                 /* PDC operating status (FBSY flag) */
    bool        fifo_empty;                 /* FIFO status (FEMPF flag) */
    bool        frame_end;                  /* Frame-end (FEF flag) */
    bool        overrun;                    /* Overrun (OVRF flag) */
    bool        underrun;                   /* Underrun (UDRF flag) */
    bool        verf_error;                 /* Vertical line count setting error (VERF flag) */
    bool        herf_error;                 /* Horizontal byte count setting error (HERF flag) */
} pdc_pcsr_stat_t;
```

```
/* Copy of PDC pin monitor status register (PCMONR) */
typedef struct st_pdc_data_cfg
{
    bool        vsync;                      /* VSYNC signal status (VSYNC flag) */
    bool        hsync;                      /* HSYNC signal status (HSYNC flag) */
} pdc_pcmonr_stat_t;
```

```
/* PDC status */
typedef struct st_pdc_data_cfg
{
    pdc_pcsr_stat_t         pcsr_stat;      /* PDC status register (PCSR) information */
    pdc_pcmonr_stat_t       pcmonr_stat;    /* PDC pin monitor status (PCMONR) information */
} pdc_stat_t;
```

```
/* R_PDC_Control control codes */
typedef enum e_pdc_command
{
    PDC_CMD_CAPTURE_START = 0,              /* Start PDC capture */
    PDC_CMD_CHANGE_POS_AND_SIZE,            /* Change PDC capture position and capture size */
    PDC_CMD_STATUS_GET,                     /* Get PDC status */
    PDC_CMD_STATUS_CLR,                     /* Clear PDC status */
    PDC_CMD_SET_INTERRUPT,                  /* PDC interrupt setting */
    PDC_CMD_DISABLE,                        /* Disable PDC receive operation */
    PDC_CMD_ENABLE,                         /* Enable PDC receive operation */
    PDC_CMD_RESET                           /* PDC reset */
} pdc_command_t;
```

```
/* Pointers to callback functions */
typedef struct
{
    void    (*pcb_receive_data_ready)(void *); /* Pointer to callback function when receive data-ready
                                                  interrupt occurs */
    void    (*pcb_frame_end)(void *);          /* Pointer to callback function when PDC FIFO is empty
                                                  after frame-end interrupt occurs */
    void    (*pcb_error)(void *);              /* Pointer to callback function when overrun error,
                                                  underrun error, vertical line count setting error, or
                                                  horizontal byte count setting error occurs */
}pdc_cb_t;
```

```
/* Callback function call source event code */
typedef enum
{
    PDC_EVT_ID_DATAREADY = 0,       /* Receive data-ready interrupt occurred. */
    PDC_EVT_ID_FRAMEEND,            /* Frame-end interrupt occurred. */
    PDC_EVT_ID_TIMEOUT,             /* Standby time elapsed but FIFO has not become empty. */
    PDC_EVT_ID_ERROR,               /* Error interrupt occurred. */
    PDC_EVT_ID_OVERRUN,             /* Overrun interrupt occurred. */
    PDC_EVT_ID_UNDERRUN,            /* Underrun interrupt occurred. */
    PDC_EVT_ID_VERTICALLINE,        /* Vertical line count setting error interrupt occurred. */
    PDC_EVT_ID_HORIZONTALBYTE       /* Horizontal byte count setting error interrupt occurred. */
}pdc_cb_event_t;
```

```
/* Argument passed to callback function */
typedef struct
{
    pdc_cb_event_t  event_id;       /* Event code of callback function call source */
}pdc_cb_arg_t;
```

2.10 Return Values

The return values of the API functions are shown below. This enumerated type and the API function prototype declarations are located in r_pdc_rx_if.h.

```
/* Function return values */
typedef enum e_pdc_return      /* PDC API error codes */
{
    PDC_SUCCESS = 0,           /* Processing finished successfully.*/
    PDC_ERR_OPENED,            /* PDC module initialized. Initialization function R_PDC_Open has been run. */
    PDC_ERR_NOT_OPEN,          /* PDC module uninitialized. R_PDC_Open has not been run. */
    PDC_ERR_INVALID_ARG,       /* Invalid argument input. */
    PDC_ERR_INVALID_COMMAND,   /* Command is invalid. */
    PDC_ERR_NULL_PTR,          /* Argument pointer value was NULL. */
    PDC_ERR_LOCK_FUNC,         /* PDC resource is in use by another process. */
    PDC_ERR_INTERNAL,          /* Module internal error detected. */
    PDC_ERR_RST_TIMEOUT        /* PDC reset was not canceled even after the specified amount of time elapsed. */
} pdc_return_t;
```

2.11 Callback Functions

(1) Receive Data-Ready Interrupt (PCDFI) and Frame-End Interrupt (PCFEI) Callback Functions

When a receive data-ready interrupt (PCDFI) occurs, or when the FIFO becomes empty after a frame-end interrupt (PCFEI) occurs, the PDC FIT module calls a callback function.

The R_PDC_Open function is used to specify the callback function. For details, see 3.1, R_PDC_Open().

When a receive data-ready interrupt occurs, the PDC FIT module calls the receive data-ready interrupt callback function. However, if the DMAC is selected for data transfer, you should set the PCDFI interrupt priority level to 0 so that no callback function is called.

When a frame-end interrupt occurs, the PDC FIT module stands by until the DTC or DMAC has transferred all the data in the FIFO of the PDC (until the FIFO of the PDC is empty). When it is confirmed that the FIFO of the PDC is empty, PDC operation is disabled, the frame-end flag is cleared to 0, and the PDC FIT module calls the frame-end interrupt callback function. Note that if an underrun occurs before the FIFO of the PDC becomes empty, the frame-end flag is cleared to 0, and the error callback function is called. Also, if the FIFO of the PDC is not empty even after the specified amount of time has elapsed, the frame-end flag is cleared to 0, and the timeout callback function is called.

When the callback function is called, the variable stored for the constant listed in Table 2.2 is passed as an argument. If an argument will be used outside the callback function, it should be copied to a global variable, or the like.

When the callback function is called as described above, group interrupt (GROUPBL0) requests should first be enabled, and then PCFEI interrupt requests, receive data-ready interrupt requests, and frame-end interrupt requests should be enabled by means of arguments passed when the R_PDC_Open function is run. For details, see 3.1, R_PDC_Open().

Table 2.2 Callback Function Arguments when Receive Data-Ready Interrupt or Frame-End Interrupt Occurs

Variable Definition	Description
PDC_EVT_ID_DATAREADY	A receive data-ready interrupt occurred.
PDC_EVT_ID_FRAMEEND	A frame-end interrupt occurred.
PDC_EVT_ID_TIMEOUT	The standby time elapsed but the FIFO did not become empty.

(2) Callback Function when Errors Occur

When an overrun, underrun, vertical line count setting error, or horizontal byte count setting error occurs, the PDC FIT module calls a callback function.

The R_PDC_Open function is used to specify the callback function. For details, see 3.1, R_PDC_Open().

When an error interrupt occurs, the PDC FIT module stops PDC operation and then calls the callback function with the argument PDC_EVT_ID_ERROR. After this, it confirms in order whether or not an overrun, underrun, vertical line count setting error, or horizontal byte count setting error has occurred. If an error has occurred, it calls the callback function. When the callback function finishes, the error flag corresponding to the error that occurred is cleared to 0 and checking resumes to determine if the next error type has occurred.

When the callback function is called with the argument PDC_EVT_ID_ERROR, make sure that at the start of processing the DTC or DMAC data transfer processing is disabled.

When the callback function is called, the variable stored for the constant listed in Table 2.3 is passed as an argument. If an argument will be used outside the callback function, it should be copied to a global variable, or the like.

When the callback function is called as described above, group interrupt (GROUPBL0) requests should first be enabled, and then PCERI interrupt requests, overrun interrupt requests, underrun interrupt requests, vertical line count setting error interrupt requests, and horizontal byte count setting error interrupt requests should be enabled by means of arguments passed when the R_PDC_Open function is run. For details, see 3.1, R_PDC_Open().

Table 2.3 Callback Function Arguments when Errors Occurs

Variable Definition	Description
PDC_EVT_ID_ERROR	Error interrupt occurred.
PDC_EVT_ID_OVERRUN	Overrun error occurred.
PDC_EVT_ID_UNDERRUN	Underrun error occurred.
PDC_EVT_ID_VERTICALLINE	Vertical line count setting error occurred.
PDC_EVT_ID_HORIZONTALBYTE	Horizontal byte count setting error occurred.

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends using “Smart Configurator” described in (1) or (3). However, “Smart Configurator” only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using “Smart Configurator” in e^2 studio
By using the “Smart Configurator” in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using “FIT Configurator” in e^2 studio
By using the “FIT Configurator” in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using “Smart Configurator” on CS+
By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.13 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /*
WAIT_LOOP */
```

3. API Functions

3.1 R_PDC_Open()

This function initializes the PDC FIT module. It must be run before using the other API functions.

Format

```
pdc_return_t R_PDC_Open(
     pdc_data_cfg_t    *p_data_cfg
)
```

Parameters

**p_data_cfg*

Pointer to PDC settings data structure

Members of Referenced pdc_data_cfg_t Structure and Their Setting Values

Parameters other than those listed below are not referenced, so they do not need to be set before the API is called.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
priority.pcdfi_level	PCDFI interrupt priority level	8-bit data 00h to 0Fh	ICU.IPR097.IPR	Sets the receive data-ready interrupt (PCDFI) priority level.
priority.groupbl0_level	GROUPBL0 interrupt priority level	8-bit data 00h to 0Fh	ICU.IPR110.IPR	Sets the fame-end interrupt and error interrupt priority level.
inticu_req.pcdfi_ien	PCDFI interrupt enabled	false	ICU.IER0C.IEN1	Disables interrupt requests for the receive data-ready interrupt (PCDFI).
		true		Enables interrupt requests for the receive data-ready interrupt (PCDFI).
inticu_req.pcfei_ien	PCFEI interrupt enabled	false	ICU.GRPBL0.EN30	Disables interrupt requests for the frame-end interrupt (PCFEI).
		true		Enables interrupt requests for the frame-end interrupt (PCFEI).
inticu_req.pceri_ien	PCERI interrupt enabled	false	ICU.GRPBL0.EN31	Disables interrupt requests for the error interrupt (PCERI).
		true		Enables interrupt requests for the error interrupt (PCERI).
intpdc_req.dfie_ien	Receive data-ready interrupt request	false	PCCR0.DFIE	Disables receive data-ready interrupt requests.
		true		Enables receive data-ready interrupt requests.
intpdc_req.feie_ien	Frame-end interrupt request	false	PCCR0.FEIE	Disables frame-end interrupt requests.
		true		Enables frame-end interrupt requests.
intpdc_req.ovie_ien	Overrun interrupt request	false	PCCR0.OVIE	Disables overrun interrupt requests.
		true		Enables overrun interrupt requests.
intpdc_req. udrie_ien	Underrun interrupt	false	PCCR0.UDRIE	Disables underrun interrupt requests.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
	request	true		Enables underrun interrupt requests.
intpdc_req. verie_ien	Vertical line count setting error interrupt request	false	PCCR0.VERIE	Disables vertical line count setting error interrupt requests.
		true		Enables vertical line count setting error interrupt requests.
intpdc_req. herie_ien	Horizontal byte count setting error interrupt request	false	PCCR0.HERIE	Disables horizontal byte count setting error interrupt requests.
		true		Enables horizontal byte count setting error interrupt requests.
vps_select	VSYNC signal polarity select	PDC_VSYNC_SIGNAL_POLARITY_HIGH	PCCR0.VPS	VSYNC signal is high-active.
		PDC_VSYNC_SIGNAL_POLARITY_LOW		VSYNC signal is low-active.
hps_select	HSYNC signal polarity select	PDC_HSYNC_SIGNAL_POLARITY_HIGH	PCCR0.HPS	HSYNC signal is high-active.
		PDC_HSYNC_SIGNAL_POLARITY_LOW		HSYNC signal is low-active.
capture_pos.vst_position	Vertical capture start line position	12-bit data 0000h to 0FFEh	VCR.VST	Vertical capture start line position
capture_pos.hst_position	Horizontal capture start line position	12-bit data 0000h to 0FFBh	HCR.HST	Horizontal capture start line position
capture_size.vsz_size	Vertical capture size	12-bit data 0001h to 0FFFh	VCR.VSZ	Vertical capture line count
capture_size.hsz_size	Horizontal capture size	12-bit data 0004h to 0FFFh	HCR.HSZ	Horizontal capture byte count
p_callback.pcb_receive_data_ready	Pointer to callback function when PCDFI interrupt occurs	other than NULL/ FIT_NO_FUNC	None	The callback function at the address indicated by the pointer runs when a receive data-ready interrupt occurs.
		NULL/ FIT_NO_FUNC		The callback function does not run even when the source occurs.
p_callback.pcb_frame_end	Pointer to callback function when PCFEI interrupt occurs	other than NULL/ FIT_NO_FUNC	None	The callback function at the address indicated by the pointer runs when the FIFO becomes empty after a frame-end interrupt occurs.
		NULL/ FIT_NO_FUNC		The callback function does not run even when the source occurs.
p_callback.pcb_error	Pointer to callback function when PCERI interrupt occurs	other than NULL/ FIT_NO_FUNC	None	The callback function at the address indicated by the pointer runs when an error interrupt occurs and when an overrun, underrun, vertical line count setting error, or horizontal byte count setting error occurs.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
		NULL/ FIT_NO_FUNC		The callback function does not run even when the source occurs.

Return Values

PDC_SUCCESS	*/* Processing finished successfully. */*
PDC_ERR_OPENED	*/* R_PDC_Open has already been run. */*
PDC_ERR_INVALID_ARG	*/* Parameter values in PDC setting information are invalid. */*
PDC_ERR_NULL_PTR	*/* Argument p_data_cfg is a NULL pointer. */*
PDC_ERR_LOCK_FUNC	*/* The PDC has already been locked by another process. */*
PDC_ERR_INTERNAL	*/* A module internal error was detected. */*
PDC_ERR_RST_TIMEOUT	*/* PDC reset was not canceled even after the specified amount of time elapsed. */*

Properties

The declaration is located in r_pdc_rx_if.h.

Description

The following processing is performed to initialize the PDC:

- Locks the PDC hardware resource using the r_bsp hardware locking function.
- Cancels PDC module stop state.
- Registers the callback functions to be called when interrupts used by the PDC occur.
- Makes settings for interrupts used by the PDC.
  Interrupt settings are made for the receive data-ready interrupt (PCDFI), frame-end interrupt (PCFEI), and error interrupt (PCERI).
- Stops PDC receive operation.
  Sets the PCE bit in PDC control register 1 (PCCR1) to “receive operation disabled.”
- Specifies the clock for parallel data transfer clock output (PCKO).
  Sets the PCKDIV bits in PDC control register 0 (PCCR0) to specify the clock.
  Specifies the parallel data transfer clock output (PCKO) setting value according to the value of PDC_CFG_PCKO_DIV in r_pdc_rx_config.h.
- Starts supply of parallel data transfer clock output (PCKO).
  Sets the PCKOE bit in PDC control register 0 (PCCR0) to “PCKO output enabled.”
- Enables PIXCLK input (PCCR0.PCKE).
  Sets the PCKE bit in PDC control register 0 (PCCR0) to “PIXCLK input enabled.”
- Resets the PDC (PCCR0.PRST).
  Starts initialization of the internal state of the PDC and of the PDC reset target registers.
- Makes vertical and horizontal capture range settings (VCR and HCR settings).
- Makes polarity settings for VSYNC and HSYNC signals (VPS and HPS).
- Makes interrupt enable/disable settings (DFIE. FEIE, OVIE, UDRIE, VERIE, and HERIE).
- Makes endianness setting (EDS).

Example

In the sample code two bytes are used to represent each dot of the image sensor output, so the horizontal dot count for the horizontal capture position and size is set to twice the actual value. The setting value should be modified as necessary to match the output characteristics of the actual image sensor used.

Case 1: Capturing image at VGA (640 × 480) resolution

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Setting values of PDC operation */
pdc_data_cfg_t           data_pdc;

/*
    Set the value 0 to PCDFI interrupt priority when using DMAC
    Set the value 1-15 to PCDFI interrupt priority level when using DTC
*/
data_pdc.priority.pcdfi_level = 0;
/* Set the values 1-15 to GROUPBL0 interrupt priority level */
data_pdc.priority.groupbl0_level = 2;
/* PCDFI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcdfi_ien = true;
/* PCFEI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcfei_ien = true;
/* PCERI interrupt request in ICU is enabled */
data_pdc.inticu_req.pceri_ien = true;
/* Generation of receive data ready interrupt requests is enabled */
data_pdc.intpdc_req.dfie_ien = true;
/* Generation of frame end interrupt requests is enabled */
data_pdc.intpdc_req.feie_ien = true;
/* Generation of overrun interrupt requests is enabled */
data_pdc.intpdc_req.ovie_ien = true;
/* Generation of underrun interrupt requests is enabled */
data_pdc.intpdc_req.udrie_ien = true;
/* Generation of vertical line number setting error interrupt requests is enabled */
data_pdc.intpdc_req.verie_ien = true;
/* Generation of horizontal byte number setting error interrupt requests is enabled */
data_pdc.intpdc_req.herie_ien = true;
/* VSYNC signal is active LOW */
data_pdc.vps_select = PDC_VSYNC_SIGNAL_POLARITY_LOW;
/* HSYNC signal is active HIGH */
data_pdc.hps_select = PDC_HSYNC_SIGNAL_POLARITY_HIGH;
/* Capture from 0 pixel of vertical direction */
data_pdc.capture_pos.vst_position = 0;
/* Capture from 0 pixel of horizontal direction */
data_pdc.capture_pos.hst_position = 0;
/* Capture 480 pixels in vertical direction */
data_pdc.capture_size.vsz_size = 480;
/* Capture 640 pixels in horizontal direction */
data_pdc.capture_size.hsz_size = (640 * 2);
/* Pointer to PCDFI interrupt callback function */
data_pdc.p_callback.pcb_receive_data_ready = (void (*) (void *)) pcdfi_callback;
/* Pointer to PCFEI interrupt callback function */
data_pdc.p_callback.pcb_frame_end = (void (*) (void *)) pcfei_callback;
/* Pointer to PCERI interrupt callback function */
data_pdc.p_callback.pcb_error = (void (*) (void *)) pceri_callback;

ret_pdc = R_PDC_Open(&data_pdc);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
```

```
}
```

Case 2: Capturing the lower right quadrant of a VGA (640 × 480) image at QVGA (320 × 240) resolution

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t      ret_pdc;
/* Setting values of PDC operation */
pdc_data_cfg_t             data_pdc;

/*
    Set the value 0 to PCDFI interrupt priority when using DMAC
    Set the value 1-15 to PCDFI interrupt priority level when using DTC
*/
data_pdc.priority.pcdfi_level = 0;
/* Set the values 1-15 to GROUPBL0 interrupt priority level */
data_pdc.priority.groupbl0_level = 2;
/* PCDFI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcdfi_ien = true;
/* PCFEI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcfei_ien = true;
/* PCERI interrupt request in ICU is enabled */
data_pdc.inticu_req.pceri_ien = true;
/* Generation of receive data ready interrupt requests is enabled */
data_pdc.intpdc_req.dfie_ien = true;
/* Generation of frame end interrupt requests is enabled */
data_pdc.intpdc_req.feie_ien = true;
/* Generation of overrun interrupt requests is enabled */
data_pdc.intpdc_req.ovie_ien = true;
/* Generation of underrun interrupt requests is enabled */
data_pdc.intpdc_req.udrie_ien = true;
/* Generation of vertical line number setting error interrupt requests is enabled */
data_pdc.intpdc_req.verie_ien = true;
/* Generation of horizontal byte number setting error interrupt requests is enabled */
data_pdc.intpdc_req.herie_ien = true;
/* VSYNC signal is active LOW */
data_pdc.vps_select = PDC_VSYNC_SIGNAL_POLARITY_LOW;
/* HSYNC signal is active HIGH */
data_pdc.hps_select = PDC_HSYNC_SIGNAL_POLARITY_HIGH;
/* Capture from 240 pixel of vertical direction */
data_pdc.capture_pos.vst_position = 240;
/* Capture from 320 pixel of horizontal direction */
data_pdc.capture_pos.hst_position = (320 * 2);
/* Capture 240 pixels in vertical direction */
data_pdc.capture_size.vsz_size = 240;
/* Capture 320 pixels in horizontal direction */
data_pdc.capture_size.hsz_size = (320 * 2);
/* Pointer to PCDFI interrupt callback function */
data_pdc.p_callback.pcb_receive_data_ready = (void (*) (void *)) pcdfi_callback;
/* Pointer to PCFEI interrupt callback function */
data_pdc.p_callback.pcb_frame_end = (void (*) (void *)) pcfei_callback;
/* Pointer to PCERI interrupt callback function */
data_pdc.p_callback.pcb_error = (void (*) (void *)) pceri_callback;

ret_pdc = R_PDC_Open(&data_pdc);
if (PDC_SUCCESS != ret_pdc)
{
```

```
    /* Error processing */
}
```

Callback function called when receive data-ready interrupt occurs

```
#include “platform.h”
#include “r_pdc_rx_if.h”

void pcdfi_callback(void * pdata)
{
    /* Stores the argument for callback function */
    pdc_cb_arg_t    * pdecode;
    pdecode = (pdc_cb_arg_t *)pdata;

    switch(pdecode->event_id)
    {
        case PDC_EVT_ID_DATAREADY:
            /* do something */
        break;

        default:
        break;
    }
}
```

Callback function called when frame-end interrupt occurs and FIFO of the PDC is empty

```
#include “platform.h”
#include “r_pdc_rx_if.h”

void pcfei_callback(void * pdata)
{
    /* Stores the argument for callback function */
    pdc_cb_arg_t    * pdecode;
    pdecode = (pdc_cb_arg_t *)pdata;

    switch(pdecode->event_id)
    {
        case PDC_EVT_ID_FRAMEEND:
            /* do something */
        break;

        case PDC_EVT_ID_TIMEOUT:
            /* do something */
        break;

        default:
        break;
    }
}
```

Callback function called when error interrupt, overrun error, underrun error, vertical line count setting error, or horizontal byte count setting error occurs

```
#include “platform.h”
#include “r_pdc_rx_if.h”

void pceri_callback(void * pdata)
{
    /* Stores the argument for callback function */
    pdc_cb_arg_t    * pdecode;
    pdecode = (pdc_cb_arg_t *)pdata;

    switch(pdecode->event_id)
    {
        case PDC_EVT_ID_ERROR:
            /* Disable the DTC or DMAC transfer */
            /* Error interrupt processing */
        break;

        case PDC_EVT_ID_OVERRUN:
            /* Overrun error processing */
        break;

        case PDC_EVT_ID_UNDERRUN:
            /* Underrun error processing */
        break;

        case PDC_EVT_ID_VERTICALLINE:
            /* Vertical Line Number Setting Error processing */
        break;

        case PDC_EVT_ID_HORIZONTALBYTE:
            /* Horizontal Byte Number Setting Error processing */
        break;

        default:
        break;
    }
}
```

Special Notes:

This API function should be run when the device and the camera module are connected. Running this API function enables PIXCLK input and then resets the PDC, but this is because the reset will not complete if PIXCLK where the camera module output is not input to the device. If the return value PDC_ERR_RST_TIMEOUT is confirmed, check the settings and hardware configuration of the camera module.

An endianness setting is applied within this API function. The endianness setting should be selected to match the corresponding compiler setting. If the compiler endianness setting is little-endian, the PDC endianness setting should be little-endian as well, and if the compiler endianness setting is big-endian, the PDC endianness setting should also be big-endian.

The arguments and return values of the registered callback function should be of type void.

3.2 R_PDC_Close()

Ends operation by the PDC and puts it into the module stop state.

Format

```
pdc_return_t R_PDC_Close(void)
```

Parameters

None

Return Values

PDC_SUCCESS */* Processing finished successfully. */*
PDC_ERR_NOT_OPEN */* R_PDC_Open has not been run. */*

Properties

The declaration is located in r_pdc_rx_if.h.

Description

Performs the following processing to shut down the PDC:

- Disables interrupts (PCFEI, PCERI, and PCDFI) used by the PDC.
- Disables PDC operation.
  Sets the PCE bit in PDC control register 1 (PCCR1) to “Operations for reception are disabled.”
- Stops supply of parallel data transfer clock output (PCKO).
  Sets the PCKOE bit in PDC control register 0 (PCCR0) to “PCKO output is disabled (fixed to the high level).”
- Disables pixel clock input from the image sensor.
  Sets the PCKE bit in PDC control register 0 (PCCR0) to “PIXCLK input is disabled.”
- Stops PDC module.
  Cancels PDC hardware resource locking using the r_bsp hardware locking function.

Example

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;

ret_pdc = R_PDC_Close();

if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Special Notes:

Use this API function after running R_PDC_Open and confirming that the return value is PDC_SUCCESS.

3.3 R_PDC_Control()

This function performs processing according to control codes.

Format

```
dmac_return_t R_PDC_Control(
    dmac_command_t    command,
    pdc_data_cfg_t      *p_data_cfg
    pdc_stat_t          *p_stat,
)
```

Parameters

command
Control code
**p_data_cfg*
Pointer to PDC settings data structure
**p_stat*
Pointer to PDC status structure

The Command Values:

/ Start capturing data from the image sensor (camera module). */*
PDC_CMD_CAPTURE_START
/ Change the range data capture from the image sensor (camera module). */*
PDC_CMD_CHANGE_POS_AND_SIZE
/ Get PDC status information. */*
PDC_CMD_STATUS_GET
/ Clear PDC status information. */*
PDC_CMD_STATUS_CLR
/ Reset PDC interrupt settings. */*
PDC_CMD_SET_INTERRUPT
/ Disable PDC receive operation. */*
PDC_CMD_DISABLE
/ Enable PDC receive operation. */*
PDC_CMD_ENABLE
/ Reset the PDC. */*
PDC_CMD_RESET

The arguments that are referenced differ according to the specified command.

- PDC_CMD_CAPTURE_START
  - — Members of referenced pdc_data_cfg_t structure and their setting values
    None
  - — Members of referenced pdc_stat_t structure and their setting values
    None

- PDC_CMD_CHANGE_POS_AND_SIZE
  - Members of referenced pdc_data_cfg_t structure and their setting values
    Parameters other than those listed below are not referenced, so they do not need to be set before the API is called.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
vst_position	Vertical capture start line position	12-bit data 0000h to 0FFEh	VCR.VST	Number of the line where capture is to start.
hst_position	Horizontal capture start byte position	12-bit data 0000h to 0FFBh	HCR.HST	Horizontal position in bytes where capture is to start.
vsz_size	Vertical capture size	12-bit data 0001h to 0FFFh	VCR.VSZ	Number of lines to be captured.
hsz_size	Horizontal capture size	12-bit data 0004h to 0FFFh	HCR.HSZ	Number of bytes to be captured horizontally.

  - Members of referenced pdc_stat_t structure and their setting values
    None

- PDC_CMD_STATUS_GET
  - Members of referenced pdc_data_cfg_t structure and their setting values
    None
  - Members of referenced pdc_stat_t structure and their setting values

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
pcsr_stat.frame_busy	Frame-busy flag	false	PCSR.FBSY	Operations for reception are stopped.
		true		Operations for reception are ongoing.
pcsr_stat.fifo_empty	FIFO-empty flag	false	PCSR.FEMPF	FIFO is not empty.
		true		FIFO is empty.
pcsr_stat.frame_end	Frame-end flag	false	PCSR.FEF	Frame end has not been generated.
		true		Frame end has been generated.
pcsr_stat.overrun	Overrun flag	false	PCSR.OVRF	FIFO overrun has not been generated.
		true		FIFO overrun has been generated.
pcsr_stat.underrun	Underrun flag	false	PCSR.UDRF	Underrun has not been generated.
		true		Underrun has been generated.
pcsr_stat.verf_error	Vertical line number setting error flag	false	PCSR.VERF	Vertical line number setting error has not been generated.
		true		Vertical line number setting error has been generated.
pcsr_stat.herf_error	Horizontal byte number setting error flag	false	PCSR.HERF	Horizontal byte number setting error has not been generated.
		true		Horizontal byte number setting error has been generated.
pcmonr_stat.vsync	VSYNC signal status flag	false	PCMONR.VSYNC	VSYNC signal is at the low level.
		true		VSYNC signal is at the high level.
pcmonr_stat.hsync	HSYNC signal status flag	false	PCMONR.HSYNC	HSYNC signal is at the low level.
		true		HSYNC signal is at the high level.

- PDC_CMD_STATUS_CLR
  - Members of referenced pdc_data_cfg_t structure and their setting values
    None
  - Members of referenced pdc_stat_t structure and their setting values
    Parameters other than those listed below are not referenced, so they do not need to be set before the API is called.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
pcsr_stat.frame_end	Frame-end flag	false	PCSR.FEF	Does nothing.
		true		Clears the frame-end flag.
pcsr_stat.overrun	Overrun flag	false	PCSR.OVRF	Does nothing.
		true		Clears the overrun flag.
pcsr_stat.underrun	Underrun flag	false	PCSR.UDRF	Does nothing.
		true		Clears the underrun flag.
pcsr_stat.verf_error	Vertical line number setting error flag	false	PCSR.VERF	Does nothing.
		true		Clears the vertical line number setting error flag.
pcsr_stat.herf_error	Horizontal byte number setting error flag	false	PCSR.HERF	Does nothing.
		true		Clears the horizontal byte number setting error flag.

- PDC_CMD_SET_INTERRUPT
  - Members of referenced pdc_data_cfg_t structure and their setting values
    Parameters other than those listed below are not referenced, so they do not need to be set before the API is called.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
iupd_select	Update target selection	10-bit data 0000h to 03FFh	None	The following parameters specify which interrupt settings are updated: Bit 0: PCDFI interrupt priority level Bit 1: GROUPBL0 interrupt priority level Bit 2: PCDFI interrupt enabled Bit 3: PCFEI interrupt enabled Bit 4: PCERI interrupt enabled Bit 5: Receive data-ready interrupt request Bit 6: Frame-end interrupt request Bit 7: Overrun interrupt request Bit 8: Underrun interrupt request Bit 9: Vertical line number setting error interrupt request Bit 10: Horizontal byte number setting error interrupt request Bits 11 to 15: Not used 0: Do not update setting. 1: Update setting.
priority.pcdfi_level	PCDFI interrupt priority level	8-bit data 00h to 0Fh	ICU.IPR097.IPR	Sets the receive data-ready interrupt (PCDFI) priority level. Note: Set bit 0 in iupd_select to 1.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
priority.groupbl0_level	GROUPBL 0 interrupt priority level	8-bit data 00h to 0Fh	ICU.IPR110.IPR	Sets the frame-end interrupt and error interrupt priority level. Note: Set bit 1 of iupd_select to 1. Setting a value smaller than the current value is invalid.
inticu_req.pcdfi_ien	PCDFI interrupt enabled	false	ICU.IER0C.IEN1	Disables receive data-ready interrupt (PCDFI) interrupt requests. Note: Set bit 2 in iupd_select to 1.
		true		Enables receive data-ready interrupt (PCDFI) interrupt requests. Note: Set bit 2 in iupd_select to 1.
inticu_req.pcfei_ien	PCFEI interrupt enabled	false	ICU.GRPBL0.EN30	Disables frame-end interrupt (PCFEI) interrupt requests. Note: Set bit 3 in iupd_select to 1.
		true		Enables frame-end interrupt (PCFEI) interrupt requests. Note: Set bit 3 in iupd_select to 1.
inticu_req.pceri_ien	PCERI interrupt enabled	false	ICU.GRPBL0.EN31	Disables error interrupt (PCERI) interrupt requests. Note: Set bit 4 in iupd_select to 1.
		true		Enables error interrupt (PCERI) interrupt requests. Note: Set bit 4 in iupd_select to 1.
intpdc_req.dfie_ien	Receive data-ready interrupt request	false	PCCR0.DFIE	Disables generation of receive data-ready interrupt requests. Note: Set bit 5 in iupd_select to 1.
		true		Enables generation of receive data-ready interrupt requests. Note: Set bit 5 in iupd_select to 1.
intpdc_req.feie_ien	Frame-end interrupt request	false	PCCR0.FEIE	Disables generation of frame-end interrupt requests. Note: Set bit 6 in iupd_select to 1.
		true		Enables generation of frame-end interrupt requests. Note: Set bit 6 in iupd_select to 1.
intpdc_req.ovie_ien	Overrun interrupt request	false	PCCR0.OVIE	Disables generation of overrun interrupt requests. Note: Set bit 7 in iupd_select to 1.
		true		Enables generation of overrun interrupt requests. Note: Set bit 7 in iupd_select to 1.
intpdc_req. udrie_ien	Underrun interrupt request	false	PCCR0.UDRIE	Disables generation of underrun interrupt requests. Note: Set bit 8 in iupd_select to 1.
		true		Enables generation of underrun interrupt requests. Note: Set bit 8 in iupd_select to 1.
intpdc_req. verie_ien	Vertical line number setting error interrupt	false	PCCR0.VERIE	Disables generation of vertical line number setting error interrupt requests. Note: Set bit 9 in iupd_select to 1.

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
	request	true		Enables generation of vertical line number setting error interrupt requests. Note: Set bit 9 in iupd_select to 1.
intpdc_req. herie_ien	Horizontal byte number setting error interrupt request	false	PCCR0.HERIE	Disables generation of horizontal byte number setting error interrupt requests. Note: Set bit 10 in iupd_select to 1.
		true		Enables generation of horizontal byte number setting error interrupt requests. Note: Set bit 10 in iupd_select to 1.

— Members of referenced pdc_stat_t structure and their setting values
None

- PDC_CMD_DISABLE/PDC_CMD_ENABLE
  — Members of referenced pdc_data_cfg_t structure and their setting values
  None
  — Members of referenced pdc_stat_t structure and their setting values
  None

- PDC_CMD_RESET
  — Members of referenced pdc_data_cfg_t structure and their setting values
  None
  — Members of referenced pdc_stat_t structure and their setting values
  None

Return Values

```
PDC_SUCCESS              /* Processing finished successfully. */
PDC_ERR_NOT_OPEN         /* R_PDC_Open has not been run. */
PDC_ERR_INVALID_ARG      /* Setting value applied to PDC register is invalid. */
PDC_ERR_INVALID_COMMAND  /* The argument command is invalid. */
PDC_ERR_NULL_PTR         /* The argument p_data_cfg or p_stat is a NULL pointer. */
PDC_ERR_RST_TIMEOUT      /* PDC reset was not canceled even after the specified amount of time elapsed. */
```

Properties

The declaration is located in r_pdc_rx_if.h.

Description

< PDC_CMD_CAPTURE_START command processing >
After reconfiguring interrupt conditions and resetting the PDC, enables PDC receive operation to start data capture.
< PDC_CMD_CHANGE_POS_AND_SIZE command processing >
After disabling PDC receive operation, resets the capture start position and capture size.
— Set the capture position and size in the horizontal direction to match the output characteristics of the image sensor used.
< PDC_CMD_STATUS_GET command processing >
Writes PDC status information to the pointer position indicated by argument p_stat.
< PDC_CMD_STATUS_CLR command processing >
Clears PDC status information indicated by argument p_stat.
< PDC_CMD_SET_INTERRUPT command >

After disabling PDC receive operation, resets PDC interrupts.

< PDC_CMD_DISABLE command >

Disables PDC receive operation.

< PDC_CMD_ENABLE command >

Enables PDC receive operation.

< PDC_CMD_RESET command processing >

After disabling PDC receive operation, resets the PDC.

Example

In the sample code two bytes are used to represent each dot of the image sensor output, so the horizontal dot count for the horizontal capture position and size is set to twice the actual value. The setting value should be modified as necessary to match the output characteristics of the actual image sensor used.

Case 1: Starting capture operation

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Unused */
pdc_stat_t               dummy_stat;

ret_pdc = R_PDC_Control(PDC_CMD_CAPTURE_START, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error Processing */
}
```

Case 2: Resetting the capture position and size

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Setting values of PDC operation */
pdc_data_cfg_t           data_pdc;
/* Unused */
pdc_stat_t               dummy_stat;

/* Capture from 0 pixel of vertical direction */
data_pdc.capture_pos.vst_position = 0;
/* Capture from 0 pixel of horizontal direction */
data_pdc.capture_pos.hst_position = 0;
/* Capture 480 pixels in vertical direction */
data_pdc.capture_pos.vsz_size = 480;
/* Capture 640 pixels in horizontal direction */
data_pdc.capture_pos.hsz_size = (640 * 2);

ret_pdc = R_PDC_Control(PDC_CMD_CHANGE_POS_AND_SIZE, &data_pdc, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 3: Getting the status

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Status values of PDC operation */
pdc_stat_t               stat_pdc;

ret_pdc = R_PDC_Control(PDC_CMD_STATUS_GET, &dummy_data, &stat_pdc);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 4: Clearing the status

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Status values of PDC operation */
pdc_stat_t               stat_pdc;

/* Clear Frame Busy Flag */
stat_pdc.pcsr_stat.frame_busy = true;
/* Clear FIFO Empty Flag */
stat_pdc.pcsr_stat.fifo_empty = true;
/* Clear Frame End Flag */
stat_pdc.pcsr_stat.frame_end = true;
/* Clear Overrun Flag */
stat_pdc.pcsr_stat.overrun = true;
/* Clear Underrun Flag */
stat_pdc.pcsr_stat.underrun = true;
/* Clear Vertical Line Number Setting Error Flag */
stat_pdc.pcsr_stat.verf_error = true;
/* Clear Horizonal Byte Number Setting Error Flag */
stat_pdc.pcsr_stat.herf_error = true;

ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 5: Resetting the interrupt settings

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Setting values of PDC operation */
pdc_data_cfg_t           p_data_pdc;
/* Unused */
pdc_stat_t               dummy_stat;

/* Update all of interrupt setting values with the contents of following */
data_pdc.iupd_select = PDC_ALL_INT_UPDATE;
/* PCDFI interrupt priority level is 8 */
data_pdc.priority.pcdfi_level = 8;
/* GROUPBL0 interrupt priority level is 2 */
data_pdc.priority.groupbl0_level = 2;
/* PCDFI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcdfi_ien = true;
/* PCFEI interrupt request in ICU is enabled */
data_pdc.inticu_req.pcfei_ien = true;
/* PCERI interrupt request in ICU is enabled */
data_pdc.inticu_req.pceri_ien = true;
/* Generation of receive data ready interrupt requests is enabled */
data_pdc.intpdc_req.dfie_ien = true;
/* Generation of frame end interrupt requests is enabled */
data_pdc.intpdc_req.feie_ien = true;
/* Generation of overrun interrupt requests is enabled */
data_pdc.intpdc_req.ovie_ien = true;
/* Generation of underrun interrupt requests is enabled */
data_pdc.intpdc_req.udrie_ien = true;
/* Generation of vertical line number setting error interrupt requests is enabled */
data_pdc.intpdc_req.verie_ien = true;
/* Generation of horizontal byte number setting error interrupt requests is enabled */
data_pdc.intpdc_req.herie_ien = true;

ret_pdc = R_PDC_Control(PDC_CMD_SET_INTERRUPT, &data_pdc, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 6: Disabling PDC receive operation only

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Unused */
pdc_stat_t               dummy_stat;

ret_pdc = R_PDC_Control(PDC_CMD_DISABLE, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 7: Enabling PDC receive operation only

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Unused */
pdc_stat_t               dummy_stat;

ret_pdc = R_PDC_Control(PDC_CMD_ENABLE, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Case 8: Resetting the PDC

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Error code of PDC FIT API */
volatile pdc_return_t    ret_pdc;
/* Unused */
pdc_data_cfg_t           dummy_data;
/* Unused */
pdc_stat_t               dummy_stat;

ret_pdc = R_PDC_Control(PDC_CMD_RESET, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* Error processing */
}
```

Special Notes:

Running this API function when receive operation is in progress will overwrite the PDC registers, thereby causing receive operation to stop. Since running this API function before the frame-end interrupt is

generated stops receive operation, capturing of image data is halted midway. To restart image capture, reset in the DMAC or DTC the pointer to the transfer destination in memory, then use the R_PDC_Control capture start command to restart capturing of image data.

When running R_PDC_Control with the command PDC_CMD_STATUS_CLR as an argument, set the status information to be cleared as “true” and the status information not to be cleared as “false”. If these settings are not made before running R_PDC_Control, status information may be cleared in an unintended manner.

3.4 R_PDC_GetFifoAddr()

This function gets the FIFO address of the PDC.

Format

```
pdc_return_t R_PDC_GetFifoAddr(
      uint32_t        *p_fifo_addr
)
```

Parameters

**p_fifo_*addr
Pointer to PDC FIFO address

Return Values

PDC_SUCCESS	*/* Processing finished successfully. */*
PDC_ERR_NOT_OPEN	*/* R_PDC_Open has not been run. **
PDC_ERR_NULL_PTR	*/* Argument p_fifo_addr is a NULL pointer. */*

Properties

The declaration is located in r_pdc_rx_if.h.

Description

Stores the address of the PDC receive data register (PCDR) in argument p_fifo_addr.

Example

Case 1: Example settings using the DMAC (RX Family DMA controller DMCA control module using Firmware Integration Technology)

```
#include “platform.h”
#include "r_pdc_rx_if.h"
#include "r_dmaca_rx_if.h"

/* Error code of PDC API */
volatile pdc_return_t     ret_pdc;
/* Error code of DMACA FIT API */
volatile dmaca_return_t   ret_dmac;
/* Setting values of dmaca_transfer information structure */
dmaca_transfer_data_cfg_t td_cfg;
/* Pointer to FIFO address of PDC */
uint32_t                  pdc_fifo_address;

/* Set PDC FIFO to DMACA transfer source address */
ret_pdc = R_PDC_GetFifoAddr(&pdc_fifo_address);
if (PDC_SUCCESS == ret_pdc)
{
    td_cfg.p_src_addr = pdc_fifo_address;
}
/* Set PCDFI to DMACA activation source */
td_cfg.act_source = IR_PDC_PCDFI;

ret_dmac = R_DMACA_Create (DMACA_CH0, &td_cfg);
if (DMACA_SUCCESS != ret_dmac)
{
    /* Error processing */
}
```

Case 2: Example settings using the DTC (RX Family DTC module using Firmware Integration Technology)

```
#include “platform.h”
#include “r_pdc_rx_if.h”
#include “r_dtc_rx_if.h”

/* Error code of PDC API */
volatile pdc_return_t    ret_pdc;
/* Error code of DTC FIT API */
volatile dtc_err_t       ret_dtc;
/* Activation source of DTC */
dtc_activation_source_t  act_source;
/* Pointer to start address of Transfer data area on RAM */
dtc_transfer_data_t     *p_transdata_dtc;
/* Pointer to setting values for transfer data */
dtc_transfer_data_cfg_t *p_data_dtc;
/* Pointer to FIFO address of PDC */
Uint32_t                 pdc_fifo_address;
/* Number of chain transfer */
uint32_t                 chain_trans_nr;

/* Set PCDFI to DTC Activation souce */
act_source = (dtc_activation_source_t)VECT_PDC_PCDFI;
/* Set PDC FIFO to DTC transfer source address */
ret_pdc = R_PDC_GetFifoAddr(&pdc_fifo_address);
if (PDC_SUCCESS == ret_pdc)
{
    p_data_dtc->source_addr = pdc_fifo_address;
}
/* Set 0 to number of chain transfer */
chain_trans_nr = 0;

ret_dtc = R_DTC_Create(act_source, p_transdata_dtc, p_data_dtc, chain_trans_nr);
if(DTC_SUCCESS != ret_dtc)
{
    /* Error processing */
}
```

Special Notes:

None

3.5 R_PDC_GetVersion()

This function returns the API version number.

Format

uint32_t R_PDC_GetVersion(void)

Parameters

None

Return Values

Version number

Properties

The declaration is located in r_pdc_rx_if.h.

Description

This function returns the version number of the currently installed PDC FIT module. The version number is encoded. The first two bytes contain the major version number and the last two bytes contain the minor version number. For example, if the version number is 4.25, the return value would be 0x00040019.

Example

```
#include “platform.h”
#include “r_pdc_rx_if.h”

/* Version number */
uint32_t version;

version = R_PDC_GetVersion();
```

Special Notes:

None

4. Pin Setting

To use the PDC FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC). The pin assignment is referred to as the “Pin Setting” in this document. Please perform the pin setting before calling the R_PDC_Open function.

When performing the Pin Setting in the e^2 studio, the Pin Setting feature of the FIT Configurator or the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT Configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.

Table 4.1 Function Output by the FIT Configurator and Smart Configurator

MCU Used	Function to be Output	Remarks
RX64M, RX65N, RX71M, RX72M	R_PDC_PinSet()	-

5. How to Use

5.1 API Usage Example

In the example presented below, the API is used to activate the DMAC and transfer input image data to the SDRAM. Example operation flowcharts and sample code are shown.

5.1.1 Example Operation Flowcharts

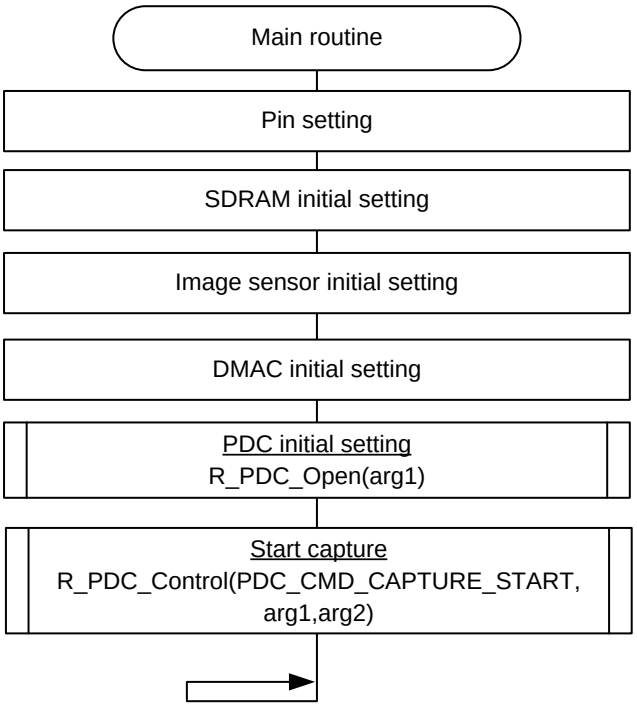

Figure 5.1 Example Operation Flowchart (1)

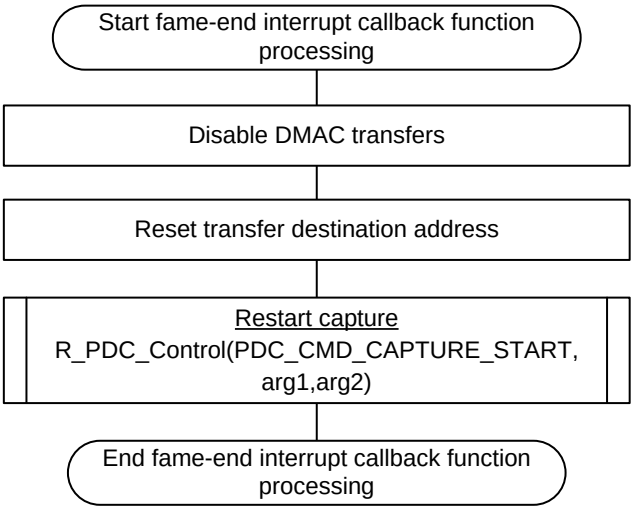

Figure 5.2 Example Operation Flowchart (2)

6. Appendices

6.1 Operation Confirmation Environment

This section describes operation confirmation environment for the PDC FIT module.

Table 6.1 Operation Confirmation Environment (Rev. 2.01)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 6.00.000
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.2.01
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2SxxxBE)

Table 6.2 Operation Confirmation Environment (Rev. 2.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.03.000
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endian	Big endian/little endian
Revision of the module	Rev.2.02

Table 6.3 Operation Confirmation Environment (Rev. 2.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.03
Board used	Renesas Starter Kit+ for RX64M (product No:RTK500564Mxxxxxx)

Table 6.4 Operation Confirmation Environment (Rev. 2.04)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.04
Board used	Renesas Starter Kit+ for RX72M (product No: RTK5572Mxxxxxxxxxx)

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_pdc_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_pdc_rx_config.h” may be wrong. Check the file “r_pdc_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7 Compile Settings for details.

(4) Q: PDC reset was not canceled even after the specified amount of time elapsed.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4 Pin Setting for details.

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
2.00	Oct. 1, 2016	—	First edition issued
2.01	Oct. 2, 2017	—	Supported RX65N-2MB version.
		5	2.4, Interrupt Vector, added
		12	2.12, Adding the FIT Module to Your Project, amended
		19	Special Notes in 3.1, R_PDC_Open(), amended
		37	4, Pin Setting, amended
		39	6.1, Operation Confirmation Environment, added
		39	6.2, Troubleshooting, added
2.02	Feb. 1, 2019	39	Table 6.2 Operation Confirmation Environment (Rev. 2.02), added
		—	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
2.03	May. 20, 2019	—	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Deleted R01AN1723, R01AN1826, R20AN0451 from Related Documents.
		1	Added Target Compilers.
		5	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		6	2.8 Code Size, amended.
		40	Table 6.3 Operation Confirmation Environment (Ver. 2.03), added.
2.04	Jul. 30, 2019	—	Supported RX72M version.
		1	Deleted R01AN1833 from Related Documents.
		5	Table 2.1 Interrupt Vector Used in the PDC FIT Module, amended.
		6	2.8 Code Size, amended.
		13	2.13 “for”, “while” and “do while” statements, added
		14-37	Delete “Reentrant” item on the API description page.
		38	Table 4.1 Function Output by the FIT Configurator and Smart Configurator, amended.
		41	Table 6.4 Operation Confirmation Environment (Ver. 2.04), added.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

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

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.