

# RX Family

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

R01AN2220EJ0100  
Rev.1.00  
Sep 03, 2014

### 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.

### Target Device

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

RX64M

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

### Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- RX64M Group User's Manual: Hardware (R01UH0377EJ)
- RX Family DMA Controller DMACA Control Module Using Firmware Integration Technology (R01AN2063EJ)
- RX Family DTC Module Using Firmware Integration Technology (R01AN1819EJ).

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

### Contents

1. Overview .....	2
2. API Information.....	3
3. API functions .....	8
4. Appendix .....	36

## 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.

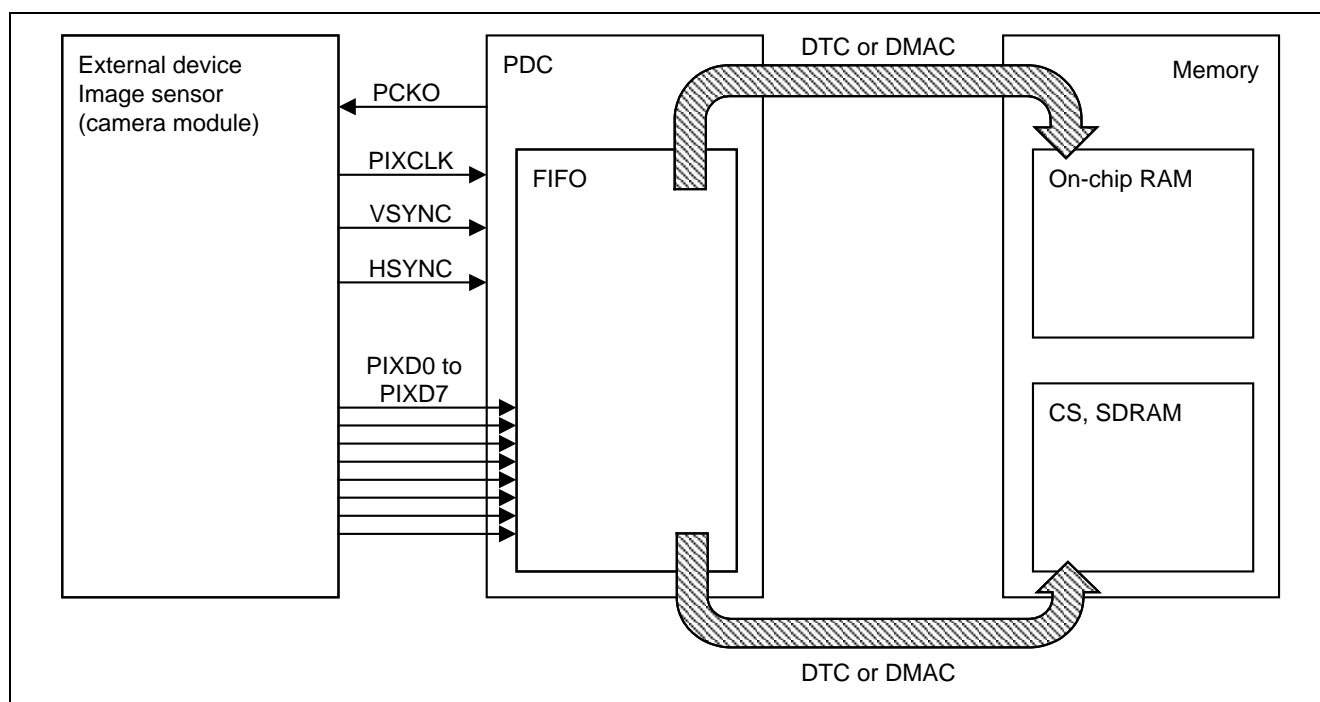


Figure 1.1 Overview of PDC

### 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 2.9, Adding the FIT Module.

### 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. Note that it is also necessary to prepare a control module, suitable for the image sensor to be used, to make settings to the image sensor.

## 2. API Information

This driver API follows 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 driver is dependent upon the following package:

- Renesas Board Support Package (r\_bsp) v2.60 or higher

---

### 2.3 Supported Toolchains

---

This driver has been confirmed to work with the following toolchain:

- Renesas RX Toolchain v.2.01.00

---

### 2.4 Header Files

---

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

---

### 2.5 Integer Types

---

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

---

### 2.6 Compile Time 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.

---

## 2.7 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 */
} pdc_ipr_dcfg_t;
```

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

```
/* PDC interrupt enable/disable */
typedef struct st_pdc_intpdc_data_cfg
{
    bool    dfie_iен;            /* Receive data-ready interrupt enabled */
    bool    feie_iен;            /* Frame-end interrupt enabled */
    bool    ovie_iен;            /* Overrun interrupt enabled */
    bool    udrie_iен;            /* Underrun interrupt enabled */
    bool    verie_iен;            /* Vertical line count setting error interrupt enabled */
    bool    herie_iен;            /* Horizontal byte count setting error interrupt 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_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;

```

```

/* R_PDC_IntWrite interrupt sources */
typedef enum e_pdc_interrupt_source
{
    PDC_INTERRUPT_PCDFI = 0,        /* Receive data-ready interrupt */
    PDC_INTERRUPT_PCFEI,           /* Frame-end interrupt */
    PDC_INTERRUPT_PCERI            /* Error interrupt */
} pdc_intsrc_t;

```

## 2.8 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_NOT_INITIALIZED,      /* PDC module uninitialized. R_PDC_Create has not been
                                   run. */
    PDC_ERR_INVALID_ARG,          /* Invalid argument input. */
    PDC_ERR_INVALID_COMMAND,      /* Command is invalid. */
    PDC_ERR_INVALID_INTERRUPT_SRC, /* Specified interrupt source is invalid. */
    PDC_ERR_INVALID_HANDLER_ADDR, /* Argument function address input is invalid.
                                   Registration of the previously registered function
                                   has been canceled. */
    PDC_ERR_NULL_PTR,             /* Argument pointer value was NULL. */
    PDC_ERR_BUSY,                 /* PDC resource is in use by another process. */
    PDC_ERR_INTERNAL              /* Module internal error detected. */
} pdc_return_t;

```

---

## 2.9 Adding Driver to Your Project

---

The FIT module must be added to each project in the e<sup>2</sup> studio.

You can use the FIT plug-in to add the FIT module to your project, or the module can be added manually.

It is recommended to use the FIT plug-in as you can add the module to your project easily and also it will automatically update the include file paths for you.

To add the FIT module using the plug-in, refer to chapter 2. “Adding FIT Modules to e<sup>2</sup> studio Projects Using FIT Plug-In” in the “Adding Firmware Integration Technology Modules to Projects” application note (R01AN1723EU).

To add the FIT module manually, refer to the following “Adding the PDC FIT Module (when not using the plug-in)”.

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the “Board Support Package Module Using Firmware Integration Technology” application note (R01AN1685EU).

### Adding the PDC FIT module (when not using the plug-in)

1. This application note is distributed with a zip file package that includes the PDC FIT module in its own folder `r_pdc_rx`.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the `r_pdc_rx` folder.
4. Open your e<sup>2</sup> Studio workspace.
5. In the e<sup>2</sup> Studio project explorer window, select the project that you want to add the PDC FIT module to.
6. Drag and drop the `r_pdc_rx` folder from the browser window into your e<sup>2</sup> Studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
  - a. Navigate to the “Add directory path” control:
    - i. ‘project name’ → properties → C/C++ Build → Settings → Compiler → Source -Add (green + icon)
  - b. Add the following paths:
    - i. “\${workspace\_loc}/\${ProjName}/r\_pdc\_rx”Whether you used the plug-in or manually added the package to your project, it is necessary to configure the module for your application.
8. Locate the `r_pdc_rx_config_reference.h` file in the `r_pdc_rx/ref` source folder in your project and copy it to the `r_config` folder in your project.
9. Change the name of the copy in the `r_pdc_rx_config_reference.h` to `r_pdc_config.h`.
10. Make the required configuration settings by editing the `r_pdc_rx_config.h` file. Refer to 2.6 “Compile Time Settings” for details.

### 3. API functions

---

#### 3.1 Summary

---

The following functions are included in this module:

Function	Description
R_PDC_Open	This function is used first to prepare the PDC for use. It makes initial settings to the PDC's registers as the first step in initializing the PDC.
R_PDC_Close	This function ends PDC operation and puts the PDC into the module stop state.
R_PDC_Create	This function makes settings to the PDC's registers. It makes initial settings to the PDC's registers as the second step in initializing the PDC.
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_IntWrite	This function registers callback functions for the interrupts used by the PDC.
R_PDC_GetVersion	This function returns the API version number.



### 3.2 R\_PDC\_Open()

This function is used first to prepare the PDC for use. It makes initial settings to the PDC's registers as the first step in initializing the PDC.

#### 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.
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).

#### Return Values

<i>PDC_SUCCESS</i>	<i>/* Processing finished successfully. */</i>
<i>PDC_ERR_OPEN</i>	<i>/* R_PDC_Open has already been run. */</i>
<i>PDC_ERR_INVALID_ARG</i>	<i>/* Parameter values in PDC setting information are invalid. */</i>
<i>PDC_ERR_NULL_PTR</i>	<i>/* Argument p_data_cfg is a NULL pointer. */</i>
<i>PDC_ERR_BUSY</i>	<i>/* The PDC has already been locked by another process. */</i>
<i>PDC_ERR_INTERNAL</i>	<i>/* A module internal error was detected. */</i>

#### Properties

The declaration is located in `r_pdc_rx_if.h`.

## Description

Performs the following processing to make initial settings for using the PDC:

- Locks the PDC hardware resource using the `r_bsp` hardware locking function.
- Cancels PDC module stop state.
- Makes initial settings to I/O ports used by the PDC.

The PDC uses the following ports:

Port	Pin Function
P24	PIXCLK (input)
P32	VSYNC (input)
P25	HSYNC (input)
P15	PIXD0 (input)
P86	PIXD1 (input)
P87	PIXD2 (input)
P17	PIXD3 (input)
P20	PIXD4 (input)
P21	PIXD5 (input)
P22	PIXD6 (input)
P23	PIXD7 (input)
P33	PCKO (output)

- Performs initial registration of callback functions for interrupts used by the PDC.

The following functions are initially registered as callback functions for interrupts used by the PDC\*<sup>1</sup>:

Interrupt Source	Function Name
Receive data-ready interrupt (PCDFI)	<code>callback_receive_data_ready</code>
Frame-end interrupt (PCFEI)	<code>callback_frame_end</code>
Error interrupt (PCERI)	<code>callback_error</code>

- Makes settings for interrupts used by the PDC.  
Receive data-ready interrupt (PCDFI), frame-end interrupt (PCFEI), and error interrupt (PCERI)\*<sup>2</sup>
- 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.”

## Reentrant

Impossible

**Example**

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            data_pdc;      /* PDC settings */

data_pdc.priority.pcdfi_level = 0;      /* Using DMAC*3 */
data_pdc.inticu_req.pcdfi_ien = true;    /* PCDFI interrupt enabled in interrupt controller */
data_pdc.inticu_req.pcfci_ien = true;    /* PCFPI interrupt enabled in interrupt controller */
data_pdc.inticu_req.pceri_ien = true;    /* PCERI interrupt enabled in interrupt controller */

ret_pdc = R_PDC_OPEN(&data_pdc);

if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}
```

**Special Notes:**

This API function makes initial settings as shown in the sample flowchart for initial setting of the PDC. For the sample flowchart, see 51.3.8, Initial Settings, in RX64M Group User's Manual: Hardware.

Before using the other API functions after a power-on reset, make sure to run this API function and confirm that the return value is PDC\_SUCCESS.

Note: 1. The initially registered functions (callback\_receive\_data\_ready, callback\_frame\_end, and callback\_error) perform interrupt and error handling as shown in the sample flowcharts in 51.3.9, Flows of Operations, in RX64M Group User's Manual: Hardware. To use these initially registered functions, add processing to stop DTC/DMAC transfers, initialize the transfer destination address, and perform any other tasks required on your end. Refer to the source code modification locations below when adding processing code.

If the initially registered functions will not be used, you will need to prepare callback processing code for each of the interrupts and reregister the callback functions using R\_PDC\_IntWrite after running this API function.

Note: 2. The frame-end interrupt (PCFPI) and error interrupt (PCERI) belong to a group interrupt (GROUPBL0). This group interrupt includes interrupts for modules other than the PDC. For this reason the FIT module does not perform settings for the group interrupt (GROUPBL0). When running this function it is necessary to make settings for the group interrupt (GROUPBL0) separately.

Note: 3. In the sample code the receive data-ready interrupt is set to level 0, based on Notes on Use of the DMAC. For details, see 51.4.5, Notes on Use of the DMAC, in RX64M Group User's Manual: Hardware.

## Source code modification locations

```
r_pdc_rx/src/r_pdc_rx.c

static void callback_receive_data_ready(void)
{
    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

Source code modification locations (continued)

r\_pdc\_rx/src/r\_pdc\_rx.c

```
static void callback_frame_end(void)
{
    volatile pdc_return_t      ret_pdc;
    pdc_data_cfg_t             dummy_data;
    pdc_stat_t                  stat_pdc;

    /* Check FIFO empty and UDRF flag. */
    do
    {
        ret_pdc = R_PDC_Control(PDC_CMD_STATUS_GET, &dummy_data, &stat_pdc);
        if (PDC_SUCCESS != ret_pdc)
        {
            /* do something */
        }
        if (true == stat_pdc.pcsr_stat.underrun)
        {
            /* Clear the PCSR.FEF flag to 0. */
            stat_pdc.pcsr_stat.fifo_empty = true;
            ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
            if (PDC_SUCCESS != ret_pdc)
            {
                /* do something */
            }
            error_processing();
            return;
        }
    }
    while (true != stat_pdc.pcsr_stat.fifo_empty);

    /* After completion of the transfer, clear the PCE bit in the PCCR1 register to 0. */
    ret_pdc = R_PDC_Control(PDC_CMD_DISABLE, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Clear the FEF flag in the PCSR register to 0. */
    stat_pdc.pcsr_stat.fifo_empty = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

Source code modification locations (continued)

r\_pdc\_rx/src/r\_pdc\_rx.c

```
static void error_processing(void)
{
    volatile pdc_return_t      ret_pdc;
    pdc_data_cfg_t            dummy_data;
    pdc_stat_t                stat_pdc;

    /* Disabling of PDC operation. */
    ret_pdc = R_PDC_Control(PDC_CMD_DISABLE, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Disable the DTC or DMAC transfer. Please edit such to call user function. */
    /* Call User Function */

    /* Read the PCSR register. */
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_GET, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Checking the overrun flag.(PCSR.OVRF) */
    if (true == stat_pdc.pcsr_stat.overrun)
    {
        /* call Overrun error processing. */
        overrun_error();

        /* Clear the PCSR.OVRF flag to 0. */
        stat_pdc.pcsr_stat.overrun = true;
        ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
        if (PDC_SUCCESS != ret_pdc)
        {
            /* do something */
        }
    }

    /* Checking the underrun flag.(PCSR.UDRF) */
    if (true == stat_pdc.pcsr_stat.underrun)
    {
        /* call Underrun error processing. */
        underrun_error();

        /* Clear the PCSR.UDRF flag to 0. */
        stat_pdc.pcsr_stat.underrun = true;
        ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
        if (PDC_SUCCESS != ret_pdc)
        {
            /* do something */
        }
    }
}
```

(Continues on following page.)

Source code modification locations (continued)

```
/* Checking the vertical line setting error flag.(PCSR.VERF) */
if (true == stat_pdc.pcsr_stat.verf_error)
{
    /* call Vertical line setting error processing. */
    vertical_line_setting_error();

    /* Clear the PCSR.VERF flag to 0. */
    stat_pdc.pcsr_stat.verf_error = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

/* Checking the horizontal byte setting error flag.(PCSR.HERF) */
if (true == stat_pdc.pcsr_stat.herf_error)
{
    /* call Horizontal byte setting error processing. */
    horizontal_byte_setting_error();

    /* Clear the PCSR.HERF flag to 0. */
    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)
    {
        /* do something */
    }
}

/* User needs to prepare processing for each, if your system need.
   Please edit such to call user function. */
/* Call user function. */
}
```

r\_pdc\_rx/src/r\_pdc\_rx.c

```
static void overrun_error(void)
{
    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

r\_pdc\_rx/src/r\_pdc\_rx.c

```
static void underrun_error(void)
{
    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

r\_pdc\_rx/src/r\_pdc\_rx.c

```
static void vertical_line_setting_error(void)
{
    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

Source code modification locations (continued)

`r_pdc_rx/src/r_pdc_rx.c`

```
static void horizontal_byte_setting_error(void)
{
    /* User needs to prepare processing for each, if your system need.
       Please edit such to call user function. */
    /* Call user function. */
}
```

### 3.3 R\_PDC\_Close()

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

#### Format

pd\_c\_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.

#### Reentrant

Impossible

#### Example

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pd_c_return_t    ret_pdc;    /* PDC API error codes */

ret_pdc = R_PDC_Close();

if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}
```

#### Special Notes:

Use this API function after running R\_PDC\_Open.



### 3.4 R\_PDC\_Create()

This function makes settings to the PDC registers. It makes initial settings to the PDC's registers as the second step in initializing the PDC.

#### Format

```

pdc_return_t R_PDC_Create(
    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
intpdc_req.dfie_ien	Receive data-ready interrupt request	false	PCCR0.DFIE	Disables generation of receive data-ready interrupt requests.
		true		Enables generation of receive data-ready interrupt requests.
intpdc_req.feie_ien	Frame-end interrupt request	false	PCCR0.FEIE	Disables generation of frame-end interrupt requests.
		true		Enables generation of frame-end interrupt requests.
intpdc_req.ovie_ien	Overrun interrupt request.	false	PCCR0.OVIE	Disables generation of overrun interrupt requests.
		true		Enables generation of overrun interrupt requests.
intpdc_req.udrie_ien	Underrun interrupt request	false	PCCR0.UDRIE	Disables generation of underrun interrupt requests.
		true		Enables generation of underrun interrupt requests.
intpdc_req.verie_ien	Vertical line count setting error interrupt request.	false	PCCR0.VERIE	Disables generation of vertical line count setting error interrupt requests.
		true		Enables generation of vertical line count setting error interrupt requests.
intpdc_req.herie_ien	Horizontal byte count setting error interrupt request	false	PCCR0.HERIE	Disables generation of horizontal byte count setting error interrupt requests.
		true		Enables generation of horizontal byte count setting error interrupt requests.

**Members of Referenced `pd_data_cfg_t` Structure and Their Setting Values (continued)**

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
vps_select	VSYNC signal polarity select	PDC_VSYNC_SIGNA L_POLARITY_HIGH	PCCR0.VPS	VSYNC signal is high-active.
		PDC_VSYNC_SIGNA L_POLARITY_LOW		VSYNC signal is low-active.
hps_select	HSYNC signal polarity select	PDC_HSYNC_SIGNA L_POLARITY_HIGH	PCCR0.HPS	HSYNC signal is high-active.
		PDC_HSYNC_SIGNA L_POLARITY_LOW		HSYNC signal is low-active.
vst_position	Vertical capture start line position	12-bit data 0000h to 0FFEh	VCR.VST	Vertical capture start line position
hst_position	Horizontal capture start byte position	12-bit data 0000h to 0FFBh	HCR.HST	Horizontal capture start byte position
vsz_size	Vertical capture size	12-bit data 0001h to 0FFFh	VCR.VSZ	Vertical capture line count
hsz_size	Horizontal capture size	12-bit data 0004h to 0FFFh	HCR.HSZ	Horizontal capture byte count

**Return Values**

<code>PDC_SUCCESS</code>	<i>/* Processing finished successfully. */</i>
<code>PDC_ERR_NOT_OPEN</code>	<i>/* R_PDC_Open has not been run. */</i>
<code>PDC_ERR_INVALID_ARG</code>	<i>/* Parameter values in PDC setting information are invalid. */</i>
<code>PDC_ERR_NULL_PTR</code>	<i>/* Argument <code>p_data_cfg</code> is a NULL pointer. */</i>

**Properties**

The declaration is located in `r_pdc_rx_if.h`.

**Description**

References the PDC setting information structure specified by the argument and makes initial settings for using the PDC.

- Enables PIXCLK input (PCCR0.PCKE).<sup>\*4</sup>
- Resets the PDC (PCCR0.PRST).<sup>\*4</sup>
- Sets the vertical and horizontal capture ranges (VCR and HCR settings).
- Sets the VSYNC and HSYNC signal polarity (VPS and HPS).
- Makes interrupt enable/disable settings (DFIE, FEIE, OVIE, UDRIE, VERIE, and HERIE).
- Sets the endianness (EDS).<sup>\*5</sup>

**Reentrant**

Possible

## Example

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

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;          /* PDC API error codes */
pdc_data_cfg_t            data_pdc;         /* PDC settings */

data_pdc.intpdc_req.dfie_ien = true;        /* Receive data-ready interrupt enabled */
data_pdc.intpdc_req.feie_ien = true;        /* Frame-end interrupt enabled */
data_pdc.intpdc_req.ovie_ien = true;        /* Overrun error interrupt enabled */
data_pdc.intpdc_req.udrie_ien = true;       /* Underrun error interrupt enabled */
data_pdc.intpdc_req.verie_ien = true;       /* Vertical line count setting error
                                             interrupt enabled */
data_pdc.intpdc_req.herie_ien = true;       /* Horizontal byte count setting error
                                             interrupt enabled */

data_pdc.vps_select = PDC_VSYNC_SIGNAL_POLARITY_LOW; /* VSYNC is low-active. */
data_pdc.hps_select = PDC_HSYNC_SIGNAL_POLARITY_HIGH; /* HSYNC is high-active. */
data_pdc.capture_pos.vst_position = 0;       /* Vertical capture starts from pixel
                                             numbered 0. */
data_pdc.capture_pos.hst_position = 0;       /* Horizontal capture starts from pixel
                                             numbered 0.*6 */
data_pdc.capture_pos.vsz_size = 480;        /* Vertical capture of 480 pixels */
data_pdc.capture_pos.hsz_size = (640 * 2);  /* Horizontal capture of 640 pixels*6 */
ret_pdc = R_PDC_Create(&data_pdc);

if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}
```

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"

volatile pdc_return_t      ret_pdc;          /* PDC API error codes */
pdc_data_cfg_t            data_pdc;         /* PDC settings */

data_pdc.intpdc_req.dfie_ien = true;        /* Receive data-ready interrupt enabled */
data_pdc.intpdc_req.feie_ien = true;        /* Frame-end interrupt enabled */
data_pdc.intpdc_req.ovie_ien = true;        /* Overrun error interrupt enabled */
data_pdc.intpdc_req.udrie_ien = true;       /* Underrun error interrupt enabled */
data_pdc.intpdc_req.verie_ien = true;       /* Vertical line count setting error
                                             interrupt enabled */
data_pdc.intpdc_req.herie_ien = true;       /* Horizontal byte count setting error
                                             interrupt enabled */

data_pdc.vps_select = PDC_VSYNC_SIGNAL_POLARITY_LOW; /* VSYNC is low-active. */
data_pdc.hps_select = PDC_HSYNC_SIGNAL_POLARITY_HIGH; /* HSYNC is high-active. */
data_pdc.capture_pos.vst_position = 240;     /* Vertical capture starts from pixel numbered
                                             240. */
data_pdc.capture_pos.hst_position = (320 * 2); /* Horizontal capture starts from pixel
                                             numbered 320.*6 */
data_pdc.capture_pos.vsz_size = 240;        /* Vertical capture of 240 pixels */
data_pdc.capture_pos.hsz_size = (320 * 2);  /* Horizontal capture of 320 pixels*6 */
ret_pdc = R_PDC_Create(&data_pdc);

if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}
```

**Special Notes:**

This API function performs processing as shown in 51.3.8, Initial Settings, in RX64M Group User's Manual: Hardware. Use this API function after running R\_PDC\_Open.

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.

Note: 4. This API function enables PIXCLK input and resets the PDC. Running this API function in a state where PIXCLK is not being input will result in a failure of the reset to complete, so make sure that PIXCLK is being input when this API function is run.

Note: 5. This API function makes endian settings. The endianness is selected to match the compiler setting. If the compiler's endian setting specifies little-endian, the PDC's endian setting will also be little-endian, and if compiler's endian setting specifies big-endian, the PDC's endian setting will also be big-endian.

Note: 6. The sample code uses two bytes of data to represent each dot of image sensor output, so the horizontal capture position and size are set to values equal to twice the horizontal byte count. These values should be revised to match the output characteristics of the image sensor used, if necessary.

### 3.5 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.

<b>Structure Member</b>	<b>Summary</b>	<b>Setting Value</b>	<b>Setting Target Register</b>	<b>Setting Description</b>
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.overflow	Overflow flag	false	PCSR.OVRF	FIFO overflow has not been generated.
		true		FIFO overflow 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_busy	Frame-busy flag	false	PCSR.FBSY	Does nothing.
		true		Clears the frame-busy flag.
pcsr_stat.fifo_empty	FIFO-empty flag	false	PCSR.FEMPF	Does nothing.
		true		Clears the FIFO-empty flag.
pcsr_stat.frame_end	Frame-end flag	false	PCSR.FEF	Does nothing.
		true		Clears the frame-end flag.
pcsr_stat.overflow	Overflow flag	false	PCSR.OVRF	Does nothing.
		true		Clears the overflow 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 `pd_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
<code>iupd_select</code>	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: PCDFI interrupt enabled Bit 2: PCFEI interrupt enabled Bit 3: PCERI interrupt enabled Bit 4: Receive data-ready interrupt request Bit 5: Frame-end interrupt request Bit 6: Overrun interrupt request Bit 7: Underrun interrupt request Bit 8: Vertical line number setting error interrupt request Bit 9: Horizontal byte number setting error interrupt request Bits 10 to 15: Not used 0: Do not update setting. 1: Update setting.
<code>priority.pcdfi_level</code>	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 <code>iupd_select</code> to 1.
<code>inticu_req.pcdfi_ien</code>	PCDFI interrupt enabled	false	ICU.IER0C.IEN1	Disables receive data-ready interrupt (PCDFI) interrupt requests. Note: Set bit 1 in <code>iupd_select</code> to 1.
		true		Enables receive data-ready interrupt (PCDFI) interrupt requests. Note: Set bit 1 in <code>iupd_select</code> to 1.
<code>inticu_req.pcfei_ien</code>	PCFEI interrupt enabled	false	ICU.GRPBL0.EN30	Disables frame-end interrupt (PCFEI) interrupt requests. Note: Set bit 2 in <code>iupd_select</code> to 1.
		true		Enables frame-end interrupt (PCFEI) interrupt requests. Note: Set bit 2 in <code>iupd_select</code> to 1.
<code>inticu_req.pceri_ien</code>	PCERI interrupt enabled	false	ICU.GRPBL0.EN31	Disables error interrupt (PCERI) interrupt requests. Note: Set bit 3 in <code>iupd_select</code> to 1.
		true		Enables error interrupt (PCERI) interrupt requests. Note: Set bit 3 in <code>iupd_select</code> to 1.

**Members of Referenced pdc\_data\_cfg\_t Structure and Their Setting Values (continued)**

Structure Member	Summary	Setting Value	Setting Target Register	Setting Description
intpdc_req.dfie_ien	Receive data-ready interrupt request	false	PCCR0.DFIE	Disables generation of receive data-ready interrupt requests. Note: Set bit 4 in iupd_select to 1.
		true		Enables generation of receive data-ready interrupt requests. Note: Set bit 4 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 5 in iupd_select to 1.
		true		Enables generation of frame-end interrupt requests. Note: Set bit 5 in iupd_select to 1.
intpdc_req.ovie_ien	Overrun interrupt request	false	PCCR0.OVIE	Disables generation of overrun interrupt requests. Note: Set bit 6 in iupd_select to 1.
		true		Enables generation of overrun interrupt requests. Note: Set bit 6 in iupd_select to 1.
intpdc_req.udrie_ien	Underrun interrupt request	false	PCCR0.UDRIE	Disables generation of underrun interrupt requests. Note: Set bit 7 in iupd_select to 1.
		true		Enables generation of underrun interrupt requests. Note: Set bit 7 in iupd_select to 1.
intpdc_req.verie_ien	Vertical line number setting error interrupt request	false	PCCR0.VERIE	Disables generation of vertical line number setting error interrupt requests. Note: Set bit 8 in iupd_select to 1.
		true		Enables generation of vertical line number setting error interrupt requests. Note: Set bit 8 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 9 in iupd_select to 1.
		true		Enables generation of horizontal byte number setting error interrupt requests. Note: Set bit 9 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**

<i>PDC_SUCCESS</i>	<i>/* Processing finished successfully. */</i>
<i>PDC_ERR_NOT_OPEN</i>	<i>/* R_PDC_Open has not been run. */</i>
<i>PDC_ERR_NOT_INITIALIZED</i>	<i>/* R_PDC_Create has not been run. */</i>
<i>PDC_ERR_INVALID_ARG</i>	<i>/* Setting value applied to PDC register is invalid. */</i>
<i>PDC_ERR_INVALID_COMMAND</i>	<i>/* The argument command is invalid. */</i>
<i>PDC_ERR_NULL_PTR</i>	<i>/* The argument p_data_cfg or p_stat is a NULL pointer. */</i>

**Properties**

The declaration is located in r\_pdc\_rx\_if.h.

**Description**

- < *PDC\_CMD\_CAPTURE\_START* command processing >  
After resetting interrupt settings and the PDC, enables PDC receive operation to start data capture.
- < *PDC\_CMD\_CHANGE\_POS\_AND\_SIZE* command processing >  
Resets the capture start position and size settings.\*<sup>7</sup>
- < *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 >  
Resets PDC interrupts.\*<sup>8</sup>
- < *PDC\_CMD\_DISABLE* command >  
Disables PDC receive operation.
- < *PDC\_CMD\_ENABLE* command >  
Enables PDC receive operation.
- < *PDC\_CMD\_RESET* command processing >  
Resets the PDC.

**Reentrant**

Possible

## Example

Case 1: Starting capture operation

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            dummy_data;    /* Not used */
pdc_stat_t                dummy_stat;    /* Not used */

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

Case 2: Resetting the capture position and size

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            data_pdc;      /* PDC settings */
pdc_stat_t                dummy_stat;    /* Not used */

data_pdc.capture_pos.vst_position = 0;    /* Vertical capture starts from pixel numbered 0. */
data_pdc.capture_pos.hst_position = 0;    /* Horizontal capture starts from pixel numbered 0. */
data_pdc.capture_pos.vsz_size = 480;      /* Vertical capture of 480 pixels */
data_pdc.capture_pos.hsz_size = (640 * 2); /* Horizontal capture of 640 pixels*9 */

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

Case 3: Getting the status

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            dummy_data;    /* Not used */
pdc_stat_t                stat_pdc;      /* PDC status */

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

## Case 4: Clearing the status

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            dummy_data;    /* Not used */
pdc_stat_t                stat_pdc;      /* PDC status */

stat_pdc.pcsr_stat.frame_busy = true;    /* Clears the frame-busy flag. */
stat_pdc.pcsr_stat.fifo_empty = true;    /* Clears the FIFO-empty flag. */
stat_pdc.pcsr_stat.frame_end = true;     /* Clears the frame-end flag. */
stat_pdc.pcsr_stat.overrun = true;       /* Clears the overrun flag. */
stat_pdc.pcsr_stat.underrun = true;      /* Clears the underrun flag. */
stat_pdc.pcsr_stat.verf_error = true;    /* Clears the vertical line number setting error flag. */
stat_pdc.pcsr_stat.herf_error = true;    /* Clears the horizontal byte number setting error flag. */

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

## Case 5: Resetting the interrupt settings

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */
pdc_data_cfg_t            p_data_pdc;    /* PDC settings */
pdc_stat_t                dummy_stat;    /* Not used */

data_pdc.iupdc_select = PDC_ALL_INT_UPDATE; /* Updates settings for all interrupts as follows: */
data_pdc.priority.pcdfi_level = 8;          /* Sets the PCDFI interrupt priority level to 8. */
data_pdc.inticu_req.pcdfi_ien = true;       /* Enables the PCDFI interrupt in the interrupt controller */
data_pdc.inticu_req.pcfci_ien = true;       /* Enables the PCFIEI interrupt in the interrupt controller */
data_pdc.inticu_req.pceri_ien = true;       /* Enables the PCERI interrupt in the interrupt controller */
data_pdc.intpdc_req.dfie_ien = true;        /* Receive data-ready interrupt enabled */
data_pdc.intpdc_req.feie_ien = true;        /* Frame-end interrupt enabled */
data_pdc.intpdc_req.ovie_ien = true;        /* Overrun error interrupt enabled */
data_pdc.intpdc_req.udrie_ien = true;       /* Underrun error interrupt enabled */
data_pdc.intpdc_req.verie_ien = true;       /* Vertical line number setting error interrupt enabled */
data_pdc.intpdc_req.herie_ien = true;       /* Horizontal byte number setting error interrupt enabled */

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

## Case 6: Disabling PDC receive operation only

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t    ret_pdc;    /* PDC API error codes */
pdc_data_cfg_t          dummy_data; /* Not used */
pdc_stat_t              dummy_stat; /* Not used */

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

## Case 7: Enabling PDC receive operation only

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t    ret_pdc;    /* PDC API error codes */
pdc_data_cfg_t          dummy_data; /* Not used */
pdc_stat_t              dummy_stat; /* Not used */

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

## Case 8: Resetting the PDC

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t    ret_pdc;    /* PDC API error codes */
pdc_data_cfg_t          dummy_data; /* Not used */
pdc_stat_t              dummy_stat; /* Not used */

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

**Special Notes:**

Use this API function after running R\_PDC\_Open and R\_PDC\_Create.

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.

Note: 7. Set the horizontal capture position and size to match the output characteristics of the image sensor used.

Note: 8. The frame-end interrupt (PCFEI) and error interrupt (PCERI) belong to a group interrupt (GROUPBL0). This group interrupt includes interrupts for modules other than the PDC. For this reason the FIT module does not perform settings for the group interrupt (GROUPBL0). When running this function it is necessary to make settings for the group interrupt (GROUPBL0) separately.

Note: 9. The sample code uses two bytes of data to represent each dot of image sensor output, so the horizontal capture position and size are set to values equal to twice the horizontal byte count. These values should be revised to match the output characteristics of the image sensor used, if necessary.

---

### 3.6 R\_PDC\_GetFifoAddr()

---

This function gets the FIFO address of the PDC.

#### Format

void\* R\_PDC\_GetFifoAddr(void)

#### Parameters

None

#### Return Values

*Address of FIFO (PDC receive data register)*

#### Properties

The declaration is located in r\_pdc\_rx\_if.h.

#### Description

Returns the address of the PDC receive data register (PCDR).

#### Reentrant

Possible

#### Example

Case 1: Using the DMAC

```
#include "platform.h"
#include "r_pdc_rx_if.h"
#include "r_dmaca_api_rx_if.h"

volatile dmaca_return_t    ret_dmac;    /* PDC API error codes */
dmaca_transfer_data_cfg_t data_dmac;    /* DMAC settings */

/* Sets the PDC's FIFO as the DMAC transfer source address. */
data_dmac.p_src_addr = R_PDC_GetFifoAddr();
/* Sets the PDC's receive data-ready interrupt (PCDFI) as the DMA transfer activation source. */
data_dmac.act_source = IR_PDC_PCDFI;
/* Other parameter settings are omitted. */

ret_dmac = R_DMACA_Create(DMACA_CH0, &data_dmac);
if (DMAC_SUCCESS != ret_dmac)
{
    /* do something */
}
```

## Case 2: Using the DTC

```
#include "platform.h"
#include "r_pdc_rx_if.h"
#include "r_dtc_api_rx_if.h"

volatile dtc_err_t      ret_dtc;          /* PDC API error codes */
dtc_activation_source_t act_source;       /* DTC activation source */
dtc_transfer_data_cfg_t data_dtc;        /* DTC settings */
dtc_transfer_data_t     *p_trandata_dtc; /* DTC transfer information storage area */
uint32_t                ch_trans_nr;     /* Chain transfer count */

/* Sets the PDC's receive data-ready interrupt (PCDFI) as the DTC transfer activation source. */
act_source = (dtc_activation_source_t)VECT_PDC_PCDFI;
/* Sets the RAM area (dtc_pdcfi) defined in the transfer information allocation destination. */
p_trandata_dtc = (dtc_transfer_data_t *)&dtc_pdcfi;
/* Sets the PDC's FIFO as the DTC transfer source address. */
data_dtc.source_addr = (uint32_t)R_PDC_GetFifoAddr();
/* Specifies 0 because chain transfer is not supported. */
ch_trans_nr = 0;
/* Other parameter settings are omitted. */

ret_dtc = R_DTC_Create(act_source, p_trandata_dtc, &data_dtc, ch_trans_nr);
if (DTC_SUCCESS != ret_dtc)
{
    /* do something */
}
```

**Special Notes:**

None



### 3.7 R\_PDC\_IntWrite()

This function registers callback functions for the following interrupts used by the PDC:

- Receive data-ready interrupt (PCDFI)
- Frame-end interrupt (PCFEI)
- Error interrupt (PCERI)

#### Format

```

pdc_return_t R_PDC_IntWrite(
    pdc_intsrc_t      interrupt_src,
    void              *p_callback
)

```

#### Parameters

*interrupt\_src*

PDC interrupt source

*\*p\_callback*

Pointer to function called when PDC interrupt occurs

#### The Interrupt src Values:

*/\* Receive data-ready interrupt \*/*

*PDC\_INTERRUPT\_PCDFI*

*/\* Frame-end interrupt \*/*

*PDC\_INTERRUPT\_PCFEI*

*/\* Error interrupt \*/*

*PDC\_INTERRUPT\_PCERI*

#### Return Values

*PDC\_SUCCESS*

*/\* Processing finished successfully. \*/*

*PDC\_ERR\_INVALID\_INTERRUPT\_ID*

*/\* A nonexistent interrupt source was specified. \*/*

*PDC\_ERR\_INVALID\_HANDLER\_ADDR*

*/\* An invalid function address was input, or the function was previously registered but the registration has been canceled. \*/*

*PDC\_ERR\_INTERNAL*

*/\* A module internal error was detected. \*/*

#### Properties

The declaration is located in `r_pdc_rx_if.h`.

#### Description

This function registers a callback function for the specified interrupt source.

Even if the interrupt source is specified correctly, `FIT_NO_FUNC` (reserved area) is reregistered as the callback function if the pointer to the callback function is `FIT_NO_FUNC` (reserved area) or `NULL`, and the registration of the previously registered callback function is canceled.

If an interrupt occurs while `FIT_NO_FUNC` is registered as the callback function, a bus error interrupt is generated. This means that the callback function has not been correctly registered, so you should revise your program code to ensure that the callback function is registered correctly.

#### Reentrant

Possible

## Example

Case 1: Registering a callback function for the receive data-ready interrupt

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */

ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCDFI, pdc_pcdfi_callback);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}

void pdc_pcdfi_callback(void)
{
    /* do something */
}
```

Case 2: Registering a callback function for the frame-end interrupt

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */

ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCFEI, pdc_pcfei_callback);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}

void pdc_pcfei_callback(void)
{
    /* do something */
}
```

Case 3: Registering a callback function for the error interrupt

```
#include "platform.h"
#include "r_pdc_rx_if.h"

volatile pdc_return_t      ret_pdc;      /* PDC API error codes */

ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCERI, pdc_pceri_callback);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}

void pdc_pceri_callback(void)
{
    /* do something */
}
```

## Special Notes:

Ensure that the arguments and return values of the registered callback function are of the type void.

### 3.8 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.

#### Reentrant

Possible

#### Example

```
#include "platform.h"
#include "r_pdc_rx_if.h"

uint32_t version;    /* Version number */

version = R_PDC_GetVersion();
```

#### Special Notes:

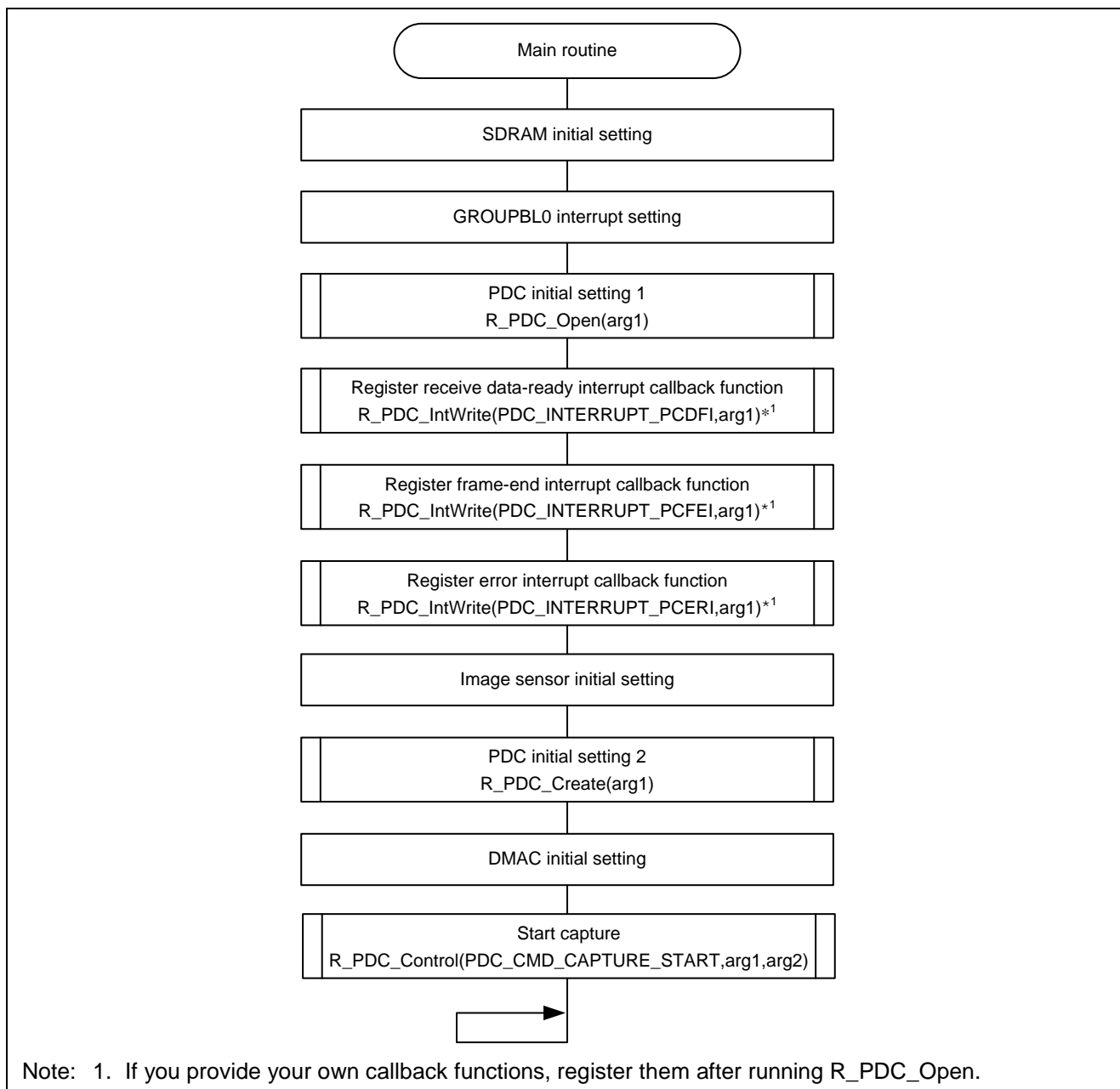
None

## 4. Appendix

### 4.1 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.

#### (1) Example Operation Flowcharts



**Figure 4.1 Example Operation Flowchart (1)**

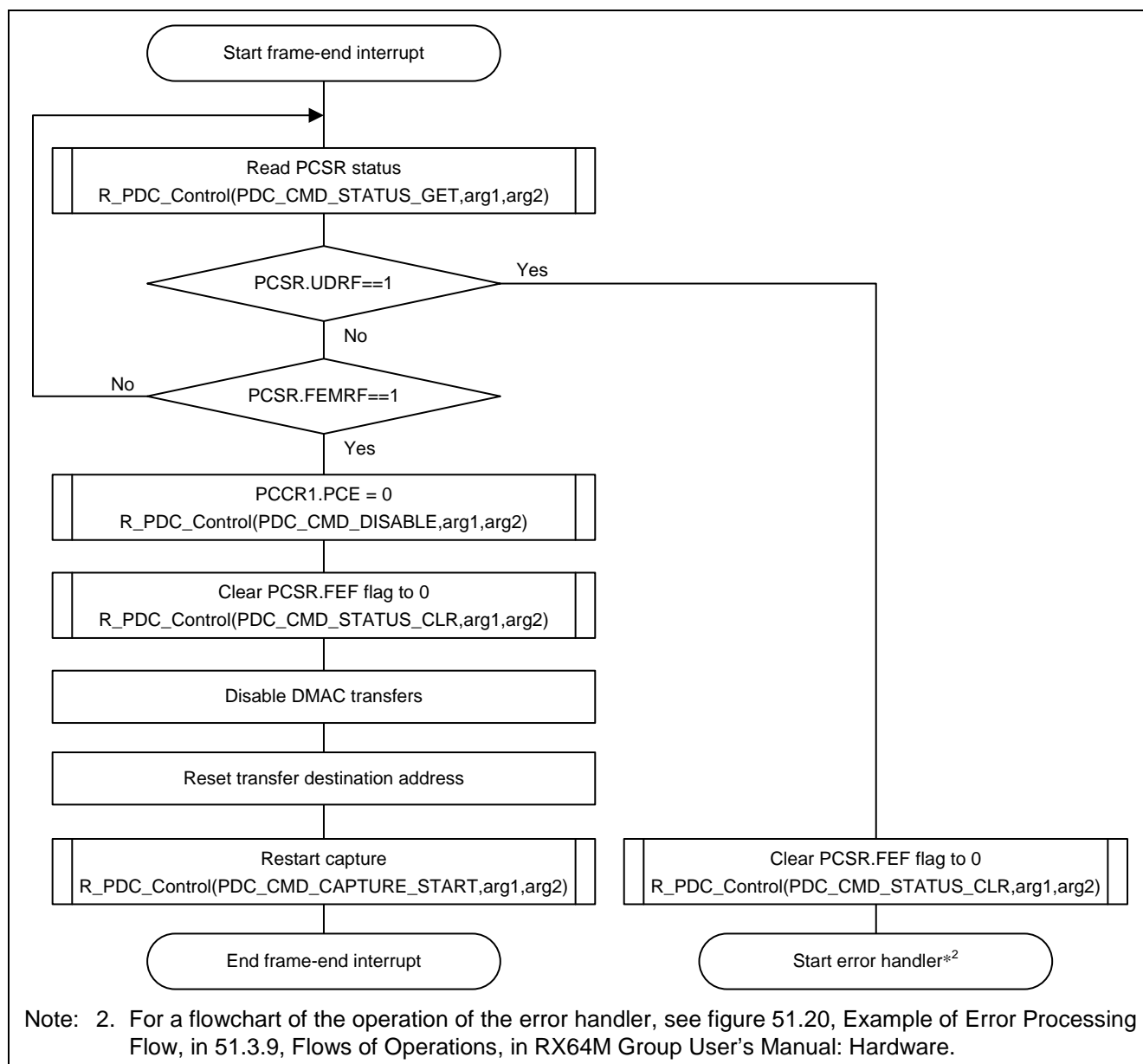


Figure 4.2 Example Operation Flowchart (2)

## (2) Sample Code

```

#include "platform.h"
#include "r_pdc_rx_if.h"
#include "dmac.h"

/*****
*****
* Function Name: main
* Description   : The main loop.
* Arguments     : none
* Return Value  : none
*****
*****/
void main(void)
{
    uint32_t ipl = 0;
    volatile pdc_return_t ret_pdc;           /* PDC API error codes */
    pdc_data_cfg_t data_pdc;                /* PDC settings */
    pdc_data_cfg_t dummy_data;              /* Not used */
    pdc_stat_t dummy_stat;                  /* Not used */

    /* SDRAM initial settings */
    initialize_sdram();

    /* Make GROUPBL0 interrupt settings: Set the GROUPBL0 interrupt priority level and
       enable GROUPBL0 interrupts. */
    ipl = 8;
    R_BSP_InterruptControl(BSP_INT_SRC_BL0_PDC_PCFEI, BSP_INT_CMD_GROUP0_INTERRUPT_ENABLE,
(void *)&ipl);

    /* PDC initial setting 1 */
    data_pdc.priority.pcdfi_level = 0;       /* Using DMAC */
    data_pdc.inticu_req.pcdfi_ien = true;    /* PCDFI interrupt enabled in interrupt controller */
    data_pdc.inticu_req.pcfci_ien = true;    /* PCFEI interrupt enabled in interrupt controller */
    data_pdc.inticu_req.pceri_ien = true;    /* PCERI interrupt enabled in interrupt controller */

    ret_pdc = R_PDC_OPEN(&data_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Register the receive data-ready interrupt callback function.
       (Implement if necessary.) */
    ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCDFI, pdc_pcdfi_callback);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Register the frame-end interrupt callback function. (Implement if necessary.) */
    ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCFEI, pdc_pcfci_callback);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Register the error interrupt callback function. (Implement if necessary.) */
    ret_pdc = R_PDC_IntWrite(PDC_INTERRUPT_PCERI, pdc_pceri_callback);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

```

```

/* Image sensor initial settings */
initialize_imagesensor();

/* PDC initial setting 2 */
data_pdc.intpdc_req.dfie_ien = true;
data_pdc.intpdc_req.feie_ien = true;
data_pdc.intpdc_req.ovie_ien = true;
data_pdc.intpdc_req.udrie_ien = true;
data_pdc.intpdc_req.verie_ien = true;
data_pdc.intpdc_req.herie_ien = true;
data_pdc.vps_select = PDC_VSYNC_SIGNAL_POLARITY_LOW;
data_pdc.hps_select = PDC_HSYNC_SIGNAL_POLARITY_HIGH;
data_pdc.capture_pos.vst_position = 0;
data_pdc.capture_pos.hst_position = 0;
data_pdc.capture_size.vsz_size = 480;
data_pdc.capture_size.hsz_size = (640 * 2);

ret_pdc = R_PDC_Create(&data_pdc);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}

/* DMAC initial settings */
DMAC_init();

/* Start capture */
ret_pdc = R_PDC_Control(PDC_CMD_CAPTURE_START, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}

while (1)
{
    /* do something */
}
} /* End of function main() */

/*****
*****
* Function Name: pdc_pcdfi_callback
* Description   : Receive data-ready interrupt callback function
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_pcdfi_callback(void)
{
    /* Don't do anything because interrupts are disabled during DMAC transfer. */
} /* End of function pdc_pcdfi_callback() */

```

```

/*****
*****
* Function Name: pdc_pcfei_callback
* Description  : Frame-end interrupt callback function
* Arguments    : none
* Return Value : none
*****
*****/
void pdc_pcfei_callback(void)
{
    volatile pdc_return_t  ret_pdc;          /* PDC API error codes */
    pdc_data_cfg_t         dummy_data;       /* Not used */
    pdc_stat_t             stat_pdc;         /* PDC status */

    /* Check FIFO empty and UDRF flag. */
    do
    {
        ret_pdc = R_PDC_Control(PDC_CMD_STATUS_GET, &dummy_data, &stat_pdc);
        if (PDC_SUCCESS != ret_pdc)
        {
            /* do something */
        }
        if (true == stat_pdc.pcsr_stat.underrun)
        {
            /* Clear the PCSR.FEF flag to 0. */
            stat_pdc.pcsr_stat.fifo_empty = true;
            ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
            if (PDC_SUCCESS != ret_pdc)
            {
                /* do something */
            }
            /* Start error handler */
            pdc_error_processing();
            return;
        }
    }
    while (true != stat_pdc.pcsr_stat.fifo_empty);

    /* After completion of the transfer, clear the PCE bit in the PCCR1 register to 0. */
    ret_pdc = R_PDC_Control(PDC_CMD_DISABLE, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Clear the FEF flag in the PCSR register to 0. */
    stat_pdc.pcsr_stat.fifo_empty = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Disable DMAC transfers */
    DMAC_stop();

    /* Reset transfer destination address */
    setting_memory();
}

```



```

/* Restart capture */
ret_pdc = R_PDC_Control(PDC_CMD_CAPTURE_START, &dummy_data, &dummy_stat);
if (PDC_SUCCESS != ret_pdc)
{
    /* do something */
}
} /* End of function pdc_pcfei_callback() */

/*****
*****
* Function Name: pdc_pceri_callback
* Description   : Error interrupt callback function
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_pceri_callback(void)
{
    /* Start error handler */
    pdc_error_processing();
} /* End of function pdc_pceri_callback() */

/*****
*****
* Function Name: pdc_error_processing
* Description   : Error handler
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_error_processing(void)
{
    volatile pdc_return_t  ret_pdc;           /* PDC API error codes */
    pdc_data_cfg_t         dummy_data;       /* Not used */
    pdc_stat_t             stat_pdc;         /* PDC status */

    /* Disabling of PDC operation. */
    ret_pdc = R_PDC_Control(PDC_CMD_DISABLE, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }

    /* Disable DMAC transfers */
    DMAC_stop();

    /* Read the PCSR register. */
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_GET, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

```

```
/* Checking the overrun flag.(PCSR.OVRF) */
if (true == stat_pdc.pcsr_stat.overrun)
{
    /* call Overrun error processing. */
    pdc_overrun_error();

    /* Clear the PCSR.OVRF flag to 0. */
    stat_pdc.pcsr_stat.overrun = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

/* Checking the underrun flag.(PCSR.UDRF) */
if (true == stat_pdc.pcsr_stat.underrun)
{
    /* call Underrun error processing. */
    pdc_underrun_error();

    /* Clear the PCSR.UDRF flag to 0. */
    stat_pdc.pcsr_stat.underrun = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

/* Checking the vertical line setting error flag.(PCSR.VERF) */
if (true == stat_pdc.pcsr_stat.verf_error)
{
    /* call Vertical line setting error processing. */
    pdc_vertical_line_setting_err();

    /* Clear the PCSR.VERF flag to 0. */
    stat_pdc.pcsr_stat.verf_error = true;
    ret_pdc = R_PDC_Control(PDC_CMD_STATUS_CLR, &dummy_data, &stat_pdc);
    if (PDC_SUCCESS != ret_pdc)
    {
        /* do something */
    }
}

/* Checking the horizontal byte setting error flag.(PCSR.HERF) */
if (true == stat_pdc.pcsr_stat.herf_error)
{
    /* call Horizontal byte setting error processing. */
    pdc_horizontal_byte_setting_err();

    /* Clear the PCSR.HERF flag to 0. */
    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)
    {
        /* do something */
    }
}

/* The sample code does nothing when an error occurs. */
} /* End of function pdc_error_processing() */
```

```

/*****
*****
* Function Name: pdc_overflow_error
* Description   : Overflow error handler
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_overflow_error(void)
{
    /* The sample code does nothing when an error occurs. */
} /* End of function pdc_overflow_error() */

/*****
*****
* Function Name: pdc_underrun_error
* Description   : Underrun error handler
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_underrun_error(void)
{
    /* The sample code does nothing when an error occurs. */
} /* End of function pdc_underrun_error() */

/*****
*****
* Function Name: pdc_vertical_line_setting_err
* Description   : Vertical line number setting error handler
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_vertical_line_setting_err(void)
{
    /* The sample code does nothing when an error occurs. */
} /* End of function pdc_vertical_line_setting_err() */

/*****
*****
* Function Name: pdc_horizontal_byte_setting_err
* Description   : Horizontal byte number setting error handler
* Arguments     : none
* Return Value  : none
*****
*****/
void pdc_horizontal_byte_setting_err(void)
{
    /* The sample code does nothing when an error occurs. */
} /* End of function pdc_horizontal_byte_setting_err() */

```

**Website and Support**

Renesas Electronics Website

<http://www.renesas.com/>

Inquiries

<http://www.renesas.com/contact/>

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

**Revision History**

Rev.	Date	Description	
		Page	Summary
1.00	Sep 03, 2014	—	First edition issued

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

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

### 1. Handling of Unused Pins

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

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

### 2. Processing at Power-on

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

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

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

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

### 3. Prohibition of Access to Reserved Addresses

Access to reserved addresses is prohibited.

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

### 4. Clock Signals

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

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

### 5. Differences between Products

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

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

## Notice

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

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

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

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
  6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
  7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
  8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
  9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
  10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
  11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
  12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.
- (Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.
- (Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.



### SALES OFFICES

### Renesas Electronics Corporation

<http://www.renesas.com>

Refer to "<http://www.renesas.com/>" for the latest and detailed information.

#### **Renesas Electronics America Inc.**

2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.  
Tel: +1-408-588-6000, Fax: +1-408-588-6130

#### **Renesas Electronics Canada Limited**

1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada  
Tel: +1-905-896-5441, Fax: +1-905-896-3220

#### **Renesas Electronics Europe Limited**

Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K.  
Tel: +44-1628-585-100, Fax: +44-1628-585-900

#### **Renesas Electronics Europe GmbH**

Arcadiastrasse 10, 40472 Düsseldorf, Germany  
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

#### **Renesas Electronics (China) Co., Ltd.**

Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China  
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

#### **Renesas Electronics (Shanghai) Co., Ltd.**

Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333  
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

#### **Renesas Electronics Hong Kong Limited**

Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong  
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

#### **Renesas Electronics Taiwan Co., Ltd.**

13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan  
Tel: +886-2-8175-9600, Fax: +886-2-8175-9670

#### **Renesas Electronics Singapore Pte. Ltd.**

80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949  
Tel: +65-6213-0200, Fax: +65-6213-0300

#### **Renesas Electronics Malaysia Sdn.Bhd.**

Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia  
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

#### **Renesas Electronics Korea Co., Ltd.**

12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea  
Tel: +82-2-558-3737, Fax: +82-2-558-5141