RX Family

DMAC Module Using Firmware Integration Technology

Introduction

This application note describes the DMA module which uses Firmware Integration Technology (FIT). This module uses DMA to transfer data without the CPU. In this document, this module is referred to as the DMA FIT module.

Target Devices

- RX230 Group, RX231 Group
- RX23W Group
- RX64M Group
- RX65N Group, RX651 Group
- RX66T Group
- RX66N Group
- RX71M Group
- RX72T Group
- RX72M Group
- RX72N Group

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 Confirmed Operation Environment".

Contents

1. Overview

1.1 DMACA FIT Module

The DMACA FIT module can be used by being implemented in a project as an API. See section 2.13 Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the DMACA FIT Module

The DMAC is a module to transfer data without the CPU. When a DMACA transfer request is generated, the DMAC transfers data stored at the transfer source address to the transfer destination address.

For details, see the “DMA Controller” section of the User’s Manual: Hardware.

(1) Transfer Modes
The DMAC supports the following transfer modes.

- Normal transfer mode
- Repeat transfer mode
- Block transfer mode

(2) Extended Repeat Area Function
The DMAC supports a function to specify the extended repeat areas on the transfer source and destination addresses. With the extended repeat areas set, the address registers repeatedly indicate the addresses of the specified extended repeat areas. However, the area (of transfer source or transfer destination) which is specified as the repeat area or block area should not be specified as the extended repeat area.

(3) Address Update Function using Offset (DMAC0 Only)
The source and destination addresses can be updated by fixing, increment, decrement, or offset addition. When the offset addition is selected, the offset specified by the DMACA offset register (DMOFR of DMAC0) is added to the address every time the DMAC performs one data transfer. This function realizes a data transfer where addresses are allocated to separated areas. Offset subtraction can also be realized by setting a negative value in DMOFR of DMAC0. In this case, the negative value must be 2’s complement.

For example, on the RX64M the offset setting ranges are 0 bytes to (16 M – 1) bytes (00000000h to 00FFFFFFh) and –16 M bytes to –1 byte (FF000000h to FFFFFFFFh).

(4) Usage Conditions of DMACA FIT Module
The usage conditions of the module are as follows.

- The r_bsp default lock function must be used.
- A single common bit must be used as the DMAC module stop setting bit and the DTC module stop setting bit.

1.3 API Overview

Table 1.1 lists the API functions of DMACA FIT module.

Table 1.1 API Functions

Function Name	Description
R_DMACA_Init()	Module information initialization processing
R_DMACA_Open()	Channel-specific initialization processing
R_DMACA_Close()	Channel-specific end processing
R_DMACA_Create()	Channel-specific register and activation source setting processing
R_DMACA_Control()	Operation setting processing
R_DMACA_Int_Callback()	Callback function registration processing for channel-specific transfer end interrupt/transfer escape end interrupt
R_DMACA_Int_Enable()	Channel-specific transfer end interrupt/transfer escape end interrupt enable processing
R_DMACA_Int_Disable()	Channel-specific transfer end interrupt/transfer escape end interrupt disable processing
R_DMACA_GetVersion()	Version information acquisition processing

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- DMAC(DMACA)
- ICU

2.2 Software Requirements

This driver is dependent upon the following FIT module:

- Renesas Board Support Package (r_bsp) v5.20 or higher.

2.3 Limitations

2.3.1 RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.

The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.

In the case of the CCRX project (e2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of the GCC project (e2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.

The default settings of the section may be changed due to the IDE version upgrade. Please check the section settings when using the latest IDE.

2.4 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 6.1, Confirmed Operation Environment.

2.5 Interrupt vector

The transfer end interrupt and the escape transfer end interrupt is enabled by executing the R_DMACA_Int_Enable() function.

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

Table 2.1 Interrupt Vector Used in the DMACA FIT Module

Device	Interrupt Vector
RX230/RX231/RX23W	DMAC0I interrupt[channel0] (vector no.:198)
	DMAC1I interrupt[channel1] (vector no.:199)
	DMAC2I interrupt[channel2] (vector no.:200)
	DMAC3I interrupt[channel3] (vector no.:201)
RX64M	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)
RX65N/RX651	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)
RX66T	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)
RX71M	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)
RX72T	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)
RX72M/RX72N/RX66N	DMAC0I interrupt[channel0] (vector no.:120)
	DMAC1I interrupt[channel1] (vector no.:121)
	DMAC2I interrupt[channel2] (vector no.:122)
	DMAC3I interrupt[channel3] (vector no.:123)
	DMAC74I interrupt[channel4-7] (vector no.:124)

2.6 Header Files

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

2.7 Integer Types

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

2.8 Configuration Overview

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

Configuration options in r_dmaca_rx_config.h	
DMACA_CFG_PARAM_CHECKING_ENABLE 1	Selects whether or not parameter checking is included in the code. 0: Parameter checking is omitted from the code at build time. 1: Parameter checking is included in the code at build time. The code size can be reduced by omitting parameter checking from the code at build time.
DMACA_CFG_USE_DTC_FIT_MODULE 0	SPECIFY WHETHER THE DTC DRIVER IS USED WITH DMACA DRIVER 0 : DTC driver is not used with DMACA driver. 1 : DTC driver is used with DMACA driver.

2.9 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.8, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.4, Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

ROM, RAM, and Stack Code Sizes							
Device		Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX231	ROM	1,598 bytes	1,253 bytes	2,840 bytes	2,296 bytes	2,860 bytes	2,352 bytes
	RAM	36 bytes	36 bytes	120 bytes	20 bytes	42 bytes	42 bytes
	Maximum stack usage	36 bytes	36 bytes	-	-	136 bytes	136 bytes
RX23W	ROM	1,548 bytes	1,203 bytes	-	-	-	-
	RAM	36 bytes	36 bytes	-	-		-
	Maximum stack usage	72 bytes	72 bytes	-	-	-	-
RX65N	ROM	1,764 bytes	1,419 bytes	3,352 bytes	2,808 bytes	3,461 bytes	2925 bytes
	RAM	72 bytes	72 bytes	40 bytes	40 bytes	76 bytes	76 bytes
	Maximum stack usage	36 bytes	36 bytes	-	-	136 bytes	136 bytes
RX66T	ROM	1,732 bytes	1,478 bytes	3,376 bytes	2,832 bytes	3,439 bytes	2,916 bytes
	RAM	72 bytes	72 bytes	40 bytes	40 bytes	76 bytes	76 bytes
	Maximum stack usage	36 bytes	36 bytes	-	-	148 bytes	148 bytes
RX71M	ROM	1,761 bytes	1,416 bytes	3,344 bytes	2,800 bytes	3,446 bytes	2,925 bytes
	RAM	72 bytes	72 bytes	40 bytes	40 bytes	76 bytes	76 bytes
	Maximum stack usage	36 bytes	36 bytes	-	-	148 bytes	148 bytes
RX72T	ROM	1,773 bytes	1,428 bytes	3,312 bytes	2,768 bytes	3,446 bytes	2,925 bytes
	RAM	72 bytes	72 bytes	40 bytes	40 bytes	76 bytes	76 bytes
	Maximum stack usage	36 bytes	36 bytes	-	-	148 bytes	148 bytes
RX72M	ROM	1,776 bytes	1,431 bytes	3,472 bytes	2,920 bytes	3,338 bytes	2,817 bytes
	RAM	72 bytes	72 bytes	40 bytes	40 bytes	72 bytes	72 bytes

	Maximum stack usage	80 bytes	80 bytes	-	-	156 bytes	156 bytes
RX72N	ROM	1832 bytes	1487 bytes	3517 bytes	2968 bytes	3175 bytes	2657 bytes
	RAM	72 bytes	72 bytes	72 bytes	72 bytes	72 bytes	72 bytes
	Maximum stack usage	32 bytes	28 bytes	-	-	96 bytes	96 bytes
RX66N	ROM	1832 bytes	1487 bytes	3520 bytes	2968 bytes	3179 bytes	2775 bytes
	RAM	72 bytes	72 bytes	72 bytes	72 bytes	72 bytes	72 bytes
	Maximum stack usage	32 bytes	28 bytes	-	-	96 bytes	96 bytes

Note 1 The memory sizes listed apply when the default settings listed in, “Configuration Overview”, are used. The memory sizes differ according to the definitions selected.

Note 2 Under confirmation conditions listed the following

- r_dmaca_rx.c
- r_dmaca_rx_target.c

Note 3 The required memory sizes differ according to the C compiler version and the compile conditions.

Note 4 The memory sizes listed apply when the little endian. The above memory sizes also differ according to endian mode.

2.10 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_dmaca_rx_if.h as are the prototype declarations of API functions.

```
typedef struct st_dmaca_transfer_data_cfg
{
    dmaca_transfer_mode_t     transfer_mode;          /* Transfer Mode  */
    dmaca_repeat_block_side_t repeat_block_side;
                              /* Repeat Area in Repeat or Block Transfer Mode */
    dmaca_data_size_t         data_size;  /* Transfer Data Size   */
    dmaca_activation_source_t act_source;       /* Activation Source    */
    dmaca_request_source_t    request_source;   /* Transfer Request Source    */
    dmaca_dti_t   dtie_request;     /* Transfer End Interrupt Request   */
    dmaca_esi_t   esie_request;     /* Transfer Escape End Interrupt Request  */
    dmaca_rpti_t  rptie_request;    /* Repeat Size End Interrupt Request      */
    dmaca_sari_t  sarie_request;    /* Source Address Extended Repeat Area
Overflow Interrupt Request    */
    dmaca_dari_t  darie_request;    /* Destination Address Extended Repeat Area
Overflow Interrupt Request    */
    dmaca_src_addr_mode_t     src_addr_mode;    /* Address Mode of Source     */
    dmaca_src_addr_repeat_area_t    src_addr_repeat_area;/* Source Address
Extended Repeat Area    */
    dmaca_des_addr_mode_t     des_addr_mode;/* Address Mode of Destination    */
    dmaca_des_addr_repeat_area_t    des_addr_repeat_area;   /* Destination
Address Extended Repeat Area  */
    uint32_t     offset_value;/* Offset value for DMA Offset Register (DMOFR) */
    dmaca_interrupt_select_t  interrupt_sel;    /* Configurable Options for
Interrupt Select  */
    void    *p_src_addr;                    /* Start Address of Source    */
    void    *p_des_addr;       /* Start Address of Destination    */
    uint32_t      transfer_count;                           /* Transfer Count */
    uint16_t      block_size;        /* Repeat Size or Block Size */
    uint8_t  rsv[2];
} dmaca_transfer_data_cfg_t;
```

```
typedef enum e_dmaca_command
{
    DMACA_CMD_ENABLE = 0,                 /* Enables DMA transfer.      */
    DMACA_CMD_ALL_ENABLE,                 /* Enables DMAC activation.   */
    DMACA_CMD_RESUME,                     /* Resumes DMA transfer.      */
    DMACA_CMD_DISABLE,                    /* Enables DMA transfer.      */
    DMACA_CMD_ALL_DISABLE,                /* Disables DMAC activation.  */
    DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ, /* SWREQ bit is cleared automatically
after DMA transfer.     */
    DMACA_CMD_SOFT_REQ_NOT_CLR_REQ,       /* SWREQ bit is not cleared after DMA
transfer.   */
    DMACA_CMD_SOFT_REQ_CLR,         /* Clears DMACA Software request flag.    */
    DMACA_CMD_STATUS_GET,           /* Gets the current status of DMACA.      */
    DMACA_CMD_ESIF_STATUS_CLR,     /* Clears Transfer Escape End Interrupt Flag.
      */
    DMACA_CMD_DTIF_STATUS_CLR       /* Clears Transfer Interrupt Flag.  */
} dmaca_command_t;
```

2.11 Return Values

This section describes return values of API functions. This enumeration is located in r_dmaca_rx_if.h as are the prototype declarations of API functions.

```
typedef enum e_dmaca_return
{
    DMACA_SUCCESS_OTHER_CH_BUSY = 0,      /* Other DMAC channels are locked,  */
                              /*  so that cannot set to module stop state.    */
    DMACA_SUCCESS_DTC_BUSY,               /* DTC is locked, */
                              /* so that cannot set to module stop state.     */
    DMACA_SUCCESS,
    DMACA_ERR_INVALID_CH,                 /* Channel is invalid.  */
    DMACA_ERR_INVALID_ARG,                /* Parameters are invalid.    */
    DMACA_ERR_INVALID_HANDLER_ADDR,       /* Invalid function address is set, */
                         /*  and any previous function has been unregistered.  */
    DMACA_ERR_INVALID_COMMAND,            /* Command is invalid.  */
    DMACA_ERR_NULL_PTR,                   /* Argument pointers are NULL.      */
    DMACA_ERR_BUSY,           /* Resource has been locked by other process.   */
    DMACA_ERR_SOFTWARE_REQUESTED,         /* DMA transfer request by software
has been generated already,   */
                                          /*  so that cannot execute command. */
    DMACA_ERR_SOFTWARE_REQUEST_DISABLED,  /* Transfer Request Source is not
Software. */
    DMACA_ERR_INTERNAL                    /* DMACA driver internal error      */
 } dmaca_return_t;
```

2.12 Callback function

In this module, the callback function specified by the user is called when the transfer end interrupt and the escape transfer end interrupt occurs.

The callback function is specified by storing the address of the user function in the “R_DMACA_Int_Callback()” structure member (see 2.10, Parameters). When the callback function is called, the variable which stores the constant is passed as the argument.

The argument is passed as void type. Thus the argument of the callback function is cast to a void pointer.

When using a value in the callback function, type cast the value.

2.13 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the 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 the 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 the Smart Configurator in 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.

2.14 “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

R_DMACA_Init()

This function is used to initialize the DMAC's internal information.

Format
void R_DMACA_Init (void)

Parameters
None.

Return Values
None.

Properties
Prototype declarations are contained in r_dmaca_rx_if.h.

Description
Initializes the usage status of each DMA channel (internal information). Also, cancels the registered callback functions for all DMAC transfer end interrupts/transfer escape end interrupts (DMAC0I, DMAC1I, DMAC2I, DMAC3I, and DMAC74I). If DMAC transfer end interrupts/transfer escape end interrupts will be used, run the R_DMACA_Init() function beforehand, and then use the R_DMACA_Int_Callback() function (described below) to register the callback functions.

Example
```
#include "r_dmaca_rx_if.h"

/* When using the DMACA driver, run the R_DMACA_Init() function first. */
R_DMACA_Init();
```

Special Notes:
When using the DMACA driver, run the R_DMACA_Init() function first. It is recommended to run at hardware setup operation.

R_DMACA_Open()

This function is run after calling R_DMACA_Init() when using the APIs of the DMACA FIT module.

Format

```
dmaca_return_t          R_DMACA_Open (
   uint8_t        channel
)
```

Parameters

uint8_t channel
DMAC channel number.

Return Values

[DMACA_SUCCESS]	*/* Successful operation*/*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*
[DMACA_ERR_BUSY]	*/* Resource has been locked by other process.*/*

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

Locks*1 the DMAC channel specified by the argument channel, then makes initial settings. Releases the DMAC from the module stop state, then activates the DMAC. Also, initializes the activation source selection register for the specified DMAC channel.

Note: 1. The DMACA FIT module uses the r_bsp default lock function. As a result, the specified DMAC channel is in the locked state after a successful end.

Example

```
#include “r_dmaca_rx_if.h”
volatile dmaca_return_t  ret;

ret = R_DMACA_Open(DMACA_CH0);
```

Special Notes:

None.

R_DMACA_Close()

This function is used to release the resources of the DMAC channel currently in use.

Format

```
dmaca_return_t  R_DMACA_Close (
    uint8_t          channel
)
```

Parameters

uint8_t channel
DMAC channel number.

Return Values

[DMACA_SUCCESS]	*/* Successful operation*/*
[DMACA_SUCCESS_OTHER_CH_BUSY]	*/* Successful operation. Other DMAC channels are locked.*/*
[DMACA_SUCCESS_DTC_BUSY]	*/* Successful operation. DTC is locked. */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid. */*
[DMACA_ERR_INTERNAL]	*/* DMACA driver internal error */*

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

Unlocks*1 the DMAC channel specified by the argument channel and clears to 0 the DMA transfer enable (DTE) bit of the specified DMAC channel to disable DMA transfers. If all DMAC channels are unlocked, the function clears the DMAC operation enable (DMST) bit to prevent DMAC activation. If in addition DTC is unlocked, the function sets the DMAC and DTC to the module stop state.*2

Note: 1. The DMACA FIT module uses the r_bsp default lock function. As a result, the specified DMAC channel is in the unlocked state after a successful end.

2. Because a shared bit is used as both the DMAC module stop setting bit and the DTC module stop setting bit, the function confirms that the DTC is unlocked before making the module stop setting. (For details, see the “Low Power Consumption” section in the User’s Manual: Hardware.

Change the processing method to match the combination of modules used, as shown below.

DMAC Control	DTC Control	Processing Method
DMACA FIT module (lock function control function present, DTC lock state checking function present)	DTC FIT module (lock function control function present, DMAC lock state checking function present)	See case 1.
Other than the above		See case 2.

Case 1: Using the r_bsp Default Lock Function and Controlling the DTC with the DTC FIT Module*[1]

The function uses the r_bsp default lock function to confirm that all DMAC channels are unlocked and that the DTC is unlocked, then puts the DMAC into the module stop state.

Note: 1. A necessary condition is that the DTC FIT module has a module stop control function that confirms the locked state of the DMAC.

Case 2: Control Other Than the Above

The user must provide code to confirm that all DMAC channels are unlocked and that the DTC is unlocked (not in use). The DMACA FIT module includes an empty function for this purpose.

If the r_bsp default lock function is not used, insert the program code for checking the locked/unlocked state of all the DMAC channels and the DTC after the line marked /* do something */ in the r_dmaca_check_DMACA_DTC_locking_byUSER() function in the file r_dmaca_rx_target.c.

Even if the r_bsp default lock function is used, if the DTC FIT module is not used to control the DTC, insert program code for checking the locked/unlocked state of the DTC after the line marked /* do something */ in the r_dmaca_check_DTC_locking_byUSER() function in the file r_dmaca_rx_target.c.

Note that the dmaca_chk_locking_sw_t type shown below should be used for the return value of the r_dmaca_check_DMACA_DTC_locking_byUSER() function or r_dmaca_check_DTC_locking_byUSER() function.

dmaca_chk_locking_sw_t type

```
DMACA_ALL_CH_UNLOCKED_AND_DTC_UNLOCKED
                                  /* All DMAC channels and DTC are unlocked.  */
DMACA_ALL_CH_UNLOCKED_BUT_DTC_LOCKED
                        /* All DMAC channels are unlocked, but DTC is locked.  */
DMACA_LOCKED_CH_EXIST                      /* Other DMAC channels are locked.  */
```

Example

```
#include “r_dmaca_rx_if.h”
volatile dmaca_return_t  ret;

ret = R_DMACA_Close(DMACA_CH0);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

When controlling the DTC without using the DTC FIT module, make sure to monitor the usage of the DTC and control locking and unlocking of the DTC so that calling this function does not set the DTC to the module stop state. Note that even if the DTC has not been activated, it is necessary to keep it in the locked state when not making DTC transfer settings.

R_DMACA_Create()

This function is used to make DMAC register settings and to specify the activation source.

Format

```
dmaca_return_t          R_DMACA_Create (
    uint8_t             channel,
    damca_transfer_data_cfg_t * p_data_cfg
)
```

Parameters

uint8_t channel
DMAC channel number.

*damca_transfer_data_cfg_t *p_data_cfg*
Pointer to dmaca_transfer_data_cfg_t DMAC transfer information structure.

Setting Values of Members of dmaca_transfer_data_cfg_t Structure

Structure Member	Short Description	Setting Value	Setting Details
transfer_mode	Transfer Mode	DMACA_TRANSFER_MODE_NORMAL	Normal transfer
		DMACA_TRANSFER_MODE_REPEAT	Repeat transfer
		DMACA_TRANSFER_MODE_BLOCK	Block transfer
repeat_block_side	Repeat Area in Repeat or Block Transfer Mode	DMACA_REPEAT_BLOCK_DESTINATION	The destination is specified as the repeat area or block area.
		DMACA_REPEAT_BLOCK_SOURCE	The source is specified as the repeat area or block area.
		DMACA_REPEAT_BLOCK_DISABLE	The repeat area or block area is not specified.
data_size	Transfer Data Size	DMACA_DATA_SIZE_BYTE	8-bit
		DMACA_DATA_SIZE_WORD	16-bit
		DMACA_DATA_SIZE_LWORD	32-bit
act_source	DMACA Activation Source	Member of enum_ir enumerated type list of constants in file Iodefine.h	Interrupt vector number of DMAC activation source
request_source	DMACA Transfer Request Source	DMACA_TRANSFER_REQUEST_SOFTWARE	Software
		DMACA_TRANSFER_REQUEST_PERIPHERAL	Interrupts from peripheral modules or external interrupt input pins.
dtie_request	Transfer End Interrupt Request	DMACA_TRANSFER_END_INTERRUPT_DISABLE	Disables the transfer end interrupt request.
		DMACA_TRANSFER_END_INTERRUPT_ENABLE	Enables the transfer end interrupt request.

esie_request	Transfer Escape End Interrupt Request	DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE	Disables the transfer escape end interrupt request.
		DMACA_TRANSFER_ESCAPE_END_INTERRUPT_ENABLE	Enables the transfer escape end interrupt request.
rptie_request	Repeat Size End Interrupt Request	DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE	Disables the repeat size end interrupt request.
		DMACA_REPEAT_SIZE_END_INTERRUPT_ENABLE	Enables the repeat size end interrupt request.
sarie_request	Source Address Extended Repeat Area Overflow Interrupt Request	DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE	Disables an interrupt request for an extended repeat area overflow on the source address
		DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_ENABLE	Enables an interrupt request for an extended repeat area overflow on the source address
darie_request	Destination Address Extended Repeat Area Overflow Interrupt Request	DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE	Disables an interrupt request for an extended repeat area overflow on the destination address
		DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_ENABLE	Enables an interrupt request for an extended repeat area overflow on the destination address
src_addr_mode	Address Mode of Source	DMACA_SRC_ADDR_FIXED	Destination address is fixed.
		DMACA_SRC_ADDR_OFFSET	Offset addition
		DMACA_SRC_ADDR_INCR	Source address is incremented
		DMACA_SRC_ADDR_DECR	Source address is decremented
src_addr_repeat_area	Source Address Extended Repeat Area	DMACA_SRC_ADDR_EXT_REP_AREA_NONE	Not specified
		DMACA_SRC_ADDR_EXT_REP_AREA_2B	2 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4B	4 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8B	8 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16B	16 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32B	32 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_64B	64 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128B	128 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_256B	256 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_512B	512 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_1KB	1K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_2KB	2K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4KB	4K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8KB	8K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16KB	16K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32KB	32K bytes 64K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_64KB	128K bytes 256K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128K	512K bytes

		B	1M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_256K B	2M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_512K B	4M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_1MB	8M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_2MB	16M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4MB	32M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8MB	64M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16M B	128M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32M B	
		DMACA_SRC_ADDR_EXT_REP_AREA_64M B	
		DMACA_SRC_ADDR_EXT_REP_AREA_128 MB	
des_addr_mode	Address Mode of Destination	DMACA_DES_ADDR_FIXED	Destination address is fixed.
		DMACA_DES_ADDR_OFFSET	Offset addition
		DMACA_DES_ADDR_INCR	Destination address is incremented.
		DMACA_DES_ADDR_DECR	Destination address is decremented.
des_addr_repeat_area	Destination Address Extended Repeat Area	DMACA_DES_ADDR_EXT_REP_AREA_NONE	Not specified
		DMACA_DES_ADDR_EXT_REP_AREA_2B	2 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_4B	4 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_8B	8 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_16B	16 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_32B	32 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_64B	64 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_128	128 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_256B	256 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_512B	512 bytes
		DMACA_DES_ADDR_EXT_REP_AREA_1KB	1K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_2KB	2K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_4KB	4K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_8KB	8K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_16KB	16K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_32KB	32K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_64KB	64K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_128K B	128K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_256K B	256K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_512K B	512K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_1MB	1M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_2MB	2M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_4MB	4M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_8MB	8M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_16M B	16M bytes
			32M bytes
			64M bytes
			128M bytes

		DMACA_DES_ADDR_EXT_REP_AREA_32MB DMACA_DES_ADDR_EXT_REP_AREA_64MB DMACA_DES_ADDR_EXT_REP_AREA_128MB	
offset_value	Offset value for DMA Offset Register (DMOFR)	32bit data 00000000h to 00FFFFFFh (0 bytes to (16M-1) bytes) FF000000h to FFFFFFFFh (-16M bytes to -1 byte) Note: Setting bits 31 to 25 is invalid. A value of bit 24 is extended to bits 31 to 25. Offset addition can be specified only for DMAC0. With R_DMACA_Create() function, setting this data is invalid except DMAC0.	Note: Offset subtraction can also be realized by setting a negative value. In this case, the negative value must be 2's complement.
interrupt_sel	Configurable Options for Interrupt Select	DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_ TRANSFER	At the beginning of transfer, clears the interrupt flag of the activation source to 0.
		DMACA_ISSUES_INTERRUPT_TO_CPU_END_OF_ TRANSFER	At the end of transfer, the interrupt flag of the activation source issues an interrupt to the CPU.
*p_src_addr	Start Address of Source	32bit data 00000000h to 0FFFFFFFh (256M bytes)	Source address
*p_des_addr	Start Address of Destination	F0000000h to FFFFFFFFh (256M bytes) Note: Setting bits 31 to 29 is invalid. A value of bit 28 is extended to bits 31 to 29.	Destination address
transfer_count	Transfer Count	32bit data [Normal Transfer Mode] 1 to 65535 When the setting is 0, no specific number of transfer operations is set (free running mode) [Repeat Transfer Mode or Block Transfer Mode]. 1 to 65536 Upper 16 bits are not used	[Normal Transfer Mode] This data is set to DMCRAL register. [Repeat Transfer Mode or Block Transfer Mode] This data is set to DMCRB register.
block_size	Repeat Size or Block Size	16bit data [Normal Transfer Mode] Invalid [Repeat Transfer Mode or Block Transfer Mode]. 1 to 1024	[Normal Transfer Mode] Invalid [Repeat Transfer Mode or Block Transfer Mode] This data is set to DMCRAL register and DMCRAH register.

Return Values

[DMACA_SUCCESS]	*/* Successful operation */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid. */*
[DMACA_ERR_INVALID_ARG]	*/* Parameters are invalid. */*
[DMACA_ERR_NULL_PTR]	*/* Argument pointers are NULL. */*

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

References the dmaca_transfer_data_cfg_t DMAC transfer information structure passed as an argument and makes register settings for the specified DMAC channel. Also specifies the activation source for the DMAC channel.

Example

Case 1: Activating the DMAC by Software

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_transfer_data_cfg_t td_cfg;
uint32_t src = 1234;
uint32_t des[3];

/* Operation – No Extended Repeat Area Function and No Offset Subtraction */
/* Source address is fixed
 * Transfer data size is 32-bit (long word).
 * DMAC transfer mode is Repeat mode & Source side is repeat area
 * At the beginning of transfer, clear the interrupt flag of the activation
source to 0.
 * Transfer Request source is software. */

/* Set Transfer data configuration. */
   td_cfg.transfer_mode                  = DMACA_TRANSFER_MODE_REPEAT;
   td_cfg.repeat_block_side                    = DMACA_REPEAT_BLOCK_SOURCE;
   td_cfg.data_size                      = DMACA_DATA_SIZE_LWORD;
   td_cfg.act_source                     = (dmaca_activation_source_t)0;
   td_cfg.request_source                 = DMACA_TRANSFER_REQUEST_SOFTWARE;
   td_cfg.dtie_request                   =
DMACA_TRANSFER_END_INTERRUPT_DISABLE;
   td_cfg.esie_request        = DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE;
   td_cfg.rptie_request       = DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE;
   td_cfg.sarie_request = DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.darie_request       =
DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.src_addr_mode                  = DMACA_SRC_ADDR_FIXED;
   td_cfg.src_addr_repeat_area           = DMACA_SRC_ADDR_EXT_REP_AREA_NONE;
   td_cfg.des_addr_mode                  = DMACA_DES_ADDR_INCR;
   td_cfg.des_addr_repeat_area           = DMACA_DES_ADDR_EXT_REP_AREA_NONE;
   td_cfg.offset_value                   = 0x00000000;
   td_cfg.interrupt_sel                  =
DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_TRANSFER;
   td_cfg.p_src_addr               = (void *)&src;
   td_cfg.p_des_addr               = (void *)des;
   td_cfg.transfer_count                 = 1;
   td_cfg.block_size               = 3;

/* Call R_DMACA_Create(). */
   ret = R_DMACA_Create(DMACA_CH0, &td_cfg);
```

Note: When the td_cfg.request_source is DMACA_TRANSFER_REQUEST_SOFTWARE (DMAC transfer request source is software), the R_DMACA_Create() function ignores the td_cfg.act_source setting.

Case 2: Using a Peripheral Module as the DMAC Activation Source (Example of Using CMI1 Interrupt)

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_transfer_data_cfg_t td_cfg;
uint32_t src = 1234;
uint32_t des[3];

/* Operation – No Extended Repeat Area Function and No Offset Subtraction */
/* Source address is fixed.
 * Transfer data size is 32-bit (long word).
 * DMAC transfer mode is Repeat mode & Source side is repeat area
 * At the beginning of transfer, clear the interrupt flag of the activation
source to 0.
 * Transfer Request source is CMI1. */

/* Set Transfer data configuration. */
   td_cfg.transfer_mode             = DMACA_TRANSFER_MODE_REPEAT;
   td_cfg.repeat_block_side         = DMACA_REPEAT_BLOCK_SOURCE;
   td_cfg.data_size                 = DMACA_DATA_SIZE_LWORD;
   td_cfg.act_source                = IR_CMT1_CMI1;
   td_cfg.request_source            = DMACA_TRANSFER_REQUEST_PERIPHERAL;
   td_cfg.dtie_request              = DMACA_TRANSFER_END_INTERRUPT_DISABLE;
   td_cfg.esie_request   = DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE;
   td_cfg.rptie_request             = DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE;
   td_cfg.sarie_request  = DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.darie_request  = DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.src_addr_mode             = DMACA_SRC_ADDR_FIXED;
   td_cfg.src_addr_repeat_area      = DMACA_SRC_ADDR_EXT_REP_AREA_NONE;
   td_cfg.des_addr_mode             = DMACA_DES_ADDR_INCR;
   td_cfg.des_addr_repeat_area      = DMACA_DES_ADDR_EXT_REP_AREA_NONE;
   td_cfg.offset_value              = 0;
   td_cfg.interrupt_sel  = DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_TRANSFER;
   td_cfg.p_src_addr                = (void *)&src;
   td_cfg.p_des_addr                = (void *)des;
   td_cfg.transfer_count            = 1;
   td_cfg.block_size                = 3;

/* Disable CMI1 interrupt request before calling R_DTC_Create(). */
IR(CMT1,CMI1) = 0;
IEN(CMT1,CMI1) = 0;

/* Call R_DMACA_Create(). */
   ret = R_DMACA_Create(DMACA_CH0, &td_cfg);
```

Special Notes:
None.

R_DMACA_Control()

This function is used to control the operation of the DMAC. This function is run after calling R_DMACA_Open().

Format

```
dmaca_return_t      R_DMACA_Control (
    uint8_t         channel,
    dmaca_command_t command,
    dmaca_stat_t    * p_stat
)
```

Parameters

uint8_t channel

DMAC channel number.

dmaca_command_t command

DMAC control command.

Command	Description
DMACA_CMD_ENABLE	Enables DMAC transfer (DMA transfer enable bit control by channel unit).
DMACA_CMD_ALL_ENABLE	Enables DMAC activation (DMAC operation enable bit control).
DMACA_CMD_RESUME	Restarts DMAC transfer (DMA transfer enable bit control by channel unit).
DMACA_CMD_DISABLE	Disables DMAC transfer (DMA transfer enable bit control by channel unit).
DMACA_CMD_ALL_DISABLE	Disables DMAC activation (DMAC operation enable bit control).
DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ	Activates the DMAC by software, and automatically clears the software activation bit.
DMACA_CMD_SOFT_REQ_NOT_CLR_REQ	Activates the DMAC by software, but does not automatically clear the software activation bit.
DMACA_CMD_SOFT_REQ_CLR	Clears the software activation bit.
DMACA_CMD_STATUS_GET	Gets the DMAC status information.
DMACA_CMD_ESIF_STATUS_CLR	Clears the transfer escape interrupt flag (ESIF).
DMACA_CMD_DTIF_STATUS_CLR	Clears the transfer end interrupt flag (DTIF).

*dmaca_stat_t *p_stat*

Pointer to dmaca_stat_t DMAC status information structure

Members of dmaca_stat_t Structure

Structure Member	Short Description	Setting Value	Setting Details
soft_req_stat	Software Request Status	false	A software transfer is not requested.
		true	A software transfer is requested.
esif_stat	Transfer Escape End Interrupt Status	false	A transfer escape end interrupt has not been generated.
		true	A transfer escape end interrupt has been generated.
dtif_stat	Transfer End Interrupt Status	false	A transfer end interrupt has not been generated.
		true	A transfer end interrupt has been generated.
act_stat	Active Flag of DMAC	false	DMAC operation is suspended.
		true	DMAC is operating.
transfer_count	Transfer Count	0000h - FFFFh	The number of normal transfer operations, block transfer operations or repeat transfer operations

Return Values

[DMACA_SUCCESS] */* Successful operation */*
[DMACA_ERR_INVALID_CH] */* Channel is invalid. */*/*
[DMACA_ERR_INVALID_COMMAND] */* Command is invalid.*/*
[DMACA_ERR_NULL_PTR] */* Argument pointers are NULL. */*
*[DMACA_ERR_SOFTWARE_REQUESTED*1]* */* DMA transfer request by software has been generated already.*/*
*[DMACA_ERR_SOFTWARE_REQUEST_DISABLED*2]* */* Transfer Request Source is not Software. */*

Note: 1. When automatic clearing of the DMA software activation bit (SWREQ bit) is specified, DMACA_ERR_SOFTWARE_REQUESTED is returned when the SWREQ bit is already set to 1. This value may be returned if, for example, the preceding software activation request was executed while automatic clearing of the DMA software activation bit was specified, but the request had not yet been accepted.

2. If issuing of transfer requests by a peripheral module is specified, DMACA_ERR_SOFTWARE_REQUEST_DISABLED is returned when a DMA transfer activation by software is executed.

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

DMACA_CMD_ENABLE command processing

Sets the DMA transfer enable (DTE) bit to enable transfer operation on the specified DMAC channel.

DMACA_CMD_ALL_ENABLE command processing

Sets the DMAC operation enable (DMST) bit to enable activation of the DMAC.

DMACA_CMD_RESUME command processing

Sets the DMA transfer enable (DTE) bit to enable a restart of transfer operation on the specified DMAC channel.

DMACA_CMD_DISABLE command processing

Clears the DMA transfer enable (DTE) bit to disable transfer operation on the specified DMAC channel.

Used to stop DMAC transfer operation or when changing the DMAC register settings.

DMACA_CMD_ALL_DISABLE command processing

Clears the DMAC operation enable (DMST) bit to disable activation of the DMAC.

Used to stop DMAC transfer operation or when changing the DMAC register settings.

DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ command processing

Enables automatic clearing of the SWREQ bit (CLRS bit = 0) and issues a DMA transfer request by software.

DMACA_CMD_SOFT_REQ_NOT_CLR_REQ command processing

Disables automatic clearing of the SWREQ bit (CLRS bit = 1) and issues a DMA transfer request by software.

DMACA_CMD_SOFT_REQ_CLR command processing

Clears the SWREQ bit of the specified DMAC channel.

DMACA_CMD_STATUS_GET command processing

Writes the status information of the specified DMAC channel to the address specified by the argument p_stat.

DMACA_CMD_ESIF_STATUS_CLR command processing

Clears the transfer escape interrupt flag (ESIF) of the specified DMAC channel.

DMACA_CMD_DTIF_STATUS_CLR command processing

Clears the transfer end interrupt flag (DTIF) of the specified DMAC channel.

Example

Case 1: Activating the DMAC by Software

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_stat_t dmac_status;

/* Call R_DMACA_Control().
Enable DMAC transfer. */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ENABLE, &dmac_status);

/* Call R_DMACA_Control().
DMAC Software request flag set & request flag is cleared automatically. */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_SOFT_REQ_NOT_CLR_REQ, &dmac_status);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}

/* DMAC transfer end check */
do
{
      ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_STATUS_GET, &dmac_status);
      if (DMACA_SUCCESS != ret)
      {
          /* do something */
      }
}while( false == (dmac_status.dtif_stat));
```

Case 2: Using a Peripheral Module as the DMAC Activation Source (Example of Using CMI1 Interrupt)

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_stat_t dmac_status;

/* Disable CMI1 interrupt request before calling R_DTC_Control(). */
IR(CMT1,CMI1) = 0;
IEN(CMT1,CMI1) = 0;

/* Call R_DMACA_Control().
Enable DMAC transfer. */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ENABLE, &dmac_status);

/* Enable CMI1 interrupt request before calling R_DTC_Create(). */
IEN(CMT1,CMI1) = 1;

/* DMAC transfer end check */
do
{
      ret = R_DMACA_Control(DMACA_CH0,  DMACA_CMD_STATUS_GET,  &dmac_status);
      if (DMACA_SUCCESS != ret)
      {
          /* do something */
      }
}while( false == (dmac_status.dtif_stat));
```

Case 3: Continuing or Restarting DMAC Transfer Operation following Case 1 or Case 2 Processing

```
/* Update register settings if necessary (see R_DMACA_Create() function). */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_RESUME, &dmac_status);
```

Case 4: Ending DMAC Transfer Operation after Case 1 or Case 2 Processing

```
/* Clear transfer end interrupt flag */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_DTIF_STATUS_CLR, &dmac_status);
/* Also use DMACA_CMD_ESIF_STATUS_CLR command to clear transfer escape
endinterrupt flag if transfer escape end interrupt is enabled. */
/* ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ESIF_STATUS_CLR, &dmac_status); */
```

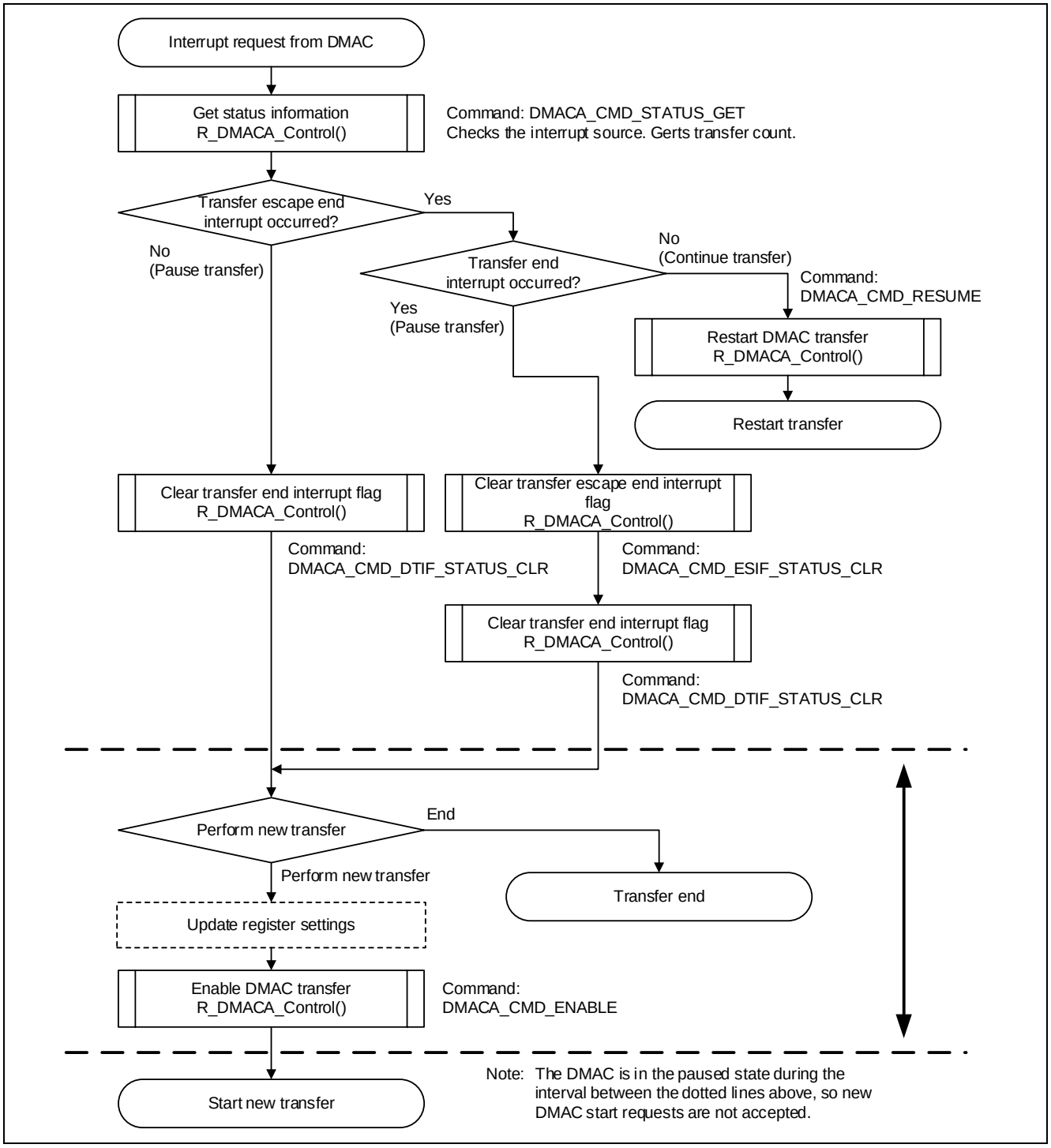

Figure 3.1 Example of Processing when DMAC Transfer Ends or Continues

Special Notes:

In the case of waiting for the transfer end by using DMAC channel 4-7 and an interrupt, please clear a transfer escape interrupt flag (ESIF) or a transfer end interrupt flag (DTIF) using a callback function for transfer end interrupts/transfer escape end interrupts.

R_DMACA_Int_Callback()

This function is used to register the callback function for the DMAC transfer end interrupt/transfer escape end interrupt.

Format

```
dmaca_return_t          R_DMACA_Int_Callback (
        uint8_t         channel,
        void            * p_callback
)
```

Parameters

uint8_t channel

DMAC channel number.

*void *p_callback*

Pointer to function that is called when a DMAC transfer end interrupt/transfer escape end interrupt occurs.

Return Values

[DMACA_SUCCESS]	*/* Successful operation */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid. */*
[DMACA_ERR_INVALID_HANDLER_ADDR]	*/* Invalid function address is set.*/*

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

Registers the callback function for the DMAC transfer end interrupt/transfer escape end interrupt of the specified channel. The registration of an already-registered callback function is canceled if FIT_NO_FUNC or NULL is passed as the callback argument. Also, the registration of an already-registered callback function is canceled if DMACA_ERR_INVALID_HANDLER_ADDR is returned.

Note: The callback function arguments and return values should be of void type.

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/* When using the DMACA driver, run the R_DMACA_Init() function once first. */
R_DMACA_Init();

/* Register the callback function for the DMAC0I interrupt (example: using a
function with the name dmac0i_callback). */
ret = R_DMACA_Int_Callback(DMACA_CH0,(void *)dmac0i_callback);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

None.

R_DMACA_Int_Enable()

This function is used to enable DMAC transfer end interrupts/transfer escape end interrupts.

Format

```
dmaca_return_t          R_DMACA_Int_Enable (
        uint8_t         channel,
        uint8_t         priority
)
```

Parameters

uint8_t channel

DMAC channel number.

uint8_t priority

DMAC transfer end interrupt/transfer escape end interrupt priority level.

Return Values

```
[DMACA_SUCCESS]          /* Successful operation  */
[DMACA_ERR_INVALID_CH]   /* Channel is invalid.   */
```

Properties

Prototype declarations are contained in r_dmaca_rx_if.h

Description

Enables the DMAC transfer end interrupt/transfer escape end interrupt for the specified channel.

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/* Enable DMAC transfer end interrupt/transfer escape end interrupt (DMAC0I) on
channel 0 with a priority level of 10. */
ret = R_DMACA_Int_Enable(DMACA_CH0,10);
if (DMAC_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

None.

R_DMACA_Init_Disable()

This function is used to disable the DMAC transfer end interrupt/transfer escape end interrupt.

Format

dmaca_return_t R_DMACA_Int_Disable (

uint8_t channel,

)

Parameters

uint8_t channel

DMAC channel number.

Return Values

[DMACA_SUCCESS] */* Successful operation */*

[DMACA_ERR_INVALID_CH] */* Channel is invalid. */*

Properties

Prototype declarations are contained in r_dmaca_rx_if.h.

Description

Disables the DMAC transfer end interrupt/transfer escape end interrupt for the specified channel.

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/* Disable DMAC transfer end interrupt/transfer escape end interrupt (DMAC0I) on
channel 0. */
ret = R_DMACA_Int_Disable(DMACA_CH0);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

None.

R_DMACA_GetVersion()

This function is used to fetch the driver version information.

Format
uint32_t R_DMACA_GetVersion (void)

Parameters
None.

Return Values
Version number.
Upper 2 bytes: major version, lower 2 bytes: minor version.

Properties
Prototype declarations are contained in r_dmaca_rx_if.h.

Description
Returns the version information.

Example
```
uint32_t version;
version = R_DMACA_GetVersion();
```

Special Notes:
None.

4. Pin Setting

DMACA FIT module don’t use pin setting.

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

5.1 dma_demo_rskrx231

The dma_demo_rskrx231 program demonstrates how to set up a DMAC in repeat transfer mode to handle ADC conversion result. As the program runs, the DMAC save ADC conversion result to a buffer of 32 bytes in sequence.

5.2 dma_demo_rskrx65n_2m

The dma_demo_rskrx65n_2m program is identical to dma_demo_rskrx231.

5.3 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects* into Workspace, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.4 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes confirmed operation environment for the DMAC FIT module.

Table 6.1 Confirmed Operation Environment (Rev.2.30)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.12.1
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
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.30
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxx)

Table 6.2 Confirmed Operation Environment (Rev.2.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
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
	GCC for Renesas RX 4.8.4.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -WI,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.12.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.20
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6.3 Confirmed Operation Environment (Rev.2.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.5.0
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.2.10
Board used	Renesas Solution Starter Kit for RX23W (product No.: RTK5523Wxxxxxxxxxx)

Table 6.4 Confirmed Operation Environment (Rev.2.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
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
	GCC for Renesas RX 4.8.4.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.2.00
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxx)

Table 6.5 Confirmed Operation Environment (Rev.1.20)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ compiler Package for RX Family V3.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian	Big endian/Little endian
Revision of the module	Rev.1.20
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 6.6 Confirmed Operation Environment (Rev.1.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ compiler Package for RX Family V3.00.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian	Big endian/Little endian
Revision of the module	Rev.1.10
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX64M (product No.: R0K50564MSxxxBE) Renesas Starter Kit for RX65N (product No.: RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit for RX71M (product No.: R0K50571MSxxxBE)

Table 6.7 Confirmed Operation Environment (Rev.1.05)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V6.0.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.07.00 (Pre-released version)
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver.1.05
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit for RX64M (product No.: R0K50564MSxxxBE) Renesas Starter Kit for RX65N (product No.: RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB (product No.: RTK50565N2SxxxxxBE) Renesas Starter Kit for RX71M (product No.: R0K50571MSxxxBE)

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 this 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_dmaca_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.

7. Reference Documents

User's Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

Not applicable technical update for this module.

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Jul 31, 2014	—	First edition issued
1.01	Aug 29, 2014	5	Added 1.3 Related Application Note.
		12	3.2 R_DMACA_Close() in Case 2: Control Other Than the Above, Changed ‘dmaca_chk_looking_sw_type’ to ‘dmaca_chk_locking_sw_type’.
1.02	Dec 26, 2014	1	Added RX71M Group in Target Devices.
		1	Added an application note (R01AN1826EJ) in Related Documents.
		3	Moved R_DMACA_Init() to top in Table 1-1, 1.2.1 Overview of APIs.
		3	Changed ‘transfer end interrupt’ to ‘transfer end interrupt/transfer escape end interrupt’ in R_DAMCA_Int_Callback(),R_DAMCA_Int_Enable() and R_DMACA_Int_Disable() of Table 1-1, 1.2.1 Overview of APIs.
		4	Changed type name of ‘Board used’ in (1)RX64M, 1.2.2 Operating Environment and Memory Sizes.
		5	Added (2)RX71M, 1.2.2 Operating Environment and Memory Sizes.
		6	Added an application note (R01AN2280EJ) in 1.3 Related Application.
		10	Changed from r_dmaca_config.h to r_dmaca_rx_config.h in 9, 2.9.1 Adding the DMACA FIT module (when not using the plug-in).
		11	Moved R_DMACA_Init() from 3.5 to 3.1 in 3. API Functions.
		11	Changed ‘transfer end interrupt’ to ‘transfer end interrupt/transfer escape end interrupt’ in Description, 3.1 R_DMACA_Init().
		11	Added contents in Special Notes, 3.1 R_DMACA_Init().
		12	Changed from ‘first’ to ‘after calling R_DMACA_Init()’ in 3.2 R_DMACA_Open().
		21	Added ‘(ESIF)’ to Description of DMACA_CMD_ESIF_STATUS_CLR in Command table, 3.5 R_DMACA_Control().
		21	Added ‘(DTIF)’ to Description of DMACA_CMD_DTIF_STATUS_CLR in Command table, 3.5 R_DMACA_Control().
		22	Added ‘(ESIF)’ to DMACA_CMD_ESIF_STATUS_CLR command processing in Description, 3.5 R_DMACA_Control().
		22	Added ‘(DTIF)’ to DMACA_CMD_DTIF_STATUS_CLR command processing in Description, 3.5 R_DMACA_Control().
		24	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in Example, 3.5 R_DMACA_Control().
		25	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in Figure 3.1 of Example, 3.5 R_DMACA_Control().
		25	Added content in Special Notes, 3.5 R_DMACA_Control().
		26	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in 3.6 R_DMACA_Int_Callback().
		26	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in Parameters and Descriptions, 3.6 R_DMACA_Int_Callback().
		28	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in 3.7 R_DMACA_Int_Enable().

1.02	Dec 26, 2014	29	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in 3.8 R_DMACA_Int_Disable().
		29	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in Descriptions and Example, 3.8 R_DMACA_Int_Disable().
1.03	Jun 15, 2015	1	Added RX230 and RX231 Group in Target Devices.
		6	Added (3)RX231, 1.2.2 Operating Environment and Memory Sizes.
1.04	Sep 30, 2016	―	Changed Title “DMA Controller DMACA Control Module Using Firmware Integration Technology” to “DMA Controller DMACA Control Module Firmware Integration Technology”.
		1	Added RX65N Group in Target Devices
		7	Added (4)RX65N, 1.2.2 Operating Environment and Memory Sizes.
		8	1.3 Related Application Note Changed title of application notes “ ---Using Firmware Integration Technology” to “ --- Firmware Integration Technology”.
		10	Added “uint8_t rsv[2]” in 2.7 Arguments.
		12	Updated explanation in 2.9 Adding Driver to Your Project.
		23	Added transfer_count of table of Members of dmaca_stat_t Structure.
		27	Added “Gets transfer count” of Figure 3.1.
1.05	Jul 07, 2017	-	Moved the following chapter contents. - Moved from 1. Overview to 1.2 Overview of APIs Changed the following chapter number. - Changed form 1.2.2 Operating Environment and Memory Size to 5.1 Operating Confirmation environment - Changed form 4. Appendices to 5.Appendices. - Changed form 5. Reference Documents to 6. Reference Documents Added the following chapter. - Added 2.4 Interrupt vector - Added 2.8 Code Size - Added 2.12 Adding FIT Module to your Project. - Added 4 Pin Setting. - Added 5.2 Troubleshooting
		1	Added RX651 Group in Target Devices.
		5	Deleted “r_cgc_rx” of 2.2 Software Requirements.
1.10	Sep 28, 2018	1	Added support for RX66T.
		6	Added Interrupt vector number for RX66T
		8	Added code size corresponding to RX66T
		33	5.1 Confirmed Operation Environment: Added Table for Rev.1.10
1.20	Feb 01, 2019	1	Added support for RX72T
		6	Added Interrupt vector number for RX72T
		8	Added code size corresponding to RX72T
		13-32	Removed ‘Reentrant’ description in each API function.
		34	Added 5. Demo Projects.
		35	Changed Renesas Starter Kit for RX66T Product No
		35	6.1 Confirmed Operation Environment: Added Table for Rev.1.20

2.00	May.20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Added the section of Target compilers. Deleted related documents.
		5	2.2 Software Requirements Requires r_bsp v5.20 or higher
		9	Updated the section of 2.8 Code Size
		34	Table 5.1 Confirmed Operation Environment: Added table for Rev.2.00
		37	Deleted the section of Website and Support.
		Program	Changed below for support GCC and IAR compiler: Replaced evenaccess with the macro definition of BSP. Replaced the declaration of interrupt functions with the macro definition of BSP.
2.10	Jun.28.19	1, 6	Added support for RX23W
		9	Added code size corresponding to RX23W
		36	Added 5. Demo Projects
		37	6.1 Confirmed Operation Environment: Added Table for Rev.2.10
		Program	Added support for RX23W. Added demo projects
2.20	Aug.15.19	1, 6	Added support for RX72M
		9-10	Added code size corresponding to RX72M
		37	6.1 Confirmed Operation Environment: Added Table for Rev.2.20 Table 6.2: Corrected board name for RX23W
		Program	Added support for RX72M.
2.30	Dec.30.19	1, 6	Added support for RX66N, RX72N
		5	2.3 Limitations Added limitations.
		10	Added code size corresponding to RX66N, RX72N
		22	Change the range of transfer_count from hexadecimal to decimal Change the range of block_size from hexadecimal to decimal
		24, 27, 28	Made some corrections in sample code
		36	6.1 Confirmed Operation Environment: Added Table for Rev.2.30
		Program	Added support for RX66N, RX72N.

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

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/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.