RX Family

DMA Controller DMACA Control Module

Firmware Integration Technology

R01AN2063EJ0104
Rev.1.04
Sep 30, 2016

Introduction

This application note explains how to use the software control module for the DMA controller (DMAC) on RX Family microcontrollers. The module is a DMAC control module using Firmware Integration Technology (FIT). The DMAC control module controls the DMAC referred to in the User's Manual: Hardware as the "DMACA". The module is referred to below as the DMACA FIT module.

In systems where the DMACA FIT module is used simultaneously with the data transfer controller (DTC), it is necessary to ensure that the DTC control software does not enable the module stop state while the DMAC is operating, because a shared bit is used as both the DMAC module stop setting bit and the DTC module stop setting bit.

Note that the initialism "DMAC" is sometimes used in the discussion below to match the descriptions in the User's Manual: Hardware, but it refers to the DMACA.

Target Device

Supported microcontrollers
RX231 Group, RX230 Group
RX64M Group, RX65N Group
RX71M Group

When applying the information in this application note to a microcontroller other than the above, modifications should be made as appropriate to match the specification of the microcontroller and careful evaluation performed.

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826EJ)

Contents

1. Overview

The DMAC is a module to transfer data without the CPU. When a DMA 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 DMA 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.

1.1 DMACA FIT Module

The DMACA FIT module can be combined with other FIT modules for easy integration into the target system.

The functions of DMACA FIT module can be incorporated into software programs by means of APIs. For information on incorporating the DMACA FIT module into projects, see 2.9, “Adding Driver to Your Project”.

1.2 Overview and Memory Size of APIs

1.2.1 Overview of APIs

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

1.2.2 Operating Environment and Memory Size

(1) RX64M

Table 1-2 lists the conditions under which operation has been confirmed, and Table 1-3 lists the required memory sizes for DMACA FIT Module.

The memory sizes listed apply when the default settings listed in 2.6, “Compile Settings”, are used. The memory sizes differ according to the definitions selected.

Table 1-2 Operation Confirmation Conditions

Item	Contents
MCU used	RX64M Group (program ROM: 4 MB, RAM: 512 KB)
Operating frequency	ICLK: 120 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.0.1.08
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00 (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. 1.02
Board used	R0K50564MSxxxBE (Renesas Starter Kit for RX64M)

Table 1-3 Required Memory Sizes

Memory	Size	Remarks
ROM	1,732 bytes (Little endian)	Under confirmation conditions listed above
RAM	72 bytes (Little endian)	Under confirmation conditions listed above
Max. user stack	24 bytes	
Max. interrupt stack	44 bytes	

The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

(2) RX71M

Table 1-4 lists the conditions under which operation has been confirmed, and Table 1-5 lists the required memory sizes for DMACA FIT Module.

The memory sizes listed apply when the default settings listed in 2.6, “Compile Settings”, are used. The memory sizes differ according to the definitions selected.

Table 1-4 Operation Confirmation Conditions

Item	Contents
MCU used	RX71M Group (program ROM: 4 MB, RAM: 512 KB)
Operating frequency	ICLK: 240 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.0.1.08
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00 (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. 1.02
Board used	R0K50571MSxxxBE (Renesas Starter Kit for RX71M)

Table 1-5 Required Memory Sizes

Memory	Size	Remarks
ROM	1,734 bytes (Little endian)	Under confirmation conditions listed above
RAM	72 bytes (Little endian)	Under confirmation conditions listed above
Max. user stack	36 bytes	
Max. interrupt stack	54 bytes	

The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

(3) RX231

Table 1-4 lists the conditions under which operation has been confirmed, and Table 1-5 lists the required memory sizes for DMACA FIT Module.

The memory sizes listed apply when the default settings listed in 2.6, “Compile Settings”, are used. The memory sizes differ according to the definitions selected.

Table 1-6 Operation Confirmation Conditions

Item	Contents
MCU used	RX231 Group (program ROM: 512 KB, RAM: 64 KB)
Operating frequency	ICLK: 32 MHz, PCLKB: 16 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.1.3.06
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.02.00 (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. 1.03
Board used	R0K505231CxxxBE (Renesas Starter Kit for RX231)

Table 1-7 Required Memory Sizes

Memory	Size	Remarks
ROM	1,516 bytes (Little endian)	Under confirmation conditions listed above
RAM	36 bytes (Little endian)	Under confirmation conditions listed above
Max. user stack	24 bytes	
Max. interrupt stack	36 bytes	

The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

(4) RX65N

Table 1-4 lists the conditions under which operation has been confirmed, and Table 1-5 lists the required memory sizes for DMACA FIT Module.

The memory sizes listed apply when the default settings listed in 2.6, “Compile Settings”, are used. The memory sizes differ according to the definitions selected.

Table 1-8 Operation Confirmation Conditions

Item	Contents
MCU used	RX65N Group (program ROM: 1 MB, RAM: 256 KB)
Operating frequency	ICLK: 120 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V5.0.0.43
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.05.00 (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. 1.04
Board used	R0K50565NxxxxxxBR (Renesas Starter Kit for RX65N)

Table 1-9 Required Memory Sizes

Memory	Size	Remarks
ROM	1,705 bytes (Little endian)	Under confirmation conditions listed above
RAM	72 bytes (Little endian)	Under confirmation conditions listed above
Max. user stack	36 bytes	
Max. interrupt stack	56 bytes	

The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

1.3 Related Application Note

The application note that is related to the DTC FIT module is listed below. Reference should also be made to this application note.

- RX Family DTC Module Firmware Integration Technology (R01AN18193EJ)

As for usage on DMACA FIT module, refer to sample program in the following application note:

- RX Family RSPI Clock Synchronous Single Master Control Module Firmware Integration Technology (R01AN1914EJ)
- RX Family QSPI Clock Synchronous Single Master Control Module Firmware Integration Technology (R01AN1940EJ)
- RX Family SCIFA Clock Synchronous Single Master Control Module Firmware Integration Technology (R01AN2280EJ)

2. API Information

The names of the APIs of the DMACA FIT module follow the Renesas API naming standard.

2.1 Hardware Requirements

The microcontroller used must support the following functionality.

- DMAC(DMACA)
- ICU

2.2 Software Requirements

The DMACA FIT module is dependent on the following packages.

- r_bsp
- r_cgc_rx (only if CGC is necessary)

2.3 Supported Toolchain

The operation of the DMACA FIT module has been confirmed with the toolchain listed in 1.2.2.

2.4 Header Files

All the API calls and interface definitions used are listed in r_dmaca_rx_if.h.

2.5 Integer Types

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

2.6 Compile Settings

The configuration option settings for the DMACA FIT module are specified in r_dmaca_rx_config.h.

The option names and setting values are described below.

Configuration options in *r_dmaca_rx_config.h*	
DMAC_CFG_PARAM_CHECKING_ENABLE Note: The default value is the value of BSP_CFG_PARAM_CHECKING_ENABLE in the r_bsp_config.h file.	Allows selection of whether or not parameter checking is included in the code. Selecting 0 causes parameter checking to be omitted from the code, resulting in a smaller code size. When the value is 0, parameter checking is omitted from the code. When the value is 1, parameter checking is included in the code.
DMACA_CFG_USE_DTC_FIT_MODULE	Allows selection of whether or not the DTC FIT module is used together with the DMACA FIT module. When the value is 0, the DTC FIT module is not used. When the value is 1, the DTC FIT module is used together with the DMACA FIT module.

2.7 Arguments

The structure for the arguments of the API functions is shown below. This structure is listed in r_dmaca_rx_if.h, along with the prototype declarations of the 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;
```

2.8 Return Values

The API function return values are shown below. This enumerated type is listed in r_dmaca_rx_if.h, along with the prototype declarations of the 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;
```

2.9 Adding Driver to Your Project

This module must be added to each project in the e^2 Studio.

There are two methods for adding to a project: using the FIT plug-in and adding manually.

When the FIT plug-in is used, FIT modules can be added to projects easily and the include file path will be updated automatically. Therefore we recommend using the FIT plug-in when adding FIT modules to a project.

There are the following methods to add FIT module using FIT plug in.

1. Use "FIT Configurator".
   This is the latest method that the plug-in function such as Lib file path automatic setting is enhanced, which we recommend the use.

   For the procedure, refer to “4.3.2 Install the FIT Modules with the FIT Plugin.” in “RX64M/RX71M Group RX Driver Package Ver.1.02 (R01AN2606EJ)” application note.

2. Use the existing ”FIT plug-in”.
   For the procedure, refer to “3. Adding FIT Modules to e2 studio Projects FIT Plug-In” in “Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)” application note.

3. API Functions

3.1 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.

Reentrant

Reentrant from a different channel is impossible.

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.

3.2 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

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.

Reentrant

Reentrant from a different channel is possible.

Example

```
#include "r_dmaca_rx_if.h"
volatile dmaca_return_t  ret;

ret = R_DMACA_Open(DMACA_CH0);
```

Special Notes:

None

3.3 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

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. */
```

Reentrant

Reentrant from a different channel is possible.

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.

3.4 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

channel

DMAC channel number

* *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.

Structure Member	Short Description	Setting Value	Setting Details
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
		DMACA_SRC_ADDR_EXT_REP_AREA_64KB	64K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128KB	128K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_256KB	256K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_512KB	512K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_1MB	1M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_2MB	2M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4MB	4M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8MB	8M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16MB	16M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32MB	32M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_64MB	64M bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128MB	128M bytes
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.

Structure Member	Short Description	Setting Value	Setting Details
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_128KB	128K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_256KB	256K bytes
		DMACA_DES_ADDR_EXT_REP_AREA_512KB	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_16MB	16M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_32MB	32M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_64MB	64M bytes
		DMACA_DES_ADDR_EXT_REP_AREA_128MB	128M bytes
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.

Structure Member	Short Description	Setting Value	Setting Details
*p_src_addr	Start Address of Source	32bit data 00000000h to 0FFFFFFFh (256M bytes) 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.	Source address
*p_des_addr	Start Address of Destination		Destination address
transfer_count	Transfer Count	32bit data [Normal Transfer Mode] 00000001h to 0000FFFFh When the setting is 0000h, no specific number of transfer operations is set (free running mode) [Repeat Transfer Mode or Block Transfer Mode]. 00000001h to 00001000h	[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]. 00000001h to 0000400h	[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.

Reentrant

Reentrant from a different channel is possible.

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_addr           = 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

3.5 R_DMACA_Control()

This function is used to control the operation of the DMAC.

Format

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

Parameters

channel

DMAC channel number

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).

** 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.

Reentrant

Reentrant from a different channel is possible.

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_WITH_AUTO_CLR_REQ,
&dmac_status);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}

/* DMAC transfer end check */
do
{
      ret = R_DMACA_Control(DMACA_CN0, 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_CN0,  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); */
```

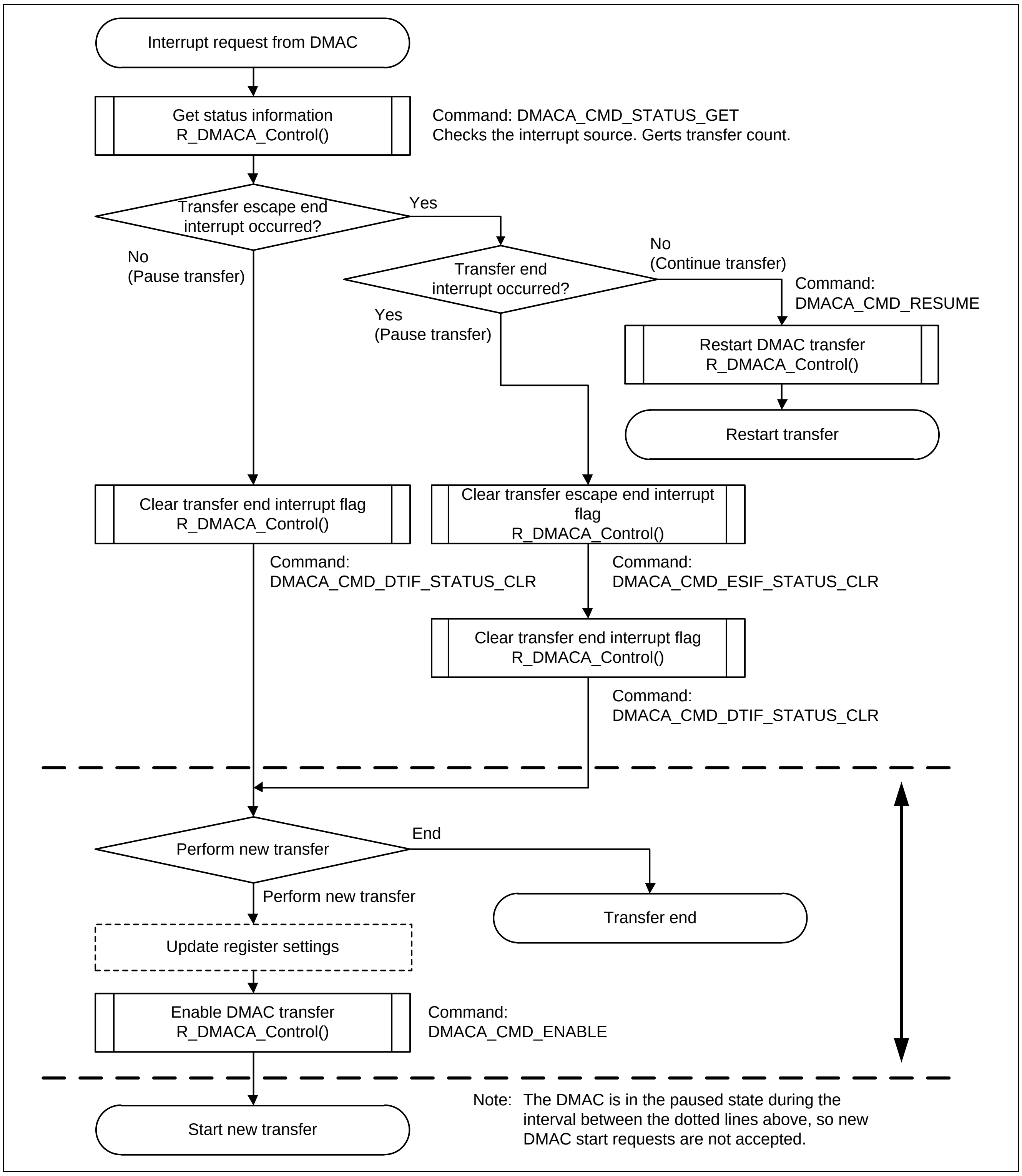
Interrupt request from DMAC
Get status information
R_DMACA_Control()
Command: DMACA_CMD_STATUS_GET
Checks the interrupt source. Gerts transfer count.
Transfer escape end interrupt occurred?
Yes
No
(Pause transfer)
Transfer end interrupt occurred?
No
(Continue transfer)
Command:
DMACA_CMD_RESUME
Yes
(Pause transfer)
Restart DMAC transfer
R_DMACA_Control()
Restart transfer
Clear transfer end interrupt flag
R_DMACA_Control()
Clear transfer escape end interrupt flag
R_DMACA_Control()
Command:
DMACA_CMD_DTIF_STATUS_CLR
Command:
DMACA_CMD_ESIF_STATUS_CLR
Clear transfer end interrupt flag
R_DMACA_Control()
Command:
DMACA_CMD_DTIF_STATUS_CLR
Perform new transfer
End
Perform new transfer
Transfer end
Update register settings
Enable DMAC transfer
R_DMACA_Control()
Command:
DMACA_CMD_ENABLE
Start new transfer
Note: The DMAC is in the paused state during the interval between the dotted lines above, so new DMAC start requests are not accepted.

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.

3.6 R_DMACA_Int_Calback()

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

channel

DMAC channel number

* *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.

Reentrant

Reentrant from a different channel is possible.

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

3.7 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

channel

DMAC channel number

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.

Reentrant

Reentrant from a different channel is possible.

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

3.8 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

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.

Reentrant

Reentrant from a different channel is possible.

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

3.9 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.

Reentrant

Reentrant from a different channel is possible.

Example

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

Special Notes:

None

4. Reference Documents

User’s Manual: Hardware

The latest version 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

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User’s Manual: RX Build Rev.1.00 (R20UT02747EJ)

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User’s Manual: RX Coding Rev.1.00 (R20UT02748EJ)

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User’s Manual: Message Rev.1.00 (R20UT02749EJ)

The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

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

Revision History

Rev.	Date	Description Page	Description 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().

		28	Changed ‘transfer escape interrupt’ to ‘transfer escape end interrupt’ in Parameters, Descriptions and Example, 3.7 R_DMACA_Int_Enable().
		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.

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.

Colophon 5.0