RE01 1500KB, 256KB Group

CMSIS Driver R_DMAC Specifications

Summary

This document describes the detailed specifications of the DMAC driver provided in the RE01 1500KB and 256KB Group CMSIS software package (hereinafter called the DMAC driver).

Target Device

RE01 1500KB Group

RE01 256KB Group

Contents

1. Overview

This is a DMAC driver for RE01 1500KB and 256KB Group devices.

2. Driver Configuration

This chapter describes the information required for using this driver.

2.1 File Configuration

The R_DMAC drive corresponds to DeviceHAL in the CMSIS Driver Package and consists of four files: "r_dmac_api.c", "r_dma_common_api.h", "r_dmac_api.h", and "r_dmac_cfg.h" in the vendor-specific file storage directory. The functions of the files are shown in Table 2-1, and the file configuration is shown in Figure 2-1.

Table 2-1 Functions of DMAC Driver Files

File Name	Description
r_dmac_api.c	Driver source file This file provides the detail of the driver function. To use the DMAC driver, it is necessary to build this file.
r_dmac_api.h	Driver header file The macro, type, and prototype definitions to be used in the driver are defined. To use the DMAC driver, it is necessary to include this file.
r_dma_common_api.h	DMAC/DTC common driver header file Macros and types used in common in DMAC and DTC are defined
r_dmac_cfg.h	Configuration definition file This provides configuration definitions that can be modified by the user.

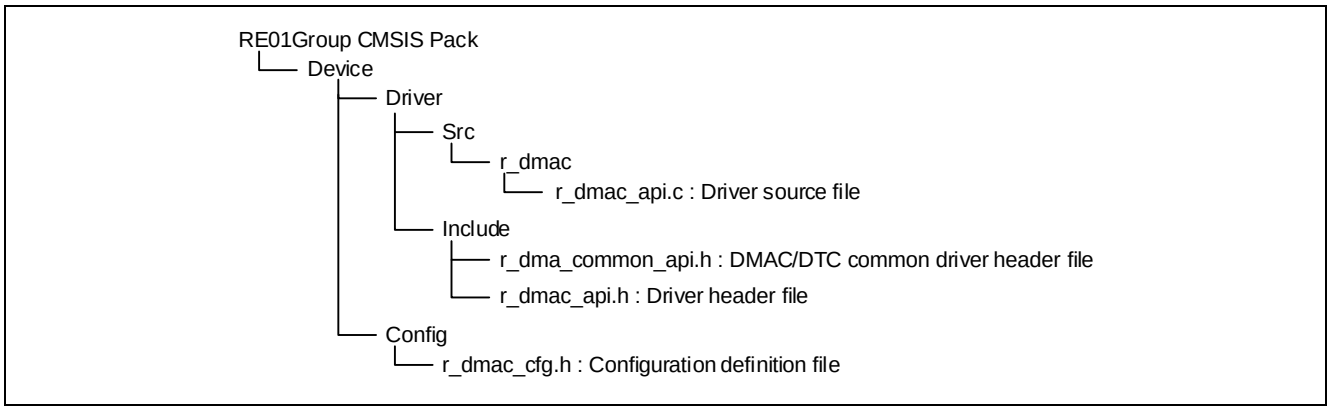


Figure 2-1 File Configuration of DMAC Driver

2.2 Driver APIs

The DMAC driver provides the ability to generate channel-specific instances. To use this driver, access the APIs by using function pointers for each instance. The list of the DMAC driver instances and the APIs contained in the instance are shown in Table 2-2 and Table 2-3, and examples of access to the DMAC driver are shown in Figure 2-3 to Figure 2-5.

Table 2-2 List of DMAC Driver Instances

Instance	Description
DRIVER_DMA Driver_DMAC0	Instance for using DMAC0
DRIVER_DMA Driver_DMAC1	Instance for using DMAC1
DRIVER_DMA Driver_DMAC2	Instance for using DMAC2
DRIVER_DMA Driver_DMAC3	Instance for using DMAC3

```
#include "r_dmac_api.h"

// DMAC0 driver instance
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA *dmac0Dev = &Driver_DMAC0;
```

Figure 2-2 Example of DMAC Driver Instance Declaration

Table 2-3 DMAC Driver APIs

API	Description	Reference
Open	Opens the DMAC driver (clears module stop and initializes RAM).	4.1.1
Close	Releases the DMAC driver (initializes registers and disables interrupts). If all of the DMAC and DTC drivers enter the unused states, it will also cause a transition to the module stop state.	4.1.2
Create	Specifies DMA transfer settings.	4.1.3
Control	Executes a control command of the DMAC. For the control commands, see Table 2-4List of Control Commands.	4.1.4
InterruptEnable	Sets a DMA interrupt. Sets the interrupt source and enables the interrupt.	4.1.5
InterruptDisable	Disables a DMA interrupt.	4.1.6
GetState	Obtains the status of the DMAC.	4.1.7
GetTransferByte	Get the number of DMA transfer bytes	4.1.9
ClearState	Clears a DMAC driver interrupt flag.	4.1.8
GetVersion	Obtains the version of the DMAC driver.	4.1.10

Table 2-4 List of Control Commands Used by DMAC Driver

Command	Description
DMA_CMD_START	Starts DMA transfer
DMA_CMD_STOP	Stops DMA transfer
DMA_CMD_SOFTWARE_TRIGGER	Issues DMA transfer request by software trigger
DMA_CMD_AUTO_CLEAR_SOFT_REQ	Enables/disables auto-clear function for DMA transfer requests generated by a software trigger
DMA_CMD_ACT_SRC_DISABLE	Disables DMA transfer, clears transfer trigger
DMA_CMD_REFRESH_DATA	Refreshes DMA transfer settings (transfer source address, transfer destination address, number of transfers)
DMA_CMD_REFRESH_EXTRA	Refreshes DMA transfer settings (transfer source address, transfer destination address, number of transfers, offset value) Also can select whether or not to hold DMAC start requests occurring from the stopping of DMA transfer to DMA transfer refresh

```
#include "r_dmac_api.h"
static void callback(void);
// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;
    /* Normal transfer mode, 8 bits, transfer source incremented, transfer destination incremented */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;
    (void)dmac0Drv->Open();                                          /* Open DMAC driver */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                          /* DMAC start trigger setting: Software trigger */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback);
                                           /* Enable DMA transfer complete interrupt */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);               /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* Issue a software trigger */
    while(1);
}
/******************************************************************************************
* callback function
******************************************************************************************/
static void callback(void)
{
    /*Describe what happens when all transfers are completed */
}
```

Figure 2-3 Example of Access to DMAC Driver (Normal Transfer)

```
#include "r_dmac_api.h"

static void callback(void);

// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;

    /* Repeat transfer mode, 8 bits, transfer source incremented, transfer destination
incremented */
    config.mode = (DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;                              /* Transfer size: 5 */
    config.block_size = 3;                                 /* Repeat size: 3 */
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                        /* Open DMAC driver */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                  /* DMAC start trigger setting: Software
trigger */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback); /* Enable DMA transfer
complete interrupt */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);               /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* Issue a software trigger */

    while(1);
}

/***********************************************************************************
* callback function
***********************************************************************************/
static void callback(void)
{
    /* Describes the processing when all transfers are completed (transfer size (5) x repeat
size (3) transfers completed) */
}
```

Figure 2-4 Example of Access to DMAC Driver (Repeat Transfer)

```
#include "r_dmac_api.h"

static void callback(void);

// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;

    /* Block transfer mode, 8 bits, transfer source incremented, transfer destination
incremented */
    config.mode = (DMA_MODE_BLOCK | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;                              /* Transfer size: 5 */
    config.block_size = 3;                                 /* Block size: 3 */
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                     /* Open DMAC driver */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                /* DMAC start trigger setting: Software
trigger */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback); /* Enable DMA transfer
complete interrupt */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);              /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* Issue a software trigger */

    while(1);
}

/***********************************************************************************
* callback function
***********************************************************************************/
static void callback(void)
{
    /* Describes the processing when all transfers are completed (5 blocks × 3 transfers) */
}
```

Figure 2-5 Example of Access to DMAC Driver (Block Transfer)

2.3 Setting DMAC Start Triggers

A DMAC start trigger is specified by the first argument of the Create function. For DMAC startup using a software trigger, DMAC_TRANSFER_REQUEST_SOFTWARE should be specified. For DMAC startup by an arbitrary interrupt request, the number of the event signal to link is specified. For details, refer to DMAC Event Link. Setting Register n (DELSRn) in the "RE01 1500KB Group User Manual: Hardware (r01uh0796)" or "RE01 256KB Group User's Manual: Hardware (r01uh0894)". In addition, the start trigger interrupt bit should be enabled.

Example: Case in which DMAC is started by an ADC140_ADI event (event number 23H)

```
dmac0Drv->Create(0x23, &config);
```

2.4 DMA Interrupts

When using a DMA interrupt, the InterruptEnable function should be used to enable the DMA interrupt that is to be used. Table 2-5 lists DMA transfer interrupt source definitions that are specified as the first argument of the InterruptEnable function.

Table 2-5 List of DMA Transfer Interrupt Sources

Definition	Description
DMA_INT_COMPLETE	DMA transfer end
DMA_INT_SRC_OVERFLOW	Transfer source address extended repeat area overflow
DMA_INT_DEST_OVERFLOW	Transfer destination address extended repeat area overflow
DMA_INT_REPEAT_END	DMA transfer repeat size end
DMA_INT_ESCAPE_END	DMA transfer escape end

By using the OR operator to combine DMA transfer interrupt source definition, multiple interrupt sources can be enabled.

Example:

```
(void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW | DMA_INT_ESCAPE_END, callback);
```

When a callback function is called at the time of occurrence of an interrupt request, the callback function is specified as the second argument. When polling the DMA interrupt occurrence flag using the GetStatus function without using a callback, NULL(0) is specified as the second argument.

When calling a callback function through a transfer source address extended repeat area overflow interrupt, a transfer destination address extended repeat area overflow interrupt, or a DMA transfer repeat size end interrupt, a transfer escape end interrupt must be simultaneously enabled. If a transfer escape end interrupt is not enabled, the callback function is not called. Figure 2-5 is a simplified logic diagram of DMA interrupt outputs.

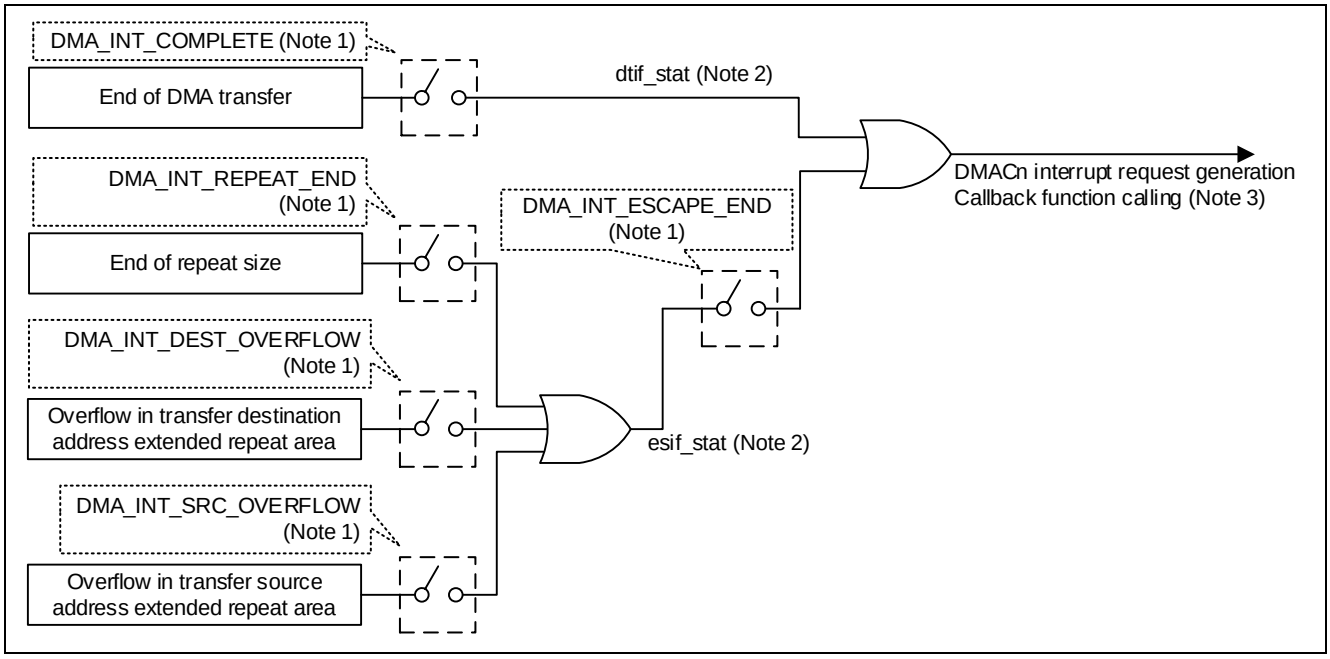


Figure 2-6 Simplified Logic Diagram of DMA Interrupt Outputs

Note 1. When this interrupt is specified as an argument of InterruptEnable function and is enabled, the interrupt request will be generated.

Note 2. If the concerned interrupt is enabled, the interrupt generation status can be acquired by the GetState function. The transfer escape end interrupt generation status and transfer end interrupt generation status are stored in the esif_state and dtif_stat members of the st_dma_state_t type structure specified as the second argument.

Note 3. When the InterruptEnable function is executed with NULL set as the second argument, a callback is not generated.

If interrupts generated by a transfer source address extended repeat area overflow, a transfer destination address extended repeat area overflow, or a DMA transfer repeat size end are enabled with the InterruptEnable function, DMA transfer stops (DMCNT.DTE = 0) when the relevant interrupt request occurs. To resume the DMA transfer, after the interrupt request occurs, execute the DMA_CMD_START command using the Control function.

Examples of using a DMA interrupt with a callback function used and with a callback function not used are shown in Figures 2-6 and 2-7. In addition, an example of DMA transfer with multiple DMA interrupts enabled is shown in Figure 2-8.

```
#include "r_dmac_api.h"

static void callback(void);

// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;

    /* Normal mode, 8 bits, transfer source incremented, transfer destination incremented */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                  /* Open DMAC driver */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                /* DMAC start trigger setting: Software
trigger */
    /* Enable the transfer source address extended repeat area overflow interrupt.
      To generate a callback, specify together with DMA_INT_ESCAPE_END using the OR
operator. */
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW  |  DMA_INT_ESCAPE_END, callback);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL);   /* Issue a software trigger
*/

    while(1);
}

/******************************************************************************************
* callback function
******************************************************************************************/
static void callback(void)
{
    /* Generate transfer source address extended repeat area overflow interrupt. */
    /* DMA transfer is stopped (DMCNT.DTE = 0). To resume transfer, execute START command.
*/
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);              /* Start DMA transfer*/
}
```

Figure 2-7 Example of Resuming DMA Transfer with Callback Function Used

```
#include "r_dmac_api.h"

// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;

    /* Normal mode, 8 bits, transfer source incremented, transfer destination incremented */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                          /* Open DMAC driver
*/
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                  /* DMAC start trigger setting: Software
trigger */
    /* Enable transfer source address extended repeat area overflow interrupt */
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW , NULL);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL);   /* Issue a software trigger
*/

    while(1)
    {
        /* Acquire DMAC status */
        dmac0Drv->GetState(dmac0_state);

        /* Transfer source address extended repeat area overflow interrupt requested? */
        if (1 == dmac0_state.esif_stat)
        {
            /* DMA transfer is stopped (DMCNT.DTE = 0).
              To resume the transfer, execute START command. */
            (void)dmac0Drv->Control(DMA_CMD_START, NULL);              /* Start DMA transfer
*/
        }
    }
}
```

Figure 2-8 Example of Resuming DMA Transfer with Callback Function Not Used

```
#include "r_dmac_api.h"

static void callback(void);

// DMAC driver instance ( DMAC0 )
extern DRIVER_DMA Driver_DMAC0;
DRIVER_DMA * dmac0Drv = &Driver_DMAC0;

// Transmit data
static uint8_t tx_data[5] = {0x11,0x12,0x13,0x14,0x15};
// Receive data
static uint8_t rx_data[5] = {0x00,0x00,0x00,0x00,0x00};

main()
{
    st_dma_transfer_data_cfg_t config;

    /* Normal mode, 8 bits, transfer source incremented, transfer destination incremented */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                              /* Open DMAC driver */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                        /* DMAC start trigger setting: Software
trigger */
    /* Enable a transfer source address extended repeat area overflow interrupt and a DMA
transfer end interrupt.
      To generate a callback with a transfer source address extended repeat area overflow
interrupt,
      specify together with DMA_INT_ESCAPE_END using the OR operator. */
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW | DMA_INT_ESCAPE_END | DMA_INT_COMPLETE,
                                callback);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* Start DMA transfer */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL);   /* Issue a software trigger
*/

    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    /* Acquire DMAC status */
    dmac0Drv->GetState(dmac0_state);
    /* Transfer source address extended repeat area overflow interrupt generated? */
    if (1 == dmac0_state.esif_stat)
    {
        /* DMA transfer is stopped (DMCNT.DTE = 0).
        To resume the transfer, execute START command. */
        (void)dmac0Drv->Control(DMA_CMD_START, NULL);              /* Start DMA transfer */
    }
    /* Is the transfer end interrupt generated? */
    if (1 == dmac0_state.dtif_stat)
    {
        /* Describe the processing to be performed when DMA transfer completes */
    }
}
```

Figure 2-9 Example of Enabling Multiple DMA Interrupts

2.5 NVIC Registration of DMAC Interrupts

When using the InterruptEnable function to enable DMA interrupts, the DMACn_INT (n=0 to 3) of the channel to be used must be registered using r_system_cfg.h in the nested vectored interrupt controller (hereafter NVIC). For details, refer to "Setting Interrupts (NVIC)" in the RE01 1500KB, 256KB Group Getting Started Guide to Development Using CMSIS Package. Figure 2-9 shows an example of DMA interrupt registration.

```
...
#define SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT
          (SYSTEM_IRQ_EVENT_NUMBER0)  /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DTC_COMPLETE
          (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  /*!< Numbers 0/4/8/12/16/20/24/28 only */
...
```

Figure 2-10 Example of Interrupt Registration to NVIC with r_system_cfg.h (DMAC0 used)

2.6 Macro and Type Definitions

For the DMAC/DTC driver, the macro and types that can be referenced by the user are defined in the r_dma_common_api.h file. Moreover, for the DMAC driver, the macro and types that can be referenced by the user are defined in the r_dmac_api.h file.

2.6.1 DMAC Control Code Definitions

DMAC control codes are DMAC control commands to be used by the Control function.

Table 2-6 List of DMAC Control Codes

Definition	Value	Description
DMA_CMD_START	(0x00)	Starts DMA transfer.
DMA_CMD_STOP	(0x01)	Disables DMA transfer.
DMA_CMD_ACT_SRC_ENABLE	(0x02)	Disabled (Note)
DMA_CMD_ACT_SRC_DISABLE	(0x03)	Disables DMAC start triggers
DMA_CMD_DATA_READ_SKIP_ENABLE	(0x04)	Disabled (Note)
DMA_CMD_CHAIN_TRANSFER_ABORT	(0x05)	Disabled (Note)
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	(0x06)	Disabled (Note)
DMA_CMD_SOFTWARE_TRIGGER	(0x07)	DMA transfer request by software trigger
DMA_CMD_AUTO_CLEAR_SOFT_REQ	(0x08)	Sets DMA software start bit auto-clear function
DMA_CMD_REFRESH_DATA	(0x09)	Refreshes DMA transfer (DMAC start request hold selection, no offset refresh)
DMA_CMD_REFRESH_EXTRA	(0x0A)	Refreshes DMA transfer (extended function) (DMAC start request hold selection, with offset refresh)

Note Can only be used with DTC driver. When this definition is specified using the Control function, DMA_ERROR is returned.

2.6.2 DMA Transfer Mode Definitions

A DMAC transfer mode definition is a definition set in the mode element of the st_dma_transfer_data_cfg_t structure. It is used in the second argument of the Create function. The mode setting is specified by combining settings.

Example: Repeat transfer, 8-bit size transfer, transfer source increment, transfer destination fixed, transfer source is repeat area

```
st_dma_transfer_data_cfg_t config;   /* DMA transfer setting information storage area */

config.mode = DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
              DMA_REPEAT_BLOCK_SRC;
```

Figure 2-10 shows the format of a DMAC transfer mode definition. Tables 2-7 to 2-11 show details of the various settings.

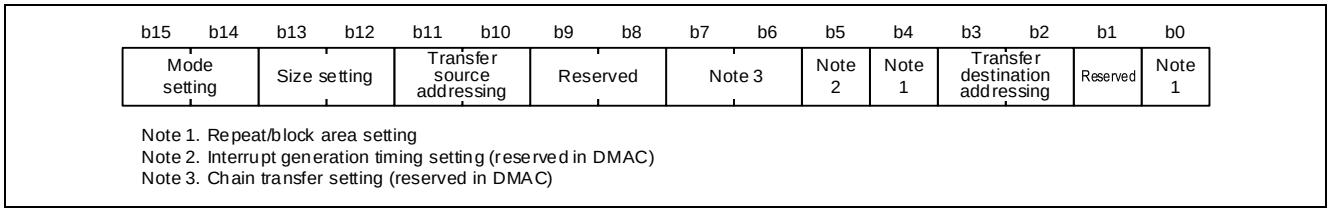


Figure 2-11 Structure of DMAC Transfer Mode Definitions

Table 2-7 List of DMA Transfer Mode Definitions

Definition	Value	Description
DMA_MODE_NORMAL	(0x0000U)	Normal transfer
DMA_MODE_REPEAT	(0x4000U)	Repeat transfer
DMA_MODE_BLOCK	(0x8000U)	Block transfer

Table 2-8 List of DMA Transfer Size Definitions

Definition	Value	Description
DMA_SIZE_BYTE	(0x0000U)	8-bit size transfer
DMA_SIZE_WORD	(0x1000U)	16-bit size transfer
DMA_SIZE_LONG	(0x2000U)	32-bit size transfer

Table 2-9 List of DMA Transfer Repeat Area Definitions

Definition	Value	Description
DMA_REPEAT_BLOCK_DEST	(0x0000U)	Transfer destination area specified as a repeat area
DMA_REPEAT_BLOCK_SRC	(0x0010U)	Transfer source area specified as a repeat area
DMA_REPEAT_BLOCK_NONE	(0x0001U)	Transfer area not specified as a repeat area

Table 2-10 List of DMA Transfer Source Address Modes

Definition	Value	Description
DMA_SRC_FIXED	(0x0000U)	Transfer source address fixed
DMA_SRC_INCR	(0x0800U)	Address incremented after transfer
DMA_SRC_DECR	(0x0C00U)	Address decremented after transfer
DMA_SRC_ADDR_OFFSET	(0x0400U)	Offset value set for transfer source address

Table 2-11 List of DMA Transfer Destination Address Modes

Definition	Value	Description
DMA_DEST_FIXED	(0x0000U)	Transfer destination address fixed
DMA_DEST_INCR	(0x0008U)	Address incremented after transfer
DMA_DEST_DECR	(0x000CU)	Address decremented after transfer
DMA_DEST_ADDR_OFFSET	(0x0004U)	Offset value set for transfer destination address

2.6.3 DMA Transfer Interrupt Source Definitions

These define interrupt sources specified by the InterruptEnable function.

Table 2-12 List of DMA Transfer Interrupt Source Definitions

Definition	Value	Description
DMA_INT_COMPLETE	(1U<<0)	DMA transfer end
DMA_INT_SRC_OVERFLOW	(1U<<1)	Transfer source address extended repeat area overflow
DMA_INT_DEST_OVERFLOW	(1U<<2)	Transfer destination address extended repeat area overflow
DMA_INT_REPEAT_END	(1U<<3)	DMA transfer repeat size end
DMA_INT_ESCAPE_END	(1U<<4)	DMA transfer escape end

2.6.4 Definitions for DMAC Status Flag Clearing

These definitions are for interrupt flags and transfer request flags specified by the ClearState function.

Table 2-13 List of Definitions for DMAC Interrupt Flags and Transfer Request Flags to be Cleared

Definition	Value	Description
DMA_CLR_STATE_ESIF	(1U<<1)	Transfer escape end interrupt flag
DMA_CLR_STATE_DTIF	(1U<<2)	Transfer end interrupt flag
DMA_CLR_STATE_SOFT_REQ	(1U<<3)	Software trigger transfer request flag

2.7 Structure Definitions

For the DMAC driver, the structures that can be referenced by the user are defined in the r_dma_common_api.h file.

2.7.1 st_dma_transfer_data_cfg_t Structure

This structure is used when the DMAC transfer setting information is specified with the Create function.

Table 2-14 st_dma_transfer_data_cfg_t Structure

Element Name	Type	Description
mode	uint16_t	Specifies the transfer mode, transfer size, repeat area, transfer source/transfer destination address mode using the OR operator
src_addr	uint32_t	Specifies the transfer source address
dest_addr	uint32_t	Specifies the transfer destination address
transfer_count	uint32_t	Specifies the number of transfers
block_size	uint32_t	Specifies the block transfer size
offset	int32_t	Specifies the offset value
src_extended_repeat	uint8_t	Specifies the transfer source extended repeat area
dest_extended_repeat	uint8_t	Specifies the transfer destination extended repeat area
*p_transfer_data	void	Disabled (Note 1)

Note 1. Can be used only with the DTC driver. This element is ignored.

2.7.2 st_dma_refresh_data_t Structure

This structure is used when the DMA transfer refresh commands (DMA_CMD_REFRESH_DATA and DMA_CMD_REFRESH_EXTRA) are specified using the Control function.

Table 2-15 st_dma_refresh_data_t Structure

Element Name	Type	Description
act_src	IRQn_Type	Disabled (Note 1)
chain_transfer_nr	uint32_t	Disabled (Note 1)
src_addr	uint32_t	Specifies the transfer source address
dest_addr	uint32_t	Specifies the transfer destination address
transfer_count	uint32_t	Specifies the number of transfers
block_size	uint32_t	Specifies the block transfer size
auto_start	bool	Specifies whether auto-start is enabled 0: Auto-start enabled 1: Auto-start disabled
keep_dma_req (Note 2)	bool	Specifies whether DMAC start requests occurring from the time DMA transfer is stopped until DMA refresh are held 0: DMAC start requests not held 1: DMAC start requests held
Offset (Note 2)	int32_t	Specifies the offset value

Note 1. Can be used only with the DTC driver. This element is ignored.

Note 2. Can be used only with the DMA transfer refresh extended command (DMA_CMD_REFRESH_EXTRA). The DMA_CMD_REFRESH_DATA command ignores this element.

2.7.3 st_dma_state_t Structure

This structure is used to store the status acquired with the GetState function.

Table 2-16 st_dma_state_t Structure

Element Name	Type	Description
in_progress	bool	Indicates the DMA transfer active status 0: Inactive 1: Active
esif_stat	bool	Indicates the status of the transfer escape end interrupt flag 0: No interrupt occurred 1: Interrupt occurred
dtif_stat	bool	Indicates the status of the transfer end interrupt flag 0: No interrupt occurred 1: Interrupt occurred
soft_req_stat	bool	Indicates the status of the software trigger transfer request flag 0: DMA transfer request flag not set 1: DMA transfer request flag set

2.8 State Transitions

The state transition diagram of the DMAC driver is shown in Figure 2-12, and state-specific events and actions are shown in Table 2-17.

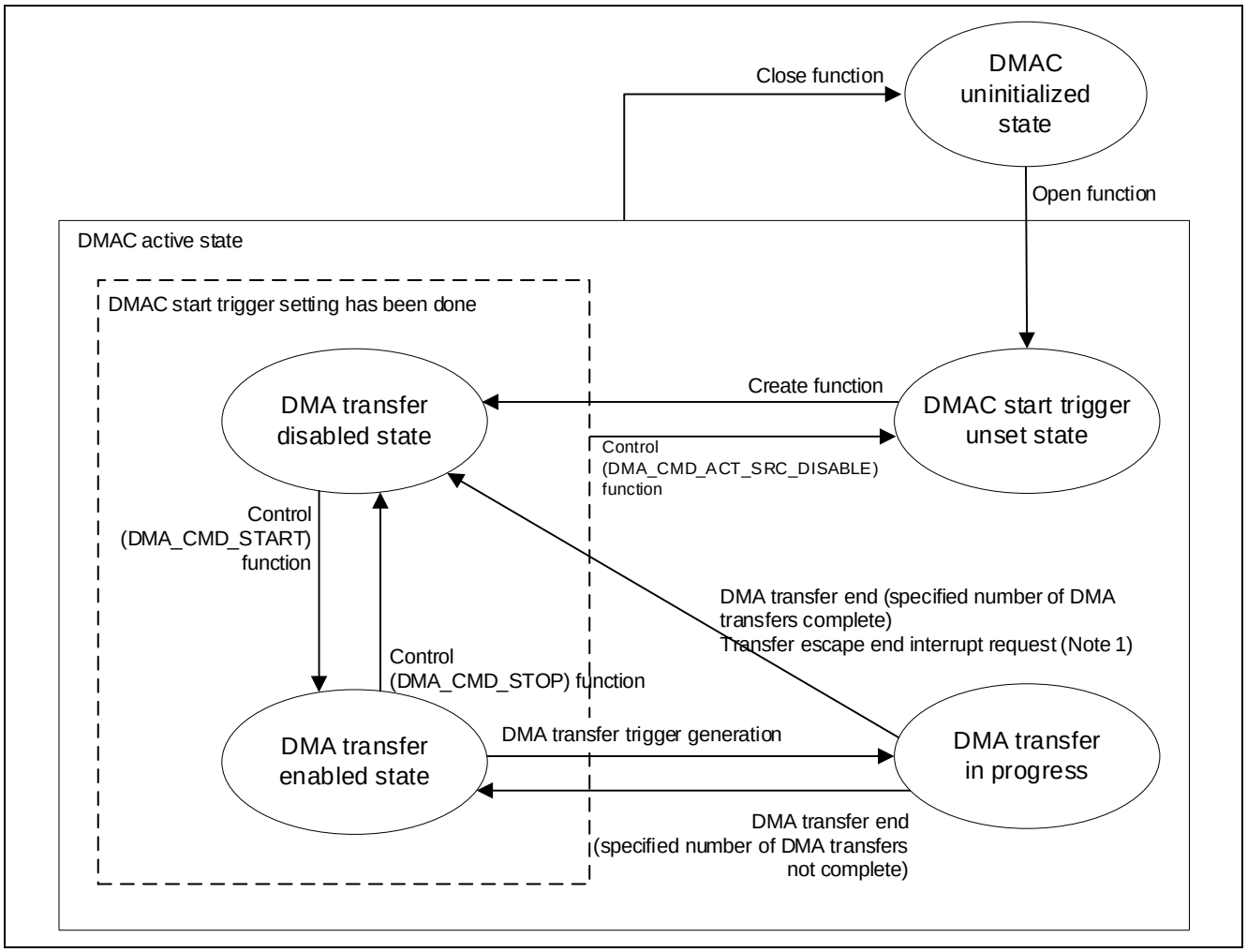


Figure 2-12 State Transitions of DMAC Driver

Note 1. When either a repeat size end interrupt, a transfer source address extended repeat area overflow interrupt, or a transfer destination address extended repeat area overflow interrupt request has occurred, there is a transition to the DMA transfer prohibition state (DMCNT.DTE = 0).

Table 2-17 Events and Actions Specific to DMAC Driver State (Note)

State	Overview	Event	Action
DMAC uninitialized state	The DMAC driver is in this state after release from a reset.	Execution of Open function	Enters the DMAC start trigger unspecified state.
DMAC start trigger unset state	DMAC operation is enabled (DMAST.DMST = 1) and DMAC start trigger is not set in this state.	Execution of Create function	Enters the DMA transfer disabled state.
DMA transfer disabled state	DMA transfer is disabled (DMCNT.DTE = 0) and DMAC start trigger has been set in this state.	Execution of Control(DMA_CMD_ACT_SRC_DISABLE) function	Enters the DMAC start trigger unspecified state.
		Execution of Control(DMA_CMD_START) function	Enters the DMA transfer enabled state.
		Execution of GetTransferByte function	Get the number of DMA transfer bytes
DMA transfer enabled state	DMA transfer is enabled (DMCNT.DTE = 1) and DMAC start trigger has been set in this state.	Execution of Control(DMA_CMD_ACT_SRC_DISABLE) function	Enters the DMAC start trigger unspecified state.
		Execution of Control(DMA_CMD_STOP) function	Enters the DMA transfer disabled state.
		Generation of DMAC start trigger	Enters DMA transfer state.
		Execution of GetTransferByte function	Get the number of DMA transfer bytes
DMA transfer state	DMA transfer is in progress in this state.	DMA transfer complete (specified number of DMA transfers not complete)	Enters the DMA transfer enabled state.
		DMA transfer complete (specified number of DMA transfers complete)	Enters the DMA transfer disabled state.
		Transfer escape end interrupt request (occurrence of either a repeat size end interrupt, transfer source address extended repeat area overflow interrupt, or transfer destination address extended repeat area overflow interrupt request)	Enters the DMA transfer disabled state.

Note The GetVersion function can be executed in any state.

3. Descriptions of Driver Operations

The DMAC driver realizes DMAC data transfers. This chapter describes the procedures for execution in different operating modes.

3.1 Normal Transfer Mode

In normal transfer mode, a start trigger will cause a 8-bit, 16-bit, or 32-bit data transfer. The number of transfers can be set to between 0 and 65,535 times. If the number of transfers is set to 0, operation is in free running mode. The procedure for normal transfers is shown in Figure 3-1.

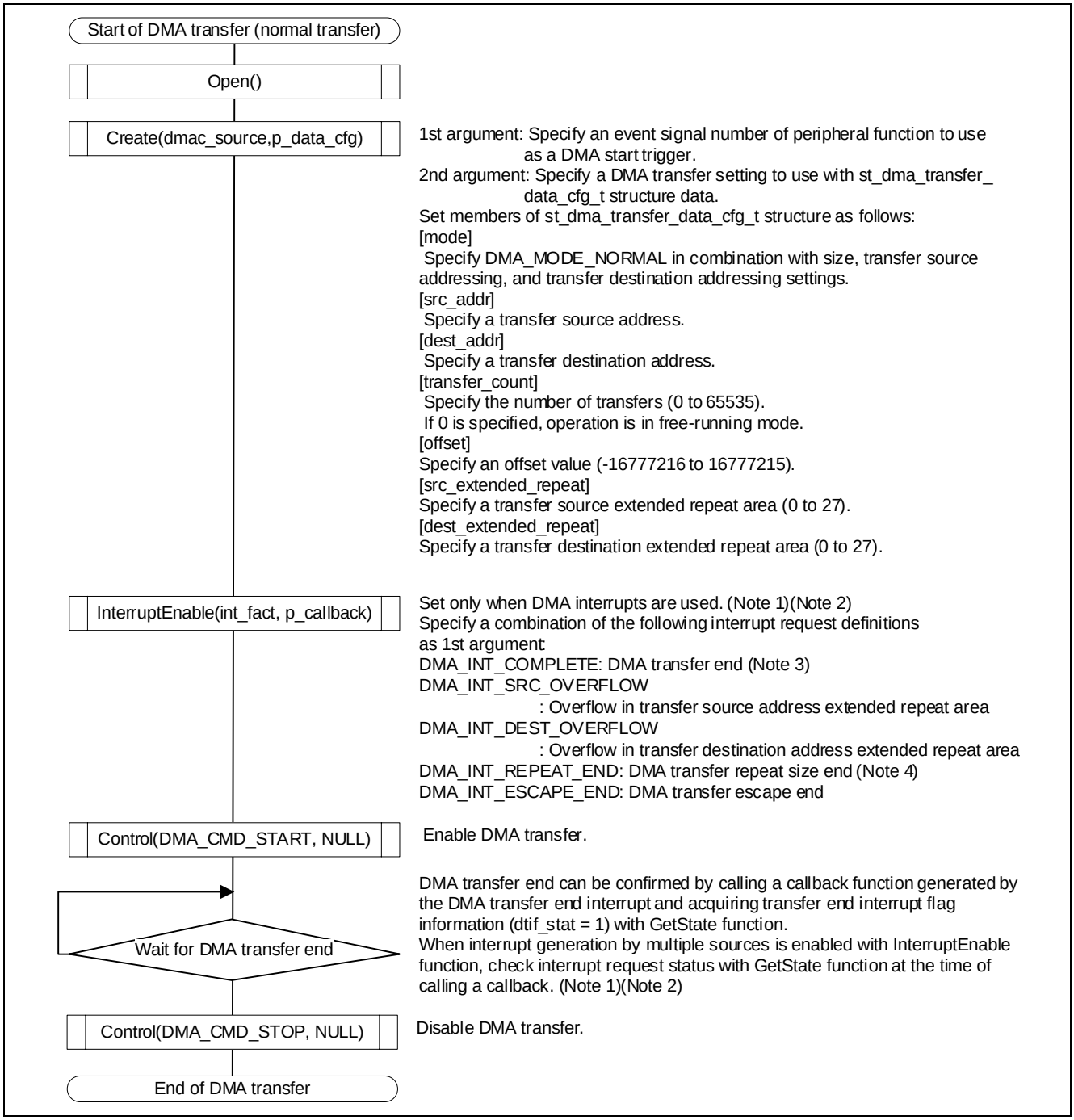


Figure 3-1 Normal Transfer Procedure

Note 1. When a callback function is called by a transfer source address extended repeat area overflow (DMA_INT_SRC_OVERFLOW) or transfer destination address extended repeat area overflow (DMA_INT_DEST_OVERFLOW) interrupt, the InterruptEnable function should be used to enable the interrupt together with the transfer escape end interrupt (DMA_INT_ESCAPE_END).

Note 2. When a transfer source address extended repeat area overflow interrupt request, transfer destination address extended repeat area overflow interrupt request, or DMA transfer repeat size end interrupt request occurs, DMA transfer stops (DMCNT.DTE = 0). When DMA transfer is to be resumed, the Control function should be used to execute the DMA_CMD_START command.

Note 3. When in free running mode, a transfer complete interrupt does not occur.

Note 4. In normal mode, a repeat size end interrupt does not occur.

3.2 Repeat Transfer Mode

In repeat transfer mode, a start trigger will cause a 8-bit, 16-bit, or 32-bit data transfer. The number of transfers can be set to between 1 and 65,536 times, and the repeat size can be set to between 1 and 1024 times. The procedure for repeat transfers is shown in Figure 3-2.

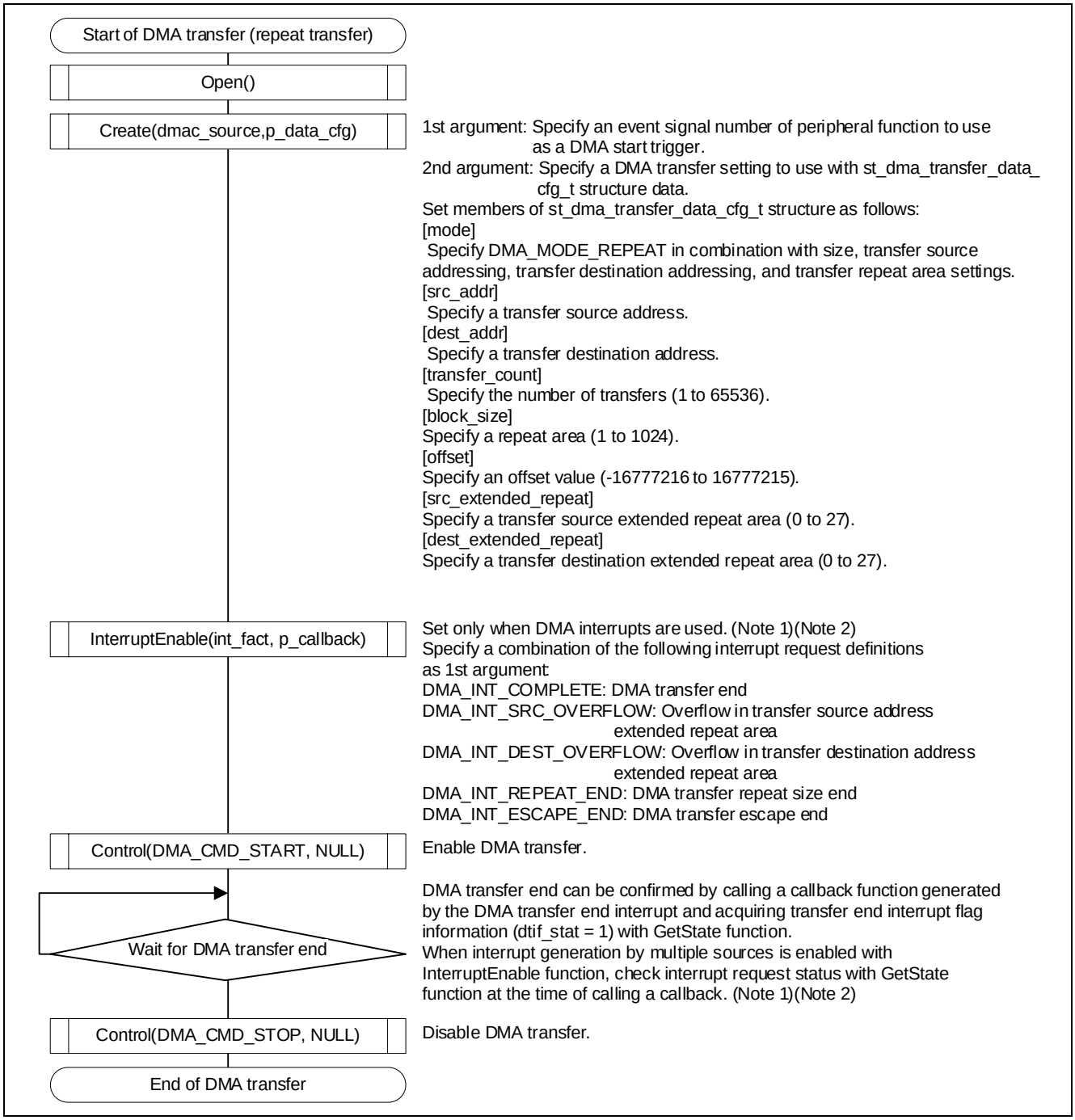


Figure 3-2 Repeat Transfer Procedure

Note 1. When a callback function is called by a transfer source address extended repeat area overflow (DMA_INT_SRC_OVERFLOW), transfer destination address extended repeat area overflow (DMA_INT_DEST_OVERFLOW), or transfer repeat size end (DMA_INT_REPEAT_END) interrupt, the InterruptEnable function should be used to enable the interrupt together with the transfer escape end interrupt (DMA_INT_ESCAPE_END).

Note 2. When a transfer source address extended repeat area overflow interrupt request, transfer destination address extended repeat area overflow interrupt request, or DMA transfer repeat size end interrupt request occurs, DMA transfer stops (DMCNT.DTE = 0). When DMA transfer is to be resumed, the Control function should be used to execute the DMA_CMD_START command.

3.3 Block Transfer Mode

In block transfer mode, a start trigger will cause the transfer of one block of data. The block size can be set to between 1 and 1024 bytes (when the transfer size is set to 8 bits), 2 and 2048 bytes (when the transfer size is set to 16 bits), or 4 and 4096 bytes (when the transfer size is set to 32 bits). The number of transfers (number of blocks) can be set to between 1 and 65,536. Figure 3-3 shows the procedure for block transfers.

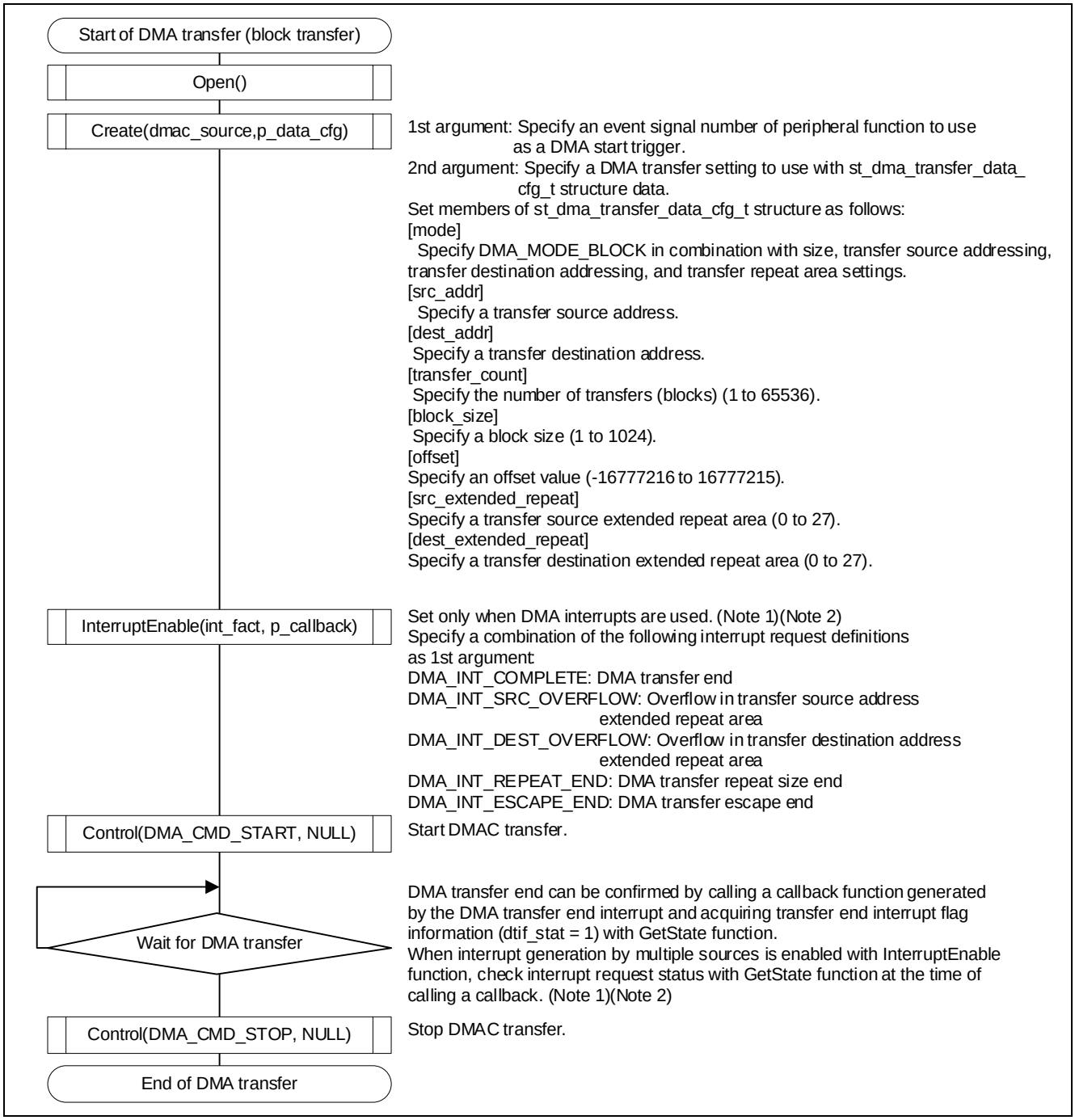


Figure 3-3 Block Transfer Procedure

Note 1. When a callback function is called by a transfer source address extended repeat area overflow (DMA_INT_SRC_OVERFLOW), transfer destination address extended repeat area overflow (DMA_INT_DEST_OVERFLOW), or transfer repeat size end (DMA_INT_REPEAT_END) interrupt, the InterruptEnable function should be used to enable the interrupt together with the transfer escape end interrupt (DMA_INT_ESCAPE_END).

Note 2. When a transfer source address extended repeat area overflow interrupt request, transfer destination address extended repeat area overflow interrupt request, or DMA transfer repeat size end interrupt request occurs, DMA transfer stops (DMCNT.DTE = 0). When DMA transfer is to be resumed, the Control function should be used to execute the DMA_CMD_START command.

3.4 DMAC Control Using the Control Function

The Control function can be used to control various DMAC functions. Table 3-1 lists the control commands and the arguments for each command.

Table 3-1 List of Operation with Control Commands and the Command-Specific Arguments

Control Command (cmd)	Command-Specific Argument (arg)	Description
DMA_CMD_START	NULL	Starts DMACn transfer (Note 2)
DMA_CMD_STOP	NULL	Stops DMACn transfer (Note 2)
DMA_CMD_ACT_SRC_DISABLE	NULL	Disables DMACn transfer triggers (Note 2)
DMA_CMD_SOFTWARE_TRIGGER	NULL	Issues software-triggered DMA transfer request
DMA_CMD_AUTO_CLEAR_SOFT_REQ	bool	Sets DMA software start bit auto-clear function to enable/disable True: Auto-clear enabled False: Auto-clear disabled
DMA_CMD_REFRESH_DATA	st_dma_refresh_data_t type pointer (Note 1)	Refreshes the transfer source, transfer destination, and number of transfers for the trigger specified by the argument
DMA_CMD_REFRESH_EXTRA	st_dma_refresh_data_t type pointer (Note 1)	Refreshes the transfer source, transfer destination, and number of transfers for the trigger specified by the argument In addition, refreshes the offset value and selects holding for DMAC start requests

Note 1. Settings for the specified variables are stored and are passed in the argument as an address.

Note 2. n = 0 to 3

3.4.1 DMA Transfer Start Command (DMA_CMD_START)

This sets the DMA transfer enabled state (DMCNT.DTE = 1). NULL(0) should be set as the argument. Figure 3-4 shows an example of using the DMA transfer start command.

```
(void)dmac0Drv->Control(DMA_CMD_START, NULL);
```

Figure 3-4 Example of DMA Transfer Start Command

3.4.2 DMA Transfer Stop Command (DMA_CMD_STOP)

This sets the DMA transfer disabled state (DMCNT.DTE = 0). NULL(0) should be set as the argument. Figure 3-5 shows an example of using the DMA transfer stop command.

```
(void)dmac0Drv->Control(DMA_CMD_STOP, NULL);
```

Figure 3-5 Example of DMA Transfer Stop Command

3.4.3 DMAC Start Trigger Disable Command (DMA_CMD_ACT_SRC_DISABLE)

DELSRn.DELS (n = 0 to 3) for the channel used is set to "0" (disabling DMAC module interrupts), and DMA transfers are stopped (DMCNT.DTE = 0). NULL(0) should be set as the argument. After DMAC start triggers have been disabled, if DMA transfers are to be performed again, the Create function must be used to set a DMA start trigger. Figure 3-6 shows an example of using the DMAC start trigger disable command.

```
(void)dmac0Drv->Control(DMA_CMD_ACT_SRC_DISABLE, NULL);
```

Figure 3-6 Example of DMAC Start Trigger Disable Command

3.4.4 DMA Software Start Command (DMA_CMD_SOFTWARE_TRIGGER)

DMREQ.SWREQ is set to "1" and a software DMA transfer request is issued. Figure 3-7 shows an example of using the DMA software start command.

```
(void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL);
```

Figure 3-7 Example of DMA Software Start Command

3.4.5 DMA Software Start Bit Set Auto-Clear Function Command (DMA_CMD_AUTO_CLEAR_SOFT_REQ)

Enables or disables the DMA software start bit auto-clear function. When the second argument is set to "0", the DMA software start bit auto-clear function is disabled, and when set to "1", the function is enabled.

When the Open function is used to initialize the DMAC, by default the software start bit auto-clear function is in the enabled state. When the software start bit auto-clear function is disabled, the software start request flag should be cleared using the ClearStatus (definition name) function. Figure 3-8 shows an example of using the DMA software start bit auto-clear function.

```
uint8_t arg = true;
  /* Enable DMA software start bit auto-clear function */
  (void)dmac0Drv->Control(DMA_CMD_AUTO_CLEAR_SOFT_REQ, &arg);
```

Figure 3-8 Example of DMA Software Start Bit Auto-Clear Function Command

3.4.6 DMA Transfer Refresh Command (DMA_CMD_REFRESH_DATA)

This command refreshes DMA transfer information.

After a DMA transfer has ended, without changing the mode, only the transfer source address (Note), transfer destination address (Note), and number of transfers are updated. When "1" is set in the auto-start member, after execution of this command, DMA transfer is resumed. If refresh is performed with the auto-start member set as "0", DMA transfer should be enabled using the DMA transfer enable command (DMA_CMD_START) at a given timing.

When NULL(0) is set as the transfer source address, the current transfer source address is held. Similarly, when NULL(0) is set as the transfer destination address, the current transfer destination address is held.

This command does not use the chain_transfer_nr member, the keep_dma_req member, or the offset member. "0" should be set. DMAC start requests occurring between the end of a DMA transfer and completion of DMA refresh are ignored (DELSRn.IR is cleared to 0).

Note The transfer source address and transfer destination address should be set to a multiple of 2 when the transfer bit size is 16 bits and to a multiple of 4 when the transfer bit size is 32 bits.

```
/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    st_dma_refresh_data_t arg;

    /* Refresh DMA transfer in DMAC0 */
    arg.act_src          = 0;                          /* Set to 0 due to not being used in DMAC. */
    arg.chain_transfer_nr = 0;                         /* Set to 0 due to not being used in DMAC. */
    arg.src_addr         = (uint32_t)(&source_ref); /* Update transfer source */
    arg.dest_addr        = NULL;                       /* Hold transfer destination */
    arg.transfer_count   = 5;                          /* Update number of transfers */
    arg.block_size        = 0;                        /* Update block size (only for block transfer) */
    arg.auto_start       = true;                       /* Enable DMA transfer after refresh command execution */
    (void)dmac0Drv->Control(DMA_CMD_REFRESH_DATA, &arg);
}
```

Figure 3-9 Example of DMA Transfer Refresh Command in Callback Function

3.4.7 DMA Transfer Refresh Extended Command (DMA_CMD_REFRESH_EXTRA)

This command refreshes the DMA transfer information. In addition to the DMA_CMD_REFRESH command functions, it is also possible to update the offset value and to select holding of DMAC start requests (Note 1).

After a DMA transfer has ended, without changing the mode, the transfer source address (Note 2), transfer destination address (Note 2), and number of transfers are updated. When "1" is set in the auto-start member, after execution of this command, DMA transfer is resumed. If refresh is performed with the auto-start member as set "0", DMA transfer should be enabled with the DMA transfer enable command (DMA_CMD_START) at a given timing.

When NULL(0) is set as the transfer source address, the current transfer source address is held. Similarly, when NULL(0) is set as the transfer destination address, the current transfer destination address is held.

As an extended function, updating of the offset value and selection to hold DMAC start requests can be performed. The offset value for updating should be set in the offset member. When holding DMAC start requests, the keep_dma_req member should be set to "1". When the function to hold DMAC start requests is enabled, one DMAC start request occurring during DMA transfer refresh after the end of a DMA transfer is held. If the keep_dma_req member is set to "0", any DMAC start requests occurring during DMA transfer refresh after the end of a DMA transfer are ignored (DELSRn.IR is cleared to 0).

The chain_transfer_nr member is not used in the DMAC, and should be set to "0".

Note 1. Refers to the DMAC start request status flag (DELSRn.IR).

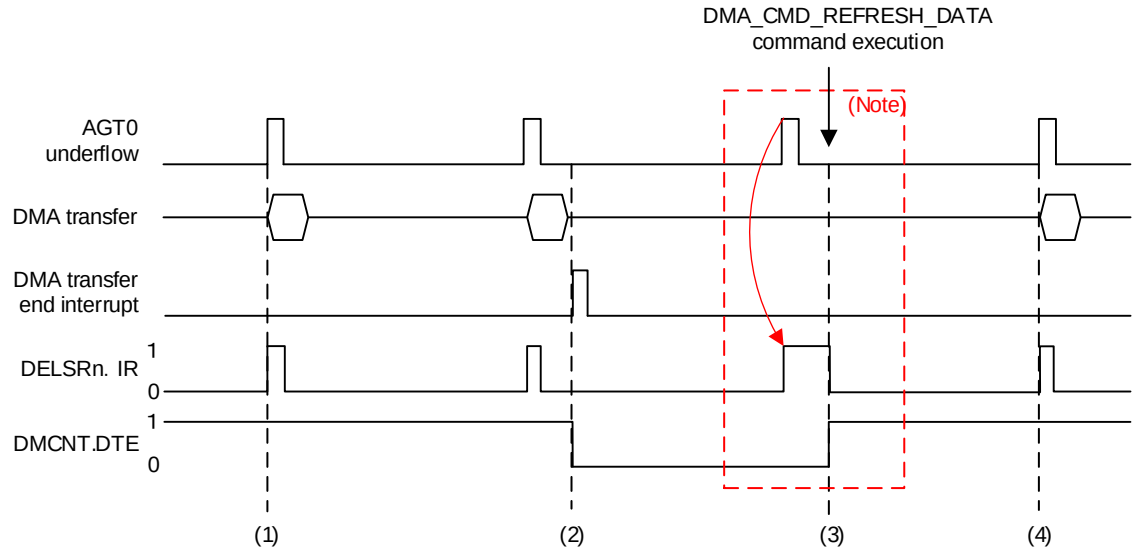
Note 2. The transfer source address and transfer destination address should be set to a multiple of 2 when the transfer bit size is 16 bits and to a multiple of 4 when the transfer bit size is 32 bits.

```
/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    st_dma_refresh_data_t arg;

    /* Refresh DMA transfer in DMAC0 */
    arg.act_src          = 0;                          /* Set to 0 due to not being used in DMAC. */
    arg.chain_transfer_nr = 0;                         /* Set to 0 due to not being used in DMAC. */
    arg.src_addr         = (uint32_t)(&source_ref); /* Update transfer source */
    arg.dest_addr        = NULL;                       /* Hold transfer destination */
    arg.transfer_count   = 5;                          /* Update number of transfers */
    arg.block_size        = 0;                         /* Update block size (only for block transfer) */
    arg.auto_start       = true;                       /* Enable DMA transfer after refresh command execution */
    keep_dma_req       = true;                       /* Enable holding of DMAC start requests generated during
DMA transfer refresh */
    offset               = 2;                          /* Refresh an offset value */
```

Figure 3-10 Example of DMA Transfer Refresh (Extended) Command

Figure 3-11 and Figure 3-12 show examples of DMA transfer refresh operations by the DMA transfer refresh command (DMA_CMD_REFRESH_DATA) and the DMA transfer refresh extended command (DMA_CMD_REFRESH_EXTRA).



Note: DMA transfer requests generated between the DMA transfer end and refresh completion are ignored.

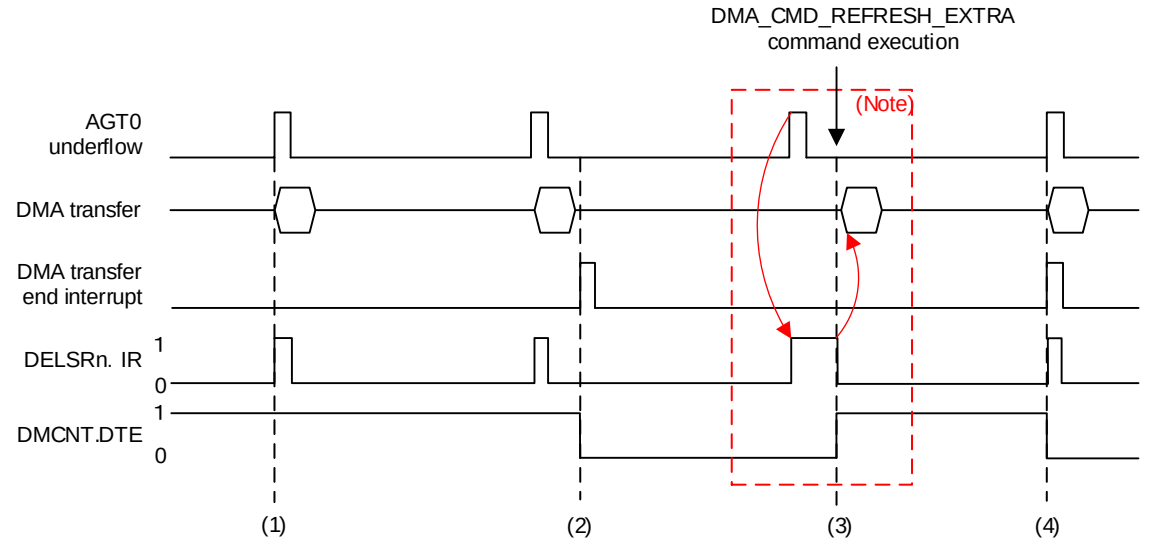
Figure 3-11 Example of DMA_CMD_REFRESH_DATA Command Operation

(1) DMA transfer is executed for one data item with AGt0 underflow as the start trigger.

(2) DMA transfer of the last data is executed. When all DMA transfers have ended, a DMA transfer end interrupt occurs. DMCNT.DTE becomes 0.

(3) The Control function is used to execute the DMA_CMD_REFRESH_DATA command, refreshing the DMA settings (number of transfers 2, auto-start enabled). If auto-start was already enabled, DMCNT.DTE becomes 1. DMA start requests occurring from the end of a DMA transfer until completion of DMA transfer refresh are ignored (DELSRn.IR = 0).

(4) With an AGT0 underflow that occurred after the DMA refresh as the start trigger, a DMA transfer is executed.



Note: When DMA start request holding function is enabled, DMA transfer is executed by a DMA start request generated between the DMA transfer end and refresh completion.

Figure 3-12 Example of DMA_CMD_REFRESH_EXTRA Command (DMA start request holding enabled) Operation

(1) DMA transfer is executed for one data item with AGt0 underflow as the start trigger.

(2) DMA transfer of the last data is executed. When all DMA transfers have ended, a DMA transfer end interrupt occurs. DMCNT.DTE becomes 0.

(3) The Control function is used to execute the DMA_CMD_REFRESH_DATA command, refreshing the DMA settings (number of transfers 2, auto-start enabled, DMA start request holding enabled). If auto-start was already enabled, DMCNT.DTE becomes 1. If DMA transfer request holding had been enabled, a DMA transfer is executed by a DMA start request that occurred between the end of a DMA transfer and completion of DMA transfer refresh.

(4) With an AGT0 underflow that occurred after the DMA refresh as the start trigger, a DMA transfer is executed. All DMA transfers end, and a DMA transfer end interrupt occurs. DMCNT.DTE becomes 0.

3.5 Get number of DMA transfer bytes

The number of DMA transfer bytes can be acquired with the GetTransferByte function. Set a variable to store the acquisition result in the second argument. The first argument is not used by the DMAC driver.

If the transfer count is set to 0 (free run mode) in normal transfer mode, the transfer byte count cannot be acquired.

When the GetTransferByte function is executed, DMA_ERROR is returned.

Example of using GetTransferByte function Figure 3-13, Example of acquiring DMA transfer byte number in normal transfer mode Table 3-2, Example of acquiring DMA transfer byte number in repeat transfer mode Table 3-3, Example of DMA in block transfer mode An example of acquiring the transfer byte count is shown in Table 3-4.

```
uint32_t transfer_byte;
/* Get DMAC transfer byte count  */
(void)dmac0Drv->GetTransferByte(0, &transfer_byte);
```

Figure 3-13 Example of using GetTransferByte function

Table 3-2 Example of acquiring DMA transfer byte count in normal transfer mode (transfer size 2 bytes)

Number of transfer factor occurrences	1	2	3	4	5	6	7
DMA transfer byte count (Note)	2	4	6	8	10	12	14

Table 3-3 Example of acquiring DMA transfer byte count in repeat transfer mode (transfer size 4 bytes, repeat size 5)

Number of transfer factor occurrences	1	2	3	4	5(Note2)	6	7
DMA transfer byte count (Note)	4	8	12	16	20	24	28

Table 3-4 Example of acquiring DMA transfer byte count in block transfer mode (transfer size 1 byte, block size 10)

Number of transfer factor occurrences	1	2	3	4	5	6	7
DMA transfer byte count (Note)	10	20	30	40	50	60	70

Note 1. Number of DMA transfer bytes acquired with the GetTransferByte function

Note 2. The DMAC driver continues counting the number of DMA transfer bytes even after the repeat size has been transferred.(Does not return to 0)

3.6 Configurations

For DMAC driver, configuration definitions that can be modified by the user are provided in the r_dmac_cfg.h file.

3.6.1 Parameter Checks

Parameter checks in the DMAC are enabled or disabled.

Name: DMAC_CFG_PARAM_CHECKING_ENABLE

Table 3-5 Setting of DMAC_CFG_PARAM_CHECKING_ENABLE

Setting	Description
0	Disables parameter checks Errors relating to validity of arguments appearing in function specifications are not detected.
1 (initial value)	Enables parameter checks Error conditions for parameters appearing in function specifications are detected.

3.6.2 DMACn_INT Interrupt Priority Level (n = 0 to 3)

This sets the priority level of the DMACn_INT interrupt.

Name: DMACn_INT_PRIORITY

Table 3-6 Setting of DMACn_INT_PRIORITY

Setting	Description
0	Sets the interrupt priority level to 0 (Highest).
1	Sets the interrupt priority level to 1.
2	Sets the interrupt priority level to 2.
3 (initial value)	Sets the interrupt priority level to 3 (Lowest).

3.6.3 Function Allocation to RAM

This initializes the settings for executing specific functions of the DMAC driver in RAM.

This configuration definition for setting function allocation to RAM has function-specific definitions.

Name: DMAC_CFG_xxx

A function name xxx should be written in all capital letters.

Example: R_DMAC_Open function → DMAC_CFG_R_DMAC_OPEN

Table 3-7 Settings of DMAC_CFG_xxx

Setting	Description
SYSTEM_SECTION_CODE	Does not allocate the function to RAM.
SYSTEM_SECTION_RAM_FUNC	Allocates the function to RAM.

Table 3-8 Initial State of Function Allocation to RAM

No.	Function Name	Allocation to RAM
1	R_DMAC_GetVersion	
2	R_DMAC_Open	
3	R_DMAC_Close	
4	R_DMAC_Create	
5	R_DMAC_Control	
6	R_DMAC_InterruptEnable	
7	R_DMAC_InterruptDisable	
8	R_DMAC_GetState	
9	R_DMAC_ClearState	
10	R_DMAC_GetTransferByte	
11	dmac_interrupt_handler (DMACn_INT (n=0 to 3) interrupt handling process)	✔

4. Detailed Information of Driver

This chapter describes the detailed specifications implementing the functions of this driver.

4.1 Function Specifications

The specifications and processing flow of each function of the DMAC driver are described in this section.

In the flow of processing, some decision methods such as conditional branching are omitted, and consequently the flowcharts do not exactly show the actual processing.

4.1.1 R_DMAC_Open Function

Table 4-1 R_DMAC_Open Function Specifications

Format	static e_dma_err_t R_DMAC_Open(st_dmac_resources_t * const p_dmac)
Description	Opens the DMAC driver (clears the module stop, initializes the RAM to be used)
Argument	st_dmac_resources_t * const p_dmac : Resources of DMAC Specifies the DMAC resource to open.
Return value	DMA_OK DMAC open success
	DMA_ERROR DMAC open failure If either of the following states is detected, the open command fails. • When the DMAC channel resource to be used is already open • When the transition to module stop fails (when an error occurs upon R_LPM_ModuleStart)
	DMA_ERROR_LOCKED DMAC resource lock failure If the DMAC channel resource to be used is locked, a lock failure occurs (when R_DMACn is locked by the R_SYS_ResourceLock function)

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv->Open();
	}

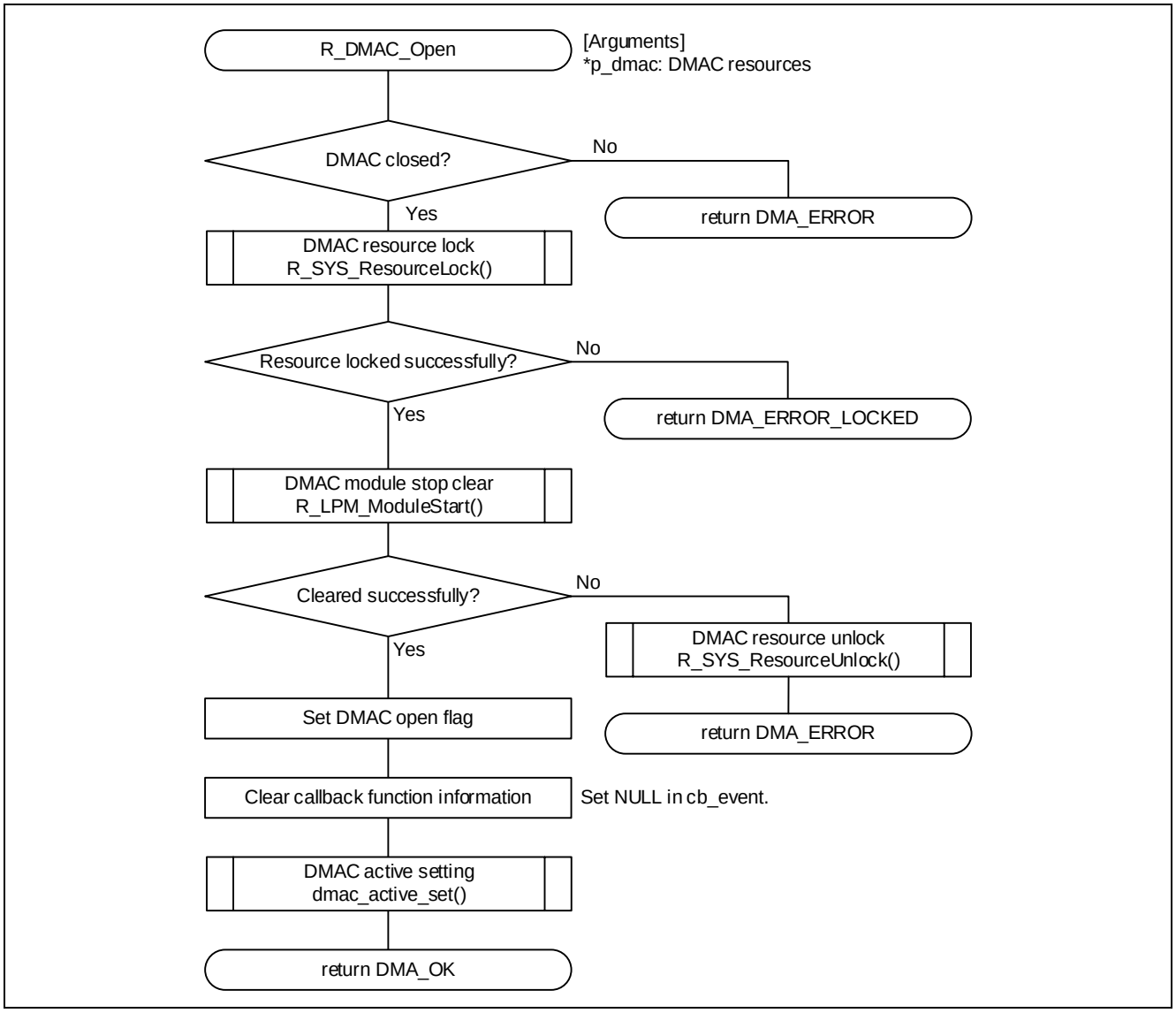


Figure 4-1 R_DMAC_Open Function Processing Flow

4.1.2 R_DMAC_Close Function

Table 4-2 R_DMAC_Close Function Specifications

Format	static e_dma_err_t R_DMAC_Close(st_dmac_resources_t * const p_dmac)
Description	Releases the DMAC driver.
Argument	st_dmac_resources_t * const p_dmac : Resources of DMAC Specifies the DMAC resource to be released.
Return value	DMA_OK DMAC released normally

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv->Close();
	}

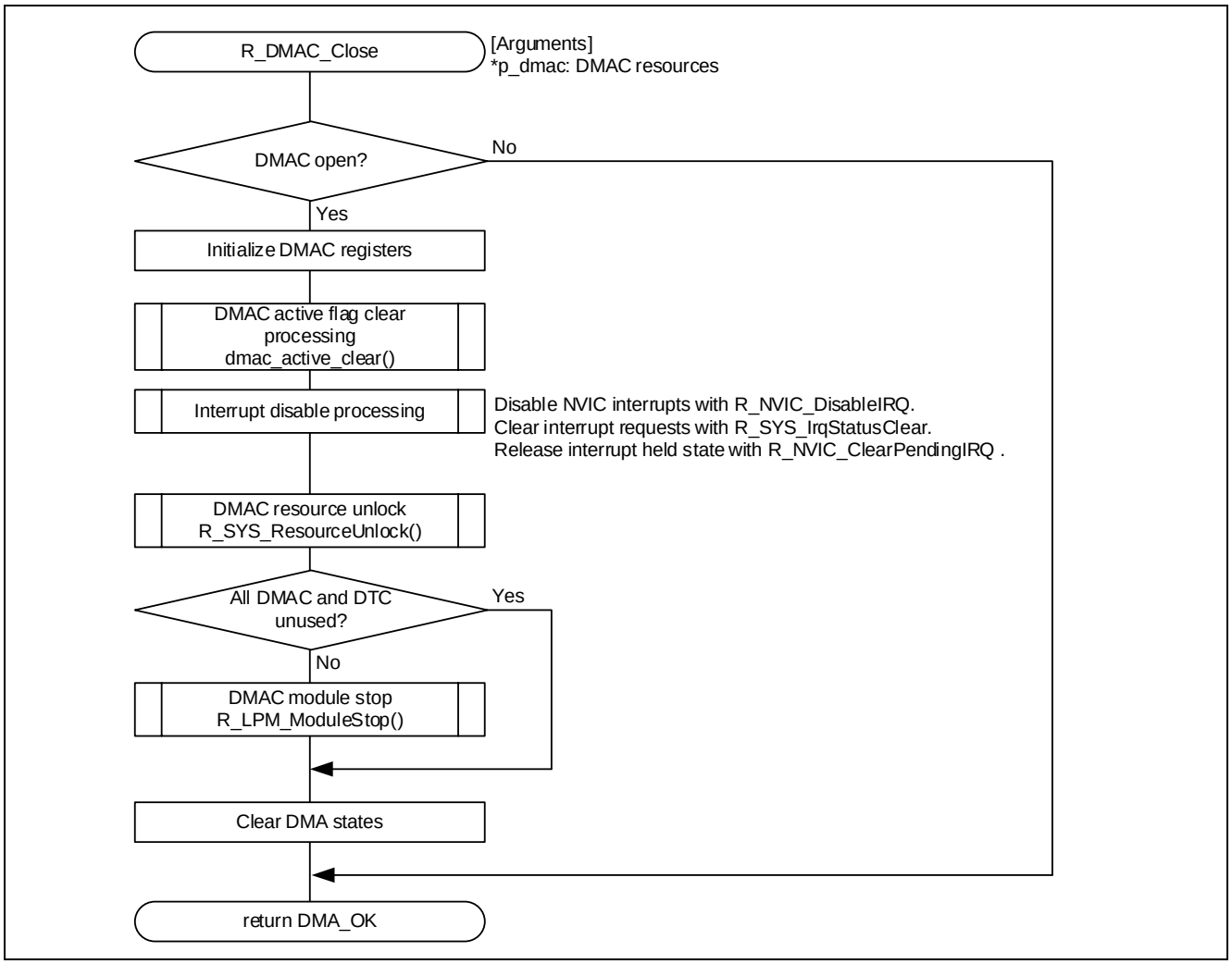


Figure 4-2 R_DMAC_Close Function Processing Flow

4.1.3 R_DMAC_Create Function

Table 4-3 R_DMAC_Create Function Specifications (1/2)

Format	static e_dma_err_t R_DMAC_Create(int16_t const dmac_source, st_dma_transfer_data_cfg_t * const p_data_cfg, st_dmac_resources_t * const p_dmac)
Description	Sets DMA transfers.
Argument	int16_t const dmac_source: DMAC start trigger Specifies the DMAC transfer trigger. When using an interrupt request from a peripheral module or a trigger from an external interrupt input pin as the transfer trigger, the number of the event signal to link should be specified. For information on event signal numbers, refer to section 16, Interrupt Controller Unit (ICU) in the "RE01 1500KB Group User Manual: Hardware (r01uh0796)" or " RE01 256KB Group User's Manual: Hardware (r01uh0894)". When using a software trigger to start a transfer, DMAC_TRANSFER_REQUEST_SOFTWARE should be specified.
	st_dma_transfer_data_cfg_t * const p_data_cfg: DMAC setting information Specifies DMA transfer information. For information on settings within structures see section 2.7.1.
	st_dmac_resources_t * const p_dmac: Resources of the DMAC Specifies the DMAC resource to set.
Return value	DMA_OK Setting success
	DMA_ERROR Setting failure When executed before opening the DMAC, setting failure results.
	DMA_ERROR_MODE Setting failure due to abnormal mode value When any of the following abnormal setting values is detected, setting failure due to an abnormal mode value results • A transfer mode setting other than normal, repeat, or block • A data size setting other than 8-bit, 16-bit, 32-bit • A repeat area setting other than transfer source repeat, transfer destination repeat, or no repeat area • A chain transfer setting (invalid in DMAC)
	DMA_ERROR_PARAMETER Setting failure due to abnormal parameter When any of the following abnormal setting values is detected, setting failure due to an abnormal setting value results • When an incorrect value is set as the DMAC start trigger (dmac_source) (possible setting range: 0x01 to 0xAA) • When a value is set as an unused bit in the transfer mode destination (p_data_cfg->mode) • When an incorrect value is set as the number of transfers (p_data_cfg->transfer_count) (For information on setting ranges see Table 4-5 Possible Setting Ranges for Number of Transfers by Transfer Mode.) • When an incorrect value is set for the block transfer size (p_data_cfg->block_size) at the time repeat transfer or block transfer is specified (possible setting range: 1 to 1024) • When, with the transfer size set to 16-bit, the lowermost bit of either the transfer source address (p_data_cfg->src_addr) or the transfer destination address (p_data_cfg->dest_addr) is not 0 • When, with the transfer size set to 32-bit, the lower 2 bits of the transfer source or the transfer destination are not 00 • When, with transfer source repeat set, the transfer mode is not normal transfer, and moreover a value other than 0 is set as the transfer source extended repeat area (p_data_cfg->src_extended_repeat)

Table 4-4 R_DMAC_Create Function Specifications (2/2)

Return value	• When, with transfer destination repeat set, the transfer mode is not normal transfer, and moreover a value other than 0 is set as the transfer destination extended repeat area (p_data_cfg->dest_extended_repeat) • When an incorrect value is set as the transfer source extended repeat area (possible setting range: 0 to 27) • When an incorrect value is set as the transfer destination extended repeat area (possible setting range: 0 to 27) • When, with an offset set for the transfer source or transfer destination, an incorrect value is set for the offset value (p_data_cfg->offset) (possible setting range: -16777216 to 16777215)

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	const uint8_t tx_data[2] = {0x51,0xA2};
	uint8_t rx_data[2] = {0x00,0x00};
	
	main()
	{
	st_dma_transfer_data_cfg_t config;
	
	config.mode = (DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR |
	DMA_DEST_INCR | DMA_REPEAT_BLOCK_SRC);
	config.src_addr = &tx_data[0];
	config.dest_addr = & rx_data[0];
	config.transfer_count = 4;
	config.block_size = 4;
	config.offset = 0;
	config.src_extended_repeat = 0;
	config.dest_extended_repeat = 0;
	
	dmac0Drv->Create (5, &config);
	}

Table 4-5 Possible Setting Ranges for Number of Transfers by Transfer Mode

Transfer Mode	Possible Setting Range
Normal transfer	0 to 65535
Repeat transfer	1 to 65536
Block transfer	

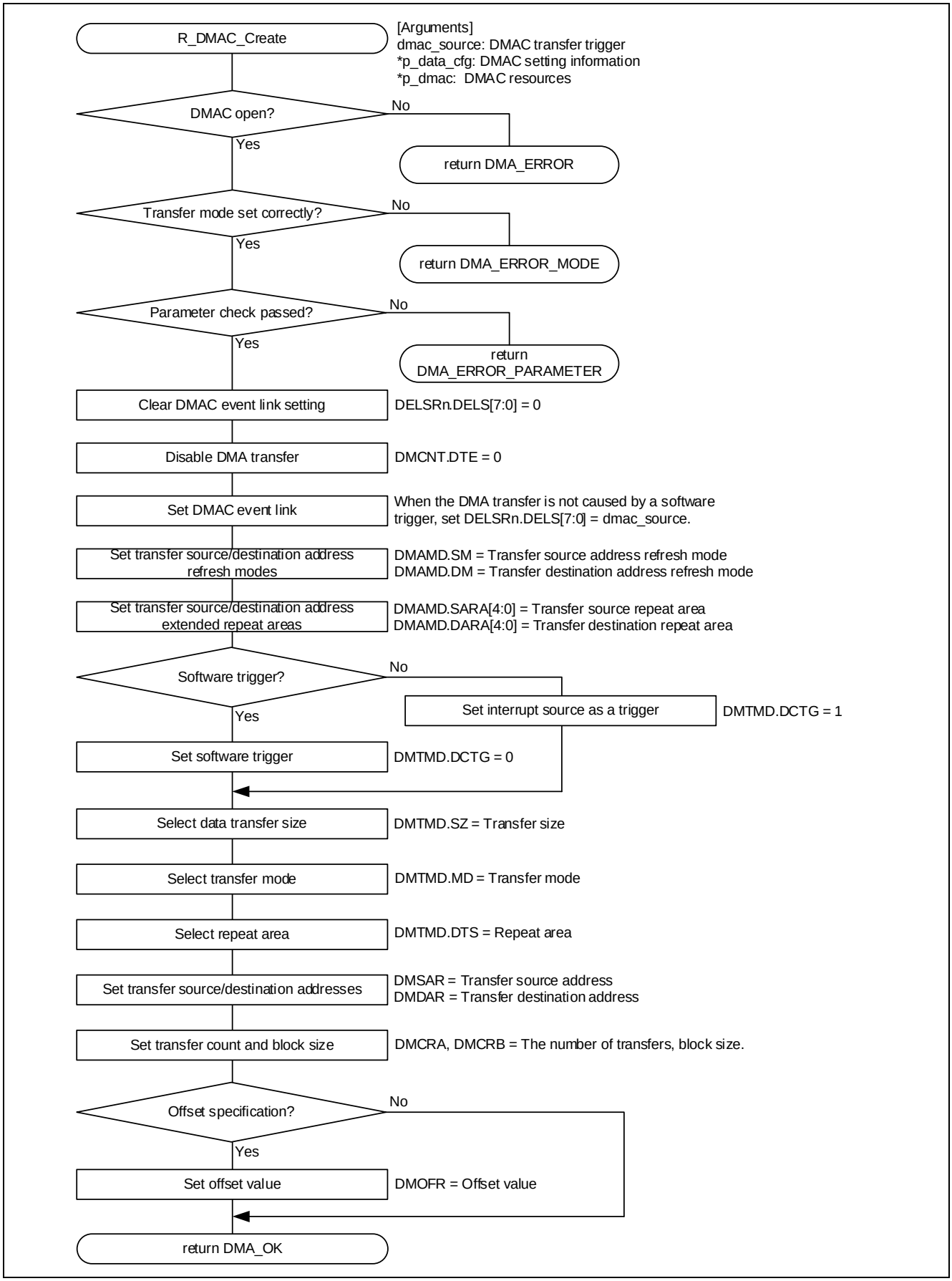


Figure 4-3 R_DMAC_Create Function Processing Flow

4.1.4 R_DMAC_Control Function

Table 4-6 R_DMAC_Control Function Specifications

Format	static e_dma_err_t R_DMAC_Control(e_dma_cmd_t cmd, void const * const p_arg, st_dmac_resources_t * const p_dmac)
Description	Executes DMAC control commands.
Argument	e_dma_cmd_t cmd: Control command The following control commands can be set: • DMA_CMD_START: DMA transfer start command • DMA_CMD_STOP: DMA transfer stop command • DMA_CMD_SOFTWARE_TRIGGER: Software trigger DMA transfer request command • DMA_CMD_AUTO_CLEAR_SOFT_REQ: DMA software start bit auto-clear setting command • DMA_CMD_ACT_SRC_DISABLE: DMAC start trigger clear command • DMA_CMD_REFRESH_DATA: DMA transfer refresh command • DMA_CMD_REFRESH_EXTRA: DMA transfer refresh (extended) command
	void const * const p_arg : Command-specific argument (For control commands and command-specific arguments, see Table 4-7.)
	st_dmac_resources_t * const p_dmac: Resources of DMAC Specifies a DMAC resource to be controlled.
Return value	DMA_OK Control command execution success
	DMA_ERROR Control command execution failure When any of the following states are detected, control command execution fails • When executed before DMAC open • When, during a software DMA transfer request, a software request command (DMA_CMD_SOFTWARE_TRIGGER) is again executed • When NULL(0) is specified as the argument in a command that requires an argument (DMA_CMD_AUTO_CLEAR_SOFT_REQ, DMA_CMD_REFRESH_DATA)
	DMA_ERROR_MODE Execution failure due to abnormal parameter When executing a DMA transfer refresh command (DMA_CMD_REFRESH_DATA, DMA_CMD_REFRESH_EXTRA), if a transfer mode other than normal, repeat, or block is specified, execution failure due to an abnormal model value results
	DMA_ERROR_PARAMETER Execution failure due to abnormal parameter When the following conditions are detected, execution failure due to an abnormal parameter results [DMA transfer refresh command (DMA_CMD_REFRESH_DATA, DMA_CMD_REFRESH_EXTRA)] • When an incorrect value is set for the number of transfers (p_data_cfg->transfer_count) For information on setting ranges see Table 4-5 Possible Setting Ranges for Number of Transfers by Transfer Mode. • Upon specifying repeat transfer or block transfer, when an incorrect value is specified for the block transfer size (p_data_cfg->block_size) (possible setting range: 1 to 1024) [DMA transfer refresh (extended) command (DMA_CMD_REFRESH_EXTRA)] • Upon setting an incorrect value for the offset value, DMA_ERROR_PARAMETER results (possible setting range: -16777216 to 16777215)

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	main()
	{
	dmac0Drv-> Control (DMA_CMD_START, NULL);
	}

Table 4-7 Behaviors Specified with Control Commands and Command-Specific Arguments

Control Command (cmd)	Command-Specific Argument (arg)	Description
DMA_CMD_START	NULL	Starts a DMA transfer. Transfer is caused by a transfer trigger specified with R_DMAC_Create.
DMA_CMD_STOP	NULL	Ends a DMA transfer.
DMA_CMD_SOFTWARE_TRIGGER	NULL	Issues a DMA transfer request by a software trigger. Valid only when DMAC_TRANSFER_REQUEST_SOFTWARE is set as the transfer trigger by the R_DMAC_Create function.
DMA_CMD_AUTO_CLEAR_SOFT_REQ	true	Clears transfer requests after start of a DMA transfer by a software trigger.
	false	Holds a transfer request even after the start of a DMA transfer by a software trigger.
DMA_CMD_ACT_SRC_DISABLE	NULL	Disables DMA transfers, clears transfer triggers.
DMA_CMD_REFRESH_DATA	Specify DMAC setting information in the st_dma_refresh_data_t type. For details of the structure, see section 2.7.2.	Updates only the number of transfers, number of transfer blocks, transfer source address, and transfer destination address. When refreshing is performed with auto-start turned on, DMA transfers are resumed after the refresh.
DMA_CMD_REFRESH_EXTRA		Updates the number of transfers, number of transfer blocks, transfer source address, and transfer destination address, and also the offset value. Also enables selection to hold DMAC start requests. When holding of DMA requests is turned on when refreshing is performed, DMA requests occurring during the DMA transfer refresh are held. When auto-start is turned on when refreshing is performed, DMA transfer is resumed after the refresh. When holding of DMAC start requests is enabled, DMAC start requests that have occurred after DMA transfer is stopped and during the DMA refresh are held.

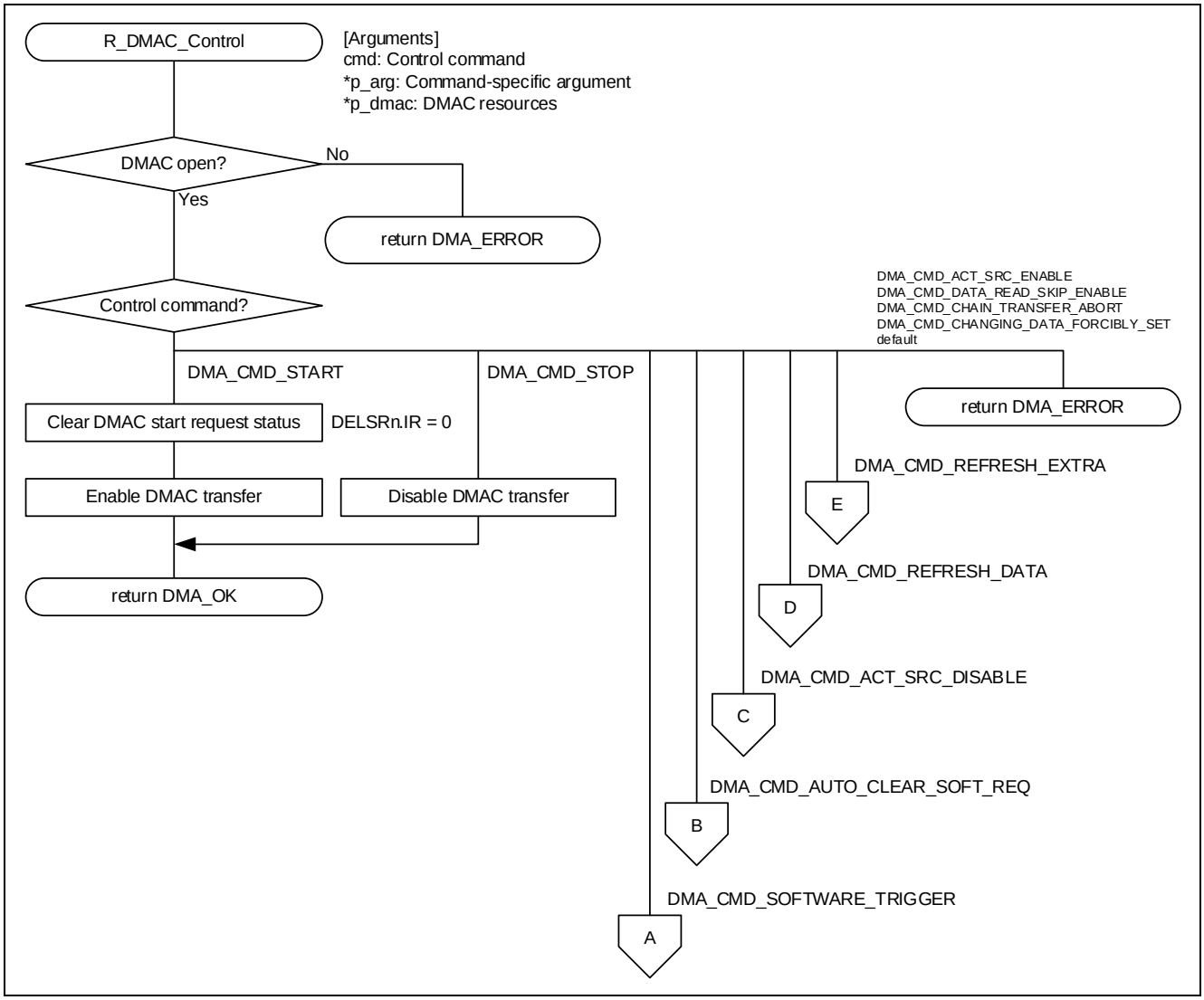


Figure 4-4 R_DMAC_Control Function Processing Flow (1/3)

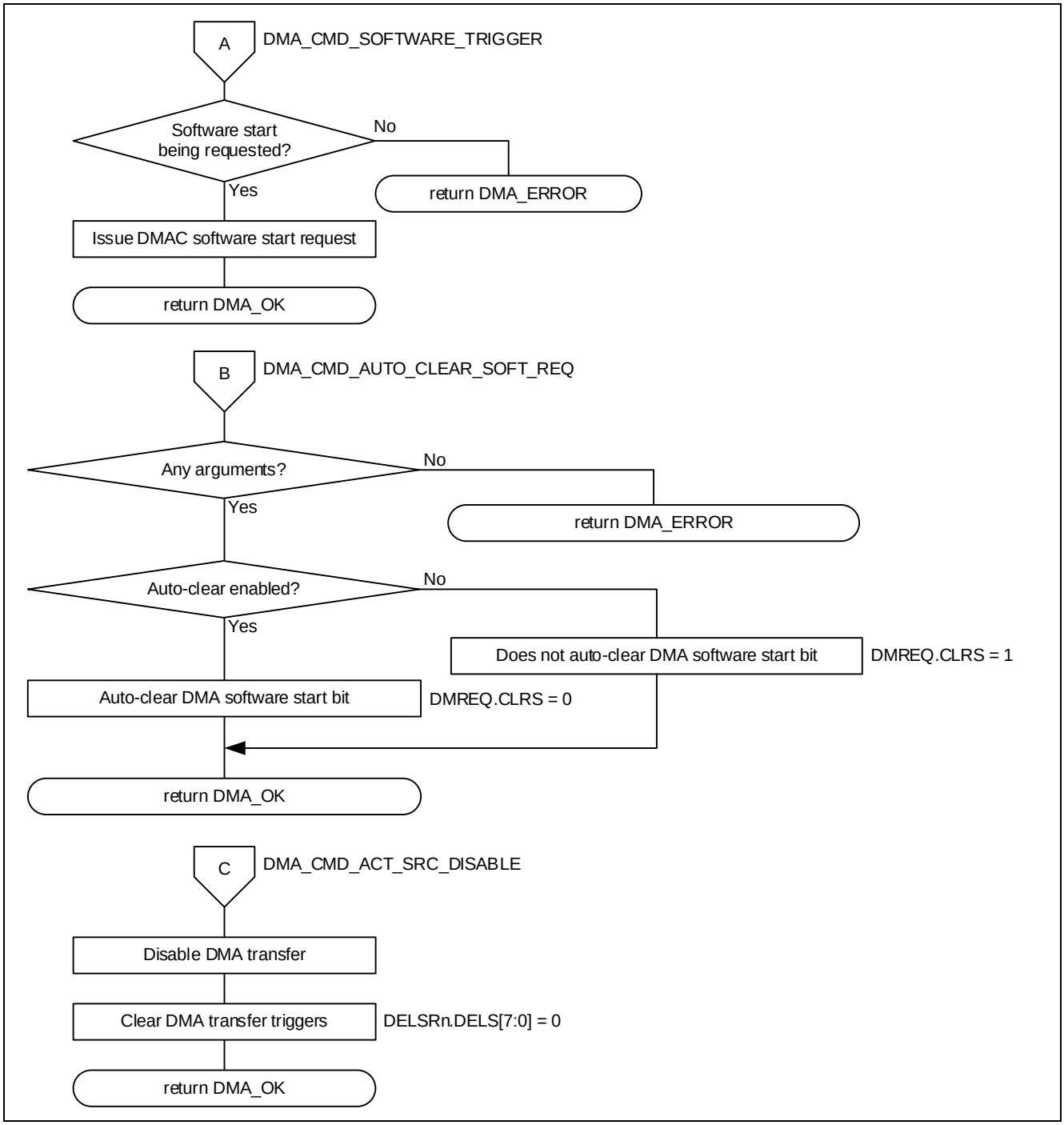


Figure 4-5 R_DMAC_Control Function Processing Flow (2/3)

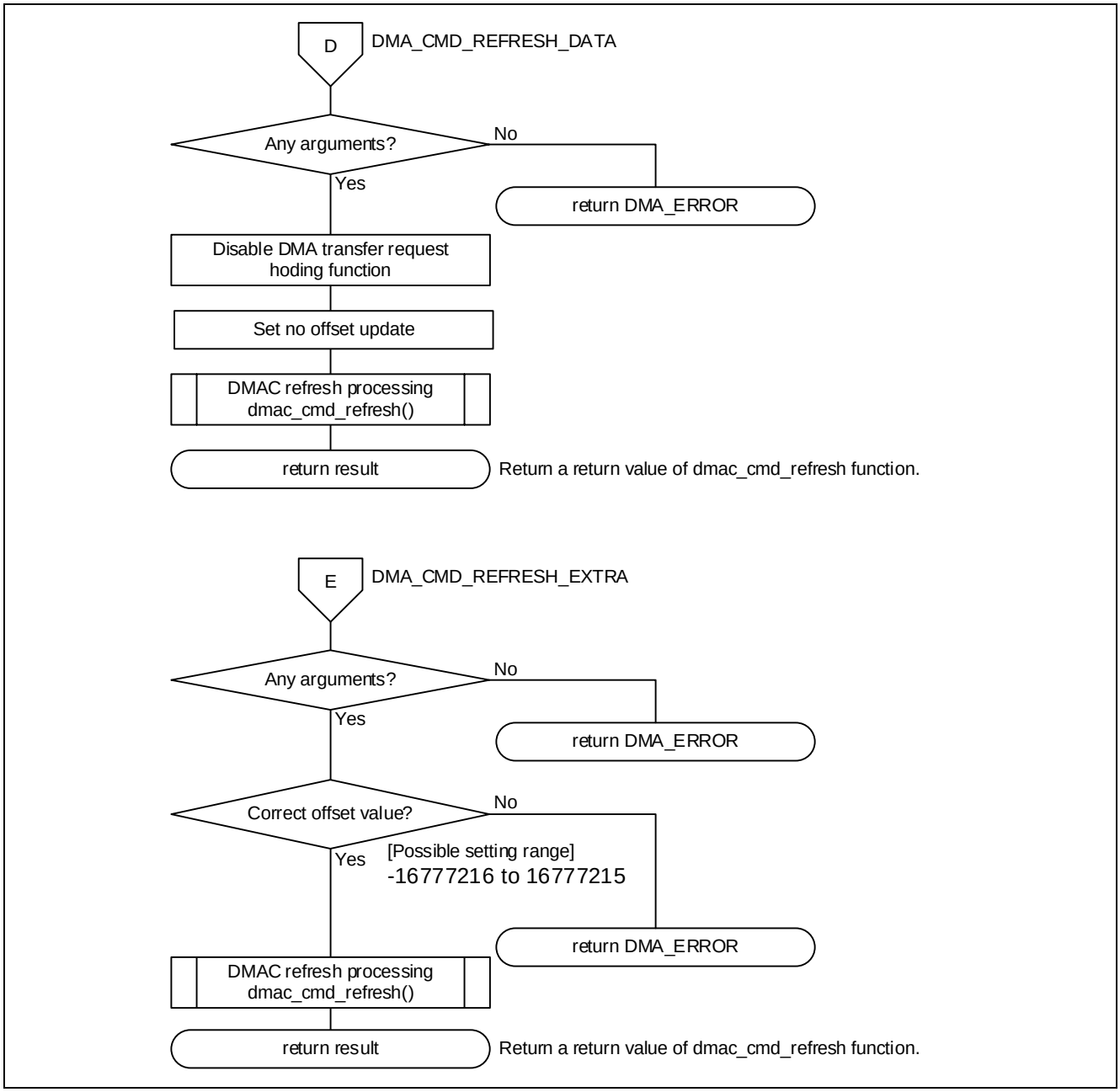


Figure 4-6 R_DMAC_Control Function Processing Flow (3/3)

4.1.5 R_DMAC_InterruptEnable Function

Table 4-8 R_DMAC_InterruptEnable Function Specifications

Format	static e_dma_err_t R_DMAC_InterruptEnable(uint16_t const int_fact, dma_cb_event_t const p_callback, st_dmac_resources_t * const p_dmac)
Description	Sets DMA transfer interrupts.
Argument	uint16_t const int_fact: Interrupt source The following interrupt sources can be set. When 0 is set, only NVIC registration of a DMA transfer complete interrupt is performed. When specifying multiple sources, the OR operator should be used in definitions. (Example: DMA_INT_COMPLETE | DMA_INT_SRC_OVERFLOW) • DMA_INT_COMPLETE: DMA transfer end • DMA_INT_SRC_OVERFLOW: Overflow in transfer source address extended repeat area • DMA_INT_DEST_OVERFLOW: Overflow in transfer destination address extended repeat area • DMA_INT_REPEAT_END: DMA transfer repeat size end • DMA_INT_ESCAPE_END: DMA transfer escape end
	dma_cb_event_t const p_callback: Callback function Specifies the callback function at the time of event occurrence. When NULL(0) is specified, a callback function is not executed.
	st_dmac_resources_t * const p_dmac: DMAC resources Specifies DMAC resources to set the interrupt.
Return value	DMA_OK Setting success
	DMA_ERROR Setting failure When either of the following states is detected, setting failure results • When execution is before DMAC open • When an incorrect value is specified for the interrupt source
	DMA_ERROR_SYSTEM_SETTING System setting failure When either of the following states is detected, system setting failure results. • When, in r_system_cfg.h, a DMAC interrupt is defined as unused (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED) • When, in r_dmac_cfg.h, the setting of DMACn_INT_PRIORITY exceeds the defined range

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	static void callback(void);
	
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> InterruptEnable (DMA_INT_SRC_OVERFLOW | DMA_INT_ESCAPE_END, callback);
	}

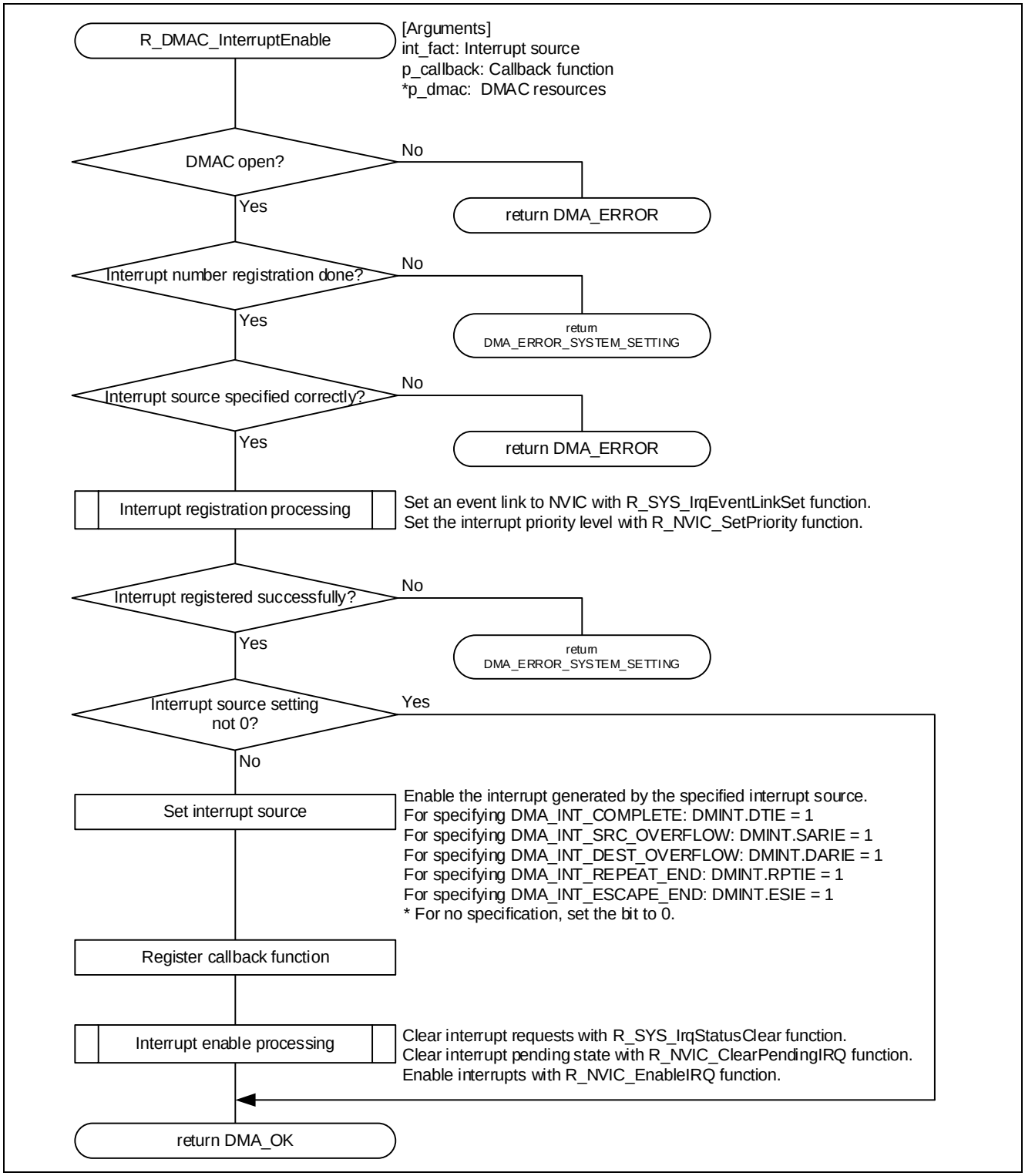


Figure 4-7 R_DMAC_InterruptEnable Function Processing Flow

4.1.6 R_DMAC_InterruptDisable Function

Table 4-9 R_DMAC_InterruptDisable Function Specifications

Format	static e_dma_err_t R_DMAC_InterruptDisable(st_dmac_resources_t * const p_dmac)
Description	Disables DMA transfer interrupts
Argument	st_dmac_resources_t * const p_dmac: DMA resources Specifies the DMA resources for interrupt prohibition.
Return value	DMA_OK Interrupt prohibition success
	DMA_ERROR Interrupt prohibition failure When executed before DMAC open, results in interrupt prohibition failure
	DMA_ERROR_SYSTEM_SETTING System setting failure When, in r_system_cfg.h, a DMAC interrupt is defined as unused (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED), system setting failure results

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> InterruptDisable ();
	}

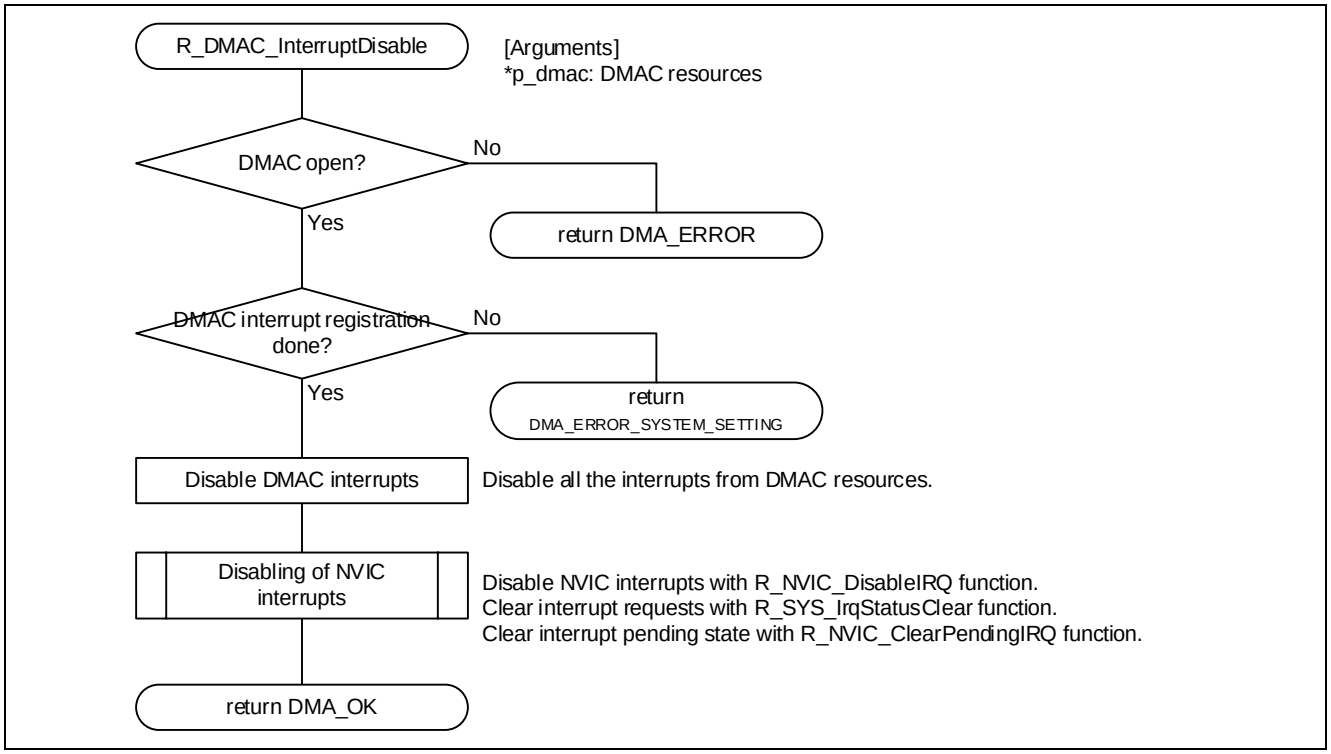


Figure 4-8 R_DMAC_InterruptDisable Function Processing Flow

4.1.7 R_DMAC_GetState Function

Table 4-10 R_DMAC_GetState Function Specifications

Format	static e_dma_err_t R_DMAC_GetState(st_dma_state_t * const state, st_dmac_resources_t * const p_dmac)
Description	Acquires the DMAC status
Argument	st_dma_state_t * const state: Status storage pointer Buffer that stores the acquired status
	st_dmac_resources_t * const p_dmac: DMAC resource Specifies the DMAC resource for which the status is to be acquired.
Return value	DMA_OK Acquisition success
	DMA_ERROR Acquisition failure When executed before DMAC open, acquisition failure results

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	st_dma_state_t dmac0_state;
	
	main()
	{
	dmac0Drv-> GetState(dmac0_state);
	}

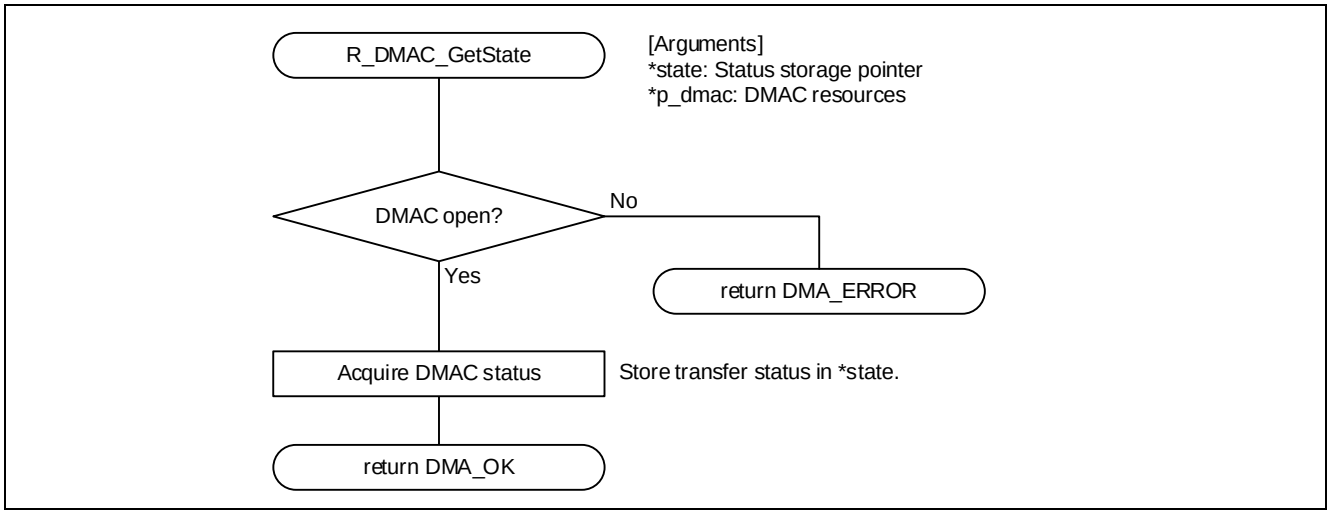


Figure 4-9 R_DMAC_GetState Function Processing Flow

4.1.8 R_DMAC_ClearState Function

Table 4-11 R_DMAC_ClearState Function Specifications

Format	static e_dma_err_t R_DMAC_ClearState(uint32_t clr_state, st_dmac_resources_t * const p_dmac)
Description	Clears the specified DMAC interrupt flag
Argument	uint32_t clr_state: Status to be cleared The following interrupt flags can be specified. To clear multiple flags, use an OR operator in the definition. (Example: DMA_CLR_STATE_ESIF | DMA_CLR_STATE_DTIF) • DMA_CLR_STATE_ESIF: Transfer escape end interrupt flag • DMA_CLR_STATE_DTIF: Transfer end interrupt flag • DMA_CLR_STATE_SOFT_REQ: Software trigger transfer request flag
Return value	DMA_OK Clear success
	DMA_ERROR Clear failure When executed before DMA open, clear failure results

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> ClearState(DMA_CLR_STATE_ESIF);
	}

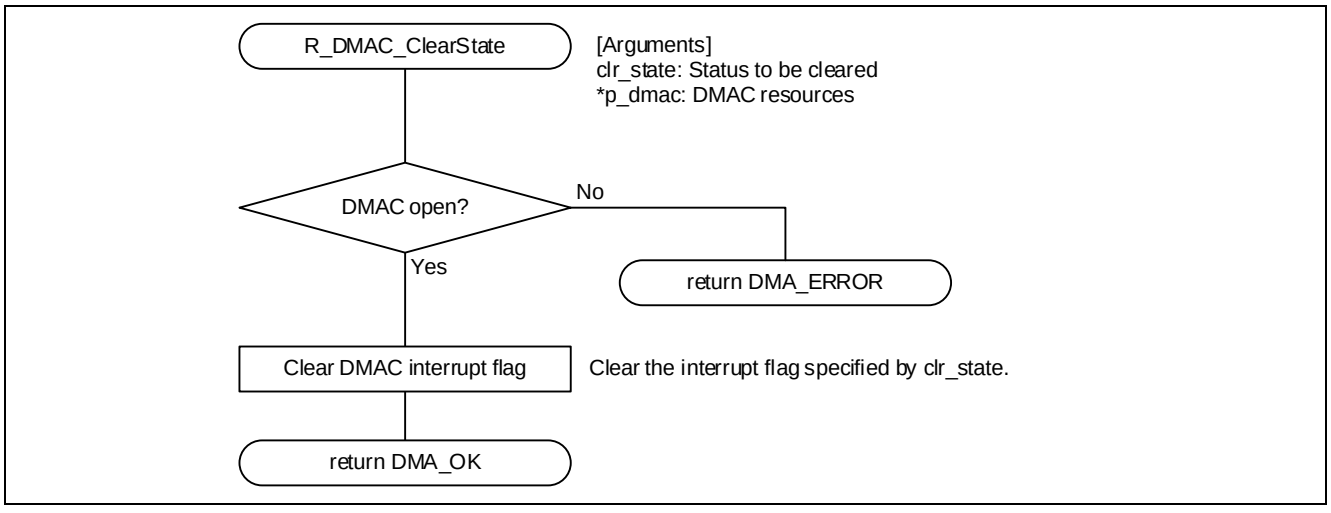


Figure 4-10 R_DMAC_ClearState Function Processing Flow

4.1.9 R_DMAC_GetTransferByte Function

Table 4-12 R_DMAC_GetTransferByte Function Specifications

Format	static e_dma_err_t R_DMAC_GetTransferByte(uint32_t * transfer_byte, st_dmac_resources_t * const p_dmac)
Description	Get the number of DMA transfer bytes
Argument	uint32_t *transfer_byte : DMA transfer byte storage pointer Specifies a pointer to store the DMA transfer byte
	st_dmac_resources_t * const p_dmac : DMAC resources Specify the DMAC resource to get the status.
Return value	DMA_OK Successful acquisition of DMA transfer byte count
	DMA_ERROR DMA transfer byte count acquisition failure When either of the following states is detected, failure results ・When executed before DMA open ・When executing in normal transfer free-run mode
	DMA_ERROR_MODE Execution failure due to abnormal mode When either of the following states is detected, mode failure results ・Invalid transfer mode ・Invalid transfer size

Remarks	When accessing from an instance, disable the first argument (set any value) and specify the DMA transfer byte storage pointer as the second argument.
	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	uint32_t byte;
	dmac0Drv-> GetTransferByte(0, &byte);
	}

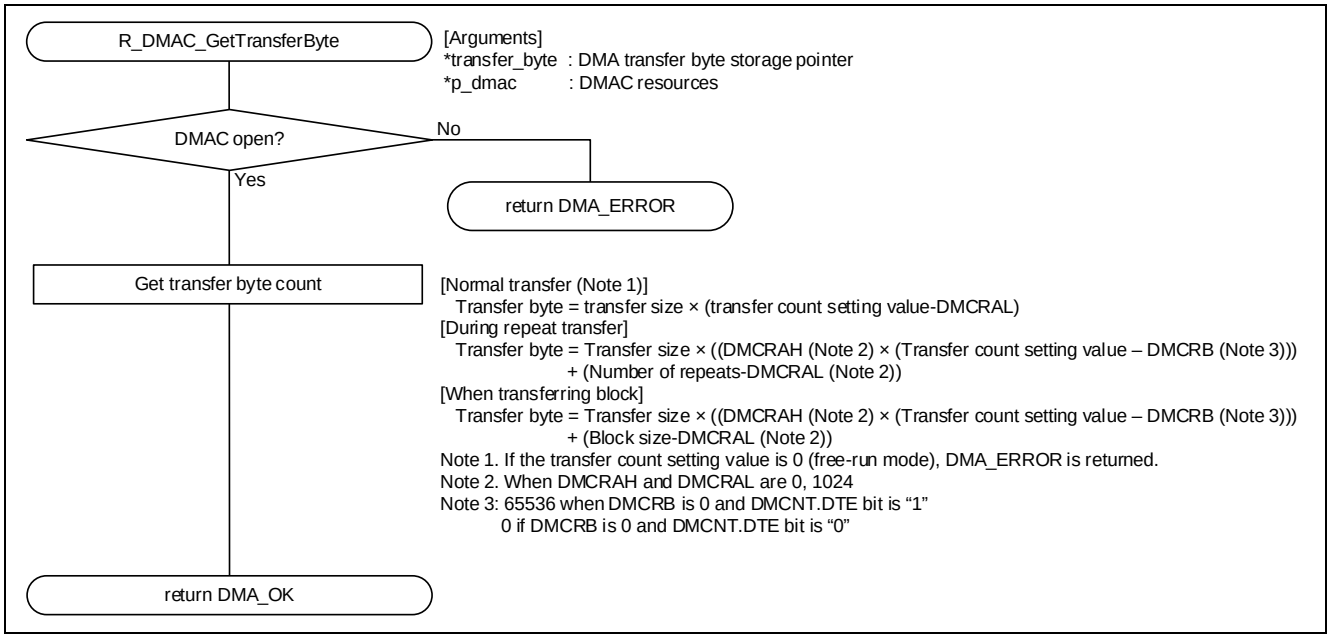


Figure 4-11 R_DMAC_GetTransferByte Function Processing Flow

4.1.10 R_DMAC_GetVersion Function

Table 4-13 R_DMAC_GetVersion Function Specifications

Format	static uint32_t R_DMAC_GetVersion(void)
Description	Acquires the DMAC driver version
Argument	None
Return value	DMAC driver version

Remarks	When this function is accessed, specifying the DMAC resources is not required.
	
	[Example of calling function from instance]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	uint32_t version;
	version = dmac0Drv-> GetVersion ();
	}

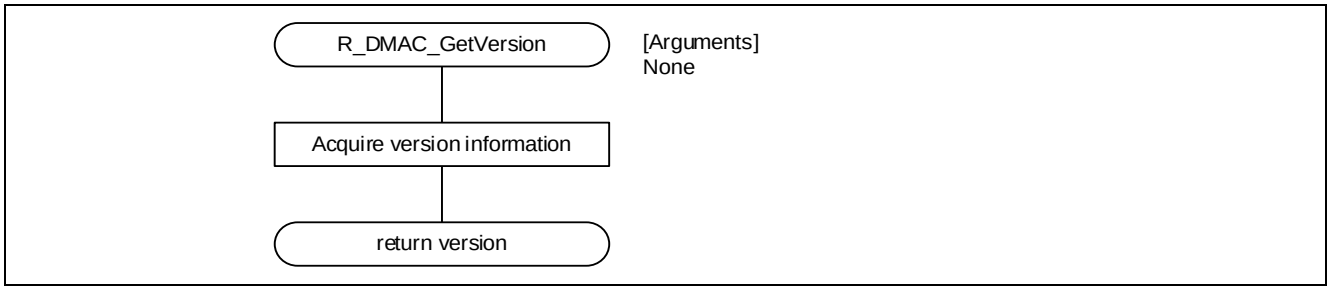


Figure 4-12 R_DMAC_GetVersion Function Processing Flow

4.1.11 dmac_active_set Function

Table 4-14 dmac_active_set Function Specifications

Format	static void dmac_active_set(uint16_t ch)
Description	Sets the specified DMAC channel to active
Argument	uint16_t ch: DMAC channel Specifies the channel number to be set to active.
Return value	None
Remarks	-

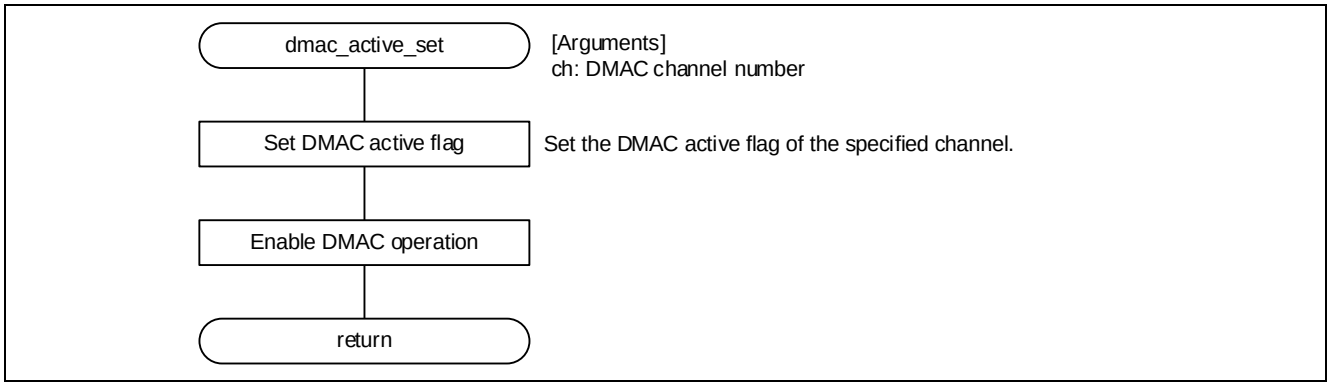


Figure 4-13 dmac_active_set Function Processing Flow

4.1.12 dmac_active_clear Function

Table 4-15 dmac_active_clear Function Specifications

Format	static void dmac_active_clear(uint16_t ch)
Description	Clears the active state of the specified DMAC channel; if all channels are in the inactive state, disables DMAC operation
Argument	uint16_t ch: DMAC channel Specifies the channel for which the active state is to be cleared.
Return value	None
Remarks	-

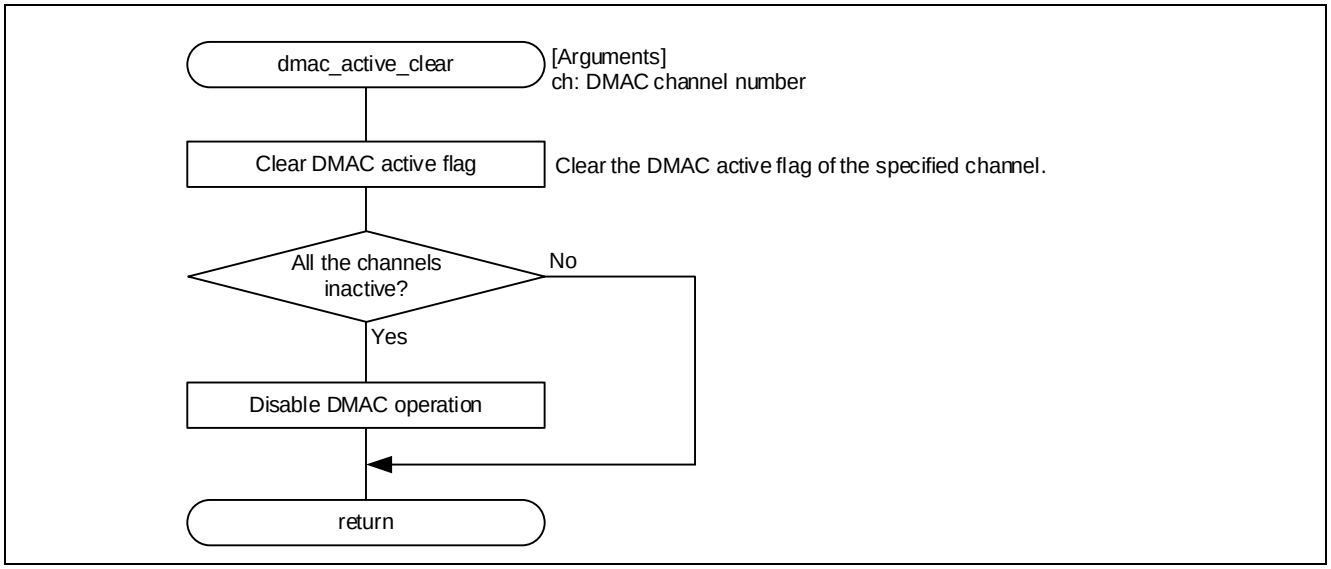


Figure 4-14 dmac_active_clear Function Processing Flow

4.1.13 dmac_cmd_refresh Function

Table 4-16 dmac_cmd_refresh Function Specifications

Format	static e_dma_err_t dmac_cmd_refresh(st_dma_refresh_data_t const * const p_data, st_dmac_resources_t * const p_dmac)
Description	Refreshes DMA transfer
Argument	st_dma_refresh_data_t const * const p_data: DMA transfer refresh data Specifies the transfer settings.
	st_dmac_resources_t * const p_dmac: DMAC resources Specifies the DMAC resource to be refreshed.
Return value	DMA_OK Refresh success
	DMA_ERROR_MODE Refresh failure due to abnormal mode value When a transfer mode setting other than normal, repeat, or block is detected, refresh failure due to an abnormal mode value results
	DMA_ERROR_PARAMETER Refresh failure due to an abnormal parameter When either of the following abnormal setting values is detected, refresh failure due to an abnormal parameter results • When an incorrect value is set for the number of transfers (p_data_cfg->transfer_count) (For information on setting ranges see Table 4-5 Possible Setting Ranges for Number of Transfers by Transfer Mode.) • When, with repeat transfer or block transfer specified, an incorrect value is set for the block transfer size (p_data_cfg->block_size) (possible setting range: 1 to 1024)
Remarks	-

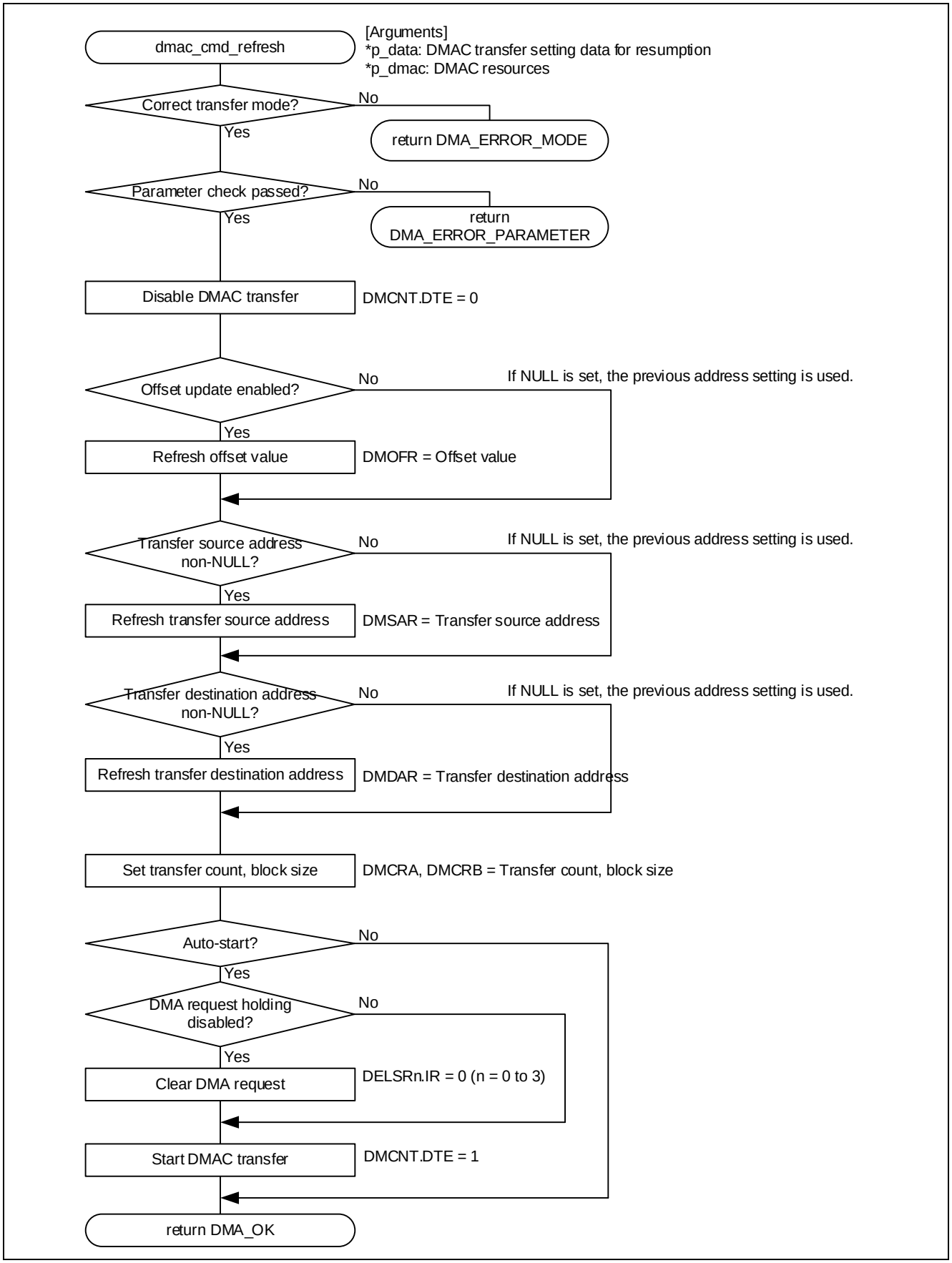


Figure 4-15 dmac_cmd_refresh Function Processing Flow

4.1.14 dmac_interrupt_handler Function

Table 4-17 dmac_interrupt_handler Function Specifications

Format	static void dmac_interrupt_handler(st_dmac_resources_t * const p_dmac)
Description	DMAC interrupt handling process
Argument	st_dmac_resources_t * const p_dmac : DMAC resources Specifies the DMAC resource to be transferred.
Return value	None
Remarks	-

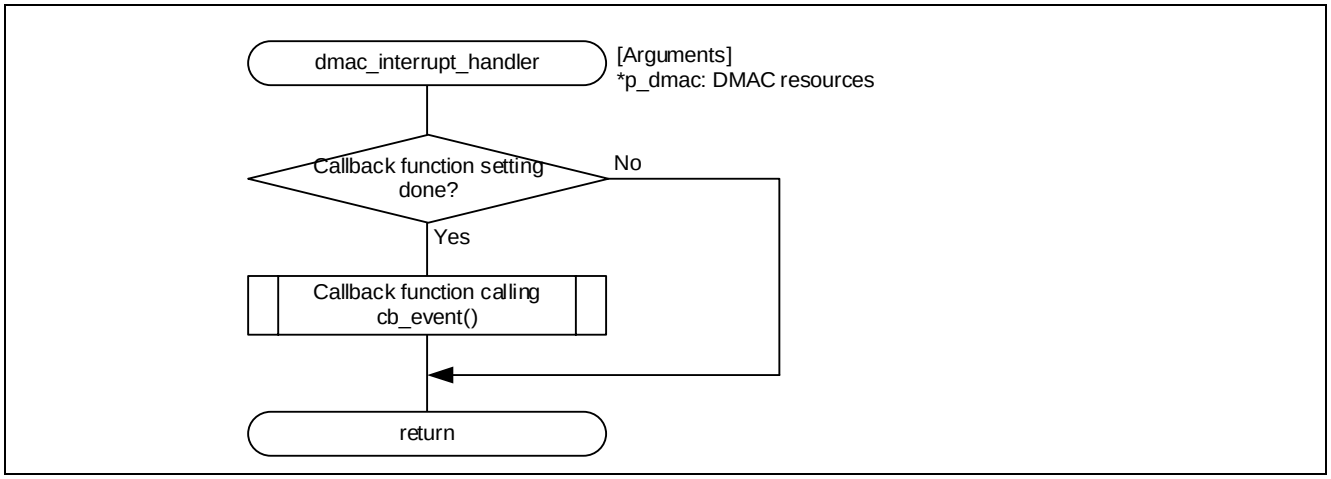


Figure 4-16 dmac_interrupt_handler Function Processing Flow

4.2 Macro and Type Definitions

This section shows the definitions of the macros used in the driver and the types of them.

4.2.1 Macro Definition List

DMAC control codes are DMAC control commands to be used by the Control function.

Table 4-18 List of Macro Definitions

Definition	Value	Description
DMA_FLAG_OPENED	(1U << 0)	Opened flag definition
DMA_FLAG_INITIALIZED	(1U << 1)	Initialized flag definition
DMA_FLAG_CREATED	(1U << 2)	Setting completed flag definition
DMAC_VERSION_MAJOR	(Note)	DMAC driver major number
DMAC_VERSION_MINOR	(Note)	DMAC driver minor number
DMA_ACTIVE_DMAC0	(0x0001)	DMAC0 active flag
DMA_ACTIVE_DMAC1	(0x0002)	DMAC1 active flag
DMA_ACTIVE_DMAC2	(0x0004)	DMAC2 active flag
DMA_ACTIVE_DMAC3	(0x0008)	DMAC3 active flag
DMA_ACTIVE_DTC	(0x8000)	DTC active flag
DMA_CLR_STATE_ESIF	(1U<<1)	Definition to clear the transfer escape end interrupt flat
DMA_CLR_STATE_DTIF	(1U<<2)	Definition to clear the transfer end interrupt flag
DMA_CLR_STATE_SOFT_REQ	(1U<<3)	Definition to clear the DMA software start flag
DMAC_PRV_MIN_COUNT_VAL	(1)	Lower limit for the number of DMA transfers
DMAC_PRV_MAX_16BITS_COUNT_VAL	(65536)	Upper limit for the number of DMAC 16-bit transfers
DMAC_PRV_MAX_8BITS_COUNT_VAL	(256)	Upper limit for the number of DMAC 8-bit transfers
DMAC_PRV_MAX_10BITS_COUNT_VAL	(1024)	Upper limit for the number of DMAC 10-bit transfers
DMAC_PRV_EVENT_NUM_MIN	(0x01)	DMAC event number lower limit
DMAC_PRV_EVENT_NUM_MAX	(0xAA)	DMAC event number upper limit
DMAC_PRV_MAX_EXTRA_REP_AREA	(27)	Address extended repeat area size upper limit
DMAC_PRV_DELSR_IR_POS	(1U << 16)	Definition to clear DMAC start requests
DMAC_PRV_MIN_ADDR_OFFSET	(-16777216)	DMAC offset register lower limit
DMAC_PRV_MAX_ADDR_OFFSET	(16777215)	DMAC offset register upper limit
DMAC_PRV_FREERUN	(0xFFFFFFFF)	Free run mode definition
DMAC_PRV_DMTMD_BYTE	(0)	Transfer size 1 byte judgment definition
DMAC_PRV_DMTMD_WORD	(1)	Transfer size 2 byte judgment definition
DMAC_PRV_DMTMD_LONG	(2)	Transfer size 4 byte judgment definition

Note. The value is set according to the version of this driver.

Example: Driver version 1.01

DMAC_VERSION_MAJOR (1)

DMAC_VERSION_MINOR (1)

4.3 Structure Definitions

4.3.1 st_dmac_resources_t Structure

This structure configures the resources of DMAC.

Table 4-19 st_dmac_resources_t Structure

Element Name	Type	Description
*reg	volatile DMAC0_Type	Indicates the DMAC register
*delsr	volatile uint32_t	Indicates the event link setting register
lock_id	e_system_mcu_lock_t	DMAC lock ID
mstp_id	e_lpm_mstp_t	DMAC module stop ID
active_flag	uint16_t	DMAC active flag
*info	st_dmac_mode_info_t	DMAC state information
dmac_int_irq	IRQn_Type	DMAC interrupt allocation number
dmac_int_iesr_val	uint32_t	IESR register setting value for DMAC interrupts
dmac_int_priority	uint32_t	DMAC interrupt priority level
dmac_int_callback	system_int_cb_t	DMAC interrupt callback function

4.3.2 st_dmac_mode_info_t Structure

This structure is used to manage the DMAC state.

Table 4-20 st_dmac_mode_info_t Structure

Element Name	Type	Description
cb_event	dma_cb_event_t	Callback function upon event occurrence When NULL, there is no callback function execution
flags	uint8_t	Driver settings flag b0: DMAC open state (0: closed, 1: open) b1: Driver initialization state (0: not initialized, 1: initialized) b2: DMAC settings completed state (0: not completed, 1: completed)

4.4 Calling External Functions

This section shows the external functions to be called from the DMAC driver APIs.

Table 4-21 External Functions Called from DMAC Driver APIs and Calling Conditions

API	Functions Called	Conditions (Note)
Open	R_SYS_ResourceLock	None
	R_LPM_ModuleStart	None
	R_SYS_ResourceUnlock	Upon module stop clear failure
Close	R_NVIC_DisableIRQ	None
	R_SYS_IrqStatusClear	None
	R_NVIC_ClearPendingIRQ	None
	R_SYS_ResourceUnlock	None
	R_LPM_ModuleStop	When all DMAC/DMAC drivers are unused
Create	-	-
Control	-	-
InterruptEnable	R_SYS_IrqEventLinkSet	None
	R_NVIC_SetPriority	None
	R_NVIC_GetPriority	None
	R_SYS_IrqStatusClear	When there are no setting errors and an interrupt source is specified
	R_NVIC_ClearPendingIRQ	
	R_NVIC_EnableIRQ	
InterruptDisable	R_NVIC_DisableIRQ	None
	R_SYS_IrqStatusClear	None
	R_NVIC_ClearPendingIRQ	None
GetState	-	-
ClearState	-	-
GetTransferByte	-	-
GetVersion	-	-

Note Even when there are no conditions, upon occurrence of an error end due to a parameter check, the called function may not be executed.

5. Usage Notes

5.1 Arguments

In a structure that specifies arguments for functions, any elements that are not used should be set to 0.

5.2 DMA Interrupt Settings

Among DMA interrupts, when using a callback function with the transfer source address extended repeat area overflow interrupt (DMA_INT_SRC_OVERFLOW), the transfer destination address extended repeat area overflow interrupt (DMA_INT_DEST_OVERFLOW), and the DMA transfer repeat size end interrupt (DMA_INT_REPEAT_END), at the time of execution of the InterruptEnable function, an OR operator must be used to simultaneously enable the transfer escape end interrupt (DMA_INT_ESCAPE_END).
For details, see section 2.4, DMA Interrupts.

5.3 Operation at the Time of Occurrence of a DMA Interrupt Request

When a transfer source address extended repeat area overflow interrupt (DMA_INT_SRC_OVERFLOW), a transfer destination address extended repeat area overflow interrupt (DMA_INT_DEST_OVERFLOW), or a DMA transfer repeat size end interrupt (DMA_INT_REPEAT_END) occurs, DMA transfer is stopped (DMCNT.DTE = 0). After occurrence of an interrupt request, if DMA transfers are to be continued, the Control function should be used to execute the DMA_CMD_START command.

5.4 DMAC Interrupt Registration in NVIC

When using DMAC interrupts (DMACn_INT (n = 0 to 3)), they should be registered with NVIC in r_system_cfg.h. If no DMAC interrupts are registered to NVIC, DMA_ERROR_SYSTEM_SETTING is returned when InterruptEnable function is executed. For details, refer to "Setting Interrupts (NVIC)" in the RE01 1500KB, 256KB Group Getting Started Guide to Development Using CMSIS Package (r01an4660).

Figure 5-1 shows an example of registration of a DMAC transfer complete interrupt.

```
...
#define SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0
                (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT
                (SYSTEM_IRQ_EVENT_NUMBER0)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DTC_COMPLETE
                (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
...
```

Figure 5-1 Example of Interrupt Registration to NVIC in r_system_cfg.h

5.5 Constraints on Transfer Source Address and Transfer Destination Address Settings

The settings for transfer source addresses and transfer destination addresses used with the Create function, DMA transfer refresh commands (DMA_CMD_REFRESH_DATA, DMA_CMD_REFRESH_EXTRA), and the DMA transfer information forced change command (DMA_CMD_CHANGING_DATA_FORCIBLY_SET) should be set to a multiple of 2 when the transfer bit size is 16 bits and to a multiple of 4 when the transfer bit size is 32 bits.

6. Reference Documents

User's Manual: Hardware

RE01 1500KB Group User's Manual: Hardware R01UH0796
RE01 256KB Group User's Manual: Hardware R01UH0894
(The latest version can be downloaded from the Renesas Electronics website.)

RE01 Group CMSIS Package Startup Guide

RE01 1500KB, 256KB Group Startup Guide to Development Using CMSIS Package R01AN4660

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

Technical Update/Technical News

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

User's Manual: Development Tools

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

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.10.19	—	First edition issued
1.01	Dec.16.2019	— program (256KB)	Compatible with 256KB group Modified to match 256KB IO definition

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

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

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

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

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

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

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

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

Notice

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

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

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

Contact information

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

© 2019 Renesas Electronics Corporation. All rights reserved.