RE01 1500KB, 256KB Group

CMSIS Driver R_DTC Specifications

Summary

This document describes the detailed specifications of the DTC driver provided in the RE01 1500KB and 256KB Group CMSIS software package .

Target Device

RE01 1500KB Group

RE01 256KB Group

Contents

1. Overview

This is a DTC driver for performing DTC transfer in 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_DTC drive corresponds to DeviceHAL in the CMSIS Driver Package and consists of four files: "r_dtc_api.c", "r_dtc_api.h", "r_dma_common_api.h", and "r_dtc_cfg.h" in the vendor-specific file storage directory. The functions of the files are shown in Table 2-1, and the file structure is shown in Figure 2-1.

Table 2-1 Functions of the Files of R_DTC Driver

File Name	Description
r_dtc_api.c	Driver source file. This file provides the detail of driver functions. To use the DTC driver, it is necessary to build this file.
r_dtc_api.h	Driver header file. The macros, types, and prototype declarations to be used in the driver are defined. To use the DTC driver, it is necessary to include this file.
r_dma_common_api.h	Common driver header file for the DMAC and DTC. The macros and types to be commonly used by the DMAC and DTC are defined.
r_dtc_cfg.h	Configuration definition file. This provides configuration definitions that can be modified by the user.

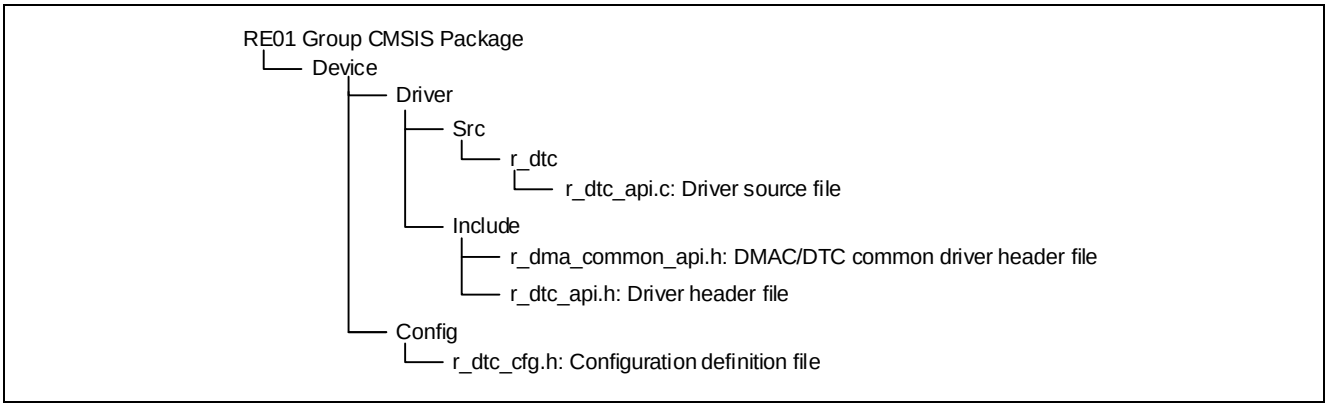


Figure 2-1 DTC Driver File Structure

2.2 Driver APIs

The DTC driver has one instance. To use the driver, access the APIs using function pointers for the instance. Table 2-2 shows the DTC driver instance. Figure 2-2 shows an example of instance declaration. Table 2-3 lists the APIs contained in the instance. Figure 2-3 to Figure 2-6 show examples of access to the DTC driver.

Table 2-2 DTC Driver Instance

Instance	Description
DRIVER_DMA Driver_DTC	Instance for using DTC

```
#include "r_dtc_api.h"

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDev = &Driver_DTC;
```

Figure 2-2 Example of DTC Driver Instance Declaration

Table 2-3 DTC Driver APIs

API	Description	Reference
Open	Opens the DTC driver (cancels the module-stop state and initializes RAM).	4.1.1
Close	Releases the DTC driver (initializes registers and disables interrupts). It will also cause a transition to the module-stop state if the DMAC and DTC drivers are both not used.	4.1.2
Create	Makes the settings for DTC transfer.	4.1.3
Control	Executes a control command of the DTC. For the control commands, see Table 2-4.	4.1.4
InterruptEnable	Sets the transfer complete interrupt (DTC_COMPLETE) of the DTC driver. When the DTC transfer complete interrupt is specified as the interrupt source, this API also sets the interrupt source and enables the interrupt.	4.1.5
InterruptDisable	Disables the transfer complete interrupt (DTC_COMPLETE) of the DTC driver.	4.1.6
GetState	Obtains the status of the DTC.	4.1.8
GetTransferByte	Get the number of DTC transfer bytes	4.1.9
ClearState	Clears the interrupt flag of the DTC driver.	4.1.10
GetVersion	Obtains the version of the DTC driver.	4.1.11

Table 2-4 Command List

Command	Description
DMA_CMD_START	Starts DTC transfer.
DMA_CMD_STOP	Stops DTC transfer.
DMA_CMD_ACT_SRC_ENABLE	Enables DTC activation using a DTC start trigger.
DMA_CMD_ACT_SRC_DISABLE	Disables DTC activation using a DTC start trigger.
DMA_CMD_DATA_READ_SKIP_ENABLE	Enables or disables read skip.
DMA_CMD_CHAIN_TRANSFER_ABORT	Aborts DTC chain transfer.
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	Changes DTC transfer information.
DMA_CMD_REFRESH_DATA	Refreshes DTC transfer settings.

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info;    /* DTC transfer information storage area  */
uint16_t src_data[5] = {0x0000, 0x1111, 0x2222, 0x4444, 0x8888 };
uint16_t dest_data= 0xFFFF;

main()
{
  st_dma_transfer_data_cfg_t config;   /* DMA transfer setting information storage area  */

    /* Normal transfer mode, 16 bits, source address incremented, destination address fixed, interrupt
*/
    /* generation when all transfers are completed, and chain transfer disabled         */
     config.mode    = DMA_MODE_NORMAL | DMA_SIZE_WORD | DMA_SRC_INCR | DMA_DEST_FIXED |
                    DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr       = (uint32_t)(&src_data[0]);    /* Set the transfer source address (RAM). */
    config.dest_addr      = (uint32_t)(&dest_data);
    config.transfer_count  = 5;                    /* Set the transfer count to 5. */
    config.p_transfer_data = &transfer_info;        /* DTC transfer information storage location */

    (void)dtcDrv->Open();                           /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                 /* Specify TMR CMIA0 as the start trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of the DTC transfer trigger
       (CMIA0 interrupt) and  */
    /* enable the interrupt.                                                          */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_dtc_complete);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

    TMR0->TCCR = 0x0E;  /* Activate the DTC transfer trigger (TMR). */
    while(1);
}

/******************************************************************************************
* callback function
******************************************************************************************/
void cb_dtc_complete(void)
{
    /* Execute the callback function when all DTC transfers are completed. */
}
```

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

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info;       /* DTC transfer information storage area  */
uint16_t src_data[5] = {0x0000, 0x1111, 0x2222, 0x4444, 0x8888 };
uint16_t dest_data= 0xFFFF;

main()
{
    st_dma_transfer_data_cfg_t config;   /* DMA transfer setting information storage area  */

    /* Repeat transfer mode, 16 bits, source address incremented,
      destination address fixed, transfer source */
    /* as the repeat area, interrupt generation when
       all transfers are completed, and chain transfer disabled  */
    config.mode = DMA_MODE_REPEAT | DMA_SIZE_WORD | DMA_SRC_INCR | DMA_DEST_FIXED |
               DMA_REPEAT_BLOCK_SRC | DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr       = (uint32_t)(&src_data[0]);
    config.dest_addr      = (uint32_t)(&dest_data);
    config.transfer_count  = 5;                   /* Set the transfer count to 5. */

    config.p_transfer_data = &transfer_info;        /* DTC transfer information storage location */

    (void)dtcDrv->Open();                           /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                 /* Specify TMR CMIA0 as the start trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of the DTC transfer trigger (CMIA0 interrupt) and
*/
    /* enable the interrupt.                                                              */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_dtc_complete);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

    TMR0->TCCR = 0x0E;  /* Activate the DTC transfer trigger (TMR). */
    while(1);
}

/****************************************
* callback function
****************************************/
void cb_dtc_complete(void)
{
    /* Execute the callback function when all DTC transfers
       (transfer count 5 x repeat count 10) are completed. */
}
```

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

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info;       /* DTC transfer information storage area  */
uint8_t src_data[3][5] = {{ 0x00, 0x01, 0x02, 0x04, 0x08 },
                    {0x10, 0x11, 0x12, 0x14, 0x18 },
                    {0x20, 0x21, 0x22, 0x24, 0x28 }};
uint8_t dest_data[5]= {0xFF, 0xFF, 0xFF, 0xFF, 0xFF};

main()
{
    st_dma_transfer_data_cfg_t config;   /* DMA transfer setting information storage area  */

    /* Block transfer mode, 8 bits, source address incremented,
       destination address incremented, transfer source*/
    /* as the block area, interrupt generation when all transfers are completed,
       and chain transfer disabled     */
    config.mode = DMA_MODE_BLOCK | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR |
             DMA_REPEAT_BLOCK_DEST | DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr       = (uint32_t)(&src_data[0][0]);   /* Set the transfer source address (RAM). */
    config.dest_addr      = (uint32_t)(&dest_data[0]);    /* Set the transfer
                                                             destination address (RAM). */
    config.transfer_count   = 3;                        /* Set the transfer count to 3. */
    config.block_size     = 5;                         /* Set the block count to 5. */
    config.p_transfer_data = &transfer_info;      /* DTC transfer information storage location */

    (void)dtcDrv->Open();                            /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                  /* Specify TMR CMIA0 as the start trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of the DTC transfer trigger
      (CMIA0 interrupt) and  */
    /* enable the interrupt.                                                              */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_dtc_complete);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

    TMR0->TCCR = 0x0E;  /* Activate the DTC transfer trigger (TMR). */
    while(1);
}

/***********************************************************************
* callback function
***********************************************************************/
void cb_dtc_complete(void)
{
    /* Execute the callback function when all DTC transfers
       (block count 5 x transfer count 3) are completed. */
}
```

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

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info[2];    /* DTC transfer information storage area  */
                                                   /* (for two chain transfers)              */
uint8_t src_data_ch1[5] = { 0x00, 0x01, 0x02, 0x04, 0x08 };
uint8_t src_data_ch2[5] = { 0x10, 0x11, 0x12, 0x14, 0x18 };
uint8_t dest_data_ch1 = 0xFF;
uint8_t dest_data_ch2 = 0xFF;

main()
{
    st_dma_transfer_data_cfg_t config[2];  /* DMA transfer setting information storage area  */
                                           /* (for two chain transfers)                      */

    /* [First chain transfer] Normal transfer mode, 8 bits, source address incremented,
        destination address fixed, */
    /* interrupt generation when all transfers are completed,
       and continuous chain transfer                    */
    config[0].mode = DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                     DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_CONTINUOUSLY;
    config[0].src_addr       = (uint32_t)(&src_data_ch1[0]);
    config[0].dest_addr      = (uint32_t)(&dest_data_ch1);
    config[0].transfer_count   = 5;
    config[0].p_transfer_data  = &transfer_info[0];

    /* [Second chain transfer] Normal transfer mode, 8 bits, source address incremented,
        destination address */
    /* fixed, interrupt generation when all transfers are completed,
       and chain transfer disabled              */
    config[1].mode = DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                     DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config[1].src_addr       = (uint32_t)(&src_data_ch2[0]);
    config[1].dest_addr      = (uint32_t)(&dest_data_ch2);
    config[1].transfer_count   = 5;

    (void)dtcDrv->Open();                          /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config[0]);
                                                 /* Specify TMR CMIA0 as the start trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of the DTC transfer trigger
       (CMIA0 interrupt) and */
    /* enable the interrupt.                                                          */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_dtc_complete);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

    TMR0->TCCR = 0x0E;  /* Activate the DTC transfer trigger (TMR). */
    while(1);
}

/***********************************************************************
* callback function
***********************************************************************/
void cb_dtc_complete(void)
{
/* Execute the callback function when all DTC transfers in the second chain transfer are completed. */
}
```

Figure 2-6 Example of Access to DTC Driver (Chain Transfer)

2.3 DTC Transfer Trigger Setting and Interrupt

The interrupt to be used as the DTC transfer trigger must first be registered to the NVIC in r_system_cfg.h. Then, specify the interrupt definition name (SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0 for IRQ0) of the peripheral function to be used in the first argument of the Create function. For setting interrupts in the NVIC in r_system_cfg.h, refer to "Interrupt (NVIC) Settings" in the RE01 1500KB, 256KB Group Getting Started Guide to Development Using CMSIS Package.

A DTC transfer will be executed when a DTC transfer trigger is input after DTC transfer has been enabled. When a DTC transfer for the specified number of times has finished, an interrupt is requested to the CPU (Note 1, Note 2). Two types of interrupt are available to the CPU: the interrupt of the DTC transfer trigger (for each channel) and the DTC transfer complete interrupt (DTC_COMPLETE: common for all channels). To use the DTC transfer complete interrupt, enable it using the InterruptEnable function. It also has to be registered to the NVIC in r_system_cfg.h.

Figure 2-7 shows an example of registration to the NVIC in which the CMIA0 interrupt of TMR is the transfer trigger and the DTC transfer complete interrupt is used. Figure 2-8 shows an example of the procedure for setting them.

Note 1. When DMA_INT_PER_SINGLE_TRANSFER is specified in the Create function, an interrupt request to the CPU is generated for each DTC data transfer. When DMA_INT_AFTER_ALL_COMPLETE is specified, an interrupt request is generated when the specified data transfer has finished.

Note 2. If DMA_CHAIN_NORMAL (chain transfer is performed when the transfer counter value changes from 1 to 0 or 1 to CRAH) is specified, interrupts to the CPU will not be generated.

Note 3. In the DTC, DMA_INT_COMPLETE is the only interrupt source that can be specified in the InterruptEnable function. If a value other than DMA_INT_COMPLETE is specified in the first argument of the InterruptEnable function, an error will be returned.

```
• • •
#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT
     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
/* Register DTC transfer complete interrupt in NVIC. */
#define SYSTEM_CFG_EVENT_NUMBER_DTC_COMPLETE       (SYSTEM_IRQ_EVENT_NUMBER0)
#define SYSTEM_CFG_EVENT_NUMBER_ICU_SNZCANCEL
     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)
• • •
#define SYSTEM_CFG_EVENT_NUMBER_IOPORT_GROUP2
     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  ///< Numbers 2/10/18/26 only
/* Register DTC transfer trigger (TMR_CMIA0) in NVIC. */
#define SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0          (SYSTEM_IRQ_EVENT_NUMBER2)
     ///< Numbers 2/10/18/26 only
#define SYSTEM_CFG_EVENT_NUMBER_GPT1_CMPC
     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)  ///< Numbers 2/10/18/26 only
• • •
```

Figure 2-7 Example of Registering Interrupts to NVIC (Registering CMIA0 and DTC Transfer Complete Interrupt)

```
    /*************************************************************************/
    /* Initialize the peripheral function used for the DTC transfer trigger. */
    /*************************************************************************/
    R_LPM_ModuleStart(LPM_MSTP_TMR);
    TMR0->TCORA = 100-1;   /* Timer setting: 100-1 */
    TMR0->TCR   = 0x48;    /* Enable the CMIA0 interrupt and clear the counter at compare match A. */

    /*******************************************************/
    /* Make DTC transfer settings.                */
    /*******************************************************/
    /* Normal transfer mode, 16 bits, source address incremented,
       destination address fixed, */
    /* interrupt generation when all transfers are completed,
       and chain transfer disabled    */
    config.mode    = DMA_MODE_NORMAL | DMA_SIZE_WORD | DMA_SRC_INCR | DMA_DEST_FIXED |
                     DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr       = (uint32_t)(&src_data[0]);    /* Set the transfer source
                                                            address (RAM). */
    config.dest_addr      = (uint32_t)(&dest_data);
    config.transfer_count  = 5;                    /* Set the transfer count to 5. */
    config.p_transfer_data = &transfer_info;         /* DTC transfer information
                                                        storage location */

    (void)dtcDrv->Open();                           /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                 /* Specify TMR CMIA0 as the start
                                                    trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* Enable the DTC. */
    (void)dtcDrv->EnableInterrupt(DMA_INT_COMPLETE, cb_dtc_complete);

/*****************************************************************************
 Make an event link setting (register in NVIC) of the DTC transfer trigger
   (CMIA0 interrupt)
 *****************************************************************************/
/* Make an event link setting (register in NVIC) of the DTC transfer trigger (CMIA0 interrupt) and
   enable the interrupt.    */

    TMR0->TCCR = 0x0E;                          /* Start counting at PCLK/8192. */
    TMR0->TCCR = 0x0E;                          /* Start counting at PCLK/8192. */
    TMR0->TCCR = 0x0E;                          /* Start counting at PCLK/8192. */
```

Figure 2-8 Example of Procedure for Setting the DTC Transfer Complete Interrupt

2.4 Macro and Type Definitions

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

2.4.1 DTC Control Code Definitions

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

Table 2-5 List of DTC Control Codes

Definition	Value	Description
DMA_CMD_START	(0x00)	DTC transfer start command
DMA_CMD_STOP	(0x01)	DTC transfer stop command
DMA_CMD_ACT_SRC_ENABLE	(0x02)	DTC start trigger enable command
DMA_CMD_ACT_SRC_DISABLE	(0x03)	DTC start trigger disable command
DMA_CMD_DATA_READ_SKIP_ENABLE	(0x04)	Read skip setting command
DMA_CMD_CHAIN_TRANSFER_ABORT	(0x05)	DTC chain transfer abort command
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	(0x06)	DTC transfer information forcible change command
DMA_CMD_SOFTWARE_TRIGGER	(0x07)	Disabled (Note 1)
DMA_CMD_AUTO_CLEAR_SOFT_REQ	(0x08)	Disabled (Note 1)
DMA_CMD_REFRESH_DATA	(0x09)	DTC transfer information refresh command
DMA_CMD_REFRESH_EXTRA	(0x0A)	Disabled (Note 1)

Note 1. This definition is not supported by the DTC driver. If it is specified in the Control function, DMA_ERROR will be returned.

2.4.2 DTC Mode Definitions

The DTC mode definitions are to be set in the "mode" element of the st_dma_transfer_data_cfg_t structure. They are used for the second argument of the Create function or an argument when using the DTC transfer information forcible change command (DMA_CMD_CHANGING_DATA_FORCIBLY_SET). Specify a combination of mode definitions in "mode".

Example: To specify the repeat transfer mode, byte-sized transfer, incrementation of transfer source address, fixed transfer destination address, transfer source as the repeat area, interrupt request to the CPU generated upon completion of specified data transfer, and no chain transfer

```
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 | DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
```

The structure of DTC mode definitions is shown in Figure 2-9, and details of each setting are shown in Table 2-6 to Table 2-12.

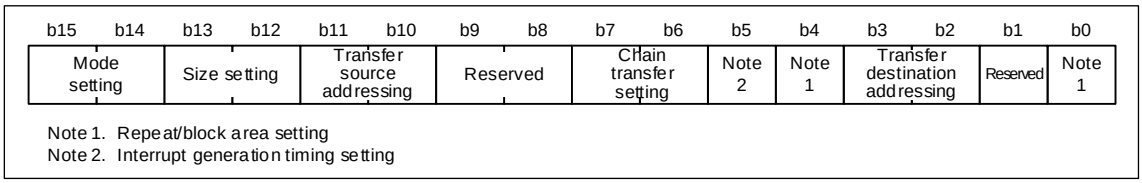


Figure 2-9 Structure of DTC Mode Definitions

Table 2-6 List of Definitions of Mode Settings

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

Table 2-7 List of Definitions of Size Settings

Definition	Value	Description
DMA_SIZE_BYTE	(0x0000U)	Byte-sized transfer(8bit)
DMA_SIZE_WORD	(0x1000U)	Word-sized transfer(16bit)
DMA_SIZE_LONG	(0x2000U)	Longword-sized transfer(32bit)

Table 2-8 List of Definitions of Transfer Source Addressing Mode Settings

Definition	Value	Description
DMA_SRC_FIXED	(0x0000U)	Transfer source address is fixed
DMA_SRC_INCR	(0x0800U)	Address is incremented after transfer
DMA_SRC_DECR	(0x0C00U)	Address is decremented after transfer
DMA_SRC_ADDR_OFFSET	(0x0400U)	Disabled (Note 1)

Note 1. This definition is not supported by the DTC driver. If it is specified in the Create or Control function, DMA_ERROR_MODE will be returned.

Table 2-9 List of Definitions of Transfer Destination Addressing Mode Settings

Definition	Value	Description
DMA_DEST_FIXED	(0x0000U)	Transfer destination address is fixed
DMA_DEST_INCR	(0x0008U)	Address is incremented after transfer
DMA_DEST_DECR	(0x000CU)	Address is decremented after transfer
DMA_DEST_ADDR_OFFSET	(0x0004U)	Disabled (Note 2)

Note 2. This definition is not supported by the DTC driver. If it is specified in the Create or Control function, DMA_ERROR_MODE will be returned.

Table 2-10 List of Definitions of Repeat or Block Area Settings

Definition	Value	Description
DMA_REPEAT_BLOCK_DEST	(0x0000U)	Transfer destination area is specified as the repeat or block area
DMA_REPEAT_BLOCK_SRC	(0x0010U)	Transfer source area is specified as the repeat or block area
DMA_REPEAT_BLOCK_NONE	(0x0001U)	Disabled (Note 3)

Note 3. This definition is not supported by the DTC driver. If it is specified in the Create or Control function, DMA_ERROR_MODE will be returned.

Table 2-11 List of Definitions of Interrupt Generation Timing Settings

Definition	Value	Description
DMA_INT_AFTER_ALL_COMPLETE	(0x0000U)	Interrupt request to the CPU is generated when the specified data transfer is completed
DMA_INT_PER_SINGLE_TRANSFER	(0x0020U)	Interrupt request to the CPU is generated each time DTC data transfer is performed

Table 2-12 List of Definitions of Chain Transfer Settings

Definition	Value	Description
DMA_CHAIN_DISABLE	(0x0000U)	Chain transfer is disabled
DMA_CHAIN_CONTINUOUSLY	(0x0080U)	Chain transfer is performed continuously
DMA_CHAIN_NORMAL	(0x00C0U)	Chain transfer is started when transfer has finished

2.4.3 DTC Interrupt Source Definitions

These are the definitions of the interrupt sources to be specified in the InterruptEnable function.

Table 2-13 List of Definitions of DTC Interrupt Sources

Definition	Value	Description
DMA_INT_COMPLETE	(1U<<0)	DTC transfer is completed
DMA_INT_SRC_OVERFLOW	(1U<<1)	Disabled (Note 4)
DMA_INT_DEST_OVERFLOW	(1U<<2)	Disabled (Note 4)
DMA_INT_REPEAT_END	(1U<<3)	Disabled (Note 4)
DMA_INT_ESCAPE_END	(1U<<4)	Disabled (Note 4)

Note 4. This definition is not supported by the DTC driver. If it is specified in the InterruptEnable function, DMA_ERROR will be returned.

2.5 Structure Definitions

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

2.5.1 st_dma_transfer_data_cfg_t Structure

This structure is used when specifying the DTC transfer setting information in 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, transfer source addressing mode, transfer destination addressing mode, repeat area, and usage of chain transfer using logical OR operators.
src_addr	uint32_t	Specifies the transfer source address.
dest_addr	uint32_t	Specifies the transfer destination address.
transfer_count	uint32_t	Specifies the transfer count.
block_size	uint32_t	Specifies the block transfer size.
offset	int32_t	Disabled (Note 1)
src_extended_repeat	uint8_t	Disabled (Note 1)
dest_extended_repeat	uint8_t	Disabled (Note 1)
*p_transfer_data	void	Specifies the address in which to store transfer information.

Note 1. This element is not supported by the DTC driver. It is ignored in the Create function.

2.5.2 st_dma_refresh_data_t Structure

This structure is used when specifying the DTC transfer information refresh command (DMA_CMD_REFRESH_DATA) in the Control function.

Table 2-15 st_dma_refresh_data_t Structure

Element Name	Type	Description
act_src	IRQn_Type	Specifies the DTC start trigger.
chain_transfer_nr	uint32_t	Specifies "chain number to be updated" − 1. Specify 0 when chain transfer is not used.
src_addr	uint32_t	Specifies the transfer source address.
dest_addr	uint32_t	Specifies the transfer destination address.
transfer_count	uint32_t	Specifies the transfer count.
block_size	uint32_t	Specifies the block transfer size.
auto_start	bool	Specifies whether auto start is available. 0: Auto start is not available. 1: Auto start is available.
keep_dma_req	bool	Disabled (Note 2)
offset	bool	Disabled (Note 2)

Note 2. This element is not supported by the DTC driver. It is ignored in the Control function.

2.5.3 st_dma_state_t Structure

This structure is used to store the transfer state that was acquired with the GetState function.

Table 2-16 st_dma_state_t Structure

Element Name	Type	Description
in_progress	bool	Shows the active state of DTC transfer. (Note 1)
esif_stat	bool	Shows the transfer escape end interrupt flag. (Note 2)
dtif_stat	bool	Shows the transfer end interrupt flag. (Note 2)
soft_req_stat	bool	Shows the flag requesting transfer by a software trigger. (Note 2)

Note 1. 0 is always returned in the GetState function.

Note 2. This element is not supported by the DTC driver. 0 is always returned in the GetState function.

2.5.4 st_dma_transfer_data_t Structure

This structure is used to store DTC transfer information.

Table 2-17 st_dma_transfer_data_t Structure

Element Name	Type	Description
mra_mrb	uint32_t	MRA/MRB register setting
sar	uint32_t	SAR register setting
dar	uint32_t	DAR register setting
cra_crb	uint32_t	CRA/CRB register setting

2.5.5 st_dma_chain_abort_t Structure

This structure is used when specifying the DTC chain transfer abort command (DMA_CMD_REFRESH_DATA) in the Control function.

Table 2-18 st_dma_chain_abort_t Structure

Element Name	Type	Description
act_src	IRQn_Type	Specifies the DTC start trigger.
chain_transfer_nr	uint32_t	Specifies the total number of transfers to be aborted.

2.5.6 st_dma_chg_data_t Structure

This structure is used when specifying the DTC transfer information forcible change command (DMA_CMD_CHANGING_DATA_FORCIBLY_SET) in the Control function.

Table 2-19 st_dma_chg_data_t Structure

Element Name	Type	Description
act_src	IRQn_Type	Specifies the DTC start trigger.
chain_transfer_nr	uint32_t	Specifies "chain number to be updated" – 1. Specify 0 when chain transfer is not used.
cfg_data	st_dma_transfer_data_cfg_t	Specifies the DTC transfer setting information.

2.6 State Transitions

The state transition diagram of the DTC driver is shown in Figure 2-10, and state-specific events and actions are shown in Table 2-20 to Table 2-22.

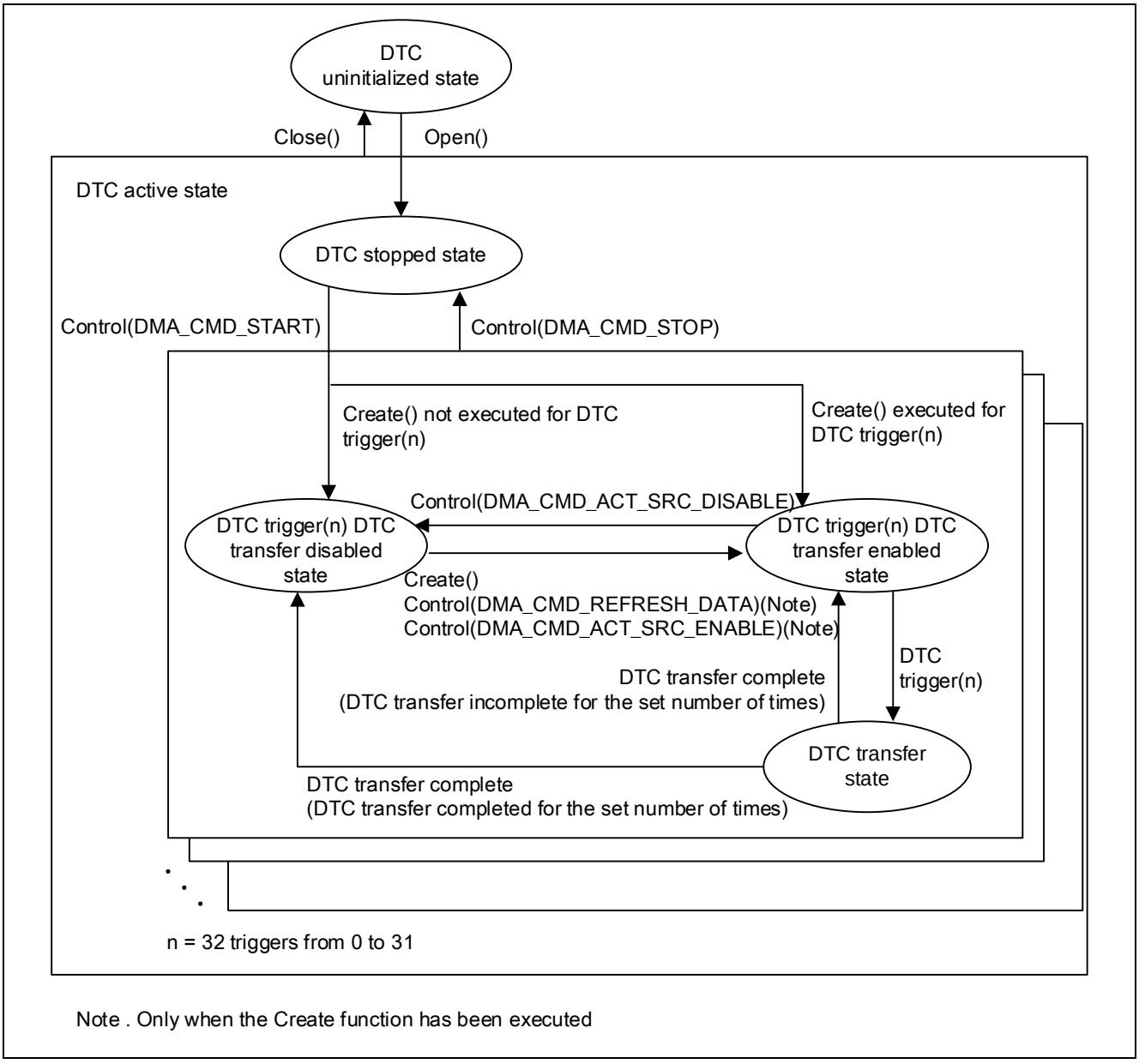


Figure 2-10 State Transitions of DTC Driver

Table 2-20 Events and Actions Specific to DTC Driver State (Note)

State	Overview	Event	Action
DTC uninitialized state	The DTC driver is in this state after release from a reset.	Executing the Open() function	Enters the DTC stopped state.
DTC stopped state	The DTC module is stopped in this state (DTCST.DTCST = 0).	Executing the Create function	Specifies the DTC transfer trigger (DTC transfer trigger enabled state).
		Executing the Control(DMA_CMD_START) function	[When Create function is not executed] Transition to DTC transfer disabled state [When the Create function has been executed] Transition to DTC transfer enabled state
		Executing the Control (DMA_CMD_ACT_SRC_ENABLE) / (DMA_CMD_ACT_SRC_DISABLE) function	Enables/Disables the specified DTC transfer trigger.
		Executing the Control (DMA_CMD_DATA_READ_SKIP_ENABLE) function	Switches between enabling and disabling of read skip.
		Executing the Control (DMA_CMD_CHAIN_TRANSFER_ABORT) function	Clears the chain transfer settings.
		Executing the Control (DMA_CMD_CHANGING_DATA_FORCIBLY_SET) function	Forcibly rewrites the DTC transfer information.
		Executing the Control (DMA_CMD_REFRESH_DATA) function	Updates the DTC transfer information (transfer source, transfer destination, and transfer count).
		Executing the InterruptEnable / InterruptDisable function	Enables/Disables the DTC transfer complete interrupt.
		Executing the Close function	Enters the DTC uninitialized state.

Note. The GetVersion, GetState, and ClearState functions can be executed in any state.

Table 2-21 Events and Actions Specific to DTC Driver State (Note)

State	Overview	Event	Action
DTC transfer disabled state	The DTC module is started (DTCST.DTCST = 1), and DTC transfer trigger activation disabled status in this state (IELSRn.DTE = 0)	Executing the Create function	Specifies the DTC transfer trigger (DTC transfer trigger enabled state).
		Executing the Control(DMA_CMD_STOP) function	Enters the DTC stopped state.
		Executing the Control (DMA_CMD_DATA_READ_SKIP_ENABLE) function	Switches between enabling and disabling of read skip.
		Executing the Control(DMA_CMD_ACT_SRC_ENABLE) function	Enables the specified DTC transfer trigger.
		Executing the Control(DMA_CMD_CHAIN_TRANSFER_ABORT) function	Clears the chain transfer settings.
		Executing the Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET) function	Forcibly rewrites the DTC transfer information.
		Executing the Control(DMA_CMD_REFRESH_DATA) function	Updates the DTC transfer information (transfer source, transfer destination, and transfer count). If the auto_start member is true (auto start enabled), transition to the DTC transfer enabled state.
		Executing the InterruptEnable / InterruptDisable function	Enables/Disables the DTC transfer complete interrupt.
		Executing the Close function	Enters the DTC uninitialized state.
DTC transfer enabled state	The DTC module is started (DTCST.DTCST = 1), and DTC transfer trigger activation disabled status in this state (IELSRn.DTE = 0)	Executing the Create function	Specifies the DTC transfer trigger (DTC transfer trigger enabled state).
		DTC transfer trigger occurs	Transition to DTC transfer state
		Executing the Control(DMA_CMD_STOP) function	Enters the DTC stopped state.
		Executing the Control (DMA_CMD_DATA_READ_SKIP_ENABLE) function	Switches between enabling and disabling of read skip.
		Executing the Control(DMA_CMD_ACT_SRC_ENABLE) function	Enables the specified DTC transfer trigger.
		Executing the Control(DMA_CMD_CHAIN_TRANSFER_ABORT) function	Clears the chain transfer settings.
		Executing the Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET) function	Forcibly rewrites the DTC transfer information.
		Executing the Control(DMA_CMD_REFRESH_DATA) function	Updates the DTC transfer information (transfer source, transfer destination, and transfer count). When auto_start member is false (auto start disabled), transition to DTC transfer disabled state.
		Executing the InterruptEnable / InterruptDisable function	Enables/Disables the DTC transfer complete interrupt.
		Executing the Close function	Enters the DTC uninitialized state.

Note. The GetVersion, GetState, and ClearState functions can be executed in any state.

Table 2-22 Events and Actions Specific to DTC Driver State (Note)

State	Overview	Event	Action
DTC transfer state	During DTC transfer	DTC transfer complete (DTC transfer incomplete for the specified number of times)	Transition to DTC transfer enabled state
		DTC transfer completed (DTC transfer completed for the specified number of times)	Transition to DTC transfer disabled state

Note. The GetVersion, GetState, and ClearState functions can be executed in any state.

3. Descriptions of Driver Operations

The DTC driver implements one or more DTC data transfers. This section shows the procedure and provides an example of operation for each operating mode.

3.1 Normal Transfer Mode

In this mode, data can be transferred in units of one byte (eight bits), one word (16 bits), or one longword (32 bits) in response to a single start trigger. The transfer count can be set to a value from 1 to 65536. Figure 3-1 shows the normal transfer procedure.

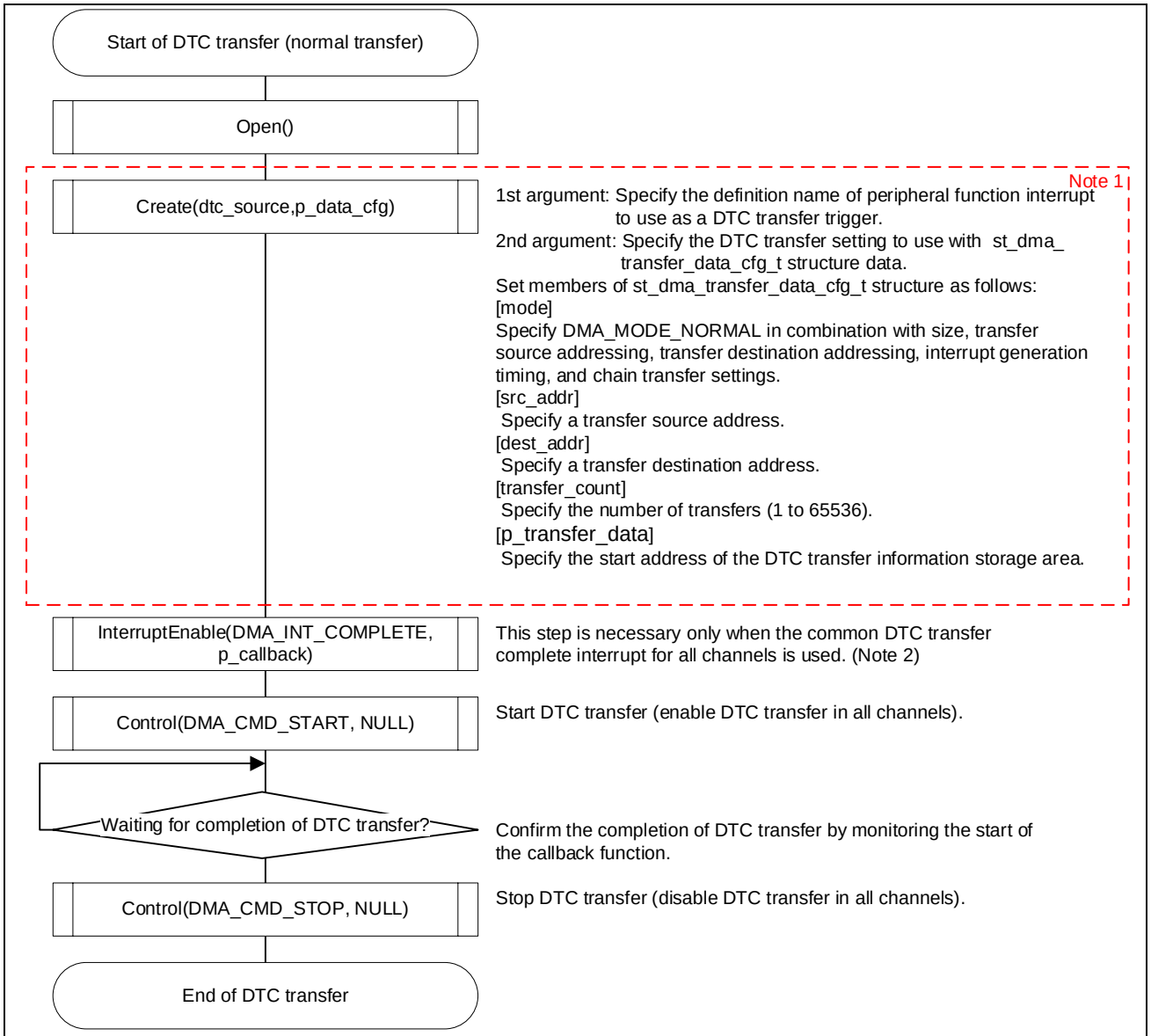


Figure 3-1 Normal Transfer Procedure

3.2 Repeat Transfer Mode

In this mode, data can be transferred in units of one byte (eight bits), one word (16 bits), or one longword (32 bits) in response to a single start trigger. The transfer count can be set to a value from 1 to 256. When DMA_INT_AFTER_ALL_COMPLETE (generating an interrupt request to the CPU when specified data transfer is completed) is specified as the interrupt generation timing, no interrupt occurs. Figure 3-2 shows the repeat transfer procedure.

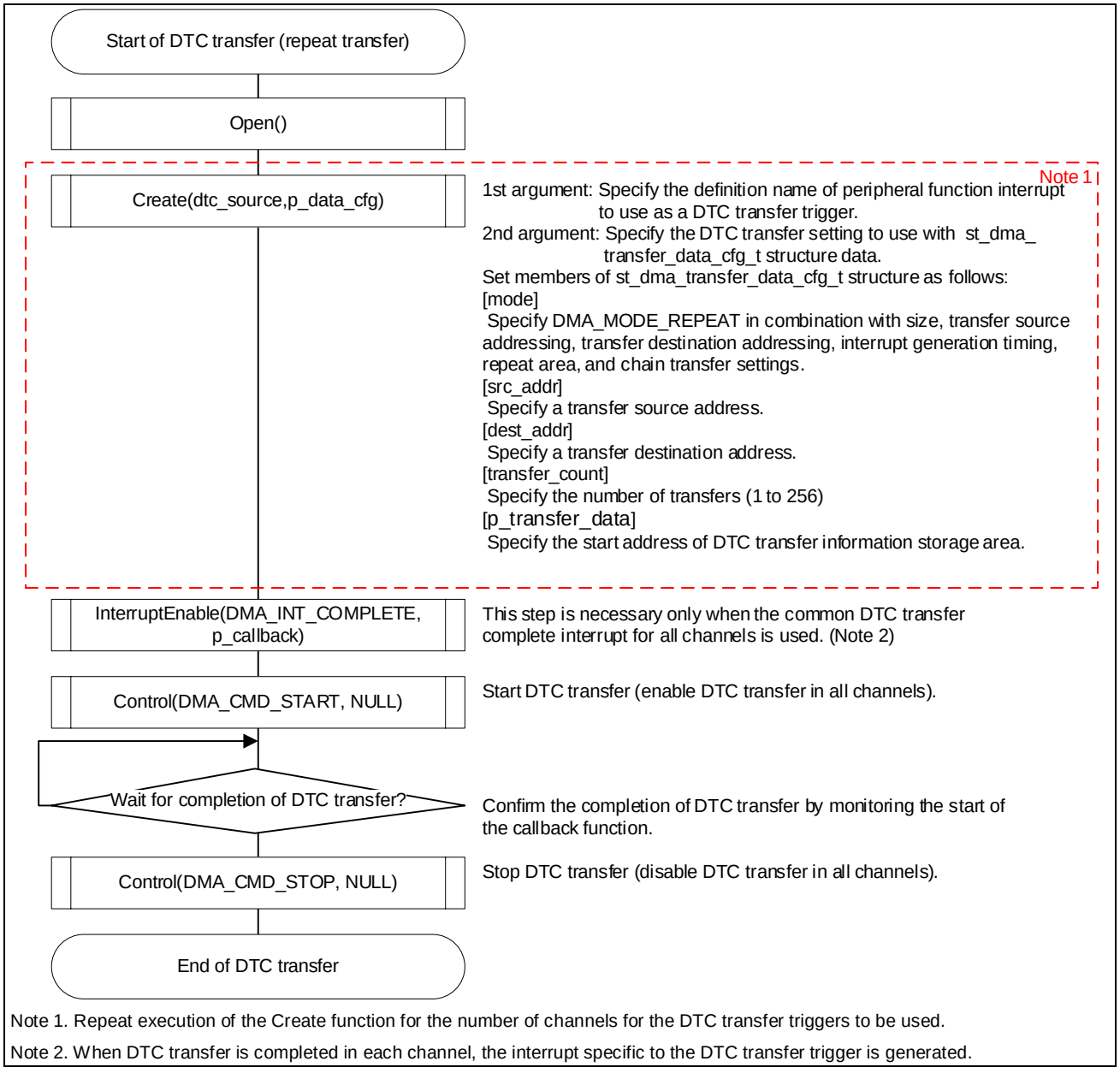


Figure 3-2 Repeat Transfer Procedure

3.3 Block Transfer Mode

In this mode, single-block data can be transferred in response to a single start trigger. The block size can be set to a value from 1 to 256 bytes, 1 to 256 words (2 to 512 bytes), or 1 to 256 longwords (4 to 1024 bytes). The transfer count (block count) can be set to a value from 1 to 65536. Figure 3-3 shows the block transfer procedure.

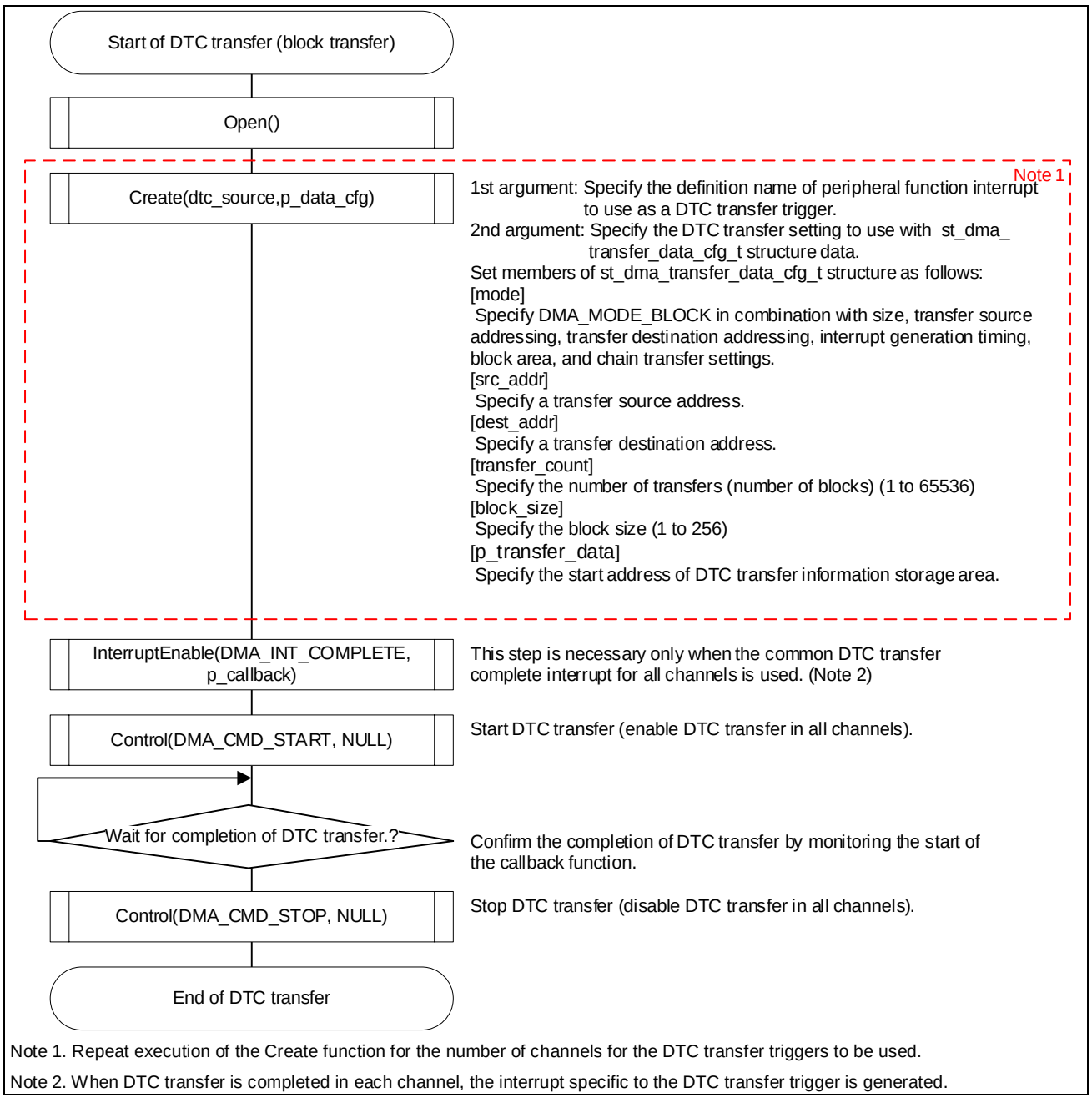


Figure 3-3 Block Transfer Procedure

3.4 Chain Transfer

Multiple data transfers can be performed sequentially in response to a single transfer trigger. Make DTC transfer settings in arrays of the st_dma_transfer_data_cfg_t type for the number of chain transfers to be executed. An area should be allocated to store DTC transfer information for the number of chain transfers and its address should be specified in the p_transfer_data member of the first array for storing DTC transfer settings; the p_transfer_data settings in the second and subsequent arrays are ignored.

Specify the start address of the array for storing DTC transfer settings as the second argument of the Create function. Figure 3-4 shows the chain transfer procedure and Figure 3-5 shows an example of chain transfer.

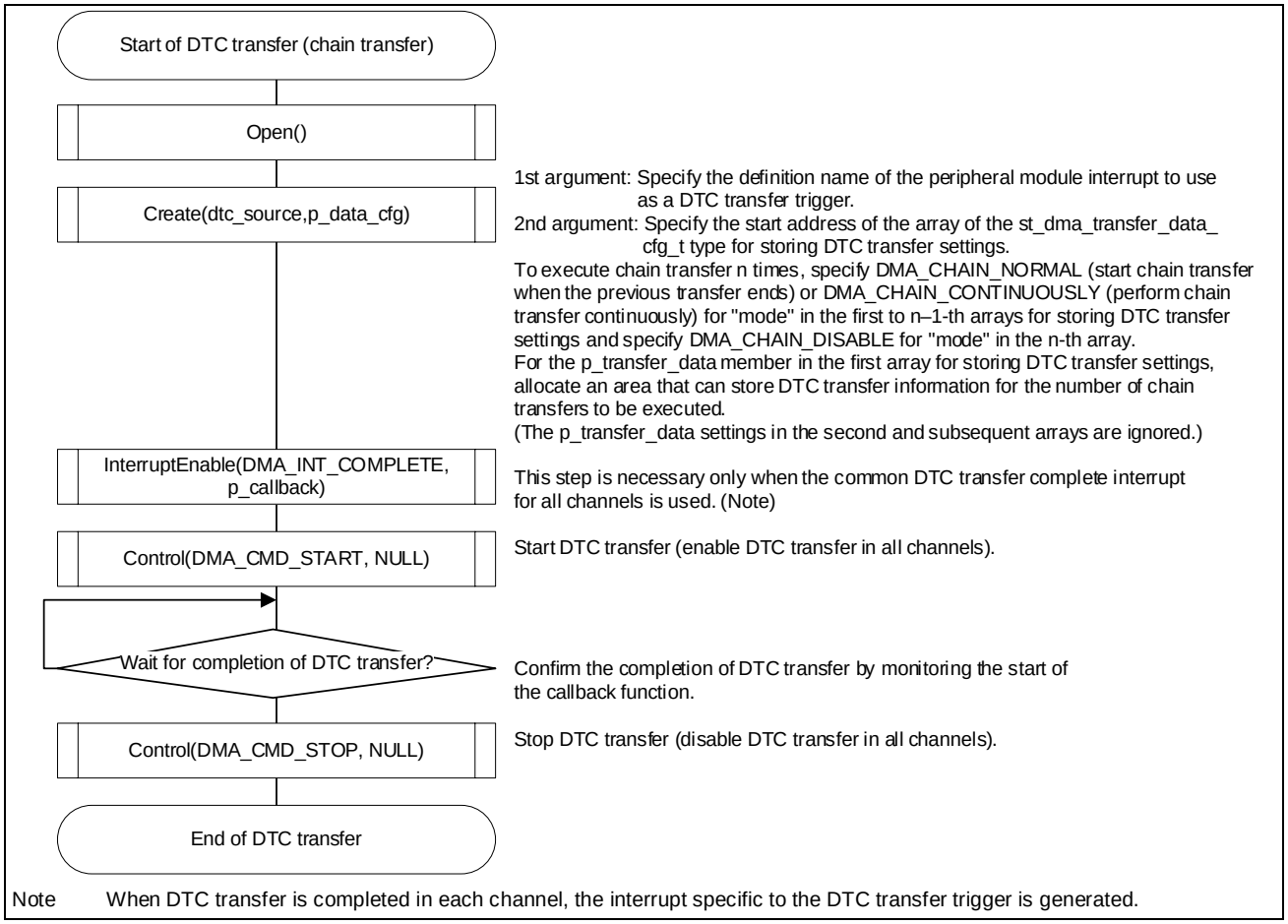


Figure 3-4 Chain Transfer Procedure

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info[2];             /* DTC transfer information storage area    */
                                                            /* (for two chain transfers) */
uint8_t src_data_ch1[5] = { 0x00, 0x01, 0x02, 0x04, 0x08 };
uint8_t src_data_ch2[5] = { 0x10, 0x11, 0x12, 0x14, 0x18 };
uint8_t dest_data_ch1 = 0xFF;
uint8_t dest_data_ch2 = 0xFF;

main()
{
    st_dma_transfer_data_cfg_t config[2];     /* DMA transfer setting information storage area
                                              /* (arrays for two chain transfers) */

    /* [First chain transfer] Normal transfer mode, 8 bits, source address incremented, destination address fixed,
*/
    /* interrupt generation when all transfers are completed, and continuous chain transfer */
    config[0].mode = DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                     DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_CONTINUOUSLY;
    config[0].src_addr          = (uint32_t)(&src_data_ch1[0]);
    config[0].dest_addr         = (uint32_t)(&dest_data_ch1);
    config[0].transfer_count    = 5;
    config[0].p_transfer_data   = &transfer_info[0];     /* Specify the DTC transfer information storage area
                                                         /* in the first DTC transfer setting information. */

    /* [Second chain transfer] Normal transfer mode, 8 bits, source address incremented, destination address    */
    /* fixed, interrupt generation when all transfers are completed, and chain transfer disabled (last chain transfer)
*/
    config[1].mode = DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                     DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config[1].src_addr          = (uint32_t)(&src_data_ch2[0]);
    config[1].dest_addr         = (uint32_t)(&dest_data_ch2);
    config[1].transfer_count    = 5;

    (void)dtcDrv->Open();                                    /* Open the DTC driver. */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config[0]);
                                                             /* Specify TMR CMIA0 as the start trigger. */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);     /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of the DTC transfer trigger (CMIA0 interrupt) and    */
    /* enable the interrupt. */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_dtc_complete);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

    TMR0->TCCR = 0x0E;    /* Activate the DTC transfer trigger (TMR). */
    while(1);
}

/******************************************************************************
* callback function
******************************************************************************/
void cb_dtc_complete(void)
{
    /* Execute the callback function when all DTC transfers in the second chain transfer are completed. */
}
```

Figure 3-5 Example of Chain Transfer

3.5 DTC Transfers Using Multiple Triggers

To use multiple DTC transfer triggers, execute the Create function multiple times. Two types of interrupts are generated: the interrupt from a transfer trigger is generated as the DTC interrupt specific to the trigger and a DTC_COMPLETE interrupt is generated as the common DTC interrupt for all triggers.

For the following sample DTC transfers using multiple triggers, Figure 3-6 to Figure 3-8 show an example of DTC settings and Figure 3-9 shows an example of interrupt generation timing.

- Repeat transfer using the AGT0_AGTI interrupt (counter underflow) as the start trigger with DMA_INT_PER_SINGLE_TRANSFER (generating an interrupt request to the CPU each time DTC data transfer is performed) specified in the Create function execution
- Normal transfer using the SCI0_RXI interrupt (reception completion) as the start trigger with DMA_INT_AFTER_ALL_COMPLETE (generating an interrupt request to the CPU when all specified data transfer is completed) specified in the Create function execution
- Normal transfer using the IRQ0 interrupt as the start trigger with DMA_INT_AFTER_ALL_COMPLETE (generating an interrupt request to the CPU when all specified data transfers are completed) specified in the Create function execution

```
#include "r_dtc_api.h"

void cb_dtc_complete(void);
void cb_agt0_agti(void);
void cb_sci0_rxi(void);
void cb_irq0(void);

// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

static st_dma_transfer_data_t transfer_info1;            /* DTC transfer information storage area */
uint8_t src_data1[5] = { 0x00, 0x01, 0x02, 0x04, 0x08 };
uint8_t dest_data1= 0xFF;
static st_dma_transfer_data_t transfer_info2;            /* DTC transfer information storage area */
uint8_t src_data2[5] = { 0x10, 0x11, 0x12, 0x14, 0x18 };
uint8_t dest_data2= 0xFF;
static st_dma_transfer_data_t transfer_info3;            /* DTC transfer information storage area */
uint8_t src_data3[5] = { 0x20, 0x21, 0x22, 0x24, 0x28 };
uint8_t dest_data3= 0xFF;

main()
{
    st_dma_transfer_data_cfg_t config;    /* DMA transfer setting information storage area    */

    (void)dtcDrv->Open();                                    /* Open the DTC driver */

    /* Repeat transfer mode, 8 bits, source address incremented, destination address fixed, transfer */
    /* destination is a block area, interrupt generation after every DTC transfer, and chain transfer disabled */
    config.mode =     DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                      DMA_REPEAT_BLOCK_SRC | DMA_INT_PER_SINGLE_TRANSFER |
                      DMA_CHAIN_DISABLE;
    config.src_addr          = (uint32_t)(&src_data1[0]);      /* Set the transfer source address (RAM). */
    config.dest_addr         = (uint32_t)(&dest_data1[0]);       /* Set the transfer destination address (RAM).*/
    config.transfer_count    = 5;                                /* Set the transfer count to 5. */
    config.p_transfer_data = &transfer_info1;                    /* DTC transfer information storage location */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &config);
                                                        /* DTC transfer trigger: AGT0_AGTI interrupt source */
```

Figure 3-6 Example of Multiple DTC Settings Using Multiple Triggers (1/3)

```
    /* Normal transfer mode, 8 bits, source address incremented, destination address fixed,                */
    /* interrupt generation when all transfers are completed, and chain transfer disabled
*/
    config.mode =     DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                      DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr           = (uint32_t)(&src_data2[0]);      /* Set the transfer source address (RAM). */
    config.dest_addr          = (uint32_t)(&dest_data2[0]);       /* Set the transfer destination address (RAM).*/
    config.transfer_count     = 3;                                /* Set the transfer count to 3. */
    config.p_transfer_data = &transfer_info2;                     /* DTC transfer information storage location */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_SCI0_TXI, &config);
                                                  /* DTC transfer trigger: SCI0 RXI interrupt source */

    /* Normal transfer mode, 8 bits, source address incremented, destination address fixed,                */
    /* interrupt generation when all transfers are completed, and chain transfer disabled
*/
    config.mode =     DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                      DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
    config.src_addr           = (uint32_t)(&src_data3[0]);      /* Set the transfer source address (RAM). */
    config.dest_addr          = (uint32_t)(&dest_data3[0]);       /* Set the transfer destination address (RAM).*/
    config.transfer_count     = 1;                                /* Set the transfer count to 1. */
    config.p_transfer_data = &transfer_info3;                     /* DTC transfer information storage location */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0, &config);
                                                  /* DTC transfer trigger: IRQ0 interrupt source */

    /* Enable the DTC_COMPLETE interrupt. */
    (void)dtcDrv->InterruptEnable(DMA_INT_COMPLETE, cb_dtc_complete);

    (void)dtcDrv->Control(DMA_CMD_START, NULL);     /* Enable the DTC. */

    /* Make an event link setting (register in NVIC) of AGT0_AGTI and enable the interrupt. */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, 0x13, cb_agt0_agti);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI);

    /*Make an event link setting (register in NVIC) of SCI0_RXI and enable the interrupt. */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_SCI0_RXI, 0x10, cb_sci0_rxi);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_SCI0_RXI, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_SCI0_RXI);

    /* Make an event link setting (register in NVIC) of IRQ0 and enable the interrupt. */
    R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0, 0x01, cb_irq0);
    R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0, 3);
    R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0);

    while(1);
}
```

Figure 3-7 Example of Multiple DTC Settings Using Multiple Triggers (2/3)

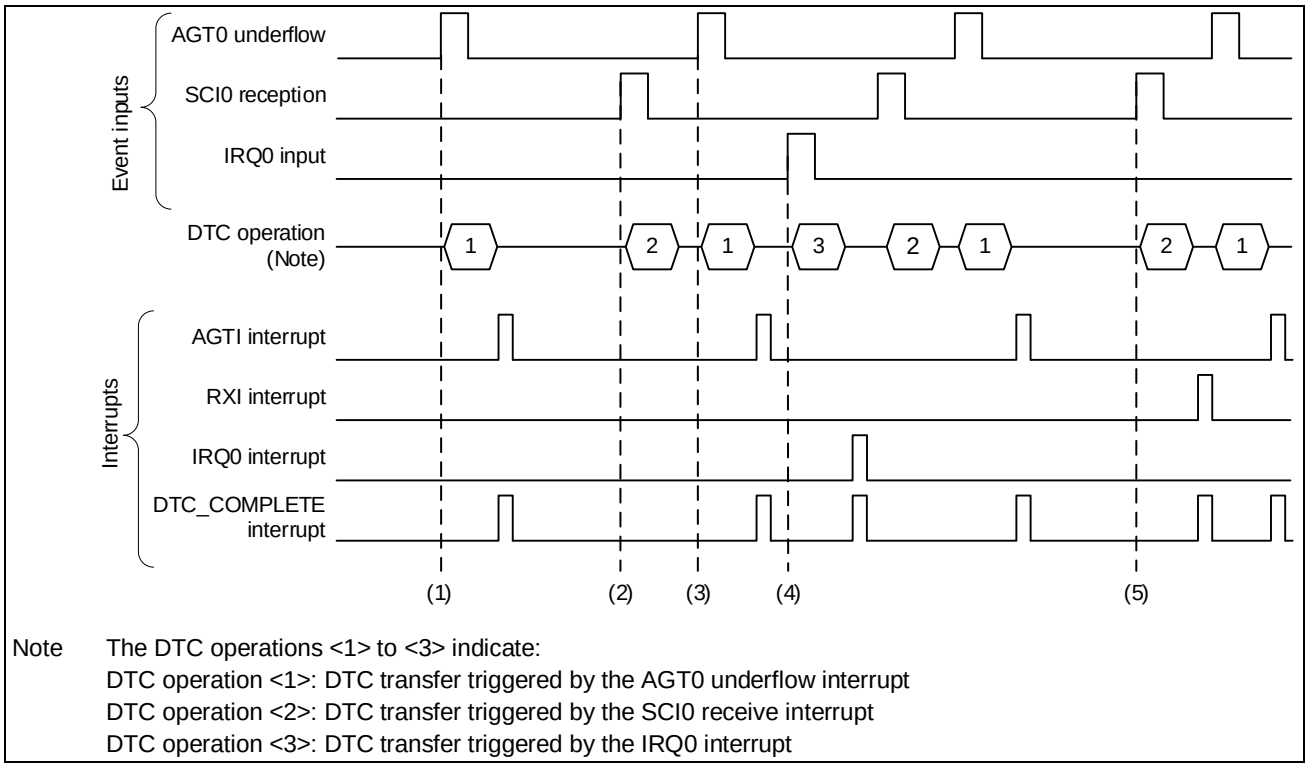
```
/***********************************************************************
* callback function for DTC_COMPLETE
***********************************************************************/
void cb_dtc_complete(void)
{
    /* Execute the callback function when the DTC transfer activated by each trigger is completed. */
}

/***********************************************************************
* callback function for AGT0_AGTI
***********************************************************************/
void cb_agt0_agti(void)
{
    /* Execute the callback function when DTC transfer activated by the AGT0_AGTI interrupt source    */
    /* is performed. */
}

/***********************************************************************
* callback function for SCI0_RXI
***********************************************************************/
void cb_sci0_rxi(void)
{
    /* Execute the callback function when all DTC transfers activated by the SCI0_RXI interrupt source    */
    /* are completed. */
}

/***********************************************************************
* callback function for IRQ0
***********************************************************************/
void cb_irq0(void)
{
    /* Execute the callback function when all DTC transfers activated by the IRQ0 interrupt source are completed.
*/
}
```

Figure 3-8 Example of Multiple DTC Settings Using Multiple Triggers (3/3)



Note The DTC operations <1> to <3> indicate:
DTC operation <1>: DTC transfer triggered by the AGT0 underflow interrupt
DTC operation <2>: DTC transfer triggered by the SCI0 receive interrupt
DTC operation <3>: DTC transfer triggered by the IRQ0 interrupt

Figure 3-9 Example of Interrupt Generation Timing when Using Multiple DTC Transfer Triggers

(1) DTC operation <1> activated by an underflow in AGT0 is performed. Every time a DTC transfer is completed, an AGTI interrupt and a DTC_COMPLETE interrupt are generated.

(2) DTC transfer <2> activated by the reception completion in the SCI is performed.

(3) Every time an underflow occurs in AGT0, DTC transfer <1> is performed and then an AGTI interrupt and a DTC_COMPLETE interrupt are generated.

(4) DTC operation <3> activated by an IRQ0 interrupt is performed. When the specified number (one time) of DTC transfer is completed, an IRQ0 interrupt and a DTC_COMPLETE interrupt are generated.

(5) When the specified number (five times) of DTC transfers activated by the reception completion in the SCI are completed, an RXI interrupt and a DTC_COMPLETE interrupt are generated.

3.6 Controlling DTC by the Control Function

The DTC operations can be controlled by the Control function. Table 3-1 is a list of control commands, command-specific arguments, and their corresponding operations.

Table 3-1 Control Commands, Command-Specific Arguments, and Corresponding Operations

Control Command (cmd)	Command-Specific Argument (arg)	Operation
DMA_CMD_START	NULL	Starts DTC transfer.
DMA_CMD_STOP	NULL	Stops DTC transfer.
DMA_CMD_ACT_SRC_ENABLE	Pointer to IRQn_Type (Note 1)	Enables the transfer trigger specified by the argument.
DMA_CMD_ACT_SRC_DISABLE	Pointer to IRQn_Type (Note 1)	Disables the transfer trigger specified by the argument.
DMA_CMD_DATA_READ_SKIP_ENABLE	Pointer to uint8_t (Note 1)	When the value of the variable specified by the argument is true: Read skip is enabled. false: Read skip is disabled.
DMA_CMD_CHAIN_TRANSFER_ABORT	Pointer to st_dma_chain_abort_t (Note 1)	Aborts the chain transfer specified by the argument.
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	Pointer to st_dma_chg_data_t (Note 1)	Forcibly changes the DTC transfer settings for the transfer trigger specified by the argument.
DMA_CMD_REFRESH_DATA	Pointer to st_dma_refresh_data_t (Note 1)	Refreshes transfer source and destination locations and transfer count for the transfer trigger specified by the argument.

Note 1. Store a desired value in the specified variable and pass its address as the argument.

3.6.1 DTC Transfer Start Command (DMA_CMD_START)

Enables DTC transfer. Set the argument to NULL. Figure 3-10 shows a usage example of the DTC transfer start command.

```
(void)dtcDrv->Control(DMA_CMD_START, NULL);     /* Enable the DTC. */
```

Figure 3-10 Usage Example of the DTC Transfer Start Command

3.6.2 DTC Transfer Stop Command (DMA_CMD_STOP)

Disables DTC transfer. Set the argument to NULL. Figure 3-11 shows a usage example of the DTC transfer stop command.

```
(void)dtcDrv->Control(DMA_CMD_STOP, NULL);     /* Disable the DTC. */
```

Figure 3-11 Usage Example of the DTC Transfer Stop Command

3.6.3 DTC Transfer Trigger Enable Command (DMA_CMD_ACT_SRC_ENABLE)

Enables the transfer trigger specified by the argument. When a DTC transfer trigger has been specified by the Create function, the trigger is enabled by default. In this case, this command is not necessary.

Use this command to enable a DTC transfer trigger after it is individually disabled by the DTC transfer trigger disable command (DMA_CMD_ACT_SRC_DISABLE) or after the DTC transfer information refresh command (DMA_CMD_REFRESH_DATA) is executed without auto start specified.

Store the DTC transfer trigger to be enabled in a variable of the IRQnType type and specify its address in the argument. Figure 3-12 shows a usage example of the DTC transfer trigger enable command.

```
IRQn_Type arg;

/* Enable DTC transfer activation by the AGTI interrupt source. */
arg = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
(void)dtcDrv->Control(DMA_CMD_ACT_SRC_ENABLE, &arg);
```

Figure 3-12 Usage Example of the DTC Transfer Trigger Enable Command

3.6.4 DTC Transfer Trigger Disable Command (DMA_CMD_ACT_SRC_DISABLE)

Disables the transfer trigger specified by the argument. This command clears the DTE bit in the IELSRn register to 0 (disable DTC activation); when the specified start trigger is generated, a request is sent to the NVIC and a callback function is executed.

Store the DTC transfer trigger to be disabled in a variable of the IRQnType type and specify its address in the argument. Figure 3-13 shows a usage example of the DTC transfer trigger disable command.

```
IRQn_Type arg;

/* Disable DTC transfer activation by the AGTI interrupt source. */
arg = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
(void)dtcDrv->Control(DMA_CMD_ACT_SRC_DISABLE, &arg);
```

Figure 3-13 Usage Example of the DTC Transfer Trigger Disable Command

3.6.5 Read Skip Setting Command (DMA_CMD_DATA_READ_SKIP_ENABLE)

Enables or disables the read skip feature. When the DTC is initialized by the Open function, read skip is enabled by default.

Store the value of true (enable) or false (disable) in a variable of the uint8_t type and specify its address in the argument. Figure 3-14 shows a usage example of the read skip setting command.

```
uint8_t arg;

/* Enable read skip. */
arg = true;
(void)dtcDrv->Control(DMA_CMD_DATA_READ_SKIP_ENABLE, &arg);
```

Figure 3-14 Usage Example of the Read Skip Setting Command

3.6.6 DTC Chain Transfer Abort Command (DMA_CMD_CHAIN_TRANSFER_ABORT)

Aborts DTC chain transfer. This command clears the CHNE bit of DTC mode register B (MRB) to 0 (disable DTC chain transfer) in the DTC transfer information for the specified count of chain transfers. Therefore, only the first DTC transfer information data (information data buf[0]) is valid.

Store the DTC transfer trigger and the chain transfer count in a variable of the st_dma_chain_abort_t type and specify its address in the argument. Figure 3-15 shows a usage example of the DTC chain transfer abort command.

```
st_dma_chain_abort_t    arg;

/* Aborts DTC chain transfer activated by the AGTI interrupt source. */
arg.act_src = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 2;
(void)dtcDrv->Control(DMA_CMD_CHAIN_TRANSFER_ABORT, &arg);
```

Figure 3-15 Usage Example of the DTC Chain Transfer Abort Command

3.6.7 DTC Transfer Information Forcible Change Command (DMA_CMD_CHANGING_DATA_FORCIBLY_SET)

Forcibly changes the DTC transfer information.

Store the target DTC start trigger of change, the value of "chain number to be changed – 1" (for example, store 1 to change the second chain), and new DTC transfer information to be set (st_dma_transfer_data_cfg_t type) in a variable of the st_dma_chg_data_t type and specify its address in the argument. DTC transfer information is forcibly changed only for the chain specified by the stored chain number. For the p_transfer_data member of the DTC transfer setting information, specify the same value as that used in the Create function. If a different value is specified, correct operation is not guaranteed.

Figure 3-16 shows an operation example when chain number 1 is specified, and Figure 3-17 shows a usage example of the DTC transfer information forcible change command.

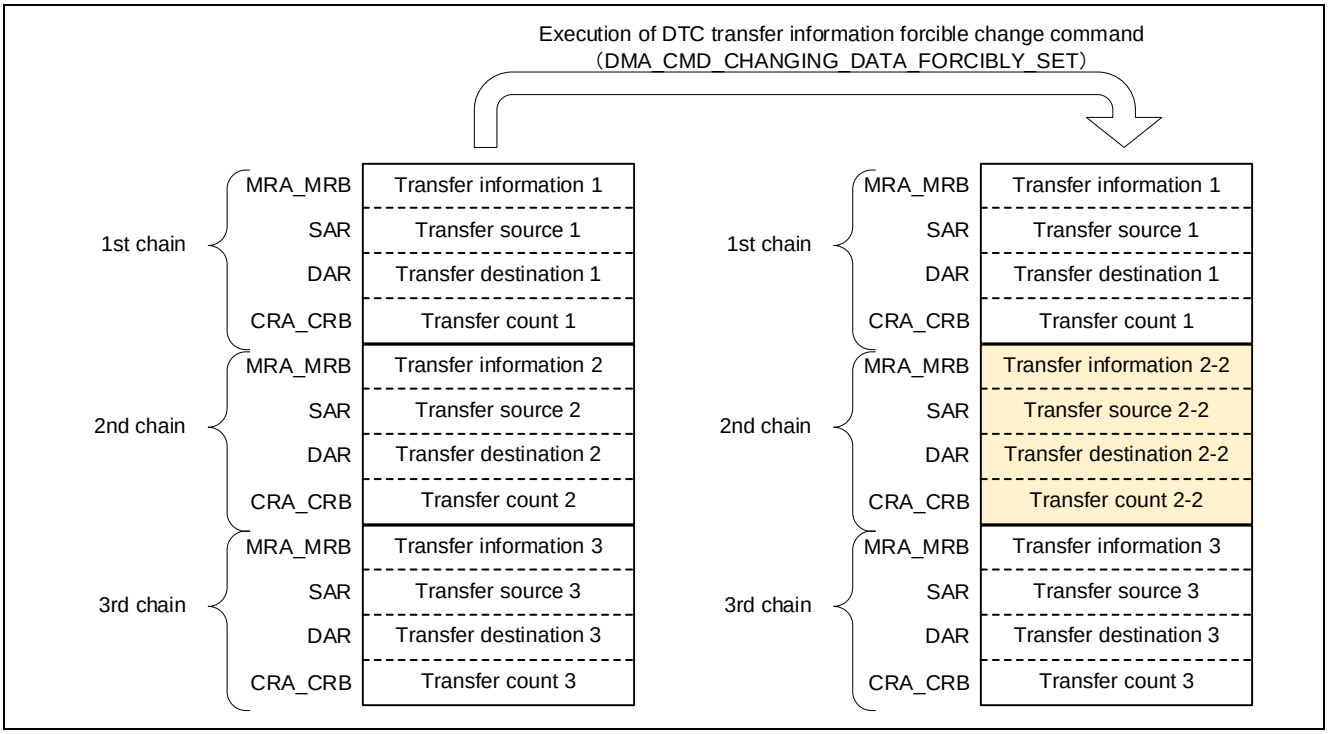


Figure 3-16 Operation Example of DTC Transfer Information Forcible Change Command

```
t_dma_chg_data_t arg;

/* Forcibly change the settings of the second chain transfer activated by the AGTI interrupt source. */
arg.act_src = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 1;
arg.cfg_data.mode = DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_FIXED |
                    DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_CONTINUOUSLY;
arg.cfg_data.src_addr          = (uint32_t)(&src_data_ch1[0]);
arg.cfg_data.dest_addr         = (uint32_t)(&dest_data_ch1);
arg.cfg_data.transfer_count    = 5;
arg.cfg_data.p_transfer_data   = &transfer_info;     /* Specify the same DTC transfer storage area as that */
                                                     /* specified in Create.                                */

(void)dtcDrv->Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET, &arg);
```

Figure 3-17 Usage Example of the DTC Transfer Information Forcible Change Command

3.6.8 DTC Transfer Information Refresh Command (DMA_CMD_REFRESH_DATA)

Refreshes the DTC transfer information.

This command only updates the transfer source and destination addresses(Note) and transfer count without changing the mode after a DTC transfer is completed. Setting the auto_start member to 1 restarts DTC transfer after this command execution. After the transfer information is updated with the auto_start member set to 0, use the DTC transfer trigger enable command (DMA_CMD_ACT_SRC_ENABLE) with the desired timing to enable DTC transfer.

If NULL (0) is set for the transfer source address, the current transfer source address is retained. Similarly, if NULL (0) is set for the transfer destination address, the current transfer destination address is retained.

For the chain_transfer_nr member, specify the value of (chain transfer number to be updated – 1). To update the information of multiple chain transfers, execute DTC transfer information refresh command multiple times. When chain transfer is not used, set the member to 0.

Figure 3-18 shows a usage example of the DTC transfer information refresh command when chain transfer is not used. Figure 3-19 shows a usage example of the DTC transfer information refresh command when chain transfer is used (chain transfer count = 2)

Note. Set the transfer source address and transfer destination address to a multiple of 2 when the transfer size is 16-bit size transfer and to a multiple of 4 when the transfer size is 32-bit size transfer.

```
    st_dma_refresh_data_t arg;

/* Update the settings of transfer activated by the AGTI interrupt source and enable DTC transfer. */
    arg.act_src       = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
    arg.chain_transfer_nr = 0;                /* Set to 0 because chain transfer is not used. */
    arg.src_addr      = (uint32_t)(&source_ref); /* Update the transfer source address. */
    arg.dest_addr     = (uint32_t)(&dest_ref);   /* Update the transfer destination address. */
    arg.transfer_count  = 5;                  /* Update the transfer count. */
    arg.block_size     = 0;                /* Update the block size (valid only for block transfer). */
    arg.auto_start     = true;              /* After the refresh command execution, enable the DTC. */
    (void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);
```

Figure 3-18 Usage Example of the DTC Transfer Information Refresh Command (Not Using Chain Transfer)

```
st_dma_refresh_data_t arg;

/* Update the settings of chain transfer 1 activated
   by the AGTI interrupt source (DTC transfer is stopped). */
arg.act_src       = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 0;                   /* Update chain number 1. */
arg.src_addr      = (uint32_t)(&source_ref_1); /* Update the transfer source address. */
arg.dest_addr     = (uint32_t)(&dest_ref_1);   /* Update the transfer destination address. */
arg.transfer_count  = 5;                     /* Update the transfer count. */
arg.block_size     = 0;              /* Update the block size (valid only for block transfer). */
arg.auto_start     = false; /* Leave DTC transfer disabled after the refresh command execution. */
(void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);

/* Update the settings of chain transfer 2 activated
    by the AGTI interrupt source and enable DTC transfer. */
arg.act_src       = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 1;                   /* Update chain number 2. */
arg.src_addr      = (uint32_t)(&source_ref_2); /* Update the transfer source address. */
arg.dest_addr     = (uint32_t)(&dest_ref_2);   /* Update the transfer destination address. */
arg.transfer_count  = 2;                     /* Update the transfer count. */
arg.block_size     = 0;               /* Update the block size (valid only for block transfer). */
arg.auto_start     = true;               /* Enable the DTC after the refresh command execution. */
(void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);
```

Figure 3-19 Usage Example of the DTC Transfer Information Refresh Command (Using Chain Transfer, Chain Transfer Count = 2)

3.7 Get number of DMA transfer bytes

The number of DTC transfer bytes can be acquired with the GetTransferByte function. Set the DTC transfer factor for acquiring the number of transfer bytes in the first argument and the variable for storing the acquisition result in the second argument.

During repeat transfer, the DTC transfer byte count returns to 0 when transfer of the specified repeat size is complete.

Example of using GetTransferByte function Figure 3-20, Example of acquiring DTC transfer byte number in normal transfer mode Table 3-2, Example of acquiring DTC transfer byte number in repeat transfer mode Table 3-3, Example of DTC in block transfer mode An example of acquiring the transfer byte count is shown in Table 3-4.

```
uint32_t transfer_byte;
/* Get DTC transfer byte count due to AGTI interrupt */
(void)dtc0Drv->GetTransferByte(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &transfer_byte);
```

Figure 3-20 Example of using GetTransferByte function

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

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

Table 3-3 Example of acquiring DTC 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
DTC transfer byte count (Note)	4	8	12	16	0	4	8

Table 3-4 Example of acquiring DTC 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
DTC transfer byte count (Note)	10	20	30	40	50	60	70

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

Note 2. The DTC driver resets (0) the number of DTC transfer bytes after the transfer for the repeat size is completed.

3.8 Configurations

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

3.8.1 Parameter Checking

This enables or disables the parameter checking in the R_DTC driver.

Name: DTC_CFG_PARAM_CHECKING_ENABLE

Table 3-5 Settings of DTC_CFG_PARAM_CHECKING_ENABLE

Setting	Description
0	Disables the parameter checking. The error conditions regarding the validity of arguments described in Function Specifications will not be detected.
1 (initial value)	Enables the parameter checking. The error conditions regarding the validity of arguments described in Function Specifications will not be detected.

3.8.2 DTC_COMPLETE Interrupt Priority Level

This specifies the priority level of the DTC_COMPLETE interrupt.

Name: DTC_COMPLETE_PRIORITY

Table 3-6 Settings of DTC_COMPLETE_PRIORITY

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

3.8.3 Function Allocation to RAM

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

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

Name: DTC_CFG_xxx

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

Example: R_DTC_Open function → DTC_CFG_R_DTC_OPEN

Table 3-7 Settings of DTC_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_DTC_GetVersion	
2	R_DTC_Open	
3	R_DTC_Close	
4	R_DTC_Create	
5	R_DTC_Control	
6	R_DTC_InterruptEnable	
7	R_DTC_InterruptDisable	
8	R_DTC_GetState	
9	R_DTC_ClearState	
10	R_DTC_GetTransferByte	
11	dtc_comp_interrupt (DTC_COMPLETE interrupt processing)	✔

4. Detailed Information of Driver

This section describes the detailed specifications implementing the features of this driver.

4.1 Function Specifications

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

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

4.1.1 R_DTC_Open Function

Table 4-1 R_DTC_Open Function Specifications

Format	static e_dma_err_t R_DTC_Open(void)	
Description	Initializes the DTC driver (initializes RAM and makes register settings).	
Argument	None	
Return value	DMA_OK	DTC initialization completed
	DMA_ERROR_LOCKED	DTC initialization failed due to locked resources The DTC initialization will fail if the DTC resources are locked. (If the DTC is already locked by the R_SYS_ResourceLock function).
	DMA_ERROR	DTC initialization failed The DTC initialization will fail if the DTC cannot be released from the module stop state.

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv->Open();
	}	

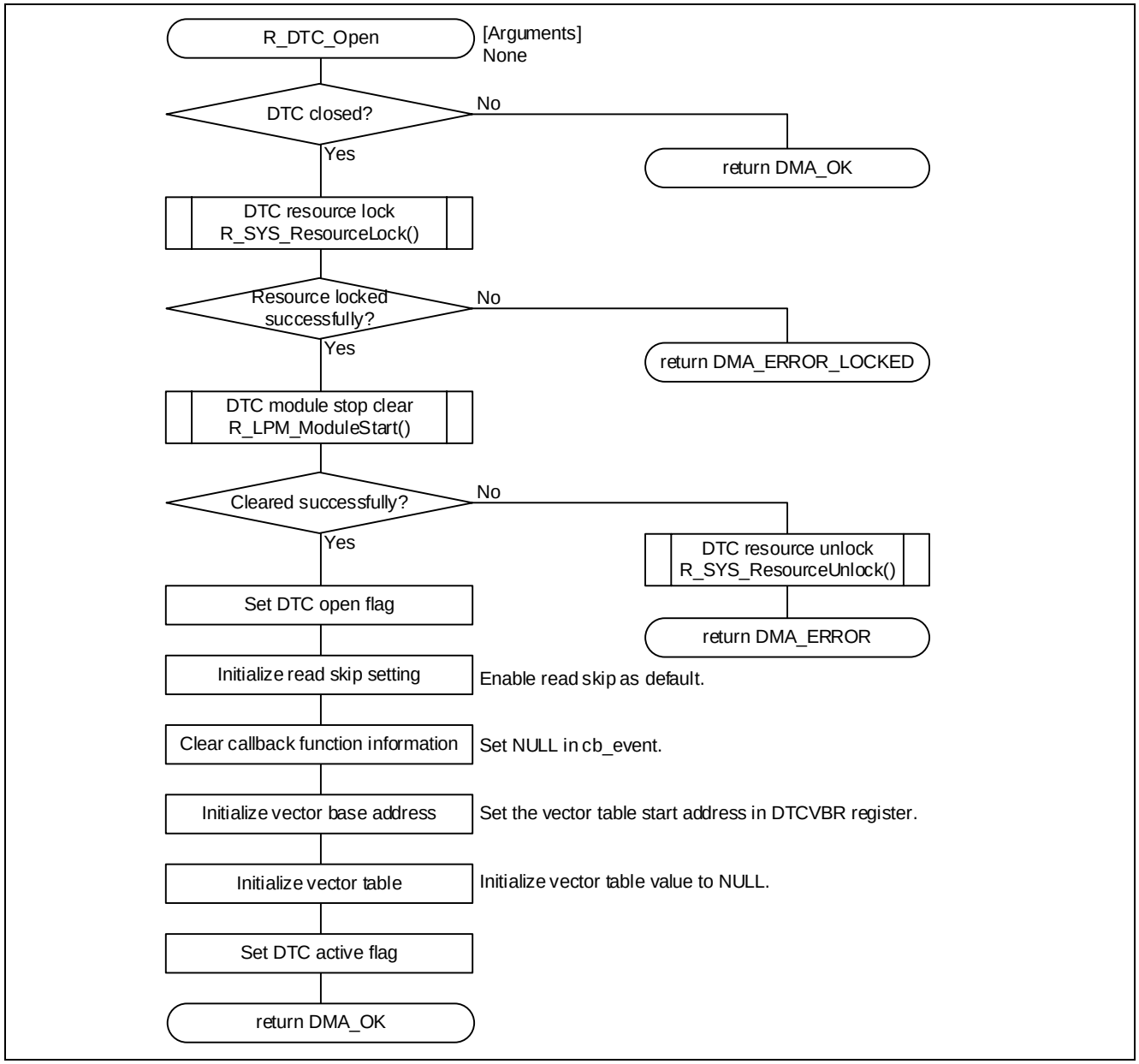


Figure 4-1 R_DTC_Open Function Processing Flow

4.1.2 R_DTC_Close Function

Table 4-2 R_DTC_Close Function Specifications

Format	static e_dma_err_t R_DTC_Close(void)
Description	Releases the DTC driver.
Argument	None
Return value	DMA_OK DTC release completed

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv->Open();
	}

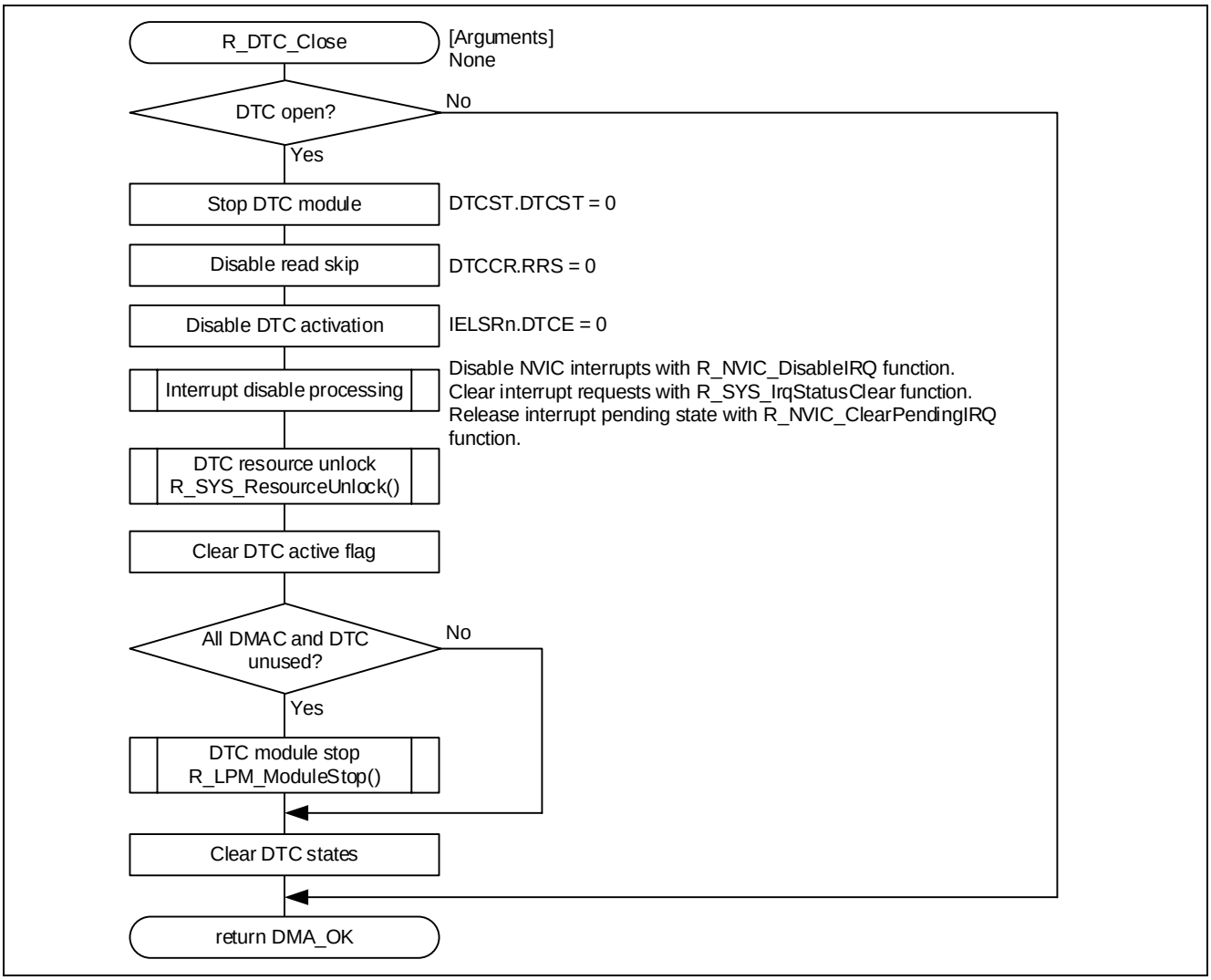


Figure 4-2 R_DTC_Close Function Processing Flow

4.1.3 R_DTC_Create Function

Table 4-3 R_DTC_Create Function Specifications

Format	static e_dma_err_t R_DTC_Create(int16_t const dtc_source, st_dma_transfer_data_cfg_t * const p_data_cfg)
Description	Makes DTC transfer settings.
Argument	int16_t const dtc_source: DTC transfer trigger Specify the definition name of the peripheral module interrupt to be used as a DTC transfer trigger. (Example: For IRQ0, specify SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0.)
	st_dma_transfer_data_cfg_t * const p_data_cfg: DTC transfer settings Specify the DTC transfer settings for the specified start trigger.
Return value	DMA_OK DTC transfer setting completed
	DMA_ERROR_PARAMETER Parameter error If one of the following conditions is detected, a parameter error will occur. • If the value specified for the DTC transfer trigger is illegal (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or a greater value, or a value smaller than 0) • If the transfer count is set outside the range of 1 to 65536 in normal transfer settings • If the transfer count is set outside the range of 1 to 256 in repeat transfer settings • If the transfer count is set outside the range of 1 to 65536 in block transfer settings • If the block count is set outside the range of 1 to 256 in block transfer settings • If the transfer source or destination address is not a multiple of 2 when the transfer size is set to words (two bytes) • If the transfer source or destination address is not a multiple of 4 when the transfer size is set to longwords (four bytes) • If a value of 1 is specified in a reserved bit of the mode setting
	DMA_ERROR_MODE Mode error If one of the following conditions is detected, a mode error will occur. • If the offset definition (DMA_SRC_ADDR_OFFSET) is specified for the transfer source addressing mode • If the offset definition (DMA_DEST_ADDR_OFFSET) is specified for the transfer destination addressing mode • If "no repeat or block area" (DMA_REPEAT_BLOCK_NONE) is specified for the repeat or block area setting • If the value of the mode setting is illegal • If the value of the size setting is illegal • If the value of the repeat or block area setting is illegal • If the value of the chain transfer setting is illegal
	DMA_ERROR DTC transfer setting failed DTC transfer setting will fail if it is executed before initialization of the DTC.

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	st_dma_transfer_data_cfg_t config;
	config.mode = DMA_MODE_NORMAL | DMA_SIZE_WORD | DMA_SRC_INCR | DMA_DEST_FIXED |
	DMA_INT_AFTER_ALL_COMPLETE | DMA_CHAIN_DISABLE;
	config.src_addr = (uint32_t)(&src_data[0]); /* Set the transfer source address (RAM). */
	config.dest_addr = (uint32_t)(&dest_data);
	config.transfer_count = 5; /* Set the transfer count to 5. */
	config.p_transfer_data = &transfer_info; /* DTC transfer information storage location */
	
	(void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
	}

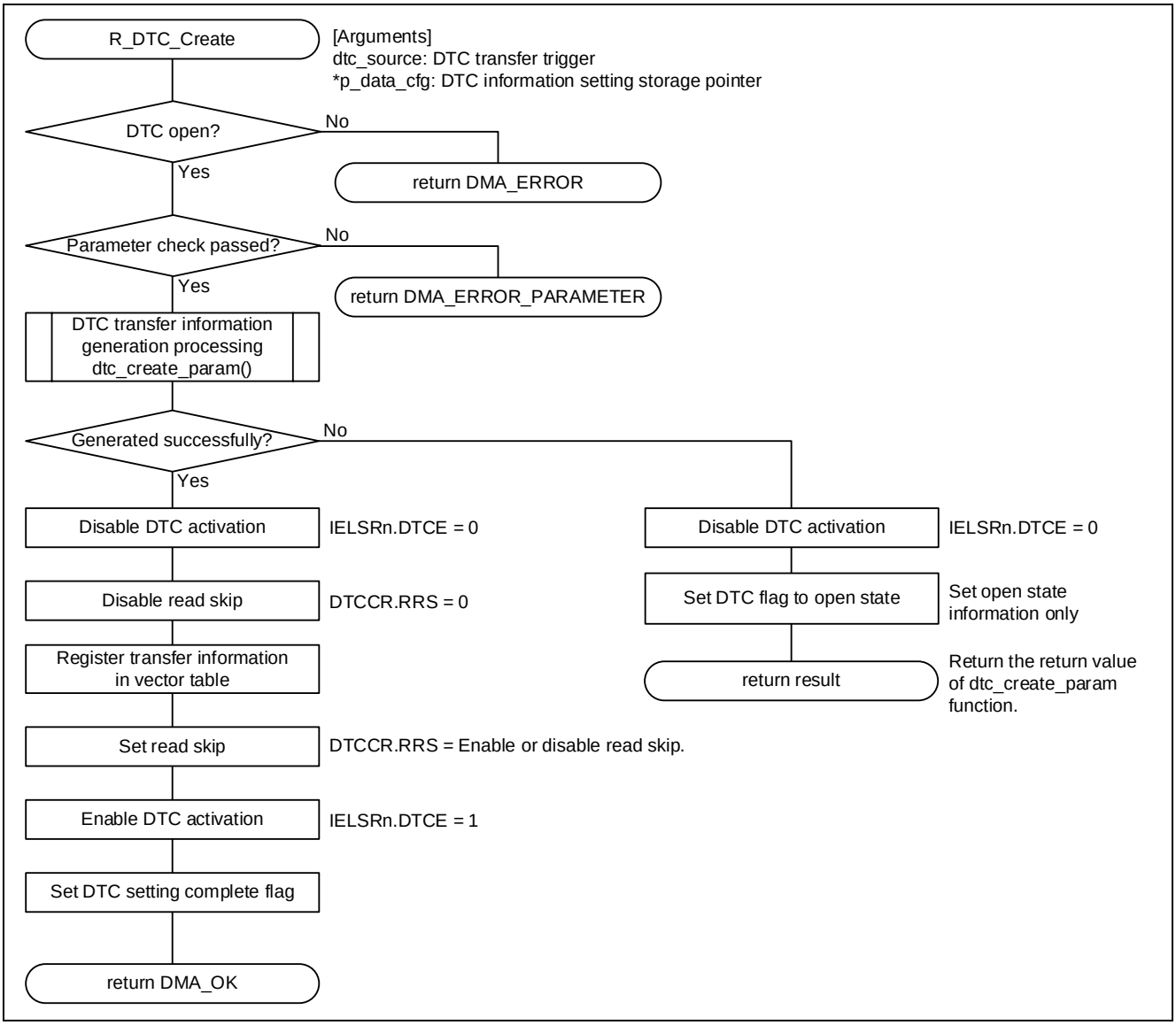


Figure 4-3 R_DTC_Create Function Processing Flow

4.1.4 R_DTC_Control Function

Table 4-4 R_DTC_Control Function Specifications

Format	static e_dma_err_t R_DTC_Control(e_dma_cmd_t cmd, void const * const p_arg)
Description	Executes a DTC control command.
Argument	e_dma_cmd_t cmd: DTC control command Specify one of the following control commands. • DMA_CMD_START: DTC transfer start command • DMA_CMD_STOP: DTC transfer stop command • DMA_CMD_ACT_SRC_ENABLE: DTC transfer trigger enable command • DMA_CMD_ACT_SRC_DISABLE: DTC transfer trigger disable command • DMA_CMD_DATA_READ_SKIP_ENABLE: Read skip setting command • DMA_CMD_CHAIN_TRANSFER_ABORT: DTC chain transfer abort command • DMA_CMD_CHANGING_DATA_FORCIBLY_SET: DTC transfer information forcible change command • DMA_CMD_REFRESH_DATA: DTC transfer information refresh command
	void const * const p_arg: Pointer to a variable that stores the command-specific setting (Refer to Table 4-5 for the relationship between control commands and argument)
Return value	DMA_OK Control command execution completed
	DMA_ERROR Control command execution failed If one of the following conditions is detected, control command execution will fail. • If this function is executed before initialization of the DTC • If any of the following commands is executed with the argument set to NULL - DMA_CMD_ACT_SRC_ENABLE command - DMA_CMD_ACT_SRC_DISABLE command - DMA_CMD_DATA_READ_SKIP_ENABLE command - DMA_CMD_CHAIN_TRANSFER_ABORT command - DMA_CMD_CHANGING_DATA_FORCIBLY_SET command - DMA_CMD_REFRESH_DATA command • If any of the following commands is executed with SYSTEM_IRQ_EVENT_NUMBER_NOT_USED (interrupt not used) or a greater value specified as the transfer trigger - DMA_CMD_ACT_SRC_ENABLE command - DMA_CMD_ACT_SRC_DISABLE command - DMA_CMD_CHAIN_TRANSFER_ABORT command - DMA_CMD_CHANGING_DATA_FORCIBLY_SET command - DMA_CMD_REFRESH_DATA command • If any of the following commands is executed while the DTC vector table for the specified transfer trigger is NULL (Create function has not been executed for the specified transfer trigger) - DMA_CMD_ACT_SRC_ENABLE command - DMA_CMD_ACT_SRC_DISABLE command - DMA_CMD_CHAIN_TRANSFER_ABORT command - DMA_CMD_CHANGING_DATA_FORCIBLY_SET command - DMA_CMD_REFRESH_DATA command • If an illegal command is executed

<table>
<tr><td>Return value</td><td>DMA_ERROR_PARAMETER Parameter error
If one of the following conditions is detected, a parameter error will occur.
[In DMA_CMD_CHANGING_DATA_FORCIBLY_SET command execution]
• If the transfer count is set outside the range of 1 to 65536 in normal transfer settings
• If the transfer count is set outside the range of 1 to 256 in repeat transfer settings
• If the transfer count is set outside the range of 1 to 65536 in block transfer settings
• If the block count is set outside the range of 1 to 256 in block transfer settings
• If the transfer source or destination address is not a multiple of 2 when the transfer size is set to words (two bytes)
• If the transfer source or destination address is not a multiple of 4 when the transfer size is set to longwords (four bytes)
• If a value of 1 is specified in a reserved bit of the mode setting
[In DMA_CMD_REFRESH_DATA command execution]
• If the transfer count is set outside the range of 1 to 65536 in normal transfer settings
• If the transfer count is set outside the range of 1 to 256 in repeat transfer settings
• If the transfer count is set outside the range of 1 to 65536 in block transfer settings
• If the block count is set outside the range of 1 to 256 in block transfer settings</td></tr>
<tr><td></td><td>DMA_ERROR_MODE Mode error
If one of the following conditions is detected during DMA_CMD_CHANGING_DATA_FORCIBLY_SET command execution, a mode error will occur.
• If the offset definition (DMA_SRC_ADDR_OFFSET) is specified for the transfer source addressing mode
• If the offset definition (DMA_DEST_ADDR_OFFSET) is specified for the transfer destination addressing mode
• If "no repeat or block area" (DMA_REPEAT_BLOCK_NONE) is specified for the repeat or block area setting
• If the value of the mode setting is illegal
• If the value of the size setting is illegal
• If the value of the repeat or block area setting is illegal
• If the value of the chain transfer setting is illegal</td></tr>
<tr><td>Remarks</td><td>[Example of calling function from instance]
// DTC driver instance
extern DRIVER_DMA Driver_DTC;
DRIVER_DMA *dtcDrv = &Driver_DTC;

main()
{
(void)dtcDrv->Control(DMA_CMD_START, NULL);
}</td></tr>
</table>

Table 4-5 Control Commands, Command-Specific Arguments, and Corresponding Operations

Control Command (cmd)	Command-Specific Argument (arg)	Operation
DMA_CMD_START	NULL	Starts DTC transfer.
DMA_CMD_STOP	NULL	Stops DTC transfer.
DMA_CMD_ACT_SRC_ENABLE	Pointer to IRQn_Type (Note 1)	Enables the transfer trigger specified by the argument.
DMA_CMD_ACT_SRC_DISABLE	Pointer to IRQn_Type (Note 1)	Disables the transfer trigger specified by the argument.
DMA_CMD_DATA_READ_SKIP_ ENABLE	Pointer to uint8_t (Note 1)	When the value of the variable specified by the argument is true: Read skip is enabled. false: Read skip is disabled.
DMA_CMD_CHAIN_TRANSFER_ ABORT	Pointer to st_dma_chain_abort_t (Note 1)	Aborts the chain transfer specified by the argument.
DMA_CMD_CHANGING_DATA_ FORCIBLY_SET	Pointer to st_dma_chg_data_t (Note 1)	Forcibly changes the DTC transfer settings for the transfer trigger specified by the argument.
DMA_CMD_REFRESH_DATA	Pointer to st_dma_refresh_data_t (Note 1)	Refreshes transfer source and destination locations and transfer count for the transfer trigger specified by the argument.

Note 1. Store a desired value in the specified variable and pass its address as the argument.

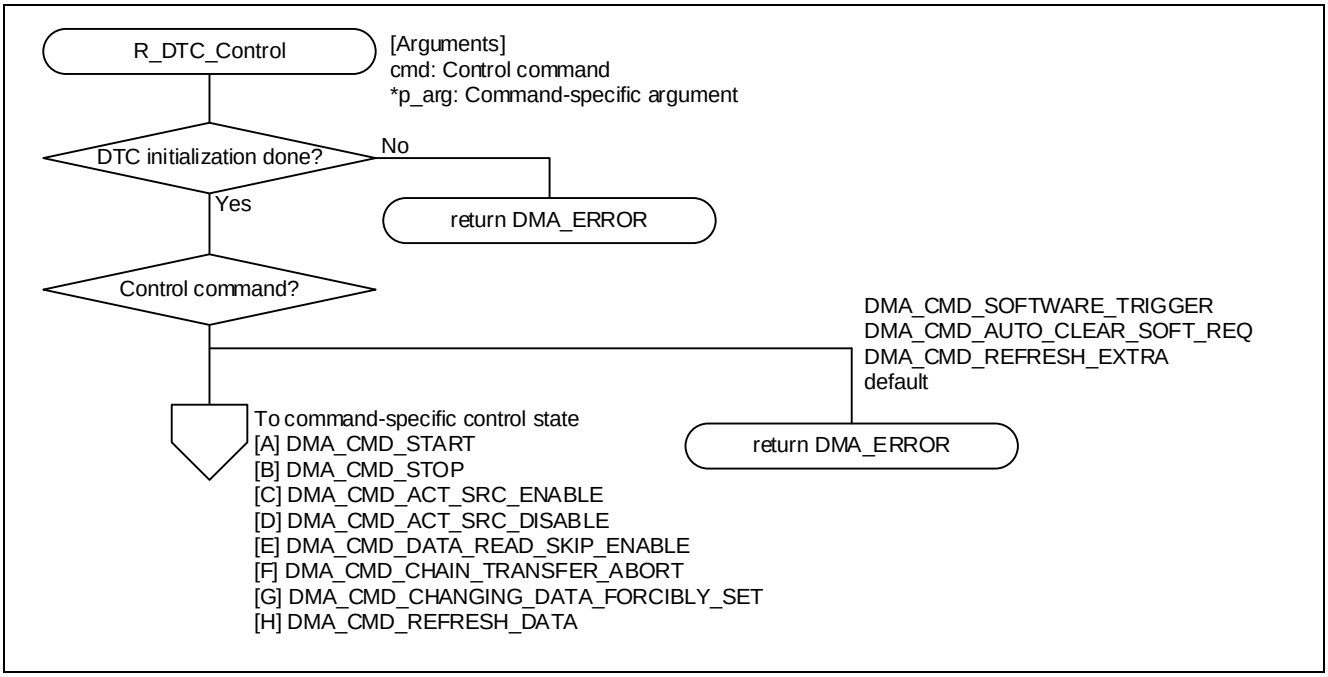


Figure 4-4 R_DTC_Control Function Processing Flow (1/3)

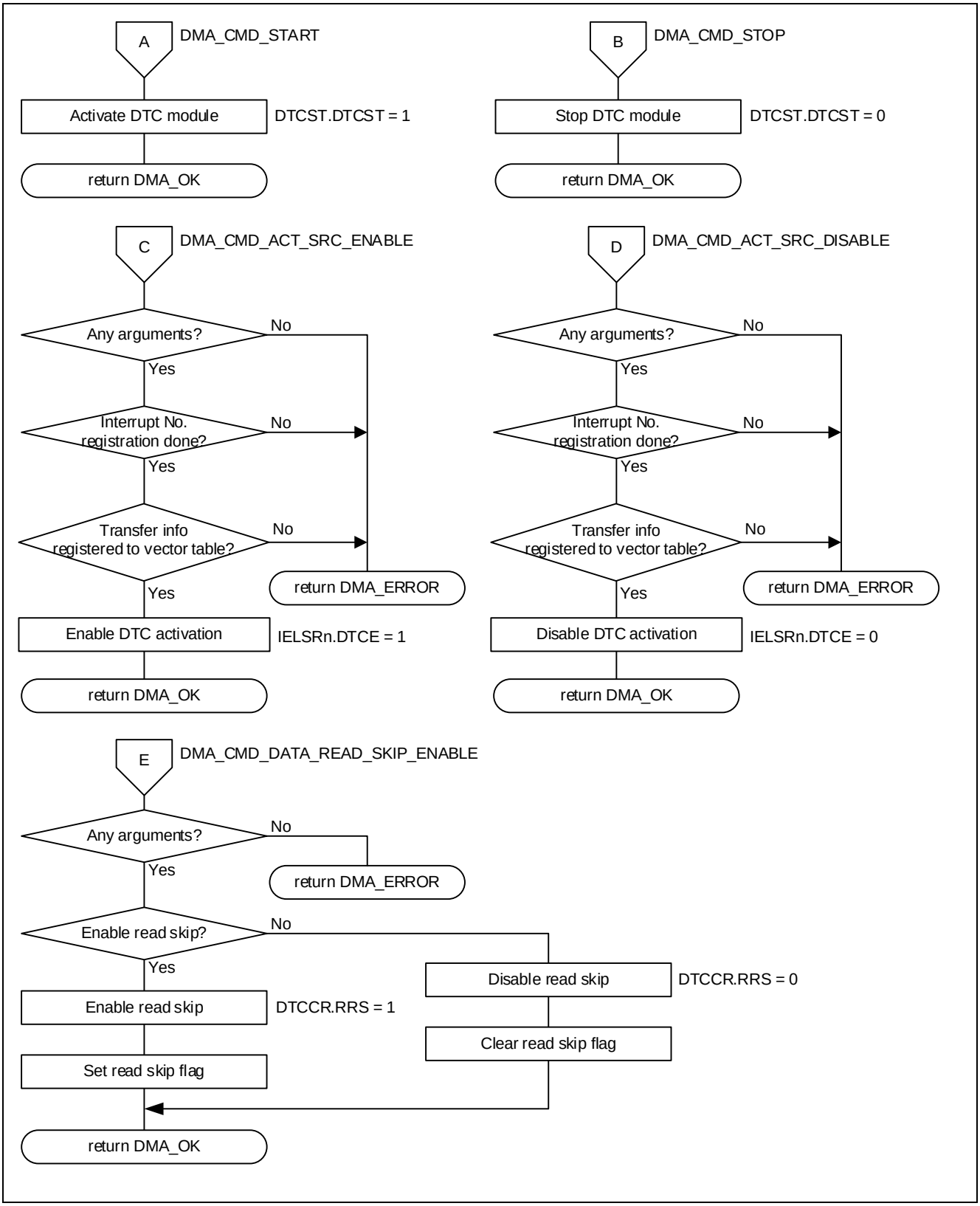


Figure 4-5 R_DTC_Control Function Processing Flow (2/3)

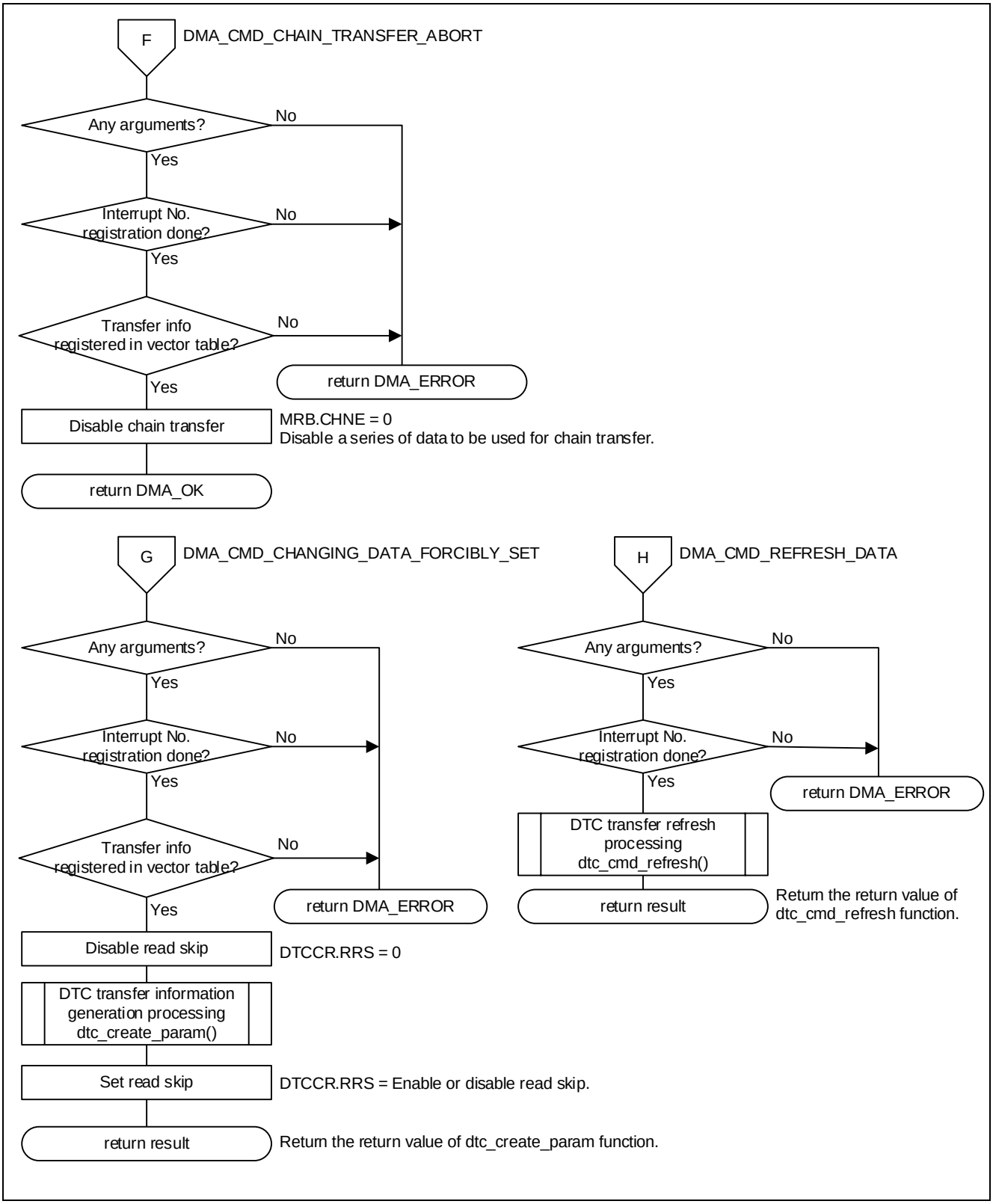


Figure 4-6 R_DTC_Control Function Processing Flow (3/3)

4.1.5 R_DTC_InterruptEnable Function

Table 4-6 R_DTC_InterruptEnable Function Specifications

Format	static e_dma_err_t R_DTC_InterruptEnable(uint16_t const int_fact, dma_cb_event_t const p_callback)
Description	Sets the transfer complete interrupt (DTC_COMPLETE) of the DTC driver.
Argument	uint16_t const int_fact: DMA interrupt source Specify DMA_INT_COMPLETE.
	dma_cb_event_t const p_callback: Callback function Specify the callback function to be executed when a DTC transfer complete interrupt occurs.
Return value	DTC_OK Interrupt setting completed
	DTC_ERROR Interrupt setting failed. If one of the following conditions is detected, interrupt enabling will fail. • If this function is executed before initialization of the DTC • If a value other than DMA_INT_COMPLETE is specified as the DMA interrupt source
	DMA_ERROR_SYSTEM_SETTING System setting error If the DTC transfer complete interrupt (DTC_COMPLETE) is set to "no interrupt used" (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or a greater value), a system setting error will occur.

Remarks	[Example of calling function from instance]
	void cb_dtc_complete(void);
	
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv-> InterruptEnable(DMA_INT_COMPLETE, cb_dtc_complete);
	}

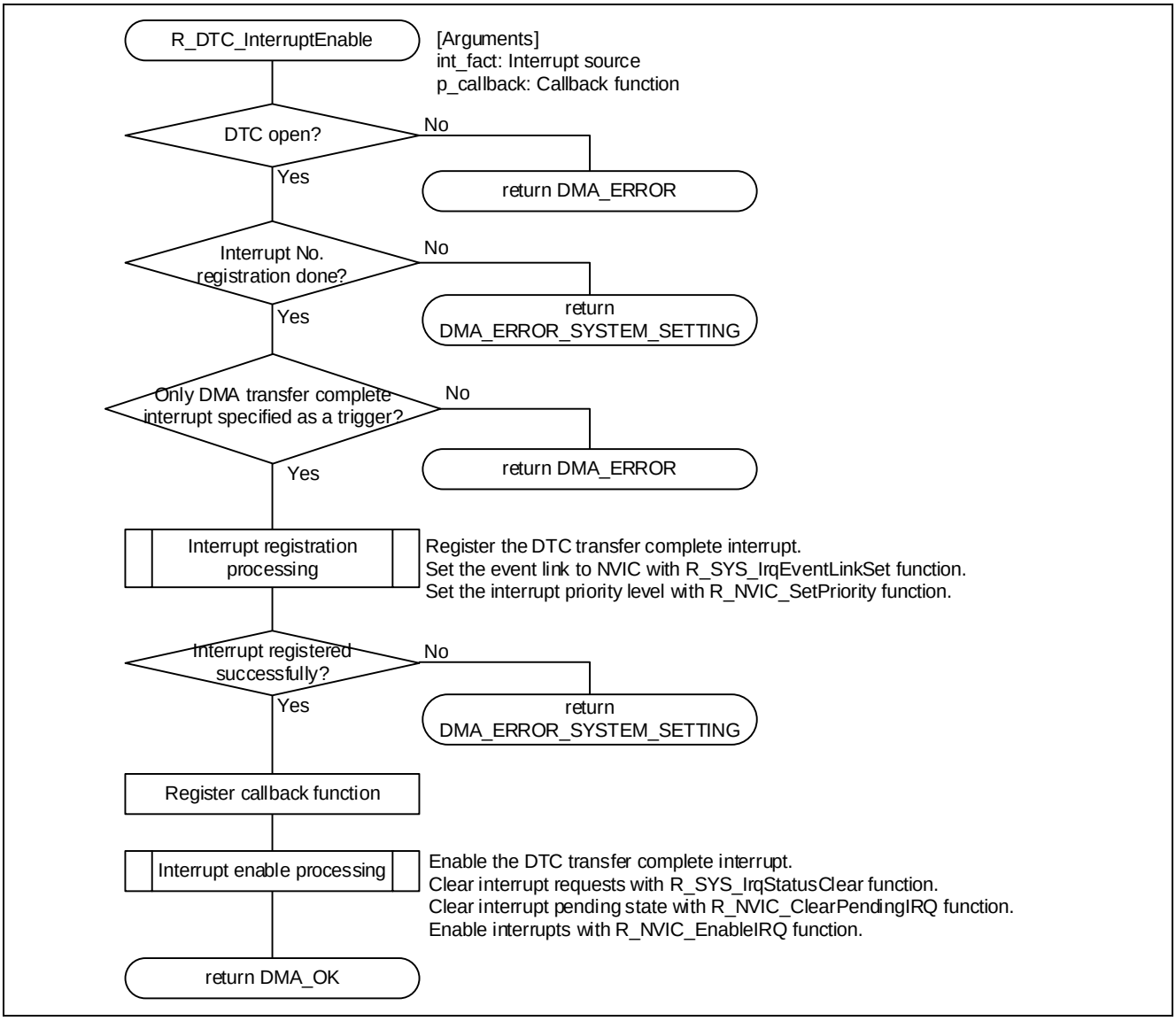


Figure 4-7 R_DTC_InterruptEnable Function Processing Flow

4.1.6 R_DTC_InterruptDisable Function

Table 4-7 R_DTC_InterruptDisable Function Specifications

Format	static e_dma_err_t R_DTC_InterruptDisable(void)
Description	Disables the transfer complete interrupt of the DTC driver
Argument	None
Return value	DTC_OK Interrupt disabled normally
	DTC_ERROR Interrupt disabling failed If this function is executed before initialization of the DTC, interrupt disabling will fail.
	DMA_ERROR_SYSTEM_SETTING System setting error If the setting of the DTC transfer complete interrupt (DTC_COMPLETE) is "no interrupt used" (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or a greater value), a system setting error will occur.

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv-> InterruptDisable();
	}

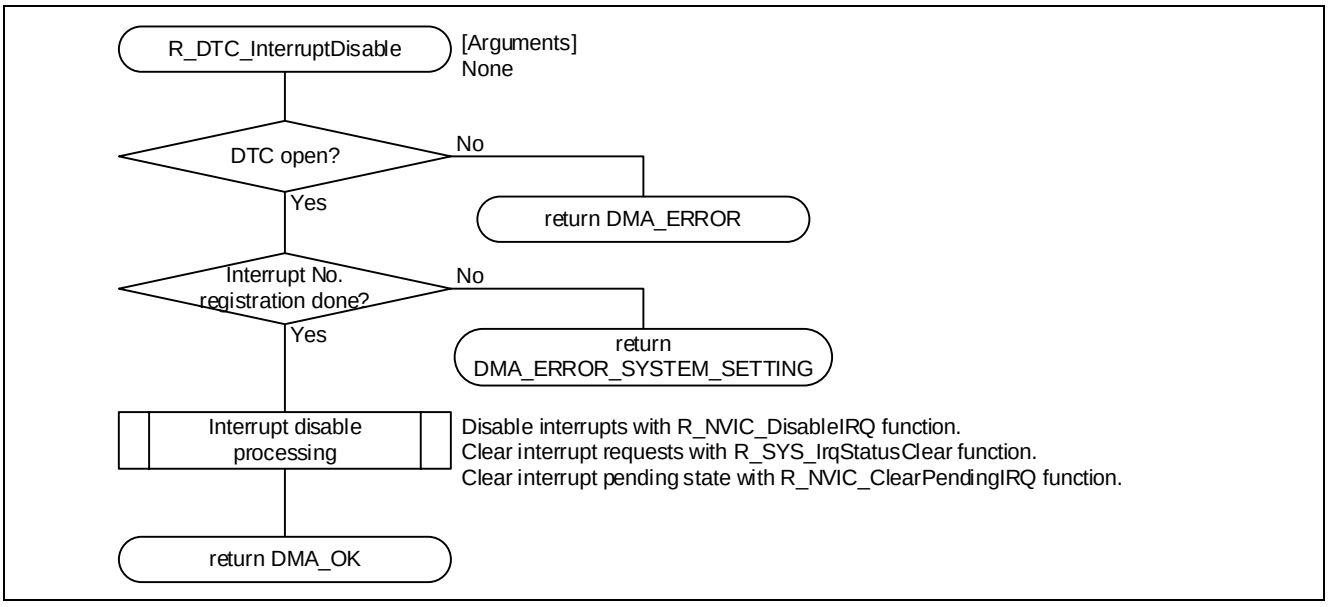


Figure 4-8 R_DTC_InterruptDisable Function Processing Flow

4.1.7 R_DTC_GetState Function

Table 4-8 R_DTC_GetState Function Specifications

Format	static e_dma_err_t R_DTC_GetState(st_dma_state_t * const state)
Description	Obtains the DTC status (the DTC always return 0).
Argument	st_dma_state_t * const state: Status storage area Specify the area for storing the obtained status (0 is always returned as the status).
Return value	ARM_DRIVER_OK Slave transmission start completed

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	st_dma_state_t status;
	
	main()
	{
	dtcDrv-> GetState(&status);
	}

4.1.8 R_DTC_ClearState Function

Table 4-9 R_DTC_ClearState Function Specifications

Format	static e_dma_err_t R_DTC_ClearState(uint32_t clr_state)
Description	Clears the status (the DTC does nothing).
Argument	uint32_t clr_state: Not used in the DTC
Return value	DMA_OK Cleared normally

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	st_dma_state_t status;
	
	main()
	{
	dtcDrv-> GetState(0);
	}

4.1.9 R_DTC_GetTransferByte Function

Table 4-10 R_DTC_GetTransferByte Function Specifications

Format	static e_dma_err_t R_DTC_GetTransferByte(uint32_t * transfer_byte, st_dtc_resources_t * const p_dtc)
Description	Get the number of DTC transfer bytes
Argument	uint32_t *transfer_byte : DTC transfer byte storage pointer Specifies a pointer to store the DTC transfer byte
	st_dtc_resources_t * const p_dtc : DTC resources Specify the DTC resource to get the status.
Return value	DMA_OK Successful acquisition of DTC transfer byte count
	DMA_ERROR DTC transfer byte count acquisition failure If one of the following conditions is detected, GetTransferByte execution will fail. ・When executed before DTC initialization ・If the specified DTC vector table is NULL (Create function not executed for specified cause)
	DMA_ERROR_PARAMETER Parameter error
	A parameter error occurs if the specified DTC transfer source is SYSTEM_IRQ_EVENT_NUMBER_NOT_USED (interrupt not used) or more, or less than 0.
	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 DTC resources is not required.
	
	[Example of calling function from instance]
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	uint32_t byte;
	dtcDrv-> GetTransferByte(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &byte);
	}

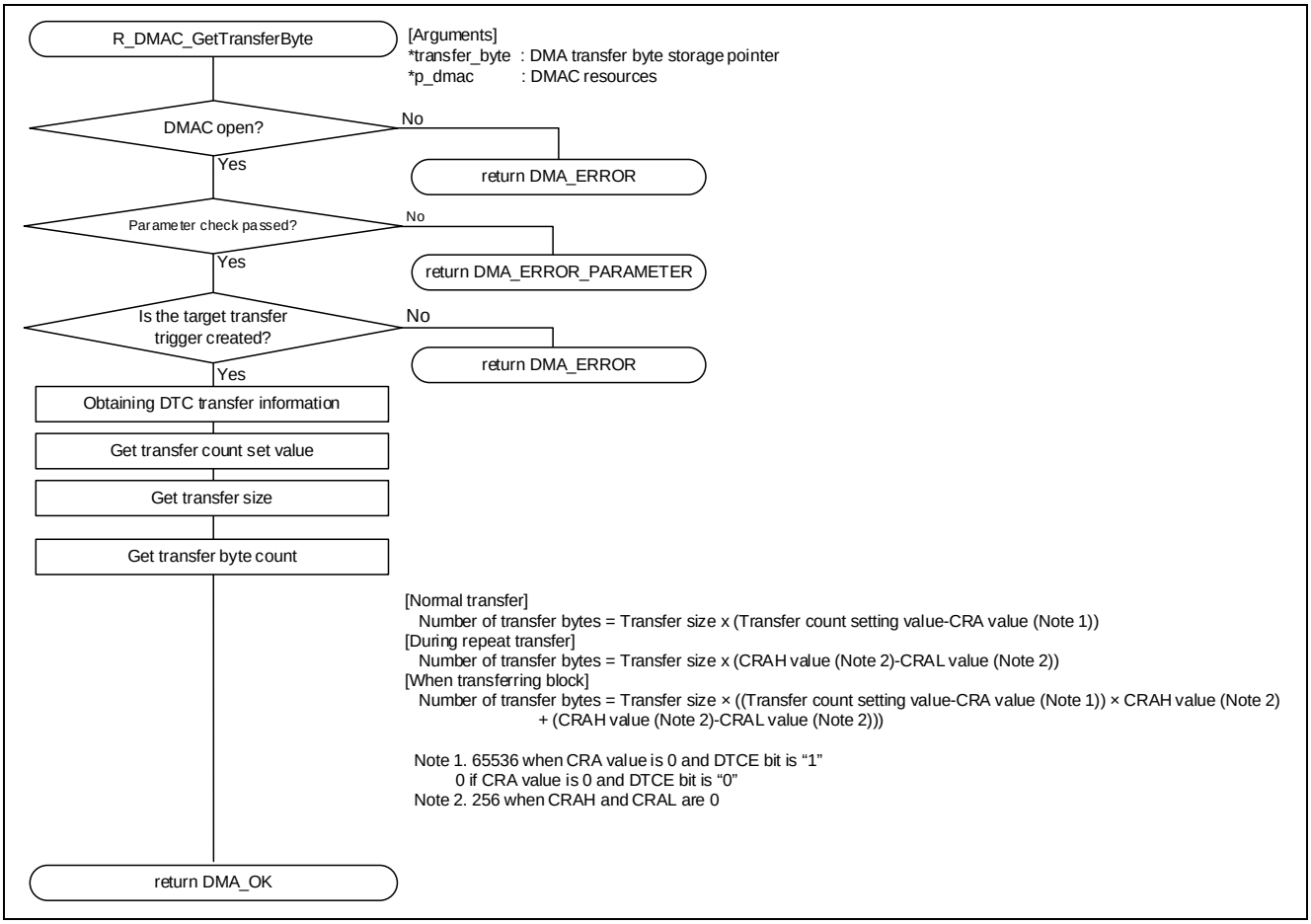


Figure 4-9 R_DTC_GetVersion Function Processing Flow

4.1.10 R_DTC_GetVersion Function

Table 4-11 R_DTC_GetVersion Function Specifications

Format	static uint32_t R_DTC_GetVersion(void)
Description	Obtains the version of the DTC driver.
Argument	None
Return value	Version of the DTC driver

Remarks	[Example of calling function from instance]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv-> GetVersion(0);
	}

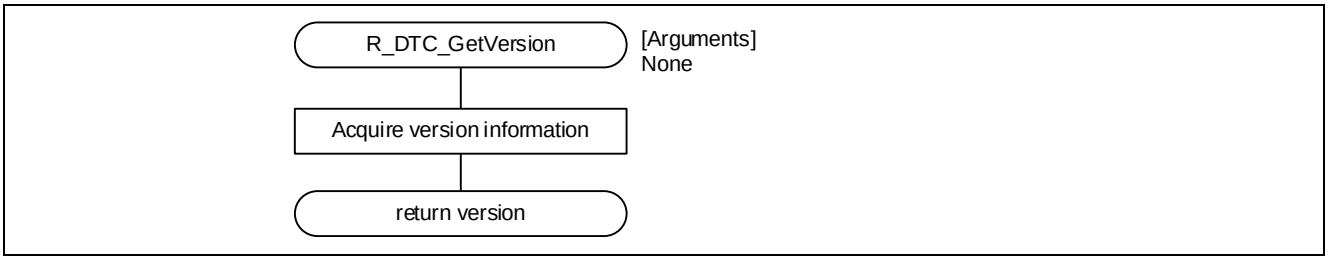


Figure 4-10 R_DTC_GetVersion Function Processing Flow

4.1.11 dtc_create_param Function

Table 4-12 dtc_create_param Function Specifications

Format	static e_dma_err_t dtc_create_param(st_dma_transfer_data_t *p_transfer_info, st_dma_transfer_data_cfg_t const * p_data_cfg, uint8_t chain_set_en)
Description	Specifies values of DTC transfer information.
Argument	st_dma_transfer_data_t *p_transfer_info: Transfer information storage area Specify the address of the area for storing transfer information.
	st_dma_transfer_data_cfg_t const * p_data_cfg: DTC transfer settings Specify the DTC transfer settings.
	uint8_t chain_set_en: Chain transfer setting enable true: Enables chain transfer settings. false: Disables chain transfer settings (enables only a single transfer).
Return value	DMA_OK DTC transfer setting completed
	DMA_ERROR_PARAMETER Parameter error If one of the following conditions is detected, a parameter error will occur. • If the value specified for the DTC transfer trigger is illegal (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED or a greater value, or a value smaller than 0) • If the transfer count is set outside the range of 1 to 65536 in normal transfer settings • If the transfer count is set outside the range of 1 to 256 in repeat transfer settings • If the transfer count is set outside the range of 1 to 65536 in block transfer settings • If the block count is set outside the range of 1 to 256 in block transfer settings • If the transfer source or destination address is not a multiple of 2 when the transfer size is set to words (two bytes) • If the transfer source or destination address is not a multiple of 4 when the transfer size is set to longwords (four bytes) • If a value of 1 is specified in a reserved bit of the mode setting
	DMA_ERROR_MODE Mode error If one of the following conditions is detected, a mode error will occur. • If the offset definition (DMA_SRC_ADDR_OFFSET) is specified for the transfer source addressing mode • If the offset definition (DMA_DEST_ADDR_OFFSET) is specified for the transfer destination addressing mode • If "no repeat or block area" (DMA_REPEAT_BLOCK_NONE) is specified for the repeat or block area setting • If the value of the mode setting is illegal • If the value of the size setting is illegal • If the value of the repeat or block area setting is illegal • If the value of the chain transfer setting is illegal
Remarks	

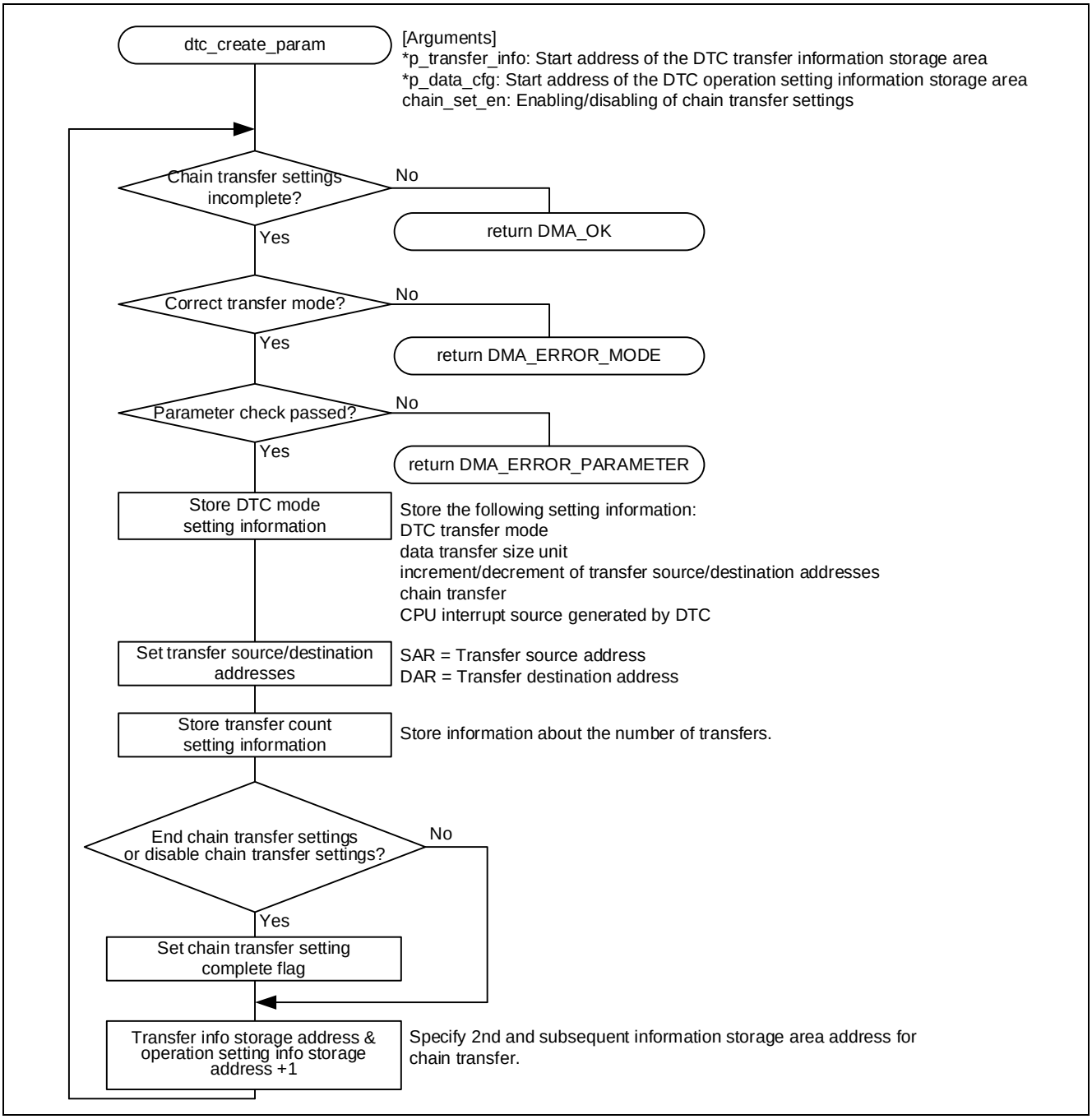


Figure 4-11 dtc_create_param Function Processing Flow

4.1.12 dtc_comp_interrupt Function

Table 4-13 dtc_comp_interrupt Function Specifications

Format	static void dtc_comp_interrupt(void)
Description	DTC_COMPLETE interrupt processing
Argument	None
Return value	None
Remarks	

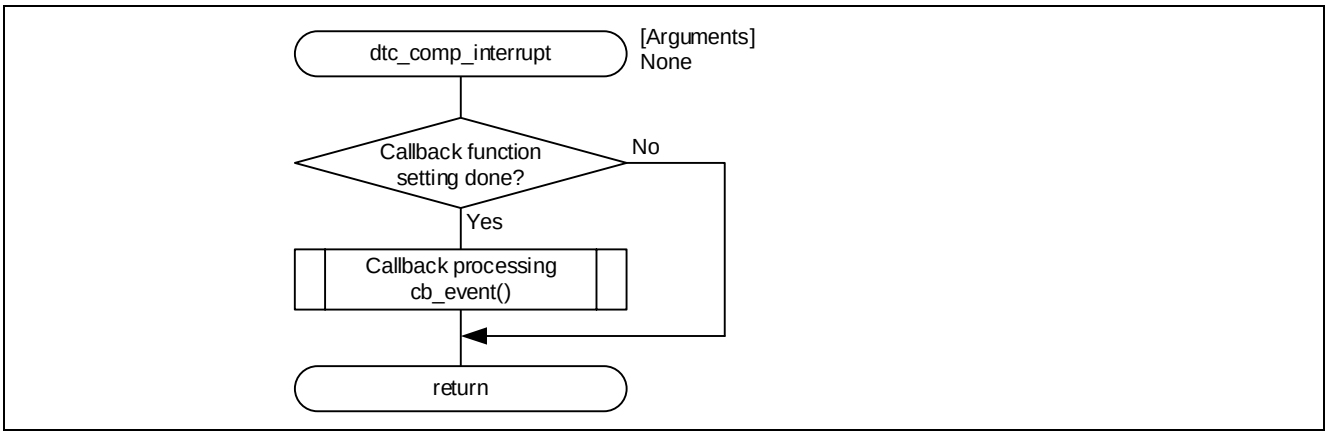


Figure 4-12 dtc_comp_interrupt Function Processing Flow

4.1.13 dtc_cmd_refresh Function

Table 4-14 dtc_cmd_refresh Function Specifications

Format	static e_dma_err_t dtc_cmd_refresh(st_dma_refresh_data_t const * const p_data)	
Description	Refreshes the transfer source and destination addresses and transfer count.	
Argument	st_dma_refresh_data_t const * const p_data: Refresh settings Specifies the new values to be set in the DTC.	
Return value	DMA_OK	Refresh completed
	DMA_ERROR	Refresh failed If one of the following conditions is detected, execution of the control command will fail. • If the DTC vector table for the specified transfer trigger is set to NULL (the Create function is not executed for the specified trigger) • If an illegal command is executed
	DMA_ERROR_PARAMETER	Parameter error If one of the following conditions is detected, a parameter error will occur. • If the transfer count is set outside the range of 1 to 65536 in normal transfer settings • If the transfer count is set outside the range of 1 to 256 in repeat transfer settings • If the transfer count is set outside the range of 1 to 65536 in block transfer settings • If the block count is set outside the range of 1 to 256 in block transfer settings
Remarks		

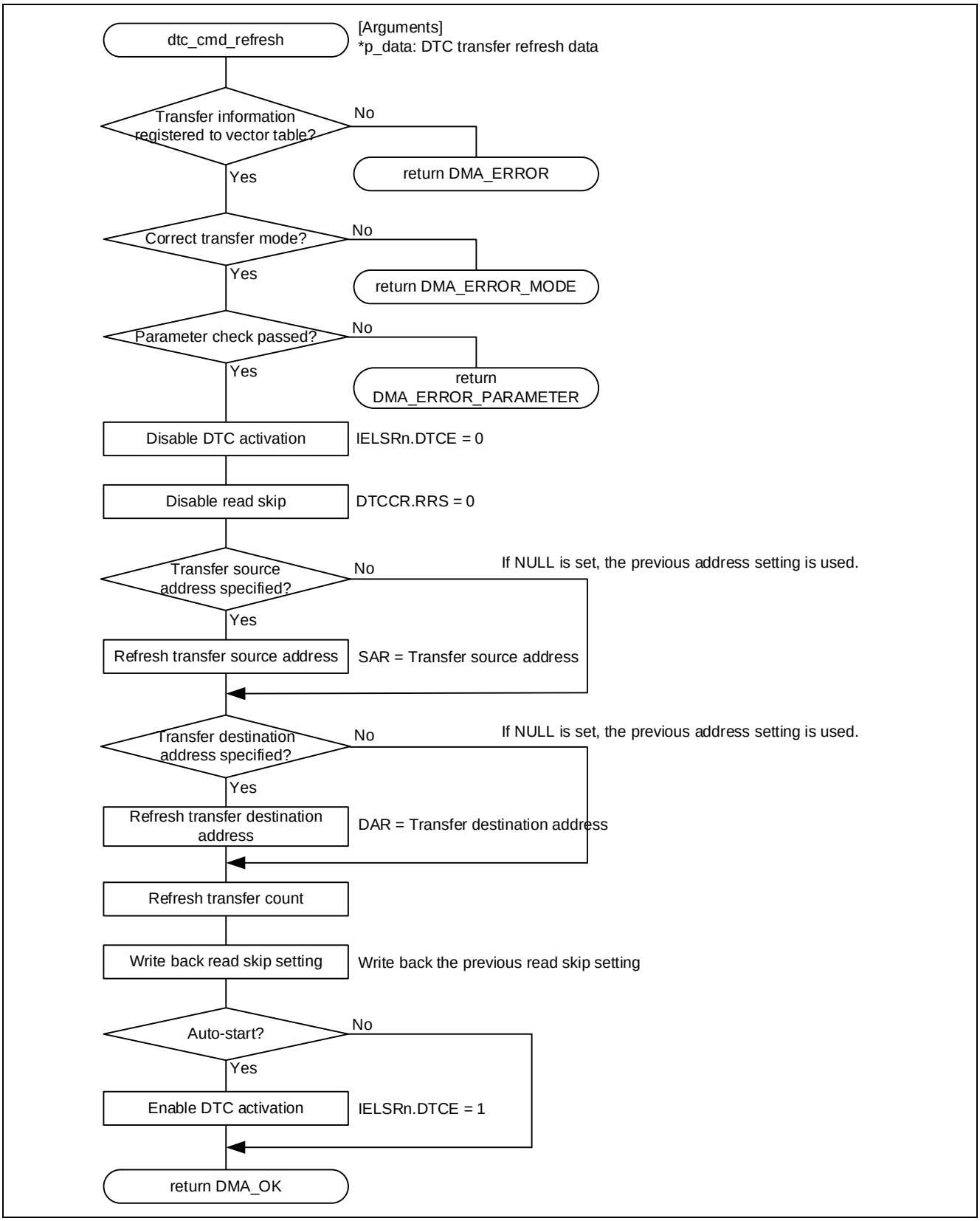


Figure 4-13 dtc_cmd_refresh Function Processing Flow

4.2 Macro and Type Definitions

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

4.2.1 Macro Definition List

Table 4-15 List of Macro Definitions

Definition	Value	Description
DMA_FLAG_OPENED	(1U << 0)	Open state flag definition
DMA_FLAG_INITIALIZED	(1U << 1)	Initialization complete flag definition
DMA_FLAG_CREATED	(1U << 2)	Setting complete flag definition
DTC_VERSION_MAJOR	(Note)	Major version information of DTC driver
DTC_VERSION_MINOR	(Note)	Minor version information of DTC driver
DMA_TRANSFER_MODE_MASK	(0xC000U)	Mask for operating mode judgment
DMA_SIZE_MASK	(0x3000U)	Mask for transfer size judgment
DMA_SRC_ADDRESSING_MASK	(0x0C00U)	Mask for transfer source addressing judgment
DMA_DEST_ADDRESSING_MASK	(0x000CU)	Mask for transfer destination addressing judgment
DMA_REPEAT_BLOCK_MASK	(0x0011U)	Mask for repeat or block area judgment
DMA_INT_SELECT_MASK	(0x0020U)	Mask for interrupt generation timing judgment
DMA_CHAIN_MODE_MASK	(0x00C0U)	Mask for chain mode judgment
DMA_INT_MASK	(0x001F)	Mask for interrupt source judgment
DMA_MODE_RASERVED	(0x0302)	Mask for unused mode area judgment
DMA_ACTIVE_DMAC0	(0x0001)	DMAC0 active
DMA_ACTIVE_DMAC1	(0x0002)	DMAC1 active
DMA_ACTIVE_DMAC2	(0x0004)	DMAC2 active
DMA_ACTIVE_DMAC3	(0x0008)	DMAC3 active
DMA_ACTIVE_DTC	(0x8000)	DTC active state
DMA_ACTIVE_MASK	(0x800F)	Mask for DMAC or DTC active state judgment
DMA_ACTIVE_DMAC_MASK	(0x000F)	Mask for DMAC active state judgment
DTC_PRV_CHAIN_ENABLE	(0x00800000)	Chain transfer enable/disable setting
DTC_PRV_MIN_COUNT_VAL	(1)	Minimum transfer count
DTC_PRV_MAX_16BITS_COUNT_VAL	(65536)	Maximum count of 16-bit transfers
DTC_PRV_MAX_8BITS_COUNT_VAL	(256)	Maximum count of 8-bit transfers
DTC_PRV_IELSR_DTCE	(0x01000000)	DTC enable/disable setting
DTC_PRV_COMPLETE_INT_IESR_VAL	(0x00000003)	Setting of DTC transfer complete interrupt as an ICU event link
DTC_PRV_MASK_MRA_MD	(0xC0000000)	MRA_MD mask definition
DTC_PRV_MRA_MD_NORMAL	(0x00000000)	Normal transfer judgment definition
DTC_PRV_MRA_MD_REPEAT	(0x40000000)	Repeat transfer judgment definition
DTC_PRV_MRA_MD_BLOCK	(0x80000000)	Block transfer judgment definition
DTC_PRV_MASK_MRA_SZ	(0x30000000)	MRA_SZ mask definition
DTC_PRV_MRA_SZ_BYTE	(0x00000000)	Transfer size 1 byte judgment definition
DTC_PRV_MRA_SZ_WORD	(0x10000000)	Transfer size 2 byte judgment definition
DTC_PRV_MRA_SZ_LONG	(0x20000000)	Transfer size 4 byte judgment definition

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

Example: Driver version 1.01

DTC_VERSION_MAJOR (1)

DTC_VERSION_MINOR (1)

4.3 Structure Definitions

4.3.1 st_dtc_resources_t Structure

This structure configures the resources of DTC.

Table 4-16 st_dtc_resources_t Structure

Element Name	Type	Description
*reg	volatile DTC_Type	Shows a target DTC register.
lock_id	e_system_mcu_lock_t	DTC lock ID
mstp_id	e_lpm_mstp_t	DTC module stop ID
*info	st_dtc_mode_info_t	DTC status information

4.3.2 st_dtc_mode_info_t Structure

This structure is used to manage the status of the DTC.

Table 4-17 st_dtc_mode_info_t Structure

Element Name	Type	Description
cb_event	dma_cb_event_t	Callback function to be executed when an event occurs. When this value is NULL, no callback function will be executed.
read_skip	uint8_t	Read skip setting 0: Read skip is disabled. 1: Read skip is enabled
flags	uint8_t	Driver setting flag Bit 0: Driver open state (0: Closed, 1: Opened) Bit 1: Driver initialization state (0: Uninitialized, 1: Initialized) Bit 2: DTC setting state (0: Not set, 1: Setting completed)

4.4 Calling External Functions

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

Table 4-18 External Functions Called from DTC Driver APIs and Calling Conditions

API	Functions Called	Conditions (Note 1)
Open	R_SYS_ResourceLock	None
	R_LPM_ModuleStart	None
	R_SYS_ResourceUnlock	Cancellation of the module stop state failed.
Close	R_NVIC_DisableIRQ	None
	R_SYS_IrqStatusClear	None
	R_NVIC_ClearPendingIRQ	None
	R_SYS_ResourceUnlock	None
	R_LPM_ModuleStop	No DMAC or DTC driver is used.
Create	-	-
Control	-	-
InterruptEnable	R_SYS_IrqEventLinkSet	None
	R_NVIC_SetPriority	None
	R_NVIC_GetPriority	None
	R_SYS_IrqStatusClear	None
	R_NVIC_ClearPendingIRQ	None
	R_NVIC_EnableIRQ	None
InterruptDisable	R_NVIC_DisableIRQ	None
	R_SYS_IrqStatusClear	None
	R_NVIC_ClearPendingIRQ	None
GetState	—	—
ClearState	—	—
GetTransferByte	—	—
GetVersion	—	—

Note 1. If operation terminates due to a parameter check error, the functions may not be called even when no condition is specified.

5. Usage Notes

5.1 Arguments

Initialize all elements of the structures used for the arguments of each function to 0 before use.

5.2 Registering DTC Transfer Complete Interrupt (DTC_COMPLETE) to NVIC

When using the DTC transfer complete interrupt (DTC_COMPLETE), register it to NVIC in r_system_cfg.h. If the DTC transfer complete interrupt is not registered in NVIC, DMA_ERROR_SYSTEM_SETTING will return when the DTC_InterruptEnable function is executed.

Figure 5-1 shows an example of registering the DTC transfer complete interrupt.

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

Figure 5-1 Example of Registering Interrupts to NVIC

5.3 Operation when DTC Transfer Trigger is Input while DTC Transfer is Disabled

If a DTC transfer trigger is input while the DTC transfer is disabled, an interrupt request of the transfer trigger is generated. If the interrupt is enabled in NVIC, DTC transfer is not performed and an interrupt occurs.

If the interrupt is disabled in NVIC, the interrupt request is held in NVIC. Even when DTC transfer is enabled, no DTC transfer request is accepted until the interrupt request in NVIC is cleared. In this case, enable the interrupt in NVIC to execute the NVIC interrupt processing, or clear the interrupt request in NVIC.

Operation example when a DTC transfer trigger is generated while DTC transfer is disabled:

[Sample program processing]

- DTC transfer is set up as follows:
  - Operating mode: Normal transfer
  - Transfer count: 2
  - Interrupt generation timing: An interrupt request is sent to the CPU when all specified data transfers are completed.
  - Transfer trigger: AGTI interrupt in AGT0
- The DTC settings are refreshed by using the DMA_CMD_REFRESH command in AGTI interrupt processing.

Figure 5-2 shows an operation example when an AGTI interrupt occurs before the DTC settings are refreshed by the DMA_CMD_REFRESH command in processing of another AGTI interrupt.

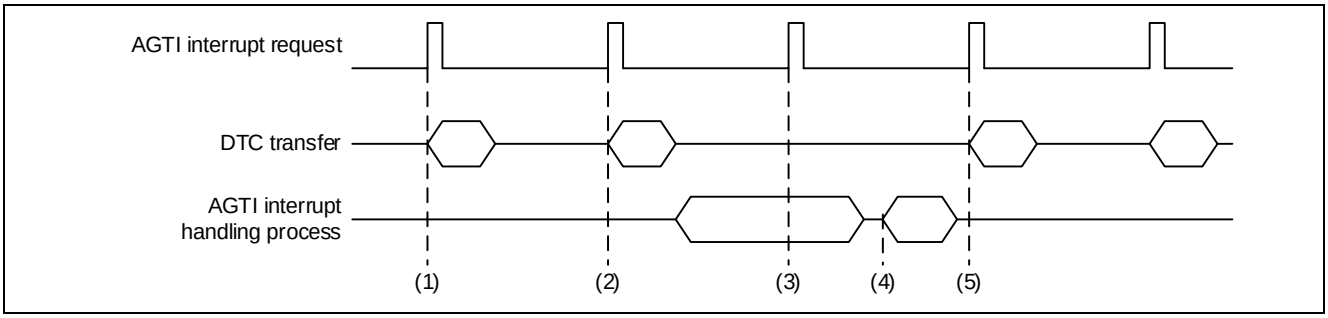


Figure 5-2 Operation Example when DTC Transfer Trigger is Generated while DTC Transfer is Disabled

(1) DTC transfer is started by an AGTI interrupt request.

(2) DTC transfer is started again by another AGTI interrupt request. After the specified count of DTC transfers are completed, the AGTI interrupt processing is executed.

(3) If another AGTI interrupt request is generated during the previous AGTI processing (before execution of the DMA_CMD_REFRESH command), the interrupt request is held in NVIC.

(4) After the AGTI interrupt processing is completed, AGTI interrupt processing is started again by the interrupt request held in (3).

(5) If an AGTI interrupt request is generated after the interrupt request in NVIC is cleared, DTC transfer is started.

If (4) behavior is not desired in the user system, clear the AGTI request after execution of the DMA_CMD_REFRESH command. The following shows an example of clearing the AGTI interrupt request.

```
/* AGT0 AGTI interrupt callback processing */
void cb_agt0_agti (void)
{
    st_dma_refresh_data_t arg;

/* Update the settings of transfer activated by the AGTI interrupt source and enable DTC transfer. */
    arg.act_src       = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
    arg.chain_transfer_nr = 0;                  /* Set to 0 because the chain transfer is not used. */
    arg.src_addr      = (uint32_t)(&source_ref); /* Update the transfer source address. */
    arg.dest_addr     = (uint32_t)(&dest_ref);   /* Update the transfer destination address. */
    arg.transfer_count  = 2;                    /* Update the transfer count. */
    arg.block_size     = 0;                /* Update the block size (valid only for block transfer). */
    arg.auto_start     = true;              /* After the refresh command execution, enable the DTC. */
    (void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);

    /* Clear the IR flag in the IELSRn register. */
    R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI);
    /* Clear the interrupt request in NVIC. */
    R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI);
}
```

Figure 5-3 Sample Procedure for Clearing the Interrupt Request Used as DTC Transfer Trigger

5.4 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.10.2019	—	First edition issued
1.01	Dec.16.2019	—	Compatible with 256KB group
1.02	Feb.18.2020	program (256KB, 1500KB)	Fixed incorrect RAM / ROM layout in r_dtc_cfg.h Added RAM allocation definition (DTC_CFG_R_DTC_GET_TRANSFER_BYTE) for R_DTC_GetTransferByte function

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/.