RE01 1500KB、256KB グループ

CMSIS ドライバ DTC 仕様書

要旨

本書では、RE01 1500KB、256KB グループ CMSIS software package の DTC ドライバ（以下、DTC ドライバ）の詳細仕様を説明します。

動作確認デバイス

RE01 1500KB グループ
RE01 256KB グループ

目次

1. 概要

DTC ドライバは、RE01 1500KB および 256KB グループで DTC 転送を行うためのドライバです。

2. ドライバ構成

本章では、本ドライバを使用するために必要な情報を記載します。

2.1 ファイル構成

DTC ドライバは CMSIS Driver Package の HAL_Driver に該当し、ベンダ独自ファイル格納ディレクトリ内の” r_dtc_api.c”、 ” r_dtc_api.h”、” r_dma_common_api.h”、” r_dtc_cfg.h”の 4 個のファイルで構成されます。各ファイルの役割を表 2-1 に、ファイル構成を図 2-1 に示します。

表 2-1 R_DTC ドライバ 各ファイルの役割

ファイル名	内容
r_dtc_api.c	ドライバソースファイルです ドライバ関数の実態を用意します DTC ドライバを使用する場合は、本ファイルをビルドする必要があります
r_dtc_api.h	ドライバヘッダファイルです ドライバ内で使用するマクロ／型／プロトタイプ宣言が定義されています DTC ドライバを使用する場合は、本ファイルをインクルードする必要があります
r_dma_common_api.h	DMAC/DTC 共通のドライバヘッダファイルです DMAC と DTC で共通して使用するマクロ／型が定義されています
r_dtc_cfg.h	コンフィグレーション定義ファイルです ユーザが設定可能なコンフィグレーション定義を用意します

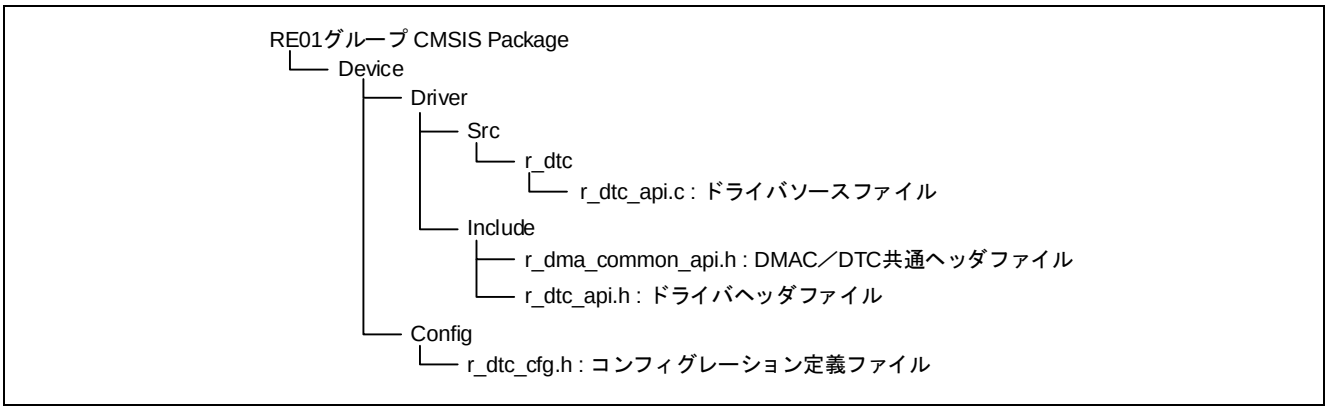

図 2-1 DTC ドライバファイル構成

2.2 ドライバ API

DTC ドライバはインスタンスを用意しています。ドライバを使用する場合は、インスタンス内の関数ポインタを使用して API にアクセスしてください。DTC ドライバのインスタンスを表 2-2 に、インスタンスの宣言例を図 2-2 に、インスタンスに含まれる API を表 2-3 に、DTC ドライバへのアクセス例を図 2-3～図 2-6 に示します。

表 2-2 DTC ドライバのインスタンス一覧

インスタンス	内容
DRIVER_DMA Driver_DTC	DTC を使用する場合のインスタンス

```
#include "r_dtc_api.h"

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

図 2-2 DTC ドライバ インスタンス宣言例

表 2-3 DTC ドライバ API

API	内容	参照
Open	DTC ドライバのオープン（モジュールストップの解除、RAM の初期化）を行います	4.1.1
Close	DTC ドライバを解放（レジスタの初期化、割り込みの禁止）します 全 DMAC、DTC ドライバが未使用になった場合、モジュールストップ状態への遷移も行います	4.1.2
Create	DTC 転送の設定を行います	4.1.3
Control	DTC の制御コマンドを実行します 制御コマンドについては表 2-4 を参照	4.1.4
InterruptEnable	DTC ドライバの転送完了割り込み（DTC_COMPLETE）を設定します 割り込み要因に DTC 転送完了割り込みを指定した場合、割り込み要因の設定と割り込み許可も行います	4.1.5
InterruptDisable	DTC ドライバの転送完了割り込み（DTC_COMPLETE）を禁止します	4.1.6
GetState	DTC の状態を取得します	4.1.7
ClearState	DTC ドライバの割り込みフラグをクリアします	4.1.8
GetTransferByte	DTC 転送バイト数を取得します	4.1.9
GetVersion	DTC ドライバのバージョンを取得します	4.1.10

表 2-4 DTC ドライバで使用可能な制御コマンド一覧

コマンド	内容
DMA_CMD_START	DTC 転送を開始します
DMA_CMD_STOP	DTC 転送を終了します
DMA_CMD_ACT_SRC_ENABLE	DTC 起動要因による DTC 起動を許可します
DMA_CMD_ACT_SRC_DISABLE	DTC 起動要因による DTC 起動を禁止します
DMA_CMD_DATA_READ_SKIP_ENABLE	リードスキップ機能の許可／禁止を設定します
DMA_CMD_CHAIN_TRANSFER_ABORT	DTC チェーン転送を中断します
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	DTC 転送情報を変更します
DMA_CMD_REFRESH_DATA	DTC 転送設定を再設定します

```
#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 転送情報格納領域 */
uint16_t src_data[5] = {0x0000, 0x1111, 0x2222, 0x4444, 0x8888 };
uint16_t dest_data= 0xFFFF;

main()
{
    st_dma_transfer_data_cfg_t config;   /* DMA 転送設定情報格納領域 */

    /* ノーマルモード、16 ビット、転送元インクリメント、転送先固定  */
    /* すべての転送完了時に割り込み発生、チェーン転送無効          */
    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]);    /* 転送元設定(RAM) */
    config.dest_addr       = (uint32_t)(&dest_data);
    config.transfer_count  = 5;                      /* 転送回数 5 回 */
    config.p_transfer_data = &transfer_info;          /* DTC 転送情報格納場所 */

    (void)dtcDrv->Open();                             /* DTC ドライバオープン */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                   /* TMR CMIA0 を起動要因に設定 */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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;  /* DTC 転送要因(TMR)の起動 */
    while(1);
}

/*****************************************************************************************************
* callback function
*****************************************************************************************************/
void cb_dtc_complete(void)
{
    /* すべての DTC 転送完了時にコールバック関数実行 */
}
```

図 2-3 DTC ドライバへのアクセス例（ノーマル転送）

```
#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 転送情報格納領域 */
uint16_t src_data[5] = {0x0000, 0x1111, 0x2222, 0x4444, 0x8888 };
uint16_t dest_data= 0xFFFF;

main()
{
    st_dma_transfer_data_cfg_t config;   /* DMA 転送設定情報格納領域 */

    /* リピート転送モード、16 ビット、転送元インクリメント、転送先固定  */
    /* 転送元がリピート領域、すべての転送完了時に割り込み発生、チェーン転送無効          */
    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;                    /* 転送回数 5 回 */

    config.p_transfer_data = &transfer_info;         /* DTC 転送情報格納場所 */

    (void)dtcDrv->Open();                            /* DTC ドライバオープン */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                  /* TMR CMIA0 を起動要因に設定 */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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;  /* DTC 転送要因(TMR)の起動 */
    while(1);
}

/***********************************************************************************************************
* callback function
***********************************************************************************************************/
void cb_dtc_complete(void)
{
    /* リピート転送ではコールバック関数は発生しません */
}
```

図 2-4 DTC ドライバへのアクセス例（リピート転送）

```
#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 転送情報格納領域 */
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 転送設定情報格納領域 */

    /* ブロック転送モード、8 ビット、転送元インクリメント、転送先インクリメント      */
    /* 転送先がブロック領域、すべての転送完了時に割り込み発生、チェーン転送無効      */
    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]);    /* 転送元設定(RAM) */
    config.dest_addr       = (uint32_t)(&dest_data[0]);     /* 転送先設定(RAM) */
    config.transfer_count   = 3;                          /* 転送回数 3 回 */
    config.block_size      = 5;                          /* ブロック数 5 */
    config.p_transfer_data = &transfer_info;               /* DTC 転送情報格納場所 */

    (void)dtcDrv->Open();                              /* DTC ドライバオープン */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                                  /* TMR CMIA0 を起動要因に設定 */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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;  /* DTC 転送要因(TMR)の起動 */
    while(1);
}

/******************************************************************************
* callback function
******************************************************************************/
void cb_dtc_complete(void)
{
    /* すべての DTC 転送完了(ブロック数 5×転送回数 3 回)時にコールバック関数実行 */
}
```

図 2-5 DTC ドライバへのアクセス例（ブロック転送）

```
#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 転送情報格納領域(チェーン転送 2 回分) */
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 転送設定情報格納領域(チェーン転送 2 回分) */

    /* [チェーン 1 回目]ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定    */
    /* すべての転送完了時に割り込み発生、連続してチェーン転送を行う                      */
    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];

    /* [チェーン 2 回目]ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定    */
    /* すべての転送完了時に割り込み発生、チェーン転送無効                              */
    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();                           /* DTC ドライバオープン */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config[0]);
                                                /* TMR CMIA0 を起動要因に設定 */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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;  /* DTC 転送要因(TMR)の起動 */
    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
void cb_dtc_complete(void)
{
    /* チェーン 2 回目のすべての DTC 転送完了時にコールバック関数実行 */
}
```

図 2-6 DTC ドライバへのアクセス例（チェーン転送）

2.3 DTC 転送要因の設定と割り込み

DTC の転送要因に使用する割り込みは、r_system_cfg.h にて NVIC の割り込み設定を行ったうえで、使用する周辺機能の割り込み定義名（IRQ0 の場合は SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0）を Create 関数の第 1 引数に指定します。r_system_cfg.h での NVIC 割り込み設定については、「RE01 1500KB、256KB グループ CMSIS パッケージを用いた開発スタートアップガイド（r01an4660）」の「割り込み（NVIC）の設定」を参照ください。

DTC 転送を許可にしたのち、DTC 転送要因を入力すると DTC 転送が実行されます。指定した回数の DTC 転送を終了したとき、CPU に対する割り込みが発生します（注 1、注 2）。CPU に対する割り込みは DTC 転送要因(チャネル毎)および DTC 転送完了割り込み(DTC_COMPLETE：全チャネル共通)となります（注 3、注 4）。DTC 転送完了割り込みを使用する場合は、InterruptEnable 関数を使用して割り込みを許可にしてください。また、r_system_cfg.h にて NVIC に登録する必要があります。

TMR の CMIA0 割り込みを要因とし、DTC 転送完了割り込みを使用する場合の NVIC の登録例を図 2-7 に、設定手順例を図 2-8 に示します。

注1. Create 関数にて DMA_INT_PER_SINGLE_TRANSFER を指定していた場合、DTC データ転送ごとに DMA_INT_AFTER_ALL_COMPLETE を指定した場合、指定されたデータ転送の終了時に CPU に対する割り込み要求が発生します。

注2. DMA_CHAIN_NORMAL（転送カウンタが 1→0、または 1→CRAH となったとき、チェーン転送を行う）を指定していた場合、CPU に対する割り込みは発生しません。

注3. DTC にて InterruptEnable 関数で指定できる割り込み要因は DMA_INT_COMPLETE のみです。InterruptEnable 関数の第 1 引数に DMA_INT_COMPLETE 以外を設定するとエラーを返します。

注4. DTC 転送要因(チャネル毎)による割り込みでは、転送要因の割り込み設定時に指定したコールバック関数（R_SYS_IrqEventLinkSet 関数で設定したコールバック関数）が呼び出されます。DTC 転送完了による割り込みでは、InterruptEnable 関数で設定したコールバック関数が呼び出されます。

```
・・・

#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT
     (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)

/* DTC 転送完了割り込みを NVIC に登録 */

#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

/* DTC 転送要因(TMR_CMIA0)を NVIC に登録 */

#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

・・・
```

図 2-7 r_system_cfg.h での NVIC への割り込み登録例（CMIA0 と DTC 転送完了割り込みの登録）

```
/*********************************************************/
/* DTC 転送要因に使用する周辺機能の初期化    */
/*********************************************************/
R_LPM_ModuleStart(LPM_MSTP_TMR);
TMR0->TCORA = 100-1;   /* タイマ設定値：100-1 */
TMR0->TCR   = 0x48;    /* CMIA0 割り込み許可、コンペアマッチ A によりカウンタクリア */

/*********************************************************/
/* DTC 転送設定                                */
/*********************************************************/
/* ノーマルモード、16 ビット、転送元インクリメント、転送先固定  */
/* すべての転送完了時に割り込み発生、チェーン転送無効          */
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]);    /* 転送元設定(RAM) */
config.dest_addr       = (uint32_t)(&dest_data);
config.transfer_count  = 5;                     /* 転送回数 5 回 */
config.p_transfer_data = &transfer_info;         /* DTC 転送情報格納場所 */

(void)dtcDrv->Open();                            /* DTC ドライバオープン */
(void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
                                               /* TMR CMIA0 を起動要因に設定 */
(void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */
(void)dtcDrv->EnableInterrupt(DMA_INT_COMPLETE, cb_dtc_complete);

/*************************************************************************************/
/* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)  */
/*************************************************************************************/
/* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
R_SYS_IrqEventLinkSet(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 0x14, cb_tmr_cmia0);
R_NVIC_SetPriority(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, 3);
R_NVIC_EnableIRQ(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0);

/*************************************/
/* DTC 転送要因(TMR)の起動  */
/*************************************/
TMR0->TCCR = 0x0E;  /* PCLK/8192 でカウント開始 */
```

図 2-8 DTC 転送完了割り込み設定手順例

2.4 マクロ／型定義

DMAC/DTC ドライバで、ユーザが参照可能なマクロ／型定義を r_dma_common_api.h ファイルで定義しています。

2.4.1 DTC コントロールコード定義

DTC コントロールコードは、Control 関数で使用する DTC 制御コマンドです。

表 2-5 DTC コントロールコード一覧

定義	値	内容
DMA_CMD_START	(0x00)	DTC 転送開始コマンド
DMA_CMD_STOP	(0x01)	DTC 転送停止コマンド
DMA_CMD_ACT_SRC_ENABLE	(0x02)	DTC 起動要因許可コマンド
DMA_CMD_ACT_SRC_DISABLE	(0x03)	DTC 起動要因禁止コマンド
DMA_CMD_DATA_READ_SKIP_ENABLE	(0x04)	リードスキップ設定コマンド
DMA_CMD_CHAIN_TRANSFER_ABORT	(0x05)	DTC チェーン転送中断コマンド
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	(0x06)	DTC 転送情報強制変更コマンド
DMA_CMD_SOFTWARE_TRIGGER	(0x07)	使用禁止(注)
DMA_CMD_AUTO_CLEAR_SOFT_REQ	(0x08)	使用禁止(注)
DMA_CMD_REFRESH_DATA	(0x09)	DTC 転送再設定コマンド
DMA_CMD_REFRESH_EXTRA	(0x0A)	使用禁止(注)

注　DMAC ドライバでのみ使用可。Control 関数で本定義を指定した場合、DMA_ERROR を返します。

2.4.2 DTC モード定義

DTC モード定義は、st_dma_transfer_data_cfg_t 構造体の mode 要素に設定する定義です。Create 関数の第 2 引数、または Control 関数の DTC 転送情報強制変更コマンド(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)使用時の引数に使用します。転送モードを設定する場合は、他のビット(b0～b14)で転送サイズ、転送元アドレッシング、リピート/ブロック領域、転送先アドレッシング、割り込み発生タイミング、チェーン転送設定も設定してください。

例）リピート転送、BYTE サイズ転送、転送元インクリメント、転送先固定、転送元をリピート領域、
指定されたデータ転送の終了時、CPU への割り込み要求が発生、チェーン転送しない設定の場合

```
st_dma_transfer_data_cfg_t config;   /* DMA 転送設定情報格納領域 */

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;
```

DTC モード定義の構成を図 2-9 に、各設定内容の詳細を表 2-6～表 2-12 に示します。

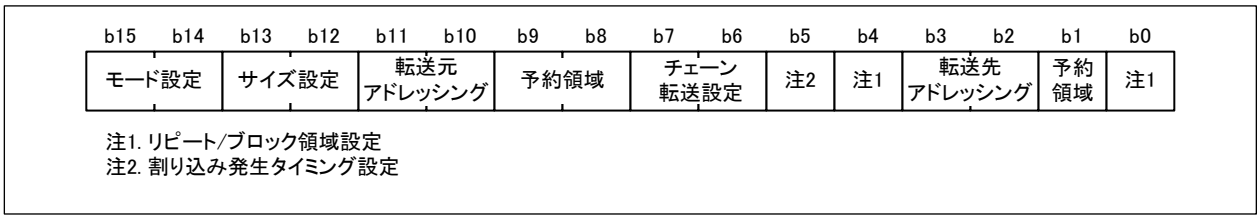

図 2-9 DTC モード定義の構成

表 2-6 モード設定定義一覧

定義	値	内容
DMA_MODE_NORMAL	(0x0000U)	ノーマル転送
DMA_MODE_REPEAT	(0x4000U)	リピート転送
DMA_MODE_BLOCK	(0x8000U)	ブロック転送

表 2-7 サイズ設定定義一覧

定義	値	内容
DMA_SIZE_BYTE	(0x0000U)	8 ビットサイズ転送
DMA_SIZE_WORD	(0x1000U)	16 ビットサイズ転送
DMA_SIZE_LONG	(0x2000U)	32 ビットサイズ転送

表 2-8 転送元アドレッシングモード設定定義一覧

定義	値	内容
DMA_SRC_FIXED	(0x0000U)	転送元アドレス固定
DMA_SRC_INCR	(0x0800U)	転送後アドレスをインクリメント
DMA_SRC_DECR	(0x0C00U)	転送後アドレスをデクリメント
DMA_SRC_ADDR_OFFSET	(0x0400U)	使用禁止(注 1)

注1. DMAC ドライバでのみ使用可。Create 関数、Control 関数で本定義を指定した場合、DMA_ERROR_MODE を返します。

表 2-9 転送先アドレッシングモード設定定義一覧

定義	値	内容
DMA_DEST_FIXED	(0x0000U)	転送先アドレス固定
DMA_DEST_INCR	(0x0008U)	転送後アドレスをインクリメント
DMA_DEST_DECR	(0x000CU)	転送後アドレスをデクリメント
DMA_DEST_ADDR_OFFSET	(0x0004U)	使用禁止(注 2)

注2. DMAC ドライバでのみ使用可。Create 関数、Control 関数で本定義を指定した場合、DMA_ERROR_MODE を返します。

表 2-10 リピート/ブロック領域設定定義一覧

定義	値	内容
DMA_REPEAT_BLOCK_DEST	(0x0000U)	転送先領域をリピート/ブロック領域に設定する
DMA_REPEAT_BLOCK_SRC	(0x0010U)	転送元領域をリピート/ブロック領域に設定する
DMA_REPEAT_BLOCK_NONE	(0x0001U)	使用禁止(注 3)

注3. DMAC ドライバでのみ使用可。Create 関数、Control 関数で本定義を指定した場合、DMA_ERROR_MODE を返します。

表 2-11 割り込み発生タイミング設定定義一覧

定義	値	内容
DMA_INT_AFTER_ALL_COMPLETE	(0x0000U)	指定されたデータ転送の終了時、CPU への割り込み要求が発生
DMA_INT_PER_SINGLE_TRANSFER	(0x0020U)	DTC データ転送のたびに、CPU への割り込み要求が発生

表 2-12 チェーン転送設定定義一覧

定義	値	内容
DMA_CHAIN_DISABLE	(0x0000U)	チェーン転送禁止
DMA_CHAIN_CONTINUOUSLY	(0x0080U)	連続してチェーン転送を行う
DMA_CHAIN_NORMAL	(0x00C0U)	転送が終了したタイミングでチェーン転送を開始する

2.4.3 DTC 割り込み要因定義

InterruptEnable 関数で指定する割り込み要因の定義です。

表 2-13 DTC 割り込み要因定義一覧

定義	値	内容
DMA_INT_COMPLETE	(1U<<0)	DTC 転送終了
DMA_INT_SRC_OVERFLOW	(1U<<1)	使用禁止(注 4)
DMA_INT_DEST_OVERFLOW	(1U<<2)	使用禁止(注 4)
DMA_INT_REPEAT_END	(1U<<3)	使用禁止(注 4)
DMA_INT_ESCAPE_END	(1U<<4)	使用禁止(注 4)

注4. DMAC ドライバでのみ使用可。InterruptEnable 関数で本定義を指定した場合、DMA_ERROR を返します。

2.5 構造体定義

DTC ドライバでは、ユーザが参照可能な構造体定義を r_dma_common_api.h ファイルで定義しています。

2.5.1 st_dma_transfer_data_cfg_t 構造体

Create 関数で DTC の転送設定情報を指定するときに使用する構造体です。

表 2-14 st_dma_transfer_data_cfg_t 構造体

要素名	型	内容
mode	uint16_t	転送モード、転送サイズ、転送元アドレスモード、転送先アドレスモード、リピート領域、チェーン転送設定を、OR 演算子を使用して指定します
src_addr	uint32_t	転送元アドレスを指定します
dest_addr	uint32_t	転送先アドレスを指定します
transfer_count	uint32_t	転送回数を指定します
block_size	uint32_t	ブロック転送サイズを指定します
offset	int32_t	使用禁止(注 1)
src_extended_repeat	uint8_t	使用禁止(注 1)
dest_extended_repeat	uint8_t	使用禁止(注 1)
*p_transfer_data	void	転送情報を格納するアドレスを指定します

注1. DMAC ドライバでのみ使用可。Create 関数では無視されます。

2.5.2 st_dma_refresh_data_t 構造体

Control 関数で DTC 転送再設定コマンド(DMA_CMD_REFRESH_DATA)を指定するときに使用する構造体です。

表 2-15 st_dma_refresh_data_t 構造体

要素名	型	内容
act_src	IRQn_Type	DTC 起動要因を指定します
chain_transfer_nr	uint32_t	更新対象のチェーン番号-1 を指定します チェーン転送を使用していない場合は 0 を指定してください
src_addr	uint32_t	転送元アドレスを指定します
dest_addr	uint32_t	転送先アドレスを指定します
transfer_count	uint32_t	転送回数を指定します
block_size	uint32_t	ブロック転送サイズを指定します
auto_start	bool	オートスタートの有無を指定します 0：オートスタートなし 1：オートスタートあり
keep_dma_req	bool	使用禁止(注 2)
offset	bool	使用禁止(注 2)

注2. DMAC ドライバでのみ使用可。Control 関数では無視されます。

2.5.3 st_dma_state_t 構造体

GetState 関数で取得した転送状態を格納するための構造体です。

表 2-16 st_dma_state_t 構造体

要素名	型	内容
in_progress	bool	DTC 転送のアクティブ状態を示します(注 1)
esif_stat	bool	転送エスケープ終了割り込みフラグを示します(注 2)
dtif_stat	bool	転送終了割り込みフラグを示します(注 2)
soft_req_stat	bool	ソフトウェアトリガによる転送要求フラグを示します(注 2)

注1. GetState 関数では必ず 0 を返します。

注2. DMAC ドライバでのみ使用可。GetState 関数では必ず 0 を返します。

2.5.4 st_dma_transfer_data_t 構造体

DTC 転送情報を格納するための構造体です。

表 2-17 st_dma_transfer_data_t 構造体

要素名	型	内容
mra_mrb	uint32_t	MRA／MRB レジスタ設定値
sar	uint32_t	SAR レジスタ設定値
dar	uint32_t	DAR レジスタ設定値
cra_crb	uint32_t	CRA／CRB レジスタ設定値

2.5.5 st_dma_chain_abort_t 構造体

Control 関数で DTC チェーン転送中断コマンド(DMA_CMD_CHAIN_TRANSFER_ABORT)を指定するときに使用する構造体です。

表 2-18st_dma_chain_abort_t 構造体

要素名	型	内容
act_src	IRQn_Type	DTC 起動要因を指定します
chain_transfer_nr	uint32_t	中断対象の総数を指定します

2.5.6 st_dma_chg_data_t 構造体

Control 関数で DTC 転送情報強制変更コマンド(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)を指定するときに使用する構造体です。

表 2-19 st_dma_chg_data_t 構造体

要素名	型	内容
act_src	IRQn_Type	DTC 起動要因を指定します
chain_transfer_nr	uint32_t	変更対象のチェーン番号-1 を指定します チェーン転送を使用していない場合は 0 を指定してください
cfg_data	st_dma_transfer_data_cfg_t	DTC 転送設定情報を指定します

2.6 状態遷移

DTC ドライバの状態遷移図を図 2-10 に、各状態でのイベント動作を表 2-20～表 2-21 に示します。

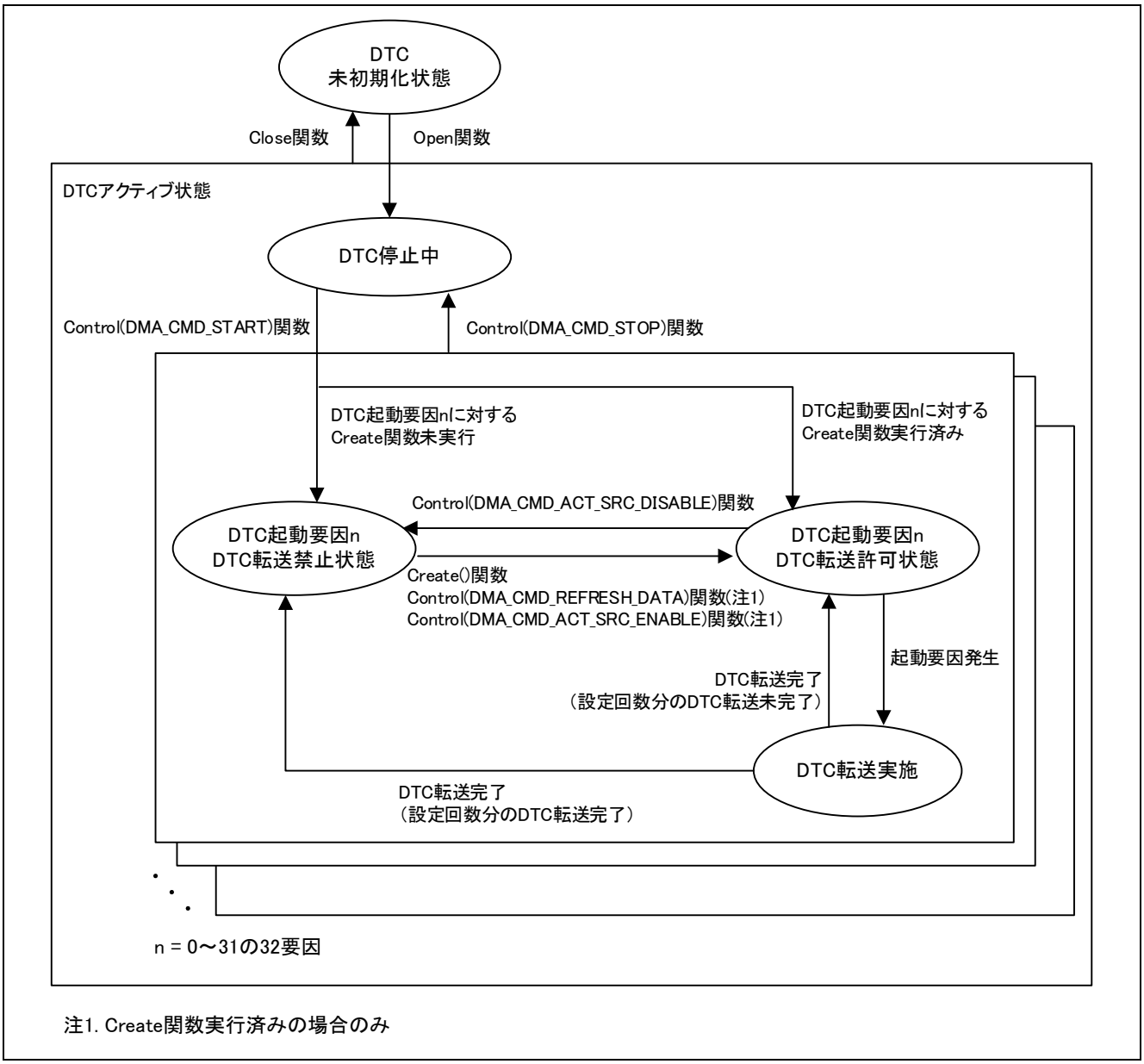

図 2-10 DTC ドライバの状態遷移

表 2-20 DTC ドライバ状態でのイベント動作(1/2)(注)

状態	概要	イベント	アクション
DTC 未初期化状態	リセット解除後の DTC ドライバの状態です	Open 関数の実行	DTC 停止状態に遷移
DTC 停止状態	DTC モジュール停止状態(DTCST.DTCST=0)です	Create 関数の実行	DTC 転送要因を設定(DTC 転送要因許可)
		Control(DMA_CMD_START)関数の実行	[Create 関数未実行の場合] DTC 転送禁止状態に遷移 [Create 関数実行済みの場合] DTC 転送許可状態に遷移
		Control(DMA_CMD_DATA_READ_SKIP_ENABLE)関数の実行	リードスキップの許可／禁止状態の切り替え
		Control(DMA_CMD_ACT_SRC_ENABLE)/ Control(DMA_CMD_ACT_SRC_DISABLE)関数の実行	対象の DTC 転送要因を許可/禁止
		Control(DMA_CMD_CHAIN_TRANSFER_ABORT)関数の実行	チェーン転送設定を解除
		Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)関数の実行	DTC 転送情報を強制的に書き換えます
		Control(DMA_CMD_REFRESH_DATA)関数の実行	DTC 転送情報(転送元、転送先、転送回数)を更新します
		InterruptEnable / InterruptDisable 関数の実行	DTC 転送完了割り込みを許可/禁止にします
		GetTransferByte 関数の実行	DTC 転送バイト数を取得します
		Close 関数の実行	DTC 未初期化状態に遷移
DTC 転送禁止状態	DTC モジュール動作状態(DTCST.DTCST=1)、DTC 転送要因の起動禁止状態 （IELSRn.DTE=0）	Create 関数の実行	DTC 転送要因を設定し､DTC 転送許可状態に遷移
		Control(DMA_CMD_STOP)関数の実行	DTC 停止状態に遷移
		Control(DMA_CMD_DATA_READ_SKIP_ENABLE)関数の実行	リードスキップの許可／禁止状態の切り替え
		Control(DMA_CMD_ACT_SRC_ENABLE)	(Create 関数実行済みの場合のみ) DTC 転送許可状態に遷移
		Control(DMA_CMD_CHAIN_TRANSFER_ABORT)関数の実行	チェーン転送設定を解除
		Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)関数の実行	DTC 転送情報を強制的に書き換えます
		Control(DMA_CMD_REFRESH_DATA)関数の実行	DTC 転送情報(転送元、転送先、転送回数)を更新します auto_start メンバが true (オートスタートする)の場合､DTC 転送許可状態に遷移
		InterruptEnable / InterruptDisable 関数の実行	DTC 転送完了割り込みを許可/禁止にします
		GetTransferByte 関数の実行	DTC 転送バイト数を取得します
		Close 関数の実行	DTC 未初期化状態に遷移

注　GetVersion、 GetState、 ClearState 関数はすべての状態で実行可能です。

表 2-21 DTC ドライバ状態でのイベント動作(2/2)(注)

状態	概要	イベント	アクション
DTC 転送許可状態	DTC モジュール動作状態(DTCST.DTCST=1)、DTC 転送要因の起動許可状態（IELSRn.DTE=1）	Create 関数の実行	DTC 転送要因を再設定
		DTC 転送要因の発生	DTC 転送状態に遷移
		Control(DMA_CMD_STOP)関数の実行	DTC 停止状態に遷移
		Control(DMA_CMD_DATA_READ_SKIP_ENABLE)関数の実行	リードスキップの許可／禁止状態の切り替え
		Control(DMA_CMD_ACT_SRC_DISABLE)関数の実行	DTC 転送禁止状態に遷移
		Control(DMA_CMD_CHAIN_TRANSFER_ABORT)関数の実行	チェーン転送設定を解除
		Control(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)関数の実行	DTC 転送情報を強制的に書き換えます
		Control(DMA_CMD_REFRESH_DATA)関数の実行	DTC 転送情報(転送元、転送先、転送回数)を更新します auto_start メンバが false (オートスタートしない)の場合､DTC 転送禁止状態に遷移
		InterruptEnable/ InterruptDisable 関数の実行	DTC 転送完了割り込みを許可/禁止にします
		GetTransferByte 関数の実行	DTC 転送バイト数を取得します
		Close 関数の実行	DTC 未初期化状態に遷移
DTC 転送状態	DTC 転送実行中	DTC 転送完了（指定回数分の DTC 転送未完了）	DTC 転送許可状態に遷移
		DTC 転送完了（指定回数分の DTC 転送完了）	DTC 転送禁止状態に遷移

注　GetVersion、 GetState、 ClearState 関数はすべての状態で実行可能です。

3. ドライバ動作説明

DTC ドライバは DTC によるデータ転送を実現します。本章では各動作モードにおける実行手順と実行例を示します。

3.1 ノーマル転送モード

ノーマル転送モードでは 1 つの起動要因で、8 ビット、16 ビット、32 ビットのデータ転送を行います。転送回数は 1～65536 回まで設定できます。ノーマル転送手順を図 3-1 に示します。

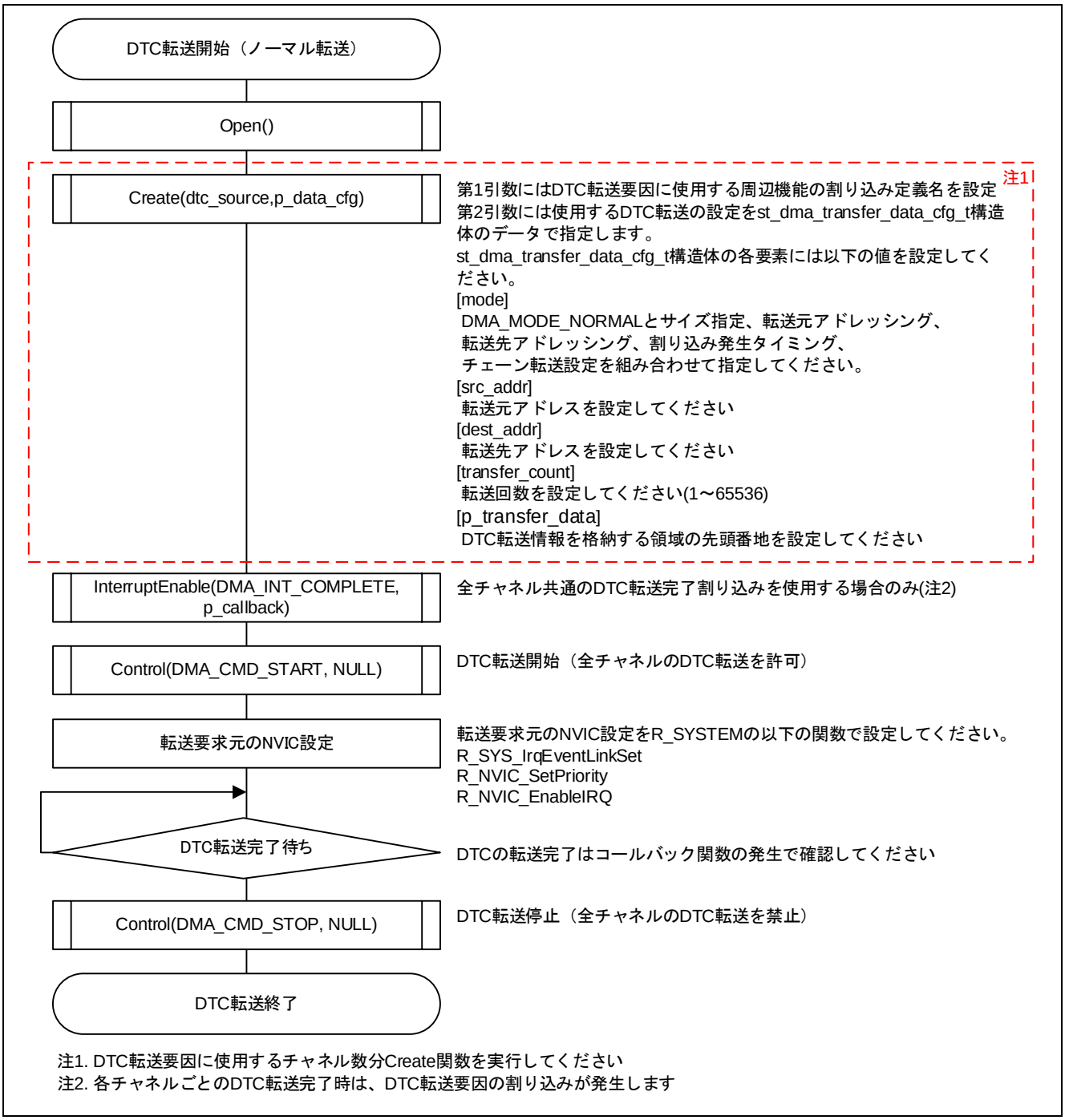

図 3-1 ノーマル転送手順

3.2 リピート転送モード

リピート転送モードでは 1 つの起動要因で、8 ビット、16 ビット、32 ビットのデータ転送を行います。転送回数は 1～256 回まで設定できます。割り込み発生タイミングを DMA_INT_AFTER_ALL_COMPLETE（指定されたデータ転送の終了時、CPU への割り込み要求が発生）にした場合、割り込みは発生しません。リピート転送手順を図 3-2 に示します。

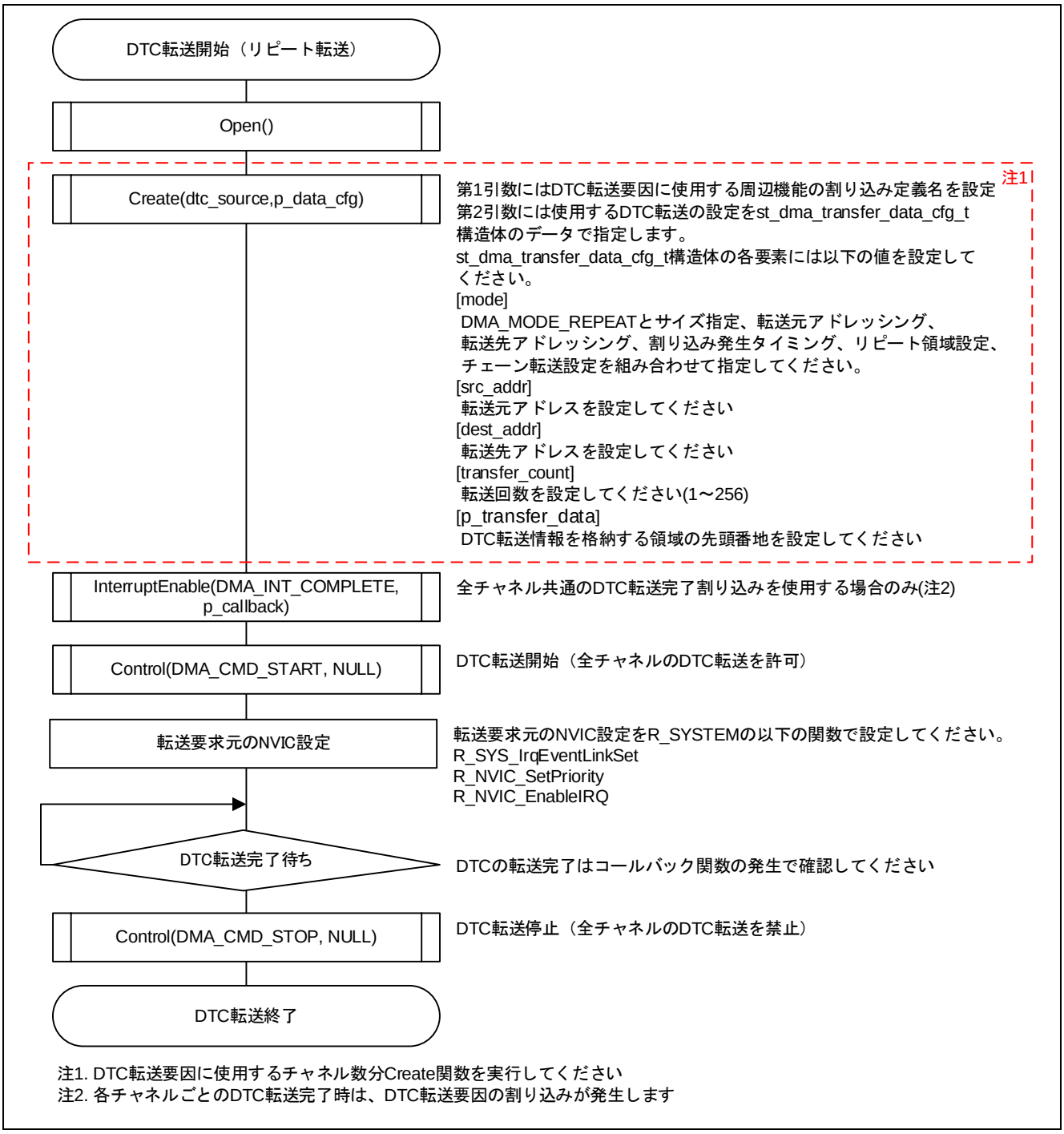

図 3-2 リピート転送手順

3.3 ブロック転送モード

ブロック転送モードでは 1 つの起動要因で、1 ブロックのデータ転送を行います。ブロックサイズは 1～256 バイト（8 ビットサイズ転送設定時）、2～512 バイト（16 ビットサイズ転送設定時）、4 バイト～1024 バイト（32 ビットサイズ転送設定時）が設定できます。転送回数（ブロック数）は 1～65536 まで設定できます。ブロック転送手順を図 3-3 に示します。

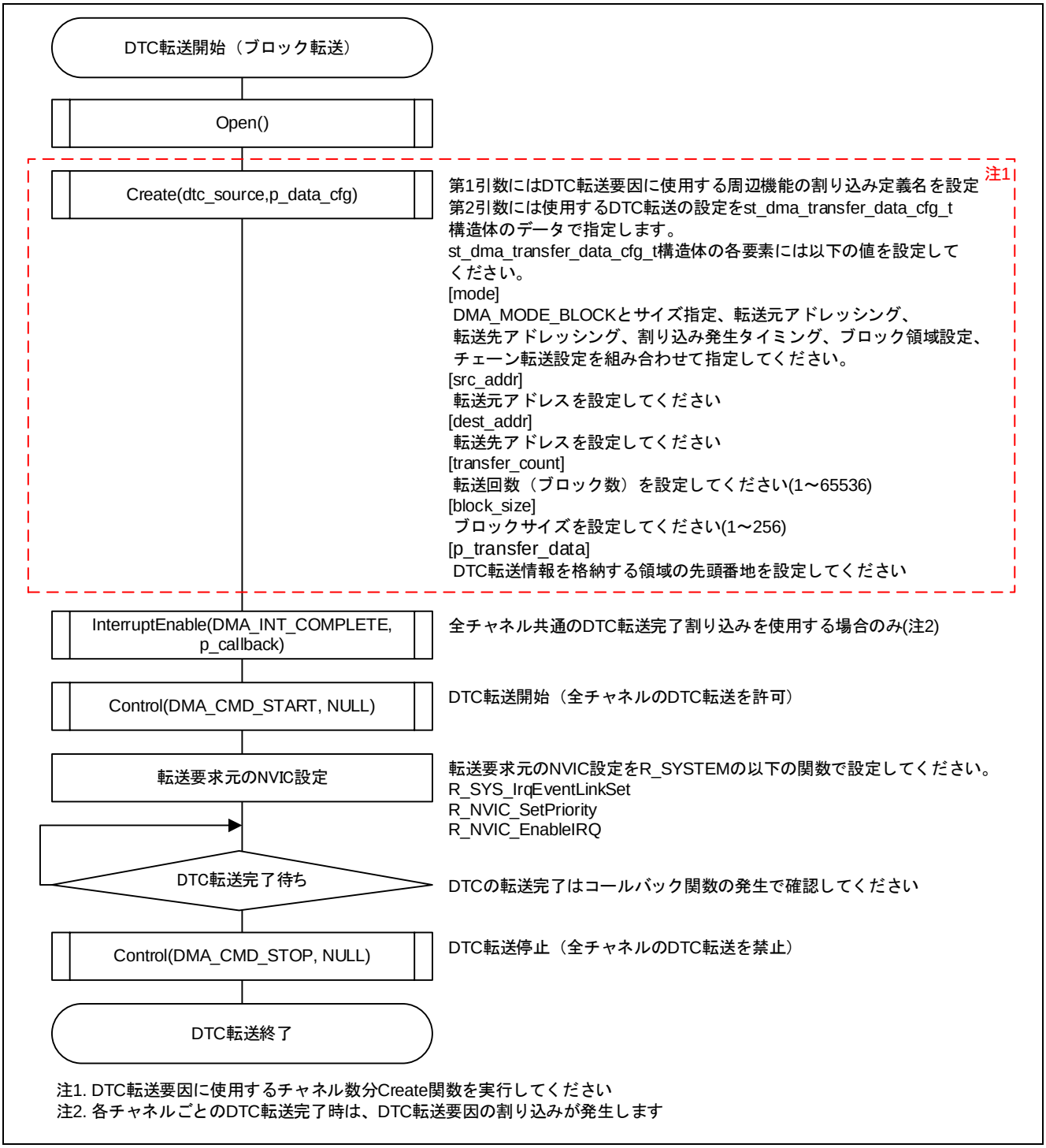

図 3-3 ブロック転送手順

3.4 チェーン転送

1 つの転送要因で複数のデータ転送を連続して行うことができます。チェーン転送を行う回数だけ st_dma_transfer_data_cfg_t 型の配列で DTC 転送設定を行ってください。DTC 転送情報の格納領域もチェーン転送分確保し、DTC 転送設定格納配列 1 番目の p_transfer_data メンバに設定してください。DTC 転送格納配列 2 番目以降の p_transfer_data の設定値は無視されます。

Create 関数の第 2 引数には DTC 転送設定格納配列の先頭番地を指定します。チェーン転送手順を図 3-4 に、チェーン転送使用例を図 3-5 に示します。

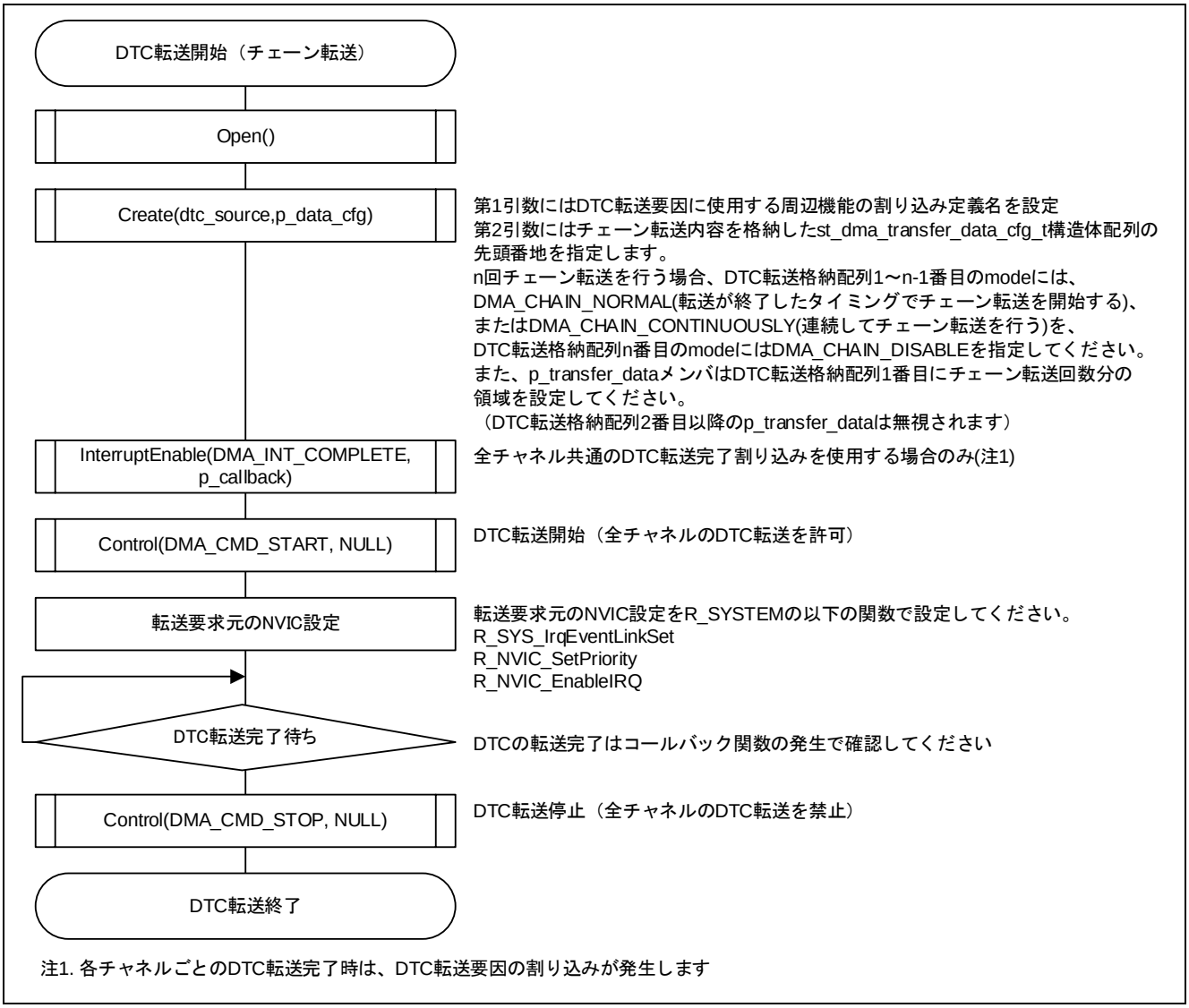

図 3-4 チェーン転送手順

```
#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 転送情報格納領域(チェーン転送 2 回分確保) */
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 転送設定情報格納領域(チェーン転送 2 回分配列で確保) */

    /* [チェーン 1 回目]ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定     */
    /* すべての転送完了時に割り込み発生、連続してチェーン転送を行う（チェーン転送継続）  */
    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];   /* 1 番目の DTC 転送情報に DTC 転送格納領域を設定 */

    /* [チェーン 2 回目]ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定     */
    /* すべての転送完了時に割り込み発生、チェーン転送無効（最終チェーン転送）           */
    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();                              /* DTC ドライバオープン */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config[0]);
                                                  /* TMR CMIA0 を起動要因に設定 */
    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* DTC 転送要因(CMIA0 割り込み)のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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;  /* DTC 転送要因(TMR)の起動 */
    while(1);
}

/**********************************************************************************************************
*****************
* callback function
**********************************************************************************************************
***************/
void cb_dtc_complete(void)
{
    /* チェーン 2 回目のすべての DTC 転送完了時にコールバック関数実行 */
}
```

図 3-5 チェーン転送使用例

3.5 複数要因の DTC 転送

DTC の転送要因を複数使用する場合は、Create 関数を複数回実行してください。各要因の DTC 転送割り込みは転送要因の割り込みが、共通の DTC 転送割り込みは DTC_COMPLETE が発生します。

複数要因の DTC 転送例として、以下の DTC 転送を行った場合の DTC 設定例を図 3-6～図 3-8 に、割り込み発生タイミング例を図 3-9 に示します。

- AGT0_AGTI 割り込み要因（カウンタのアンダフロー）によるリピート転送で、Create 関数実行時に DMA_INT_PER_SINGLE_TRANSFER（DTC データ転送のたびに、CPU への割り込み要求が発生）を指定
- SCI0_RXI 割り込み要因（受信完了）によるノーマル転送で、Create 関数実行時に DMA_INT_AFTER_ALL_COMPLETE（指定されたデータ転送の終了時、CPU への割り込み要求が発生）を指定
- IRQ0 割り込み要因によるノーマル転送で、Create 関数実行時に DMA_INT_AFTER_ALL_COMPLETE（指定されたデータ転送の終了時、CPU への割り込み要求が発生）を指定

```
#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 転送情報格納領域 */
uint8_t src_data1[5] = { 0x00, 0x01, 0x02, 0x04, 0x08 };
uint8_t dest_data1= 0xFF;
static st_dma_transfer_data_t transfer_info2;       /* DTC 転送情報格納領域 */
uint8_t src_data2[5] = { 0x10, 0x11, 0x12, 0x14, 0x18 };
uint8_t dest_data2= 0xFF;
static st_dma_transfer_data_t transfer_info3;       /* DTC 転送情報格納領域 */
uint8_t src_data3[5] = { 0x20, 0x21, 0x22, 0x24, 0x28 };
uint8_t dest_data3= 0xFF;

main()
{
    st_dma_transfer_data_cfg_t config;   /* DMA 転送設定情報格納領域 */

    (void)dtcDrv->Open();                           /* DTC ドライバオープン */

    /* リピート転送モード、8 ビット、転送元インクリメント、転送先固定              */
    /* 転送先がブロック領域、DTC 転送ごとに割り込み発生、チェーン転送無効          */
    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]);    /* 転送元設定(RAM) */
    config.dest_addr      = (uint32_t)(&dest_data1[0]);     /* 転送先設定(RAM) */
    config.transfer_count   = 5;                        /* 転送回数 5 回 */
    config.p_transfer_data = &transfer_info1;               /* DTC 転送情報格納場所 */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &config);
                                                    /* DTC 転送要因: AGT0_AGTI 割り込み要因 */
```

図 3-6 DTC 複数要因設定例（1/3）

```
    /* ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定                */
    /* すべての転送完了時に割り込み発生、チェーン転送無効                          */
    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]);    /* 転送元設定(RAM) */
    config.dest_addr       = (uint32_t)(&dest_data2[0]);     /* 転送先設定(RAM) */
    config.transfer_count   = 3;                          /* 転送回数 3 回 */
    config.p_transfer_data = &transfer_info2;               /* DTC 転送情報格納場所 */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_SCI0_RXI, &config);
                                                /* DTC 転送要因: SCI0 RXI 割り込み要因 */

    /* ノーマル転送モード、8 ビット、転送元インクリメント、転送先固定                */
    /* すべての転送完了時に割り込み発生、チェーン転送無効                          */
    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]);    /* 転送元設定(RAM) */
    config.dest_addr       = (uint32_t)(&dest_data3[0]);     /* 転送先設定(RAM) */
    config.transfer_count   = 1;                          /* 転送回数 1 回 */
    config.p_transfer_data = &transfer_info3;               /* DTC 転送情報格納場所 */
    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0, &config);
                                                /* DTC 転送要因: IRQ0 割り込み要因 */

    /* DTC_COMPLETE 割り込みの許可 */
    (void)dtcDrv->InterruptEnable(DMA_INT_COMPLETE, cb_dtc_complete);

    (void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */

    /* AGT0_AGTI のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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);

    /* SCI0_RXI のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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);

    /* IRQ0 のイベントリンク設定(NVIC 登録)、割り込みの許可 */
    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);
}
```

図 3-7 DTC 複数要因設定例（2/3）

```
/***********************************************************************************************
* callback function for DTC_COMPLETE
***********************************************************************************************/
void cb_dtc_complete(void)
{
    /* 各要因の DTC 転送完了時にコールバック関数実行 */
}

/***********************************************************************************************
* callback function for AGT0_AGTI
***********************************************************************************************/
void cb_agt0_agti(void)
{
    /* AGT0_AGTI 割り込み要因による DTC 転送発生時にコールバック関数実行 */
}

/***********************************************************************************************
* callback function for SCI0_RXI
***********************************************************************************************/
void cb_sci0_rxi(void)
{
    /* SCI0_RXI 割り込み要因による DTC 転送がすべて完了したときにコールバック関数実行 */
}

/***********************************************************************************************
* callback function for IRQ0
***********************************************************************************************/
void cb_irq0(void)
{
    /* IRQ0 割り込み要因による DTC 転送がすべて完了したときにコールバック関数実行 */
}
```

図 3-8 DTC 複数要因設定例（3/3）

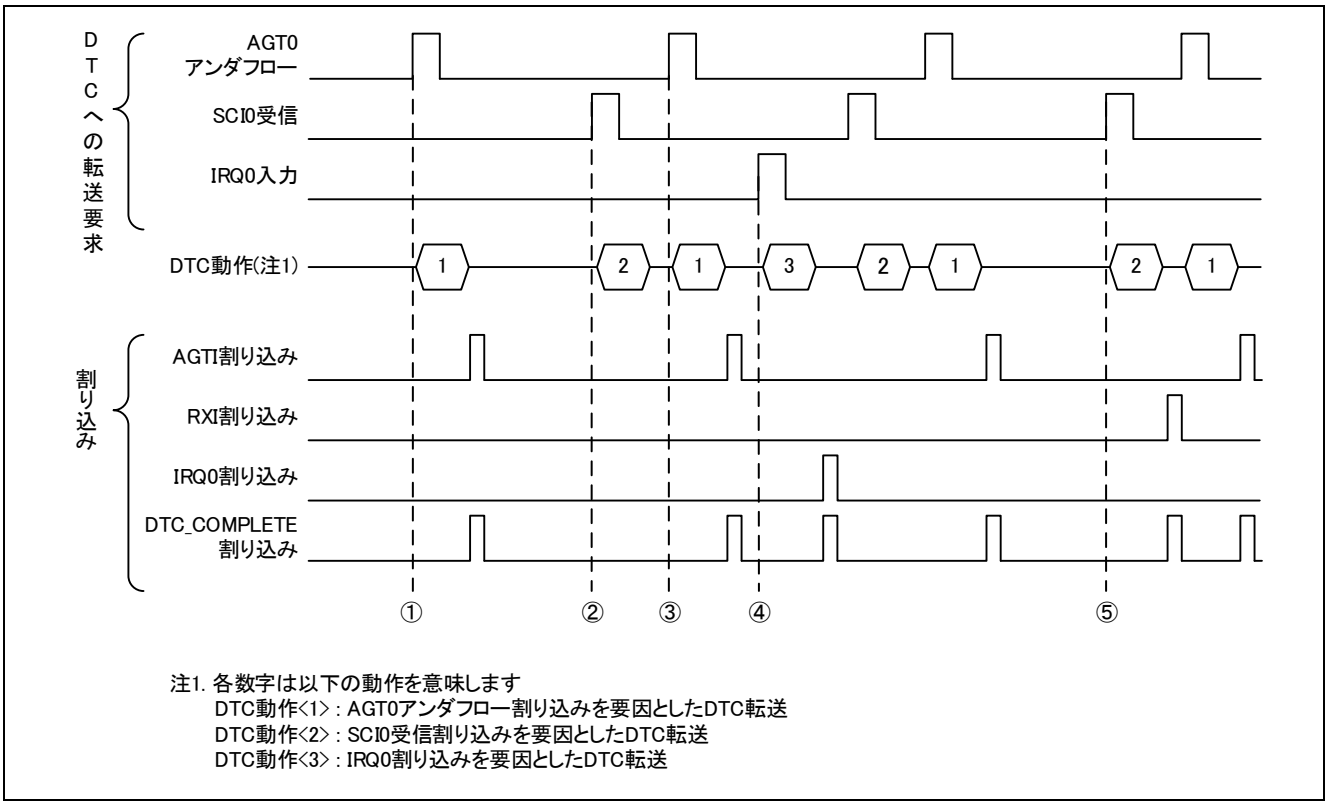

図 3-9 DTC 複数要因動作時の割り込み発生タイミング例

① AGT0 のアンダフローを要因として DTC 動作<1>が実行されます。DTC 転送が完了ごとに AGTI 割り込み、および DTC_COMPLETE 割り込みが発生します。

② SCI 受信完了を要因として DTC 転送<2>が実行されます。

③ AGT0 のアンダフローごとに DTC 転送<1>転送後、AGTI 割り込み、DTC_COMPLETE 割り込みが発生します。

④ IRQ0 割り込みを要因として DTC 動作<3>が実行されます。指定した回数(1 回)DTC 転送が完了時に、IRQ0 割り込み、および DTC_COMPLETE 割り込みが発生します。

⑤ SCI 受信完了による DTC 転送が指定回数(3 回)完了時に、RXI 割り込み、および DTC_COMPLETE 割り込みが発生します。

3.6 Control 関数による DTC 制御

Control 関数にて DTC の各制御を行うことができます。制御コマンド、およびコマンド別引数による動作一覧を表 3-1 に示します。

表 3-1　制御コマンドおよびコマンド別引数による動作一覧

制御コマンド(cmd)	コマンド別引数(arg)	内容
DMA_CMD_START	NULL	DTC 転送を開始します
DMA_CMD_STOP	NULL	DTC 転送を停止します
DMA_CMD_ACT_SRC_ENABLE	IRQn_Type 型のポインタ(注)	引数で指定した転送要因を許可にします
DMA_CMD_ACT_SRC_DISABLE	IRQn_Type 型のポインタ(注)	引数で指定した転送要因を禁止にします
DMA_CMD_DATA_READ_SKIP_ENABLE	uint8_t 型のポインタ(注)	引数で指定した変数の内容が true の場合 ： リードスキップ機能許可 false の場合：リードスキップ機能禁止
DMA_CMD_CHAIN_TRANSFER_ABORT	st_dma_chain_abort_t 型のポインタ(注)	引数で指定したチェーン転送を中断します
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	st_dma_chg_data_t 型のポインタ(注)	引数で指定した要因に対して、DTC 転送設定を強制的に変更します
DMA_CMD_REFRESH_DATA	st_dma_refresh_data_t 型のポインタ(注)	引数で指定した要因に対して、転送元、転送先、転送回数を再設定します

注　引数には指定の変数に設定値を格納し、アドレス渡しで指定します。

3.6.1 DTC 転送開始コマンド（DMA_CMD_START）

DTC 転送を許可状態にします。引数には NULL を設定してください。DTC 転送開始コマンド使用例を図 3-10 に示します。

```
(void)dtcDrv->Control(DMA_CMD_START, NULL);   /* DTC 許可 */
```

図 3-10 DTC 転送開始コマンド使用例

3.6.2 DTC 転送停止コマンド（DMA_CMD_STOP）

DTC 転送を禁止状態にします。引数には NULL を設定してください。DTC 転送停止コマンド使用例を図 3-11 に示します。

```
(void)dtcDrv->Control(DMA_CMD_STOP, NULL);   /* DTC 禁止 */
```

図 3-11 DTC 転送停止コマンド使用例

3.6.3 DTC 転送要因許可コマンド（DMA_CMD_ACT_SRC_ENABLE）

引数で指定した転送要因を許可にします。Create 関数にて DTC 転送要因を設定したときは、デフォルト設定で許可状態になっているため、本コマンドを使用する必要はありません。

DTC 転送要因禁止コマンド（DMA_CMD_ACT_SRC_DISABLE）で個別に DTC 転送要因を禁止にした後、または DTC 転送再設定コマンド（DMA_CMD_REFRESH_DATA）でオートスタートしない設定で実行した後、DTC 転送要因を許可にしたい場合にご使用ください。

引数には IRQnType 型の変数に許可にする DTC 転送要因を設定し、変数のアドレスを指定してください。DTC 転送要因許可コマンド使用例を図 3-12 に示します。

```
IRQn_Type arg;

/* AGTI 割り込み要因による DTC 転送を許可にする */
arg = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
(void)dtcDrv->Control(DMA_CMD_ACT_SRC_ENABLE, &arg);
```

図 3-12 DTC 転送要因許可コマンド使用例

3.6.4 DTC 転送要因禁止コマンド（DMA_CMD_ACT_SRC_DISABLE）

引数で指定した転送要因を禁止にします。本コマンドでは IELSRn レジスタの DTE ビットを”0”（DTC 起動禁止）にするため、対象の要因が発生した場合、NVIC 側に要求が伝わりコールバック関数が実行されます。

引数には IRQnType 型の変数に禁止にする DTC 転送要因を設定し、変数のアドレスを指定してください。DTC 転送要因禁止コマンド使用例を図 3-13 に示します。

```
IRQn_Type arg;

/* AGTI 割り込み要因による DTC 転送を禁止にする */
arg = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
(void)dtcDrv->Control(DMA_CMD_ACT_SRC_DISABLE, &arg);
```

図 3-13 DTC 転送要因禁止コマンド使用例

3.6.5 リードスキップ設定コマンド（DMA_CMD_DATA_READ_SKIP_ENABLE）

リードスキップ機能を許可／禁止にします。Open 関数で DTC を初期化したときは、リードスキップ機能はデフォルトで許可状態となっています。

引数には uint8_t 型の変数に true（許可）、または false（禁止）を設定し、変数のアドレスを指定してください。リードスキップ機能設定コマンド使用例を図 3-14 に示します。

```
uint8_t arg;

/* リードスキップ機能を許可にする */
arg = true;
(void)dtcDrv->Control(DMA_CMD_DATA_READ_SKIP_ENABLE, &arg);
```

図 3-14 リードスキップ設定コマンド使用例

3.6.6 DTC チェーン転送中断コマンド（DMA_CMD_CHAIN_TRANSFER_ABORT）

DTC のチェーン転送を解除します。本コマンド実行時、指定したチェーン回数分だけ DTC 転送情報の DTC モードレジスタ B（MRB）の CHNE ビットが”0”（DTC チェーン転送禁止）になります。そのため、1 番目のチェーン転送設定（情報転送テーブルの配列[0]）のみ有効になります。（チェーン転送は実行されません）

引数には st_dma_chain_abort_t 型の変数に DTC 転送要因およびチェーン転送回数を設定し、変数のアドレスを指定してください。DTC チェーン転送中断コマンド使用例を図 3-15 に示します。

```
st_dma_chain_abort_t  arg;

/* AGTI 割り込み要因の DTC 転送のチェーン転送を中断する */
arg.act_src = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 2;
(void)dtcDrv->Control(DMA_CMD_CHAIN_TRANSFER_ABORT, &arg);
```

図 3-15 DTC チェーン転送中断コマンド使用例

3.6.7 DTC 転送情報強制変更コマンド（DMA_CMD_CHANGING_DATA_FORCIBLY_SET）

DTC 転送情報を強制的に変更します。

引数には st_dma_chg_data_t 型の変数に変更対象の DTC 起動要因、変更するチェーン番号-1（チェーン 2 回目であれば 1 を格納）、変更する DTC 転送設定情報（st_dma_transfer_data_cfg_t 型）を設定し、変数のアドレスを指定してください。強制変更される DTC 転送情報は、変更するチェーン番号として指定したチェーン番号のみ。また、DTC 転送設定情報の p_transfer_data 要素には、Create 時の設定と同じ値を設定してください。異なる値を設定した場合の動作は保証されません。

変更するチェーン番号が 2 の場合の動作例を図 3-16 に、DTC 転送情報強制変更コマンド使用例を図 3-17 に示します。

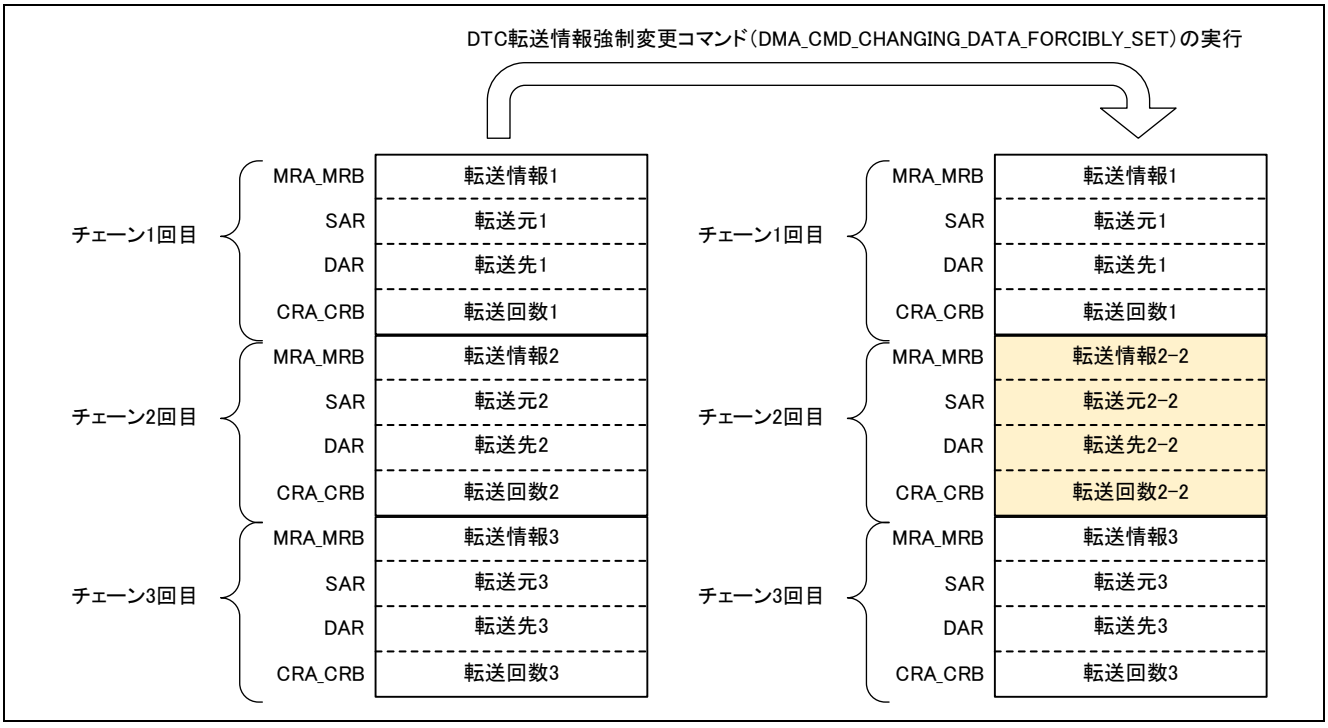

図 3-16 DTC 転送情報強制変更コマンド動作例（チェーン 2 回目を変更する例）

```
t_dma_chg_data_t arg;

/* AGTI 割り込み要因の 2 番目のチェーン転送設定を強制的に書き換える*/
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;   /* Create 時に指定した DTC 転送格納領域を設定 */

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

図 3-17 DTC 転送情報強制変更コマンド使用例

3.6.8 DTC 転送再設定コマンド（DMA_CMD_REFRESH_DATA）

DTC 転送情報の再設定を行います。

DTC 転送が完了した後、モードは変更せずに、転送元アドレス（注）、転送先アドレス（注）、転送回数のみ更新します。auto_start メンバに”1”を設定すると、本コマンド実行後に DTC 転送が再開します。auto_start 要素を”0”で更新した場合は、任意のタイミングで DTC 転送要因許可コマンド（DMA_CMD_ACT_SRC_ENABLE）を使用して DTC 転送を許可にしてください。

転送元アドレスに NULL(0)を設定すると、現在の転送元アドレスが保持されます。同様に転送先アドレスに NULL(0)を設定すると現在の転送先アドレスが保持されます。

chain_transfer_nr メンバは、更新するチェーン転送番号- 1 を指定します。複数のチェーン転送情報を更新する場合は、複数回 DTC 転送再設定コマンドを実行してください。チェーン転送未使用の場合は、0 を設定してください。

チェーン転送未使用時の DTC 転送再設定コマンド使用例を図 3-18 に、チェーン転送使用時（チェーン回数 2 回）の DTC 転送再設定コマンド使用例を図 3-19 に示します。

注　転送元アドレス、および転送先アドレスは、転送サイズが 16 ビットサイズ転送のときは 2 の倍数、32 ビットサイズ転送の場合は 4 の倍数になるよう設定してください。

```
st_dma_refresh_data_t arg;

/* AGTI 割り込み要因の転送設定を更新し、DTC 動作許可にする */
arg.act_src        = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 0;                  /* チェーン転送未使用のため 0 設定 */
arg.src_addr       = (uint32_t)(&source_ref); /* 転送元更新 */
arg.dest_addr      = NULL;                /* 転送先保持 */
arg.transfer_count  = 5;                    /* 転送回数の更新 */
arg.block_size      = 0;                   /* ブロックサイズの更新(ブロック転送のみ有効) */
arg.auto_start      = true;                /* 更新コマンド実行後、DTC 許可 */
(void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);
```

図 3-18 DTC 転送再設定コマンド使用例（チェーン転送未使用）

```
st_dma_refresh_data_t arg;

/* AGTI 割り込み要因の転送設定、チェーン番号 1 の情報を更新（DTC 転送は停止状態のまま）*/
arg.act_src        = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 0;                    /* チェーン番号 1 更新*/
arg.src_addr       = (uint32_t)(&source_ref_1); /* 転送元更新 */
arg.dest_addr      = (uint32_t)(&dest_ref_1);   /* 転送先更新 */
arg.transfer_count  = 5;                     /* 転送回数の更新 */
arg.block_size      = 0;                    /* ブロックサイズの更新(ブロック転送のみ有効) */
arg.auto_start      = flase;                /* 更新コマンド実行後、DTC 禁止のまま */
(void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);

/* AGTI 割り込み要因の転送設定、チェーン番号 2 の情報を更新し、DTC 転送許可にする */
arg.act_src        = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
arg.chain_transfer_nr = 1;                    /* チェーン番号 2 更新*/
arg.src_addr       = (uint32_t)(&source_ref_2); /* 転送元更新 */
arg.dest_addr      = (uint32_t)(&dest_ref_2);   /* 転送先更新 */
arg.transfer_count  = 2;                     /* 転送回数の更新 */
arg.block_size      = 0;                    /* ブロックサイズの更新(ブロック転送のみ有効) */
arg.auto_start      = true;                 /* 更新コマンド実行後、DTC 許可 */
(void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);
```

図 3-19 DTC 転送再設定コマンド使用例（チェーン転送使用、チェーン回数 2 回時）

3.7 DTC 転送バイト数の取得

GetTransferByte 関数にて、DTC 転送バイト数を取得できます。第 1 引数には転送バイト数を取得する DTC 転送要因を、第 2 引数には取得結果を格納する変数を設定してください。

リピート転送時は、指定したリピートサイズの転送完了時 DTC 転送バイト数は 0 に戻ります。

GetTransferByte 関数使用例を図 3-20 に、ノーマル転送モードでの DTC 転送バイト数取得例を表 3-2 に、リピート転送モードでの DTC 転送バイト数取得例を表 3-3 に、ブロック転送モードでの DTC 転送バイト数取得例を表 3-4 に示します。

```
uint32_t transfer_byte;
/* AGTI 割り込みを要因とした DTC 転送バイト数を取得*/
(void)dtcDrv-> GetTransferByte(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &transfer_byte);
```

図 3-20 GetTransferByte 関数使用例

表 3-2 ノーマル転送モードでの DTC 転送バイト数取得例（転送サイズ 2 バイト）

転送要因発生回数	1	2	3	4	5	6	7
DTC 転送バイト数（注 1）	2	4	6	8	10	12	14

表 3-3 リピート転送モードでの DTC 転送バイト数取得例（転送サイズ 4 バイト、リピートサイズ 5）

転送要因発生回数	1	2	3	4	5(注 2)	6	7
DTC 転送バイト数（注 1）	4	8	12	16	0	4	8

表 3-4 ブロック転送モードでの DTC 転送バイト数取得例（転送サイズ 1 バイト、ブロックサイズ 10）

転送要因発生回数	1	2	3	4	5	6	7
DTC 転送バイト数（注 1）	10	20	30	40	50	60	70

注1. GetTransferByte 関数で取得した DTC 転送バイト数

注2. DTC ドライバではリピートサイズ分の転送完了後、DTC 転送バイト数はリセット(0)されます。

3.8 コンフィグレーション

DTC ドライバは、ユーザが設定可能なコンフィグレーションを r_dtc_cfg.h ファイルに用意します。

3.8.1 パラメータチェック

R_DTC ドライバにおけるパラメータチェックの有効/無効を設定します。

名称：DTC_CFG_PARAM_CHECKING_ENABLE

表 3-5 DTC_CFG_PARAM_CHECKING_ENABLE の設定

設定値	内容
0	パラメータチェックを無効にします 関数仕様に記載している引数の妥当性判定に関するエラーの検出を行いません
1（初期値）	パラメータチェックを有効にします 関数仕様に記載しているパラメータ判定エラー条件の検出を行います

3.8.2 DTC_COMPLETE 割り込み優先レベル

DTC_COMPLETE 割り込みの優先レベルを設定します。

名称：DTC_COMPLETE_PRIORITY

表 3-6 DTC_COMPLETE_PRIORITY の設定

設定値	内容
0	割り込み優先レベルを 0（最高）に設定
1	割り込み優先レベルを 1 に設定
2	割り込み優先レベルを 2 に設定
3（初期値）	割り込み優先レベルを 3（最低）に設定

3.8.3 関数の RAM 配置

DTC ドライバの特定関数を RAM で実行するための設定を行います。

関数の RAM 配置を設定するコンフィグレーションは、関数ごとに定義を持ちます。

名称：DTC_CFG_xxx

xxx には関数名をすべて大文字で記載

例） R_DTC_Open 関数 → DTC_CFG_R_DTC_OPEN

表 3-7 DTC_CFG_xxx の設定

設定値	内容
SYSTEM_SECTION_CODE	関数を RAM に配置しません
SYSTEM_SECTION_RAM_FUNC	関数を RAM に配置します

表 3-8 各関数の RAM 配置初期状態

番号	関数名	RAM 配置
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 割り込み処理）	✔

4. ドライバ詳細情報

本章では、本ドライバ機能を構成する詳細仕様について説明します。

4.1 関数仕様

DTC ドライバの各関数の仕様と処理フローを示します。

処理フロー内では条件分岐などの判定方法の一部を省略して記述しているため、実際の処理と異なる場合があります。

4.1.1 R_DTC_Open 関数

表 4-1 R_DTC_Open 関数仕様

書式	static e_dma_err_t R_DTC_Open(void)	
仕様説明	DTC ドライバの初期化（RAM の初期化、レジスタ設定）を行います	
引数	なし	
戻り値	DMA_OK	DTC の初期化成功
	DMA_ERROR_LOCKED	リソースロックによる DTC 初期化失敗 DTC のリソースがロックされている場合にリソースロックによる DTC 初期化失敗となります （すでに R_SYS_ResourceLock 関数にて DTC がロックされている場合）
	DMA_ERROR	DTC の初期化失敗 DTC のモジュールストップ解除に失敗した場合、DTC の初期化失敗となります

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	　　dtcDrv->Open();
	}	

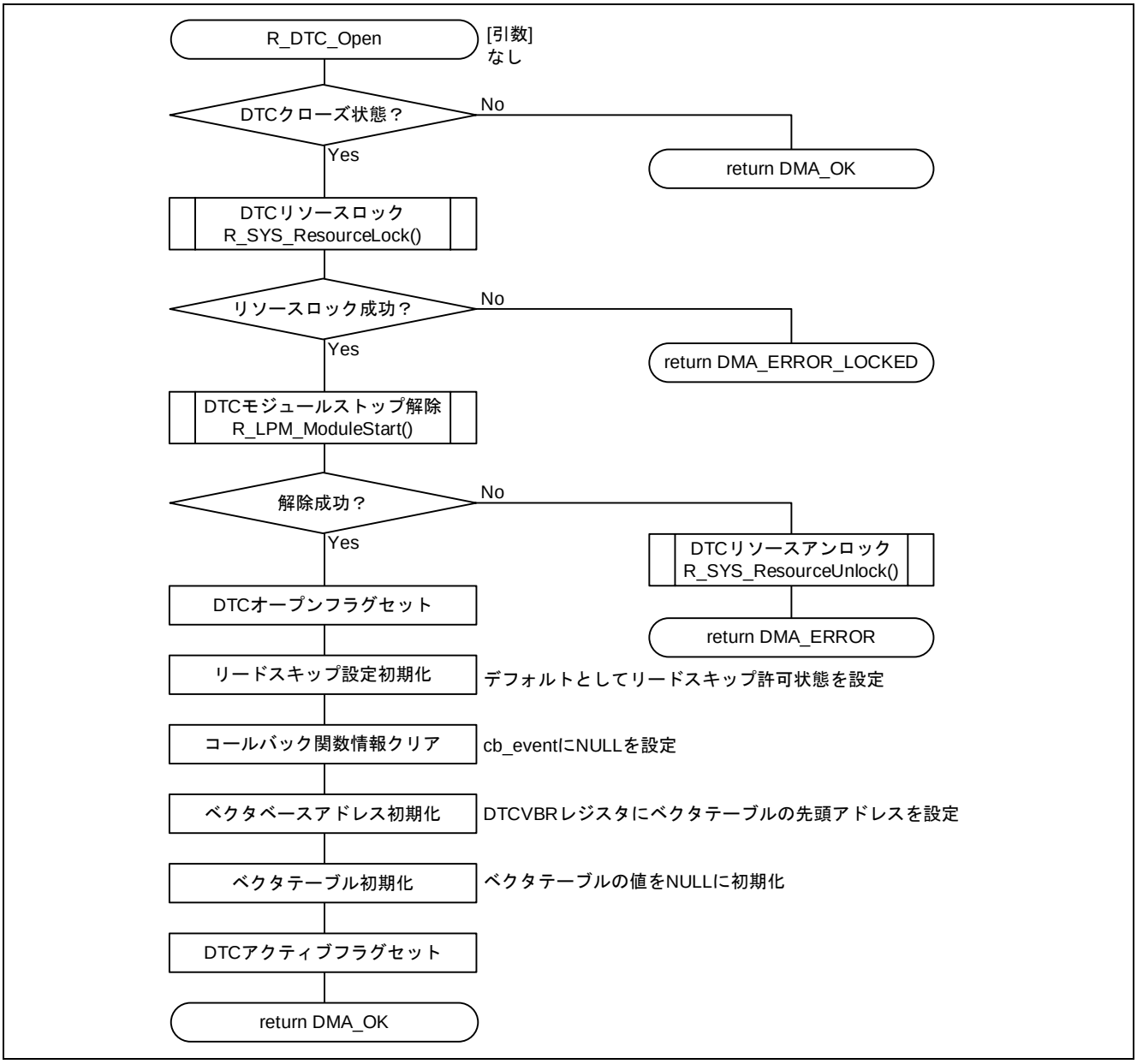

図 4-1 R_DTC_Open 関数処理フロー

4.1.2 R_DTC_Close 関数

表 4-2 R_DTC_Close 関数仕様

書式	static e_dma_err_t R_DTC_Close(void)
仕様説明	DTC ドライバを解放します
引数	なし
戻り値	DMA_OK　　DTC の解放成功

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	　　dtcDrv->Close();
	}

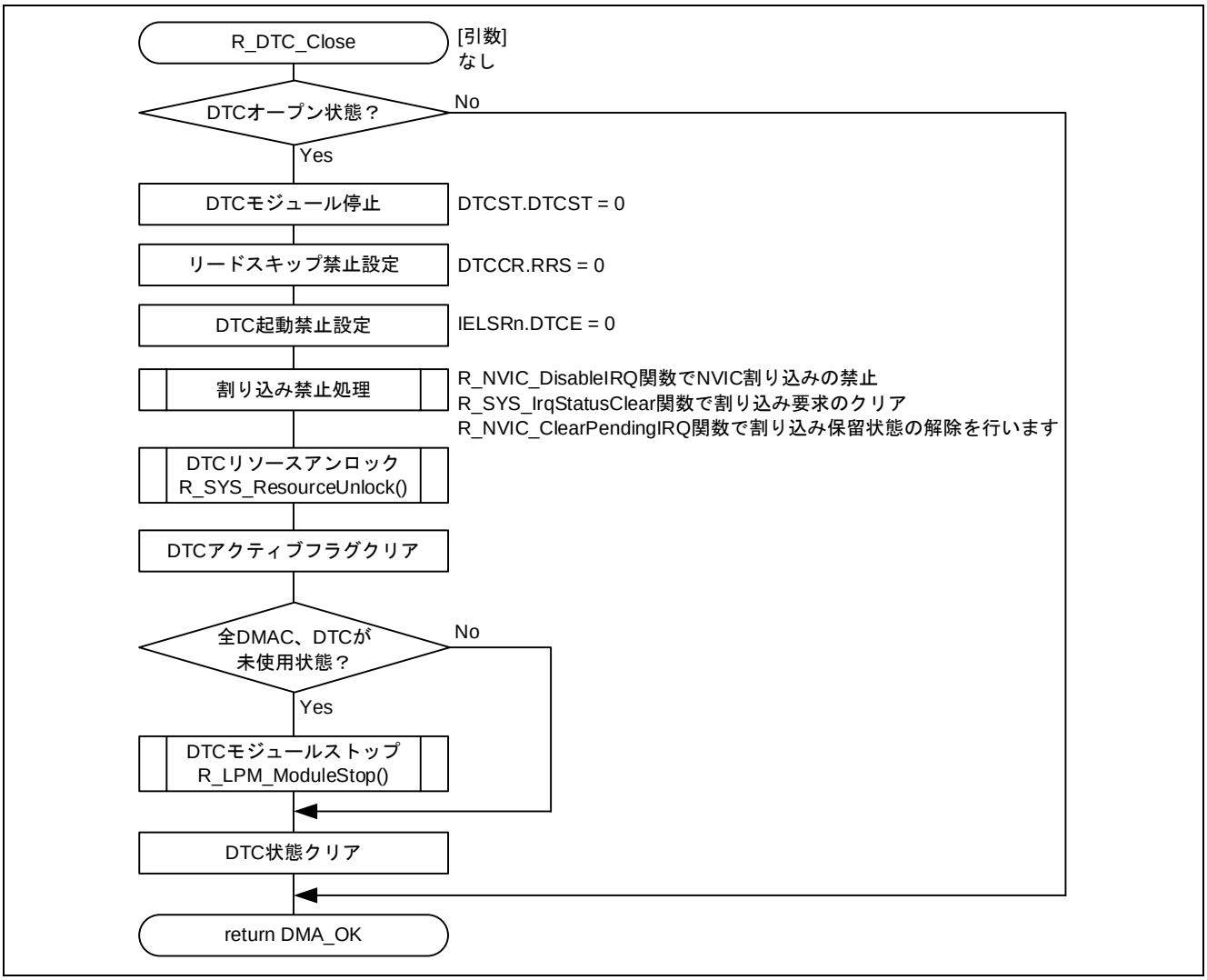

図 4-2 R_DTC_Close 関数処理フロー

4.1.3 R_DTC_Create 関数

表 4-3 R_DTC_Create 関数仕様

書式	static e_dma_err_t R_DTC_Create(int16_t const dtc_source, st_dma_transfer_data_cfg_t * const p_data_cfg)
仕様説明	DTC 転送設定を行います
引数	int16_t const dtc_source : DTC 転送要因 DTC 転送要因に設定する周辺機能の割り込み定義名を指定します （IRQ0 の場合は SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0）
	st_dma_transfer_data_cfg_t * const p_data_cfg : DTC 転送設定 指定した要因に対する DTC 転送設定を指定します
戻り値	DMA_OK　DTC 転送設定成功
	DMA_ERROR_PARAMETER　パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります ・DTC 転送要因に不正な値（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED 以上、もしくは 0 未満）を設定した場合 ・ノーマル転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・リピート転送設定時に、転送回数を 1～256 の範囲外を設定した場合 ・ブロック転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・ブロック転送設定時に、ブロック数を 1～256 の範囲外を設定した場合 ・転送サイズに 16 ビットサイズ転送指定時に、転送元、転送先アドレスが 2 の倍数でない場合 ・転送サイズに 32 ビットサイズ転送指定時に、転送元、転送先アドレスが 4 の倍数でない場合 ・モード指定の予約ビット領域に 1 を設定した場合
	DMA_ERROR_MODE　モードエラー 以下のいずれかの条件を検出するとモードエラーとなります ・転送元アドレッシング指定にオフセット指定（DMA_SRC_ADDR_OFFSET）を設定した場合 ・転送先アドレッシング指定にオフセット指定（DMA_DEST_ADDR_OFFSET）を設定した場合 ・リピート/ブロック領域指定にリピート/ブロック領域なし（DMA_REPEAT_BLOCK_NONE）を指定した場合 ・モード指定に不正な値を設定した場合 ・サイズ指定に不正な値を設定した場合 ・リピート/ブロック領域指定に不正な値を指定した場合 ・チェーン転送指定に不正な値を指定した場合
	DMA_ERROR　DTC 転送設定失敗 DTC 初期化前に実行した場合、DTC 転送設定失敗となります
備考	[インスタンスからの関数呼び出し例]

```
// 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]);        /* 転送元設定(RAM) */
    config.dest_addr         = (uint32_t)(&dest_data);
    config.transfer_count    = 5;                               /* 転送回数 5 回 */
    config.p_transfer_data = &transfer_info;                    /* DTC 転送情報格納場所 */

    (void)dtcDrv->Create(SYSTEM_CFG_EVENT_NUMBER_TMR_CMIA0, &config);
}
```

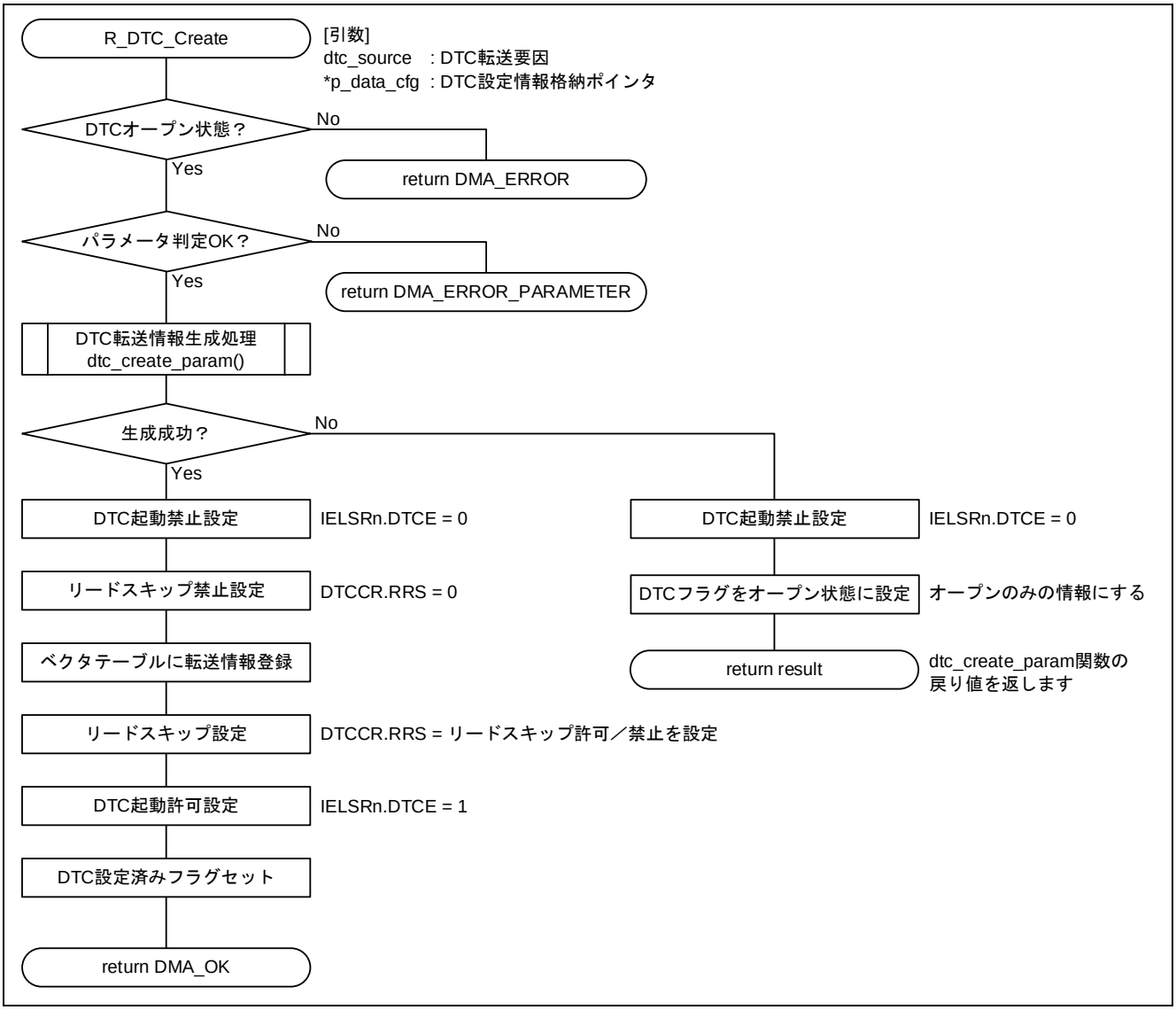

図 4-3 R_DTC_Create 関数処理フロー

4.1.4 R_DTC_Control 関数

表 4-4 R_DTC_Control 関数仕様

書式	static e_dma_err_t R_DTC_Control(e_dma_cmd_t cmd, void const * const p_arg)
仕様説明	DTC の制御コマンドを実行します
引数	e_dma_cmd_t cmd: DTC 制御コマンド 以下のいずれかの制御コマンドを指定します ・DMA_CMD_START : DTC 転送開始コマンド ・DMA_CMD_STOP : DTC 転送停止コマンド ・DMA_CMD_ACT_SRC_ENABLE : DTC 転送要因許可コマンド ・DMA_CMD_ACT_SRC_DISABLE : DTC 転送要因禁止コマンド ・DMA_CMD_DATA_READ_SKIP_ENABLE ： リードスキップ設定コマンド ・DMA_CMD_CHAIN_TRANSFER_ABORT ：DTC チェーン転送中断コマンド ・DMA_CMD_CHANGING_DATA_FORCIBLY_SET ：DTC 転送情報強制変更コマンド ・DMA_CMD_REFRESH_DATA : DTC 転送再設定コマンド
	void const * const p_arg: コマンド別の設定値を格納した変数ポインタを指定します（制御コマンドと引数の関係については表 4-5 参照）
戻り値	DMA_OK 制御コマンド実行成功
	DMA_ERROR 制御コマンド実行失敗 以下のいずれかの条件を検出すると制御コマンド実行失敗となります ・DTC 未初期化状態で実行した場合 ・以下のコマンド実行時に引数が NULL の場合 - DMA_CMD_ACT_SRC_ENABLE コマンド - DMA_CMD_ACT_SRC_DISABLE コマンド - DMA_CMD_DATA_READ_SKIP_ENABLE コマンド - DMA_CMD_CHAIN_TRANSFER_ABORT コマンド - DMA_CMD_CHANGING_DATA_FORCIBLY_SET コマンド - DMA_CMD_REFRESH_DATA コマンド ・以下のコマンド実行時に指定した要因が SYSTEM_IRQ_EVENT_NUMBER_NOT_USED（割り込み未使用）以上の場合 - DMA_CMD_ACT_SRC_ENABLE コマンド - DMA_CMD_ACT_SRC_DISABLE コマンド - DMA_CMD_CHAIN_TRANSFER_ABORT コマンド - DMA_CMD_CHANGING_DATA_FORCIBLY_SET コマンド - DMA_CMD_REFRESH_DATA コマンド ・以下のコマンド実行時、指定した要因の DTC ベクタテーブルが NULL の場合（指定した要因に対する Create 関数未実行） - DMA_CMD_ACT_SRC_ENABLE コマンド - DMA_CMD_ACT_SRC_DISABLE コマンド - DMA_CMD_CHAIN_TRANSFER_ABORT コマンド - DMA_CMD_CHANGING_DATA_FORCIBLY_SET コマンド - DMA_CMD_REFRESH_DATA コマンド ・不正なコマンドを実行した場合

戻り値	DMA_ERROR_PARAMETER　　パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります [DMA_CMD_CHANGING_DATA_FORCIBLY_SET コマンド実行時] ・ノーマル転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・リピート転送設定時に、転送回数を 1～256 の範囲外を設定した場合 ・ブロック転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・ブロック転送設定時に、ブロック数を 1～256 の範囲外を設定した場合 ・転送サイズに 16 ビットサイズ転送指定時に、転送元、転送先アドレスが 2 の倍数でない場合 ・転送サイズに 32 ビットサイズ転送指定時に、転送元、転送先アドレスが 4 の倍数でない場合 ・モード指定の予約ビット領域に 1 を設定した場合 [DMA_CMD_REFRESH_DATA コマンド実行時] ・ノーマル転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・リピート転送設定時に、転送回数を 1～256 の範囲外を設定した場合 ・ブロック転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・ブロック転送設定時に、ブロック数を 1～256 の範囲外を設定した場合
	DMA_ERROR_MODE　　モードエラー DMA_CMD_CHANGING_DATA_FORCIBLY_SET コマンド実行時に以下のいずれかの条件を検出するとモードエラーとなります ・転送元アドレッシング指定にオフセット指定（DMA_SRC_ADDR_OFFSET）を設定した場合 ・転送先アドレッシング指定にオフセット指定（DMA_DEST_ADDR_OFFSET）を設定した場合 ・リピート/ブロック領域指定にリピート/ブロック領域なし（DMA_REPEAT_BLOCK_NONE）を指定した場合 ・モード指定に不正な値を設定した場合 ・サイズ指定に不正な値を設定した場合 ・リピート/ブロック領域指定に不正な値を指定した場合 ・チェーン転送指定に不正な値を指定した場合

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	　　(void)dtcDrv->Control(DMA_CMD_START, NULL);
	}

表 4-5　制御コマンドおよびコマンド別引数による動作一覧

制御コマンド(cmd)	コマンド別引数(arg)	内容
DMA_CMD_START	NULL	DTC 転送を開始します
DMA_CMD_STOP	NULL	DTC 転送を停止します
DMA_CMD_ACT_SRC_ENABLE	IRQn_Type 型のポインタ(注)	引数で指定した転送要因を許可にします
DMA_CMD_ACT_SRC_DISABLE	IRQn_Type 型のポインタ(注)	引数で指定した転送要因を禁止にします
DMA_CMD_DATA_READ_SKIP_ENABLE	uint8_t 型のポインタ(注)	引数で指定した変数の内容が true の場合 ： リードスキップ機能許可 false の場合：リードスキップ機能禁止
DMA_CMD_CHAIN_TRANSFER_ABORT	st_dma_chain_abort_t 型のポインタ(注)	引数で指定したチェーン転送を中断します
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	st_dma_chg_data_t 型のポインタ(注)	引数で指定した要因に対して、DTC 転送設定を強制的に変更します
DMA_CMD_REFRESH_DATA	st_dma_refresh_data_t 型のポインタ(注)	引数で指定した要因に対して、転送元、転送先、転送回数を再設定します

注　引数には指定の変数に設定値を格納し、アドレス渡しで指定します。

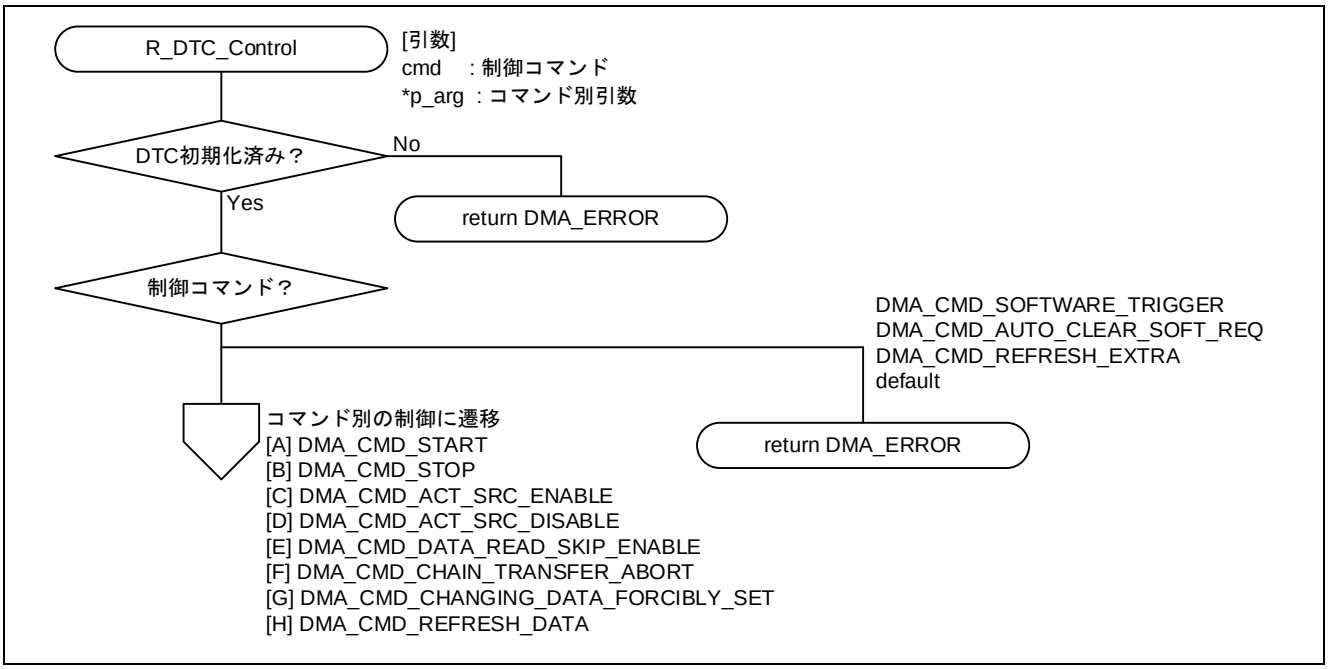

図 4-4 R_DTC_Control 関数処理フロー（1/3）

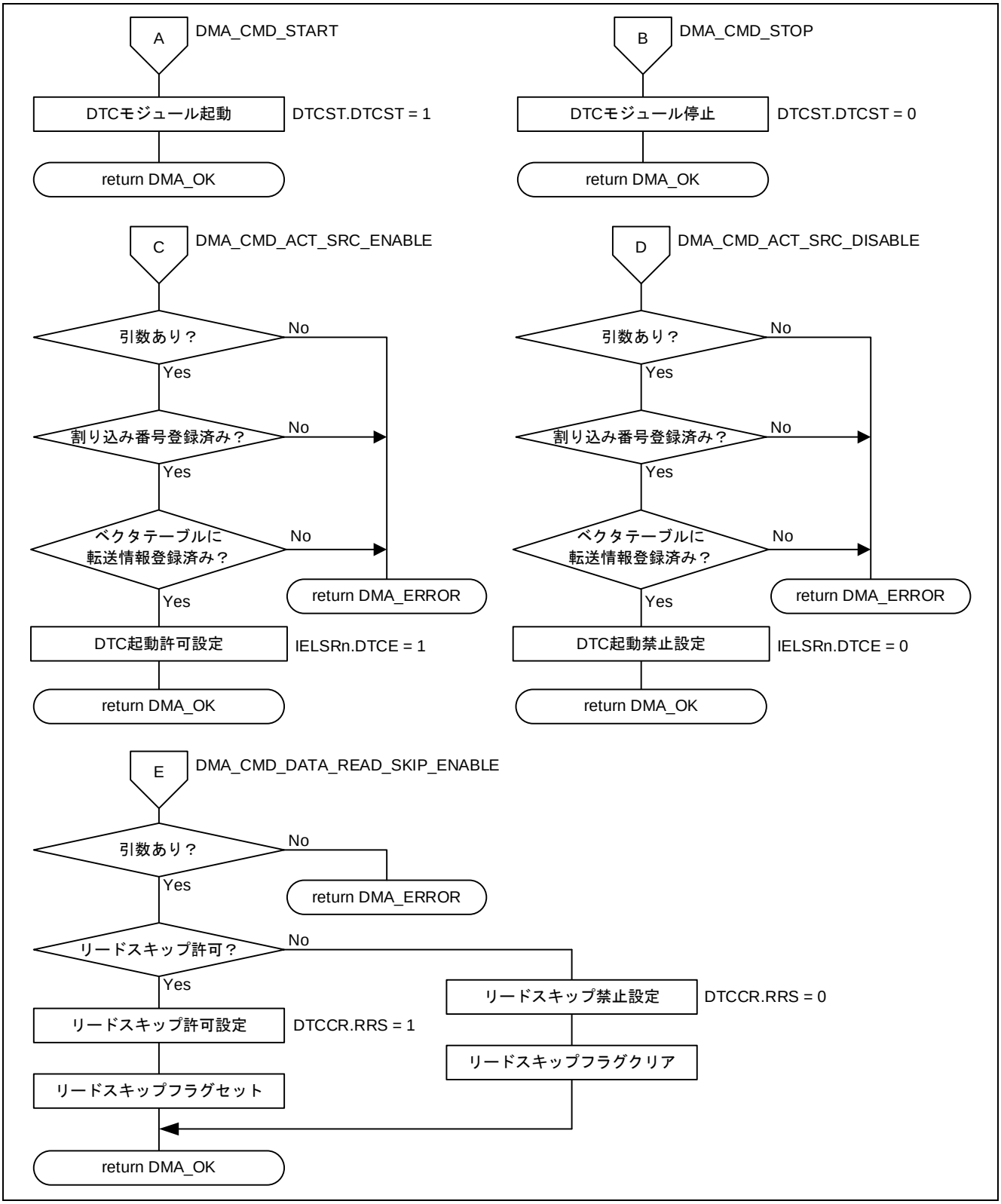

図 4-5 R_DTC_Control 関数処理フロー（2/3）

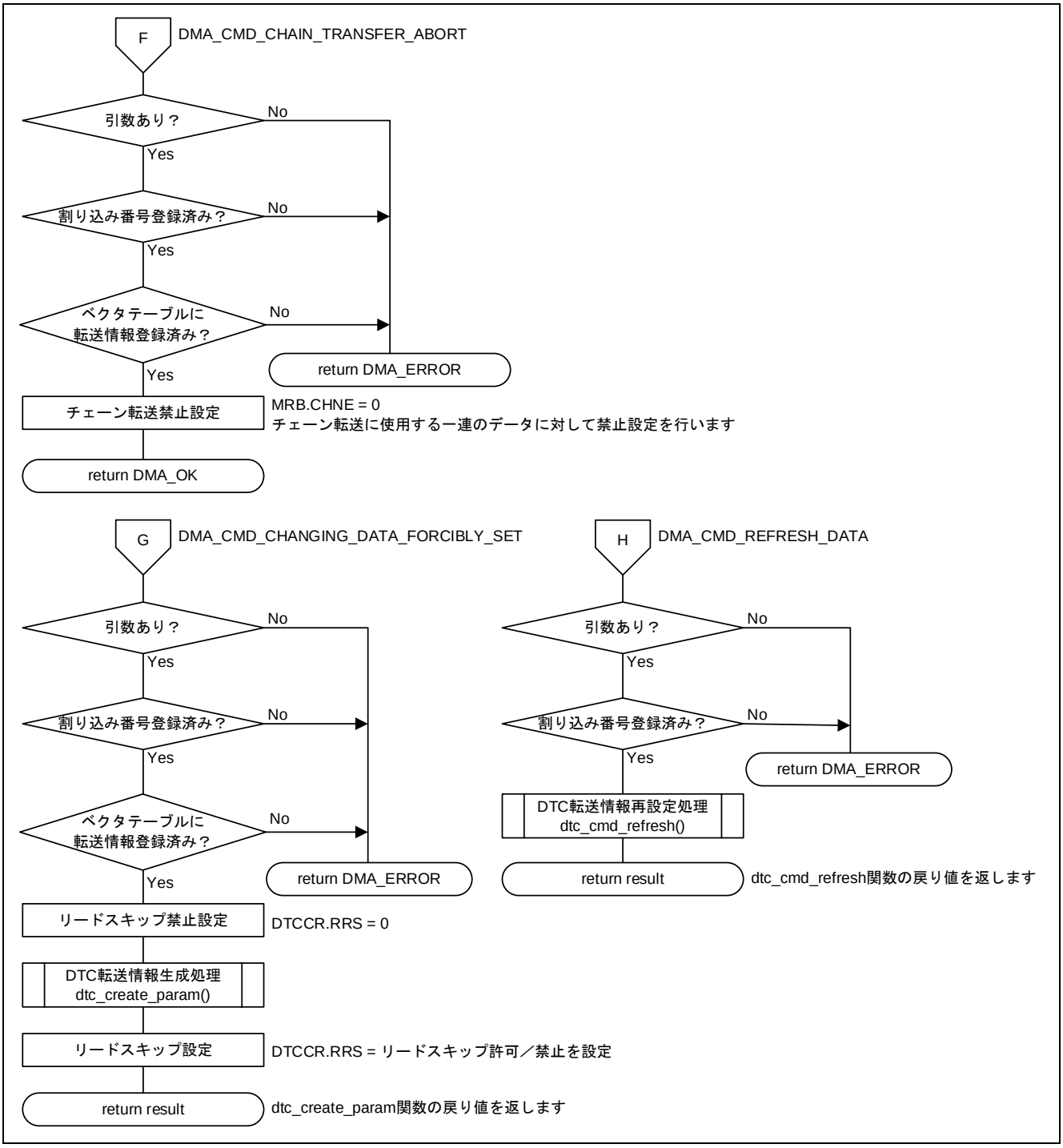

図 4-6 R_DTC_Control 関数処理フロー（3/3）

4.1.5 R_DTC_InterruptEnable 関数

表 4-6 R_DTC_InterruptEnable 関数仕様

書式	static e_dma_err_t R_DTC_InterruptEnable(uint16_t const int_fact, dma_cb_event_t const p_callback)
仕様説明	DTC ドライバの転送完了割り込み（DTC_COMPLETE）を設定します
引数	uint16_t const int_fact: DMA 割り込み要因 DMA_INT_COMPLETE を指定します
	dma_cb_event_t const p_callback :コールバック関数 DTC 転送完了割り込み発生時のコールバック関数を指定します
戻り値	DMA_OK 割り込み設定成功
	DMA_ERROR 割り込み設定失敗 以下のいずれかの条件を検出すると割り込み設定失敗となります ・DTC 未初期化状態で実行した場合 ・DMA 割り込み要因に DMA_INT_COMPLETE 以外を設定した場合
	DMA_ERROR_SYSTEM_SETTING システム設定エラー DTC 転送完了割り込み（DTC_COMPLETE）の割り込み設定が割り込み未使用（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED 以上）の場合、システム設定エラーとなります

備考	[インスタンスからの関数呼び出し例]
	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);
	}

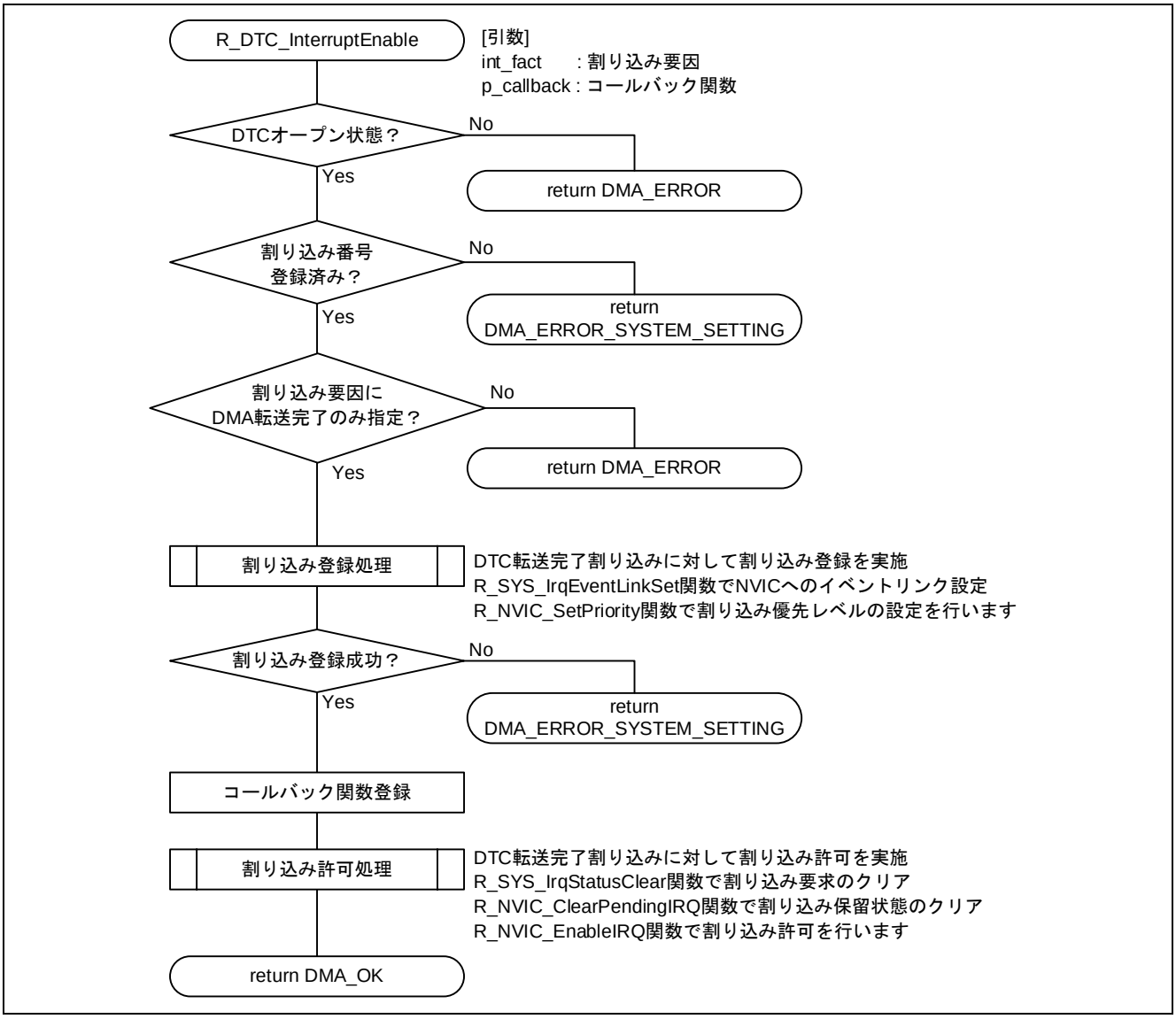

図 4-7 R_DTC_InterruptEnable 関数処理フロー

4.1.6 R_DTC_InterruptDisable 関数

表 4-7 R_DTC_InterruptDisable 関数仕様

書式	static e_dma_err_t R_DTC_InterruptDisable(void)
仕様説明	DTC ドライバの転送完了割り込みを禁止にします
引数	なし
戻り値	DMA_OK 割り込み禁止設定成功
	DMA_ERROR 割り込み禁止設定失敗 DTC 未初期化状態で実行した場合、割り込み禁止設定失敗となります。
	DMA_ERROR_SYSTEM_SETTING システム設定エラー DTC 転送完了割り込み（DTC_COMPLETE）の割り込み設定が割り込み未使用（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED 以上）の場合、システム設定エラーとなります

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv-> InterruptDisable();
	}

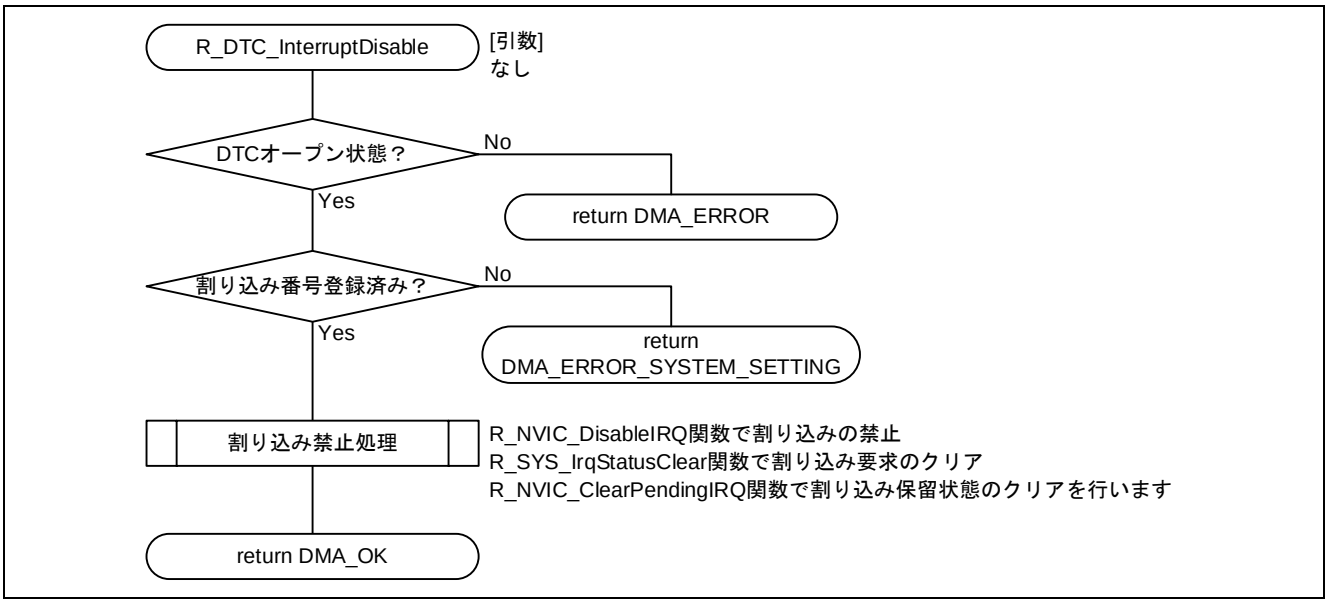

図 4-8 R_DTC_InterruptDisable 関数処理フロー

4.1.7 R_DTC_GetState 関数

表 4-8 R_DTC_GetState 関数仕様

書式	static e_dma_err_t R_DTC_GetState(st_dma_state_t * const state)
仕様説明	DTC ステータスを取得します（DTC ではすべて 0 を返します）
引数	st_dma_state_t * const state : ステータス格納領域 取得したステータスを格納する領域を指定します（0 のみ返します）
戻り値	DMA_OK ステータス取得成功

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	st_dma_state_t status;
	
	main()
	{
	dtcDrv-> GetStatus(&status);
	}

4.1.8 R_DTC_ClearState 関数

表 4-9 R_DTC_ClearState 関数仕様

書式	static e_dma_err_t R_DTC_ClearState(uint32_t clr_state)
仕様説明	ステータスをクリアする（DTC では何も処理しません）
引数	uint32_t clr_state : DTC では使用しません
戻り値	DMA_OK クリア成功

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	dtcDrv-> ClearState(0);
	}

4.1.9 R_DTC_GetTransferByte 関数

表 4-10 R_DTC_GetTransferByte 関数仕様

書式	static e_dma_err_t R_DTC_GetTransferByte(int16_t const dtc_source, uint32_t *transfer_byte)	
仕様説明	DTC 転送バイト数を取得します	
引数	int16_t const dtc_source : DTC 転送要因 転送バイト数を取得する DTC 転送要因を周辺機能の割り込み定義名で指定します （IRQ0 の場合は SYSTEM_CFG_EVENT_NUMBER_PORT_IRQ0）	
	uint32_t *transfer_byte :DTC 転送バイト格納ポインタ DTC 転送バイトを格納するポインタを指定します	
戻り値	DMA_OK	DTC 転送バイト数取得成功
	DMA_ERROR	DTC 転送バイト数取得失敗 以下のいずれかの条件を検出すると DTC 転送バイト数取得失敗となります ・DTC 初期化前に実行した場合 ・指定した要因の DTC ベクタテーブルが NULL の場合 （指定した要因に対する Create 関数未実行）
	DMA_ERROR_PARAMETER	パラメータエラー 指定した DTC 転送要因が SYSTEM_IRQ_EVENT_NUMBER_NOT_USED（割り込み未使用）以上の場合、または 0 未満の場合、パラメータエラーとなります。
	DMA_ERROR_MODE	モードエラー 以下のいずれかの条件を検出するとモードエラーとなります ・転送モードが不正 ・転送サイズが不正

備考	[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	
	main()
	{
	uint32_t byte;
	dtcDrv-> GetTransferByte(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI, &byte);
	}	

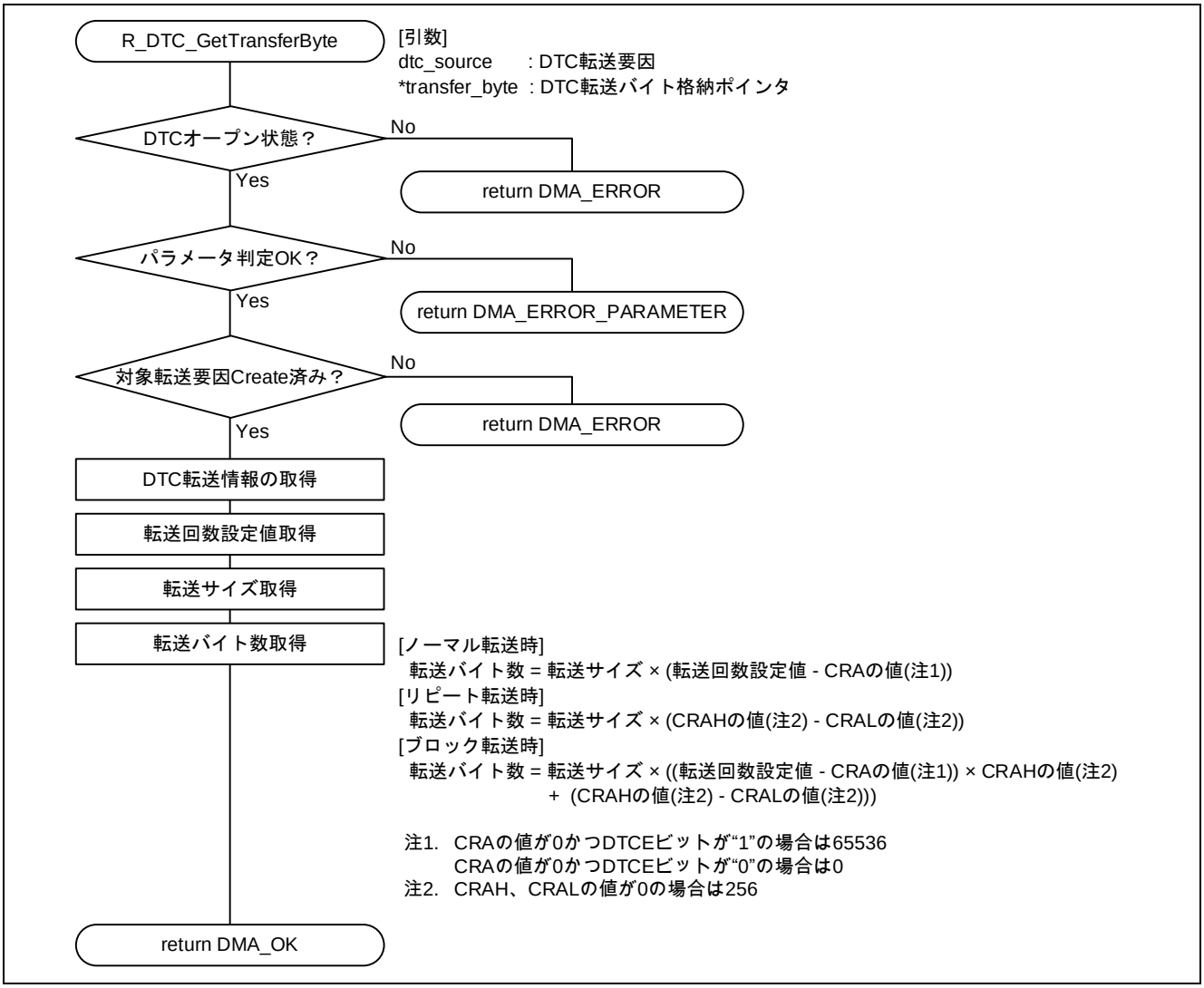

図 4-9 R_DTC_GetTransferByte 関数処理フロー

4.1.10 R_DTC_GetVersion 関数

表 4-11 R_DTC_GetVersion 関数仕様

書式	static uint32_t R_DTC_GetVersion(void)
仕様説明	DTC ドライバのバージョンを取得します
引数	なし
戻り値	DTC ドライバのバージョン

備考	[[インスタンスからの関数呼び出し例]
	// DTC driver instance
	extern DRIVER_DMA Driver_DTC;
	DRIVER_DMA *dtcDrv = &Driver_DTC;
	uint32_t version;
	
	main()
	{
	version = dtcDrv-> GetVersion();
	}

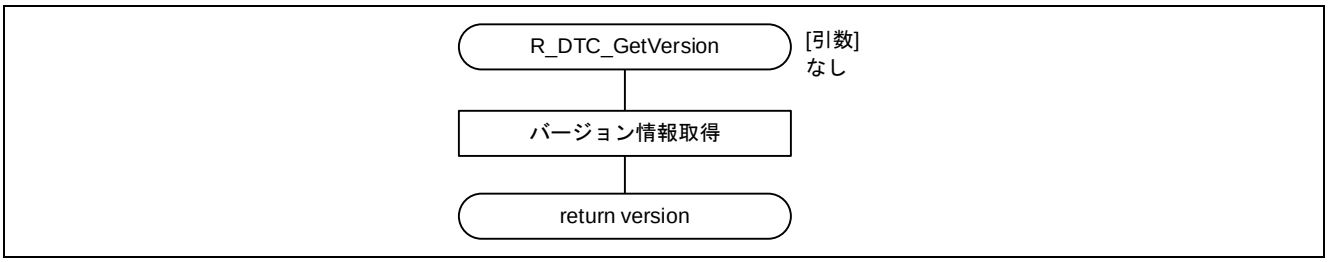

図 4-10 R_DTC_GetVersion 関数処理フロー

4.1.11 dtc_create_param 関数

表 4-12 dtc_create_param 関数仕様

書式	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)
仕様説明	DTC 転送情報に値を設定します
引数	st_dma_transfer_data_t *p_transfer_info : 転送情報格納領域 転送情報を格納するアドレスを指定します
	st_dma_transfer_data_cfg_t const * p_data_cfg : DTC 転送設定 DTC 転送設定を指定します
	uint8_t chain_set_en: チェーン転送設定許可設定 true : チェーン転送分の設定許可 false : チェーン転送分の設定禁止（1 回分の設定のみ実施）
戻り値	DMA_OK　DTC 転送設定成功
	DMA_ERROR_PARAMETER　パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります ・DTC 転送要因に不正な値（SYSTEM_IRQ_EVENT_NUMBER_NOT_USED 以上、もしくは 0 未満）を設定した場合 ・ノーマル転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・リピート転送設定時に、転送回数を 1～256 の範囲外を設定した場合 ・ブロック転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・ブロック転送設定時に、ブロック数を 1～256 の範囲外を設定した場合 ・転送サイズに 16 ビットサイズ転送指定時に、転送元、転送先アドレスが 2 の倍数でない場合 ・転送サイズに 32 ビットサイズ転送指定時に、転送元、転送先アドレスが 4 の倍数でない場合 ・モード指定の予約ビット領域に 1 を設定した場合
	DMA_ERROR_MODE　モードエラー 以下のいずれかの条件を検出するとモードエラーとなります ・転送元アドレッシング指定にオフセット指定（DMA_SRC_ADDR_OFFSET）を設定した場合 ・転送先アドレッシング指定にオフセット指定（DMA_DEST_ADDR_OFFSET）を設定した場合 ・リピート/ブロック領域指定にリピート/ブロック領域なし（DMA_REPEAT_BLOCK_NONE）を指定した場合 ・モード指定に不正な値を設定した場合 ・サイズ指定に不正な値を設定した場合 ・リピート/ブロック領域指定に不正な値を指定した場合 ・チェーン転送指定に不正な値を指定した場合
備考	なし

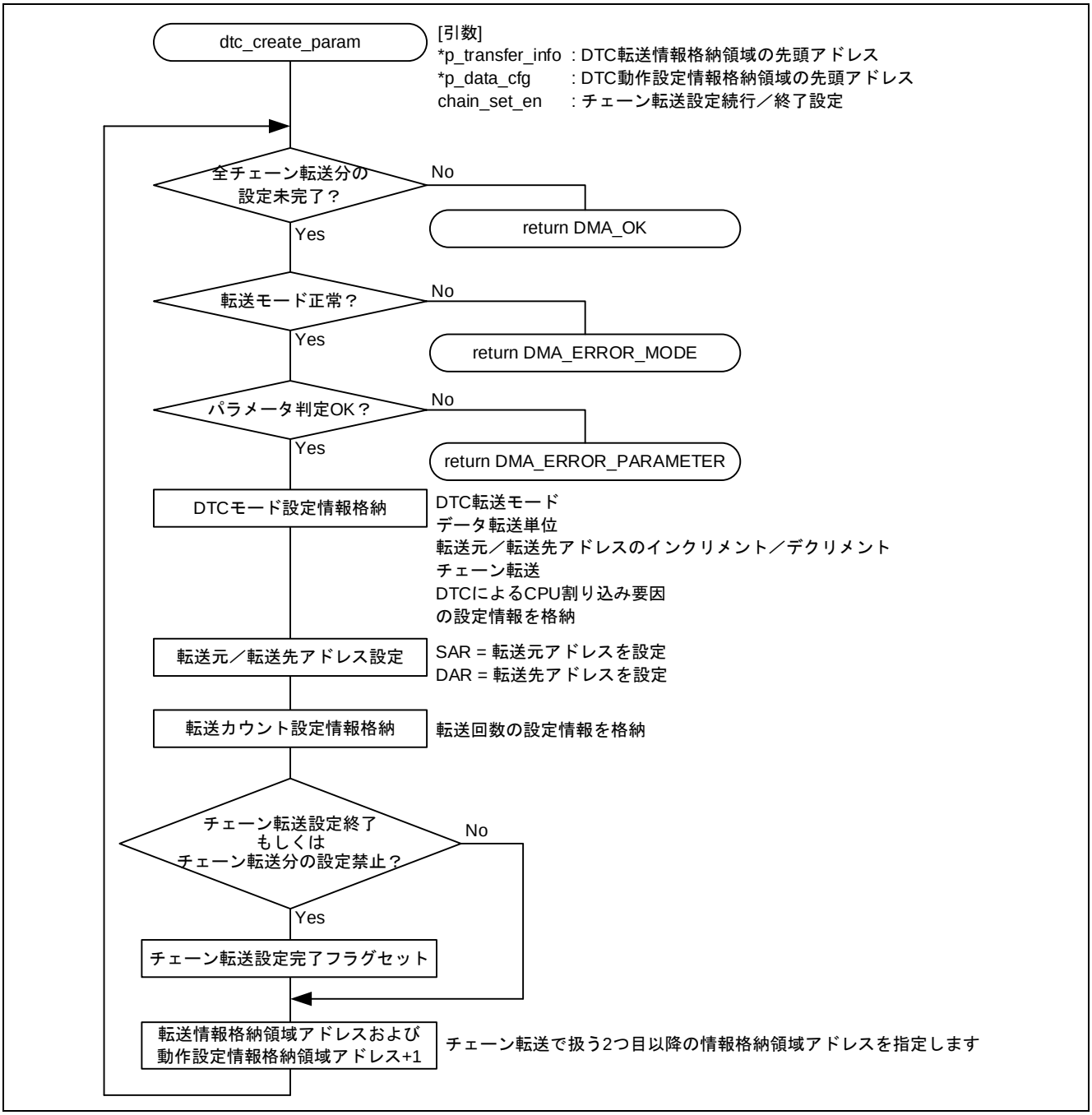

図 4-11 dtc_create_param 関数処理フロー

4.1.12 dtc_comp_interrupt 関数

表 4-13 dtc_comp_interrupt 関数仕様

書式	static void dtc_comp_interrupt(void)
仕様説明	DTC_COMPLETE 割り込み処理
引数	なし
戻り値	なし
備考	なし

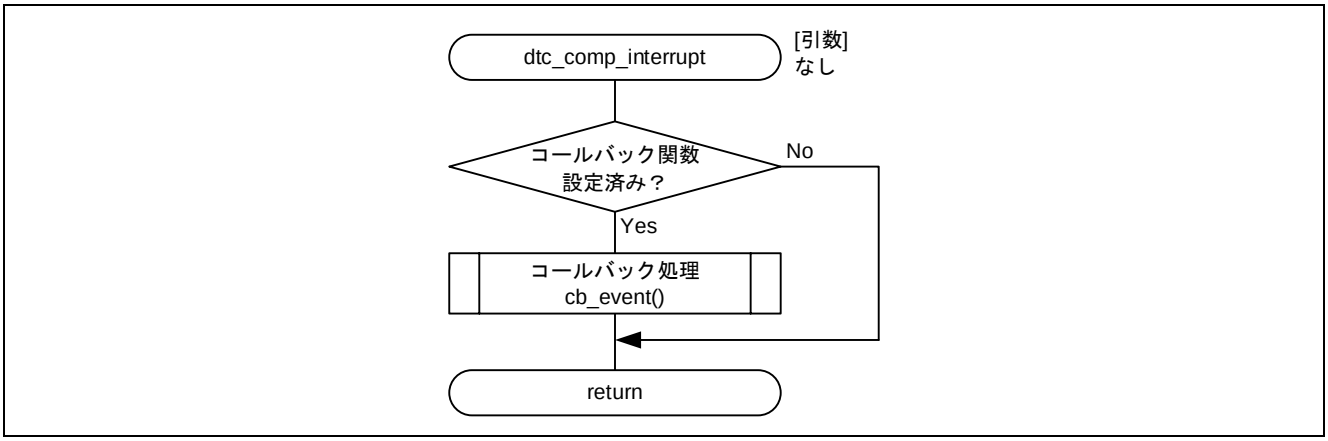

図 4-12 dtc_comp_interrupt 関数処理フロー

4.1.13 dtc_cmd_refresh 関数

表 4-14 dtc_cmd_refresh 関数仕様

書式	static e_dma_err_t dtc_cmd_refresh(st_dma_refresh_data_t const * const p_data)
仕様説明	転送元、転送先、転送回数を再設定します
引数	st_dma_refresh_data_t const * const p_data: 再設定情報 DTC の再設定値を指定します
戻り値	DMA_OK 再設定成功
	DMA_ERROR 再設定失敗 以下のいずれかの条件を検出すると制御コマンド実行失敗となります ・指定した要因の DTC ベクタテーブルが NULL の場合 （指定した要因に対する Create 関数未実行） ・不正なコマンドを実行した場合
	DMA_ERROR_PARAMETER パラメータエラー 以下のいずれかの条件を検出するとパラメータエラーとなります ・ノーマル転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・リピート転送設定時に、転送回数を 1～256 の範囲外を設定した場合 ・ブロック転送設定時に、転送回数を 1～65536 の範囲外を設定した場合 ・ブロック転送設定時に、ブロック数を 1～256 の範囲外を設定した場合
備考	なし

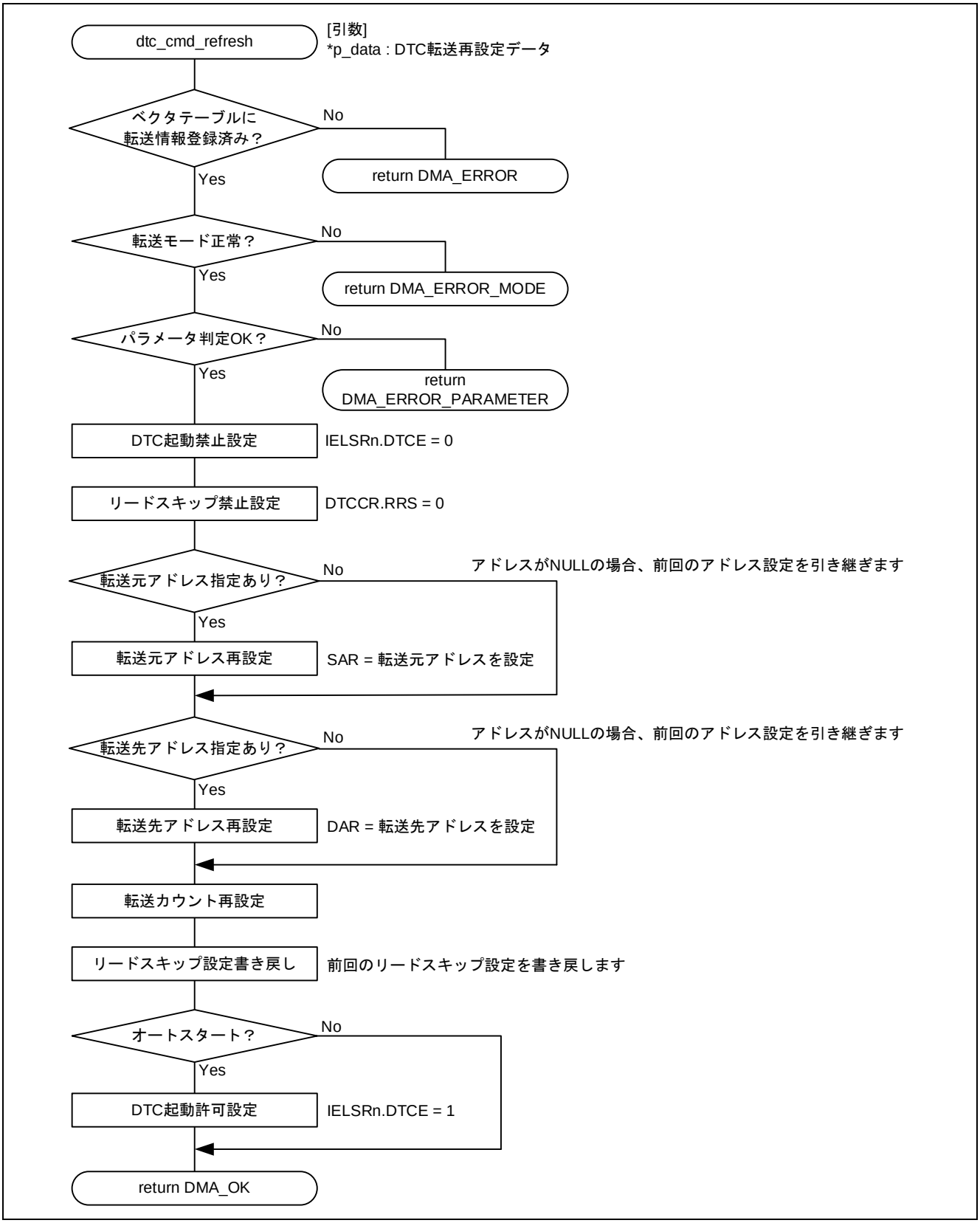

図 4-13 dtc_cmd_refresh 関数処理フロー

4.2 マクロ／型定義

ドライバ内部で使用するマクロ／型定義を示します。

4.2.1 マクロ定義一覧

表 4-15 マクロ定義一覧

定義	値	内容
DMA_FLAG_OPENED	(1U << 0)	オープン済みフラグ定義
DMA_FLAG_INITIALIZED	(1U << 1)	初期化済みフラグ定義
DMA_FLAG_CREATED	(1U << 2)	設定済みフラグ定義
DTC_VERSION_MAJOR	(注)	DTC ドライバ メジャーバージョン情報
DTC_VERSION_MINOR	(注)	DTC ドライバ マイナーバージョン情報
DMA_TRANSFER_MODE_MASK	(0xC000U)	動作モード判定マスク定義
DMA_SIZE_MASK	(0x3000U)	転送サイズ判定マスク定義
DMA_SRC_ADDRESSING_MASK	(0x0C00U)	転送元アドレッシング判定マスク定義
DMA_DEST_ADDRESSING_MASK	(0x000CU)	転送先アドレッシング判定マスク定義
DMA_REPEAT_BLOCK_MASK	(0x0011U)	リピート／ブロック領域判定マスク定義
DMA_INT_SELECT_MASK	(0x0020U)	割り込み発生タイミング判定マスク定義
DMA_CHAIN_MODE_MASK	(0x00C0U)	チェーンモード判定マスク定義
DMA_INT_MASK	(0x001F)	割り込み要因判定マスク定義
DMA_MODE_RASERVED	(0x0302)	モード未使用領域判定マスク
DMA_ACTIVE_DMAC0	(0x0001)	DMAC0 アクティブ定義
DMA_ACTIVE_DMAC1	(0x0002)	DMAC1 アクティブ定義
DMA_ACTIVE_DMAC2	(0x0004)	DMAC2 アクティブ定義
DMA_ACTIVE_DMAC3	(0x0008)	DMAC3 アクティブ定義
DMA_ACTIVE_DTC	(0x8000)	DTC アクティブ定義
DMA_ACTIVE_MASK	(0x800F)	DMAC、DTC アクティブ判定マスク定義
DMA_ACTIVE_DMAC_MASK	(0x000F)	DMAC アクティブ判定マスク定義
DTC_PRV_CHAIN_ENABLE	(0x00800000)	チェーン転送許可／禁止設定用定義
DTC_PRV_MIN_COUNT_VAL	(1)	転送回数下限値
DTC_PRV_MAX_16BITS_COUNT_VAL	(65536)	16 ビット転送カウント指定時の上限値
DTC_PRV_MAX_8BITS_COUNT_VAL	(256)	8 ビット転送カウント指定時の上限値
DTC_PRV_IELSR_DTCE	(0x01000000)	DTC 起動許可／禁止設定用定義
DTC_PRV_COMPLETE_INT_IESR_VAL	(0x00000003)	DTC 転送完了割り込みを ICU イベントリンクに設定するための定義
DTC_PRV_MASK_MRA_MD	(0xC0000000)	MRA_MD マスク定義
DTC_PRV_MRA_MD_NORMAL	(0x00000000)	ノーマル転送判定定義
DTC_PRV_MRA_MD_REPEAT	(0x40000000)	リピート転送判定定義
DTC_PRV_MRA_MD_BLOCK	(0x80000000)	ブロック転送判定定義
DTC_PRV_MASK_MRA_SZ	(0x30000000)	MRA_SZ マスク定義
DTC_PRV_MRA_SZ_BYTE	(0x00000000)	転送サイズ 1 バイト判定定義
DTC_PRV_MRA_SZ_WORD	(0x10000000)	転送サイズ 2 バイト判定定義
DTC_PRV_MRA_SZ_LONG	(0x20000000)	転送サイズ 4 バイト判定定義

注　本ドライバのバージョンに応じた値が設定されています
例）ドライババージョン 1.01 の場合
DTC_VERSION_MAJOR (1)
DTC_VERSION_MINOR (01)

4.3 構造体定義

4.3.1 st_dtc_resources_t 構造体

DTC のリソースを構成する構造体です。

表 4-16 st_dtc_resources_t 構造体

要素名	型	内容
*reg	volatile DTC_Type	対象の DTC レジスタを示します
lock_id	e_system_mcu_lock_t	DTC ロック ID
mstp_id	e_lpm_mstp_t	DTC モジュールストップ ID
*info	st_dtc_mode_info_t	DTC 状態情報

4.3.2 st_dtc_mode_info_t 構造体

DTC の状態を管理するための構造体です。

表 4-17 st_dtc_mode_info_t 構造体

要素名	型	内容
cb_event	dma_cb_event_t	イベント発生時のコールバック関数 NULL の場合はコールバック関数実行しない
read_skip	uint8_t	リードスキップ設定 0： リードスキップ禁止 1： リードスキップ許可
flags	uint8_t	ドライバ設定フラグ b0： ドライバオープン状態(0:クローズ、1:オープン) b1： ドライバ初期化状態(0:未初期化、1:初期化済み) b2 : DTC 設定済み状態(0:未設定、1:設定済み)

4.4 外部関数の呼び出し

DTC ドライバ API から呼び出される外部関数を示します。

表 4-18 DTC ドライバ API から呼び出される外部関数と呼び出し条件

API	呼び出し関数	条件（注）
Open	R_SYS_ResourceLock	なし
	R_LPM_ModuleStart	なし
	R_SYS_ResourceUnlock	モジュールストップ解除失敗時
Close	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_SYS_ResourceUnlock	なし
	R_LPM_ModuleStop	全 DMAC／DTC ドライバ未使用時
Create	-	-
Control	-	-
InterruptEnable	R_SYS_IrqEventLinkSet	なし
	R_NVIC_SetPriority	なし
	R_NVIC_GetPriority	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
	R_NVIC_EnableIRQ	なし
InterruptDisable	R_NVIC_DisableIRQ	なし
	R_SYS_IrqStatusClear	なし
	R_NVIC_ClearPendingIRQ	なし
GetState	-	-
ClearState	-	-
GetTransferByte	-	-
GetVersion	-	-

注　条件なしの場合でも、パラメータチェックによるエラー終了発生時には呼び出し関数が実行されない可能性があります。

5. 使用上の注意

5.1 引数について

各関数の引数で指定する構造体の内、使用しない要素には 0 を設定してください。

5.2 NVIC への DTC 転送完了割り込み（DTC_COMPLETE）登録

DTC 転送完了割り込み（DTC_COMPLETE）を使用する場合は、r_system_cfg.h にて NVIC 登録してください。NVIC に DTC 転送完了割り込みが登録されていない場合、DTC_InterruptEnable 関数実行時に DMA_ERROR_SYSTEM_SETTING が戻ります。

DTC 転送完了割り込みの登録例を図 5-1 に示します。

```
・・・

#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 */

・・・
```

図 5-1 NVIC への割り込み登録例

5.3 DTC 転送禁止状態での DTC 転送要因入力時の動作について

DTC 転送禁止状態で DTC 転送要因が入力された場合、転送要因の割り込み要求が発生します。NVIC にて割り込み許可状態にしていた場合、DTC 転送は行われず転送要因の割り込みが発生します。

NVIC にて割り込み禁止状態にしていた場合は、NVIC 側の割り込みが保持されたままとなります。DTC 転送を許可にしても、NVIC 側の割り込み要求がクリアされるまで、DTC 転送要求は受け付けません。NVIC 側の割り込みを許可状態にして NVIC 割り込み処理を実行するか、NVIC 側の割り込み要求をクリアしてください。

DTC 転送禁止状態で DTC 転送が発生した場合の動作例

[プログラム処理例]

・DTC 転送を以下の設定で動作

- 動作モード ： ノーマル転送
- 転送回数 ：2 回
- 割り込み発生タイミング ： 指定されたデータ転送の終了時、CPU への割り込み要求が発生
- 転送要因 ：AGT0 の AGTI 割り込みを使用

・AGTI 割り込みにて DMA_CMD_REFRESH コマンドを使用して、DTC の再設定を行う

AGTI 割り込みにて、DMA_CMD_REFRESH コマンドによる再設定を実行するまえに AGTI 割り込みが発生した場合の動作例を図 5-2 に示します。

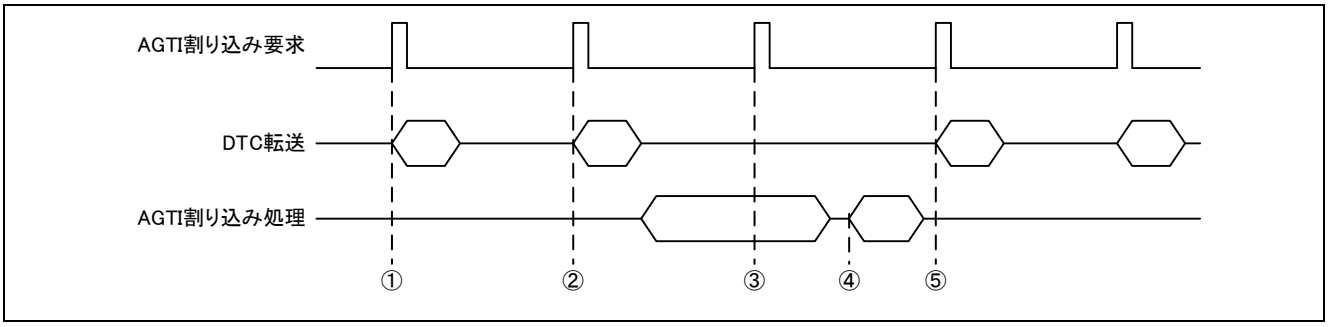

図 5-2 DTC 禁止状態で DTC 転送要因が発生した場合の動作例

① AGTI 割り込み要求にて DTC 転送発生。

② 再度 AGTI 割り込み要求にて DTC 転送が発生。指定回数の DTC 転送完了で AGTI 割り込み処理が実行。

③ AGTI 割り込み処理実行中（DMA_CMD_REFRESH コマンド実行前）に AGTI 割り込み要求が発生すると、NVIC 側に割り込み要求が保持されます。

④ AGTI 割り込みが終了後、③で保持されていた割り込み要求により、再度 AGTI 割り込み処理が実行されます。

⑤ NVIC 側の割り込み要求がクリアされた状態で AGTI 割り込み要求が発生すると、DTC 転送が実行されます。

上記の④で AGTI 割り込みが発生する現象が問題となる場合、DMA_CMD_REFRESH コマンド実行後、AGTI 割り込み要求をクリアしてください。AGTI 割り込み要求クリア例を図 5-3 に示します。

```
/* AGT0 AGTI 割り込みコールバック処理 */
void cb_agt0_agti (void)
{
    st_dma_refresh_data_t arg;

    /* AGTI 割り込み要因の転送設定を更新し、DTC 動作許可にする */
    arg.act_src        = SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI;
    arg.chain_transfer_nr = 0;                  /* チェーン転送未使用のため 0 設定 */
    arg.src_addr       = (uint32_t)(&source_ref); /* 転送元更新 */
    arg.dest_addr      = (uint32_t)(&dest_ref);   /* 転送先更新 */
    arg.transfer_count  = 2;                   /* 転送回数の更新 */
    arg.block_size      = 0;                  /* ブロックサイズの更新(ブロック転送のみ有効) */
    arg.auto_start      = true;                /* 更新コマンド実行後、DTC 許可 */
    (void)dtcDrv->Control(DMA_CMD_REFRESH_DATA, &arg);

    /* IELSRn レジスタの IR フラグをクリア */
    R_SYS_IrqStatusClear(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI);
    /* NVIC 側の割り込み要求をクリア */
    R_NVIC_ClearPendingIRQ(SYSTEM_CFG_EVENT_NUMBER_AGT0_AGTI);
}
```

図 5-3 DTC 転送要因の割り込み要求クリア手順例

5.4 転送元アドレス、転送先アドレスの設定値の制限

Create 関数、または Create 関数の DTC 転送再設定コマンド（DMA_CMD_REFRESH_DATA）、DTC 転送情報強制変更コマンド（DMA_CMD_CHANGING_DATA_FORCIBLY_SET）での転送元アドレス、転送先アドレスの設定値は、転送サイズが 16 ビットサイズ転送のときは 2 の倍数、32 ビットサイズ転送の場合は 4 の倍数になるよう設定してください。

6. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

RE01 1500KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0796
RE01 256KB グループ ユーザーズマニュアル　ハードウェア編 R01UH0894

（最新版をルネサス エレクトロニクスホームページから入手してください。）

RE01Group CMSIS Package スタートアップガイド

RE01 1500KB、256KB グループ CMSIS パッケージを用いた開発スタートアップガイド R01AN4660

（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境

（最新版をルネサス エレクトロニクスホームページから入手してください。）

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Oct.10.19	ー	初版発行
1.01	Dec.16.2019	ー	256KB グループに対応
1.02	Feb.18.2020	プログラム (256KB, 1500KB)	r_dtc_cfg.h の RAM/ROM 配置不備を修正 R_DTC_GetTransferByte 関数用の RAM 配置定義 (DTC_CFG_R_DTC_GET_TRANSFER_BYTE)を追加

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2020 Renesas Electronics Corporation. All rights reserved.