RE01 1500KB、256KB グループ

CMSIS ドライバ DMAC 仕様書

要旨

本書では、RE01 1500KB、256KB グループ CMSIS software package の DMAC ドライバ（以下、DMAC 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 4

2. ドライバ構成 4
2.1 ファイル構成 4
2.2 ドライバ API 5
2.3 DMAC 起動要因の設定 9
2.4 DMA 割り込み 9
2.5 DMAC 割り込みの NVIC 登録 14
2.6 マクロ／型定義 14
2.6.1 DMAC コントロールコード定義 14
2.6.2 DMA 転送モード定義 15
2.6.3 DMA 転送割り込み要因定義 16
2.6.4 DMAC ステータスフラグクリア対象定義 16
2.7 構造体定義 17
2.7.1 st_dma_transfer_data_cfg_t 構造体 17
2.7.2 st_dma_refresh_data_t 構造体 17
2.7.3 st_dma_state_t 構造体 18
2.8 状態遷移 19

3. ドライバ動作説明 21
3.1 ノーマル転送モード 21
3.2 リピート転送モード 23
3.3 ブロック転送モード 25
3.4 Control 関数による DMAC 機能制御 27
3.4.1 DMA 転送開始コマンド（DMA_CMD_START） 27
3.4.2 DMA 転送停止コマンド（DMA_CMD_STOP） 27
3.4.3 DMAC 起動要因禁止コマンド（DMA_CMD_ACT_SRC_DISABLE） 28
3.4.4 DMA ソフトウェア起動コマンド（DMA_CMD_SOFTWARE_TRIGGER） 28
3.4.5 DMA ソフトウェア起動ビット自動クリア機能設定コマンド（DMA_CMD_AUTO_CLEAR_SOFT_REQ） 28
3.4.6 DMA 転送再設定コマンド（DMA_CMD_REFRESH_DATA） 29
3.4.7 DMA 転送再設定拡張コマンド（DMA_CMD_REFRESH_EXTRA） 30
3.5 DMA 転送バイト数の取得 33
3.6 コンフィグレーション 34
3.6.1 パラメータチェック 34
3.6.2 DMACn_INT 割り込み優先レベル(n=0～3) 34
3.6.3 関数の RAM 配置 35

4. ドライバ詳細情報 36
4.1 関数仕様 36
4.1.1 R_DMAC_Open 関数 36
4.1.2 R_DMAC_Close 関数 38
4.1.3 R_DMAC_Create 関数 39
4.1.4 R_DMAC_Control 関数 42
4.1.5 R_DMAC_InterruptEnable 関数 47

4.1.6 R_DMAC_InterruptDisable 関数 49
4.1.7 R_DMAC_GetState 関数 50
4.1.8 R_DMAC_ClearState 関数 51
4.1.9 R_DMAC_GetTransferByte 関数 52
4.1.10 R_DMAC_GetVersion 関数 54
4.1.11 dmac_active_set 関数 55
4.1.12 dmac_active_clear 関数 56
4.1.13 dmac_cmd_refresh 関数 57
4.1.14 dmac_interrupt_handler 関数 59
4.2 マクロ／型定義 60
4.2.1 マクロ定義一覧 60
4.3 構造体定義 61
4.3.1 st_dmac_resources_t 構造体 61
4.3.2 st_dmac_mode_info_t 構造体 61
4.4 外部関数の呼び出し 62

5. 使用上の注意 63
5.1 引数について 63
5.2 DMA 割り込み設定について 63
5.3 DMA 割り込み要求発生時の動作について 63
5.4 NVIC への DMAC 割り込み登録 63
5.5 転送元アドレス、転送先アドレスの設定値の制限 63

6. 参考ドキュメント 64

改訂記録 65

1. 概要

DMAC ドライバは、RE01 1500KB、256KB グループで DMA 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

DMAC 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_dmac_api.c”、 “r_dma_common_api.h”、” r_dmac_api.h”、” r_dmac_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 DMAC ドライバ 各ファイルの役割

ファイル名	内容
r_dmac_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 DMAC 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_dmac_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 DMAC 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_dmac_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

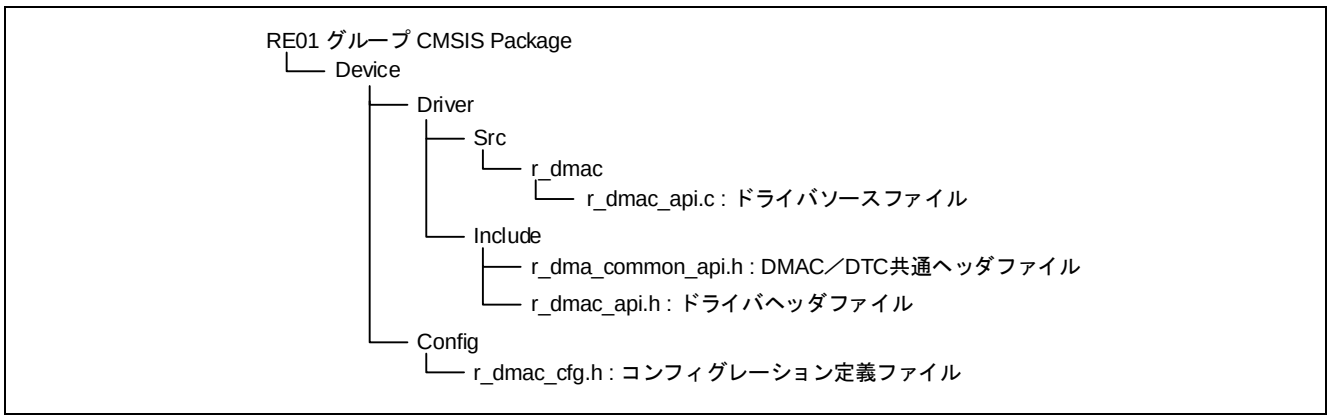

図 2-1 DMAC ドライバファイル構成

2.2 ドライバ API

DMAC ドライバはチャネル別に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。DMAC ドライバのインスタンス一覧を表 2-2 に、インスタンスに含まれる API を表 2-3 に、DMAC ドライバへのアクセス例を図 2-3～図 2-5 に示します。

表 2-2 DMAC ドライバのインスタンス一覧

インスタンス	内容
DRIVER_DMA Driver_DMAC0	DMAC0 を使用する場合のインスタンス
DRIVER_DMA Driver_DMAC1	DMAC1 を使用する場合のインスタンス
DRIVER_DMA Driver_DMAC2	DMAC2 を使用する場合のインスタンス
DRIVER_DMA Driver_DMAC3	DMAC3 を使用する場合のインスタンス

```
#include "r_dmac_api.h"

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

図 2-2 DMAC0 ドライバ インスタンス宣言例

表 2-3 DMAC ドライバ API

API	内容	参照
Open	DMAC 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	DMAC 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	DMA 転送の設定を行います	4.1.3
Control	DMAC 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	DMA 割り込みを設定します 割り込み要因の設定と割り込み許可を行います	4.1.5
InterruptDisable	DMA 割り込みを禁止します	4.1.6
GetState	DMAC の状態を取得します	4.1.7
ClearState	DMAC ドライバの割り込みフラグをクリアします	4.1.8
GetTransferByte	DMA 転送バイト数を取得します	4.1.9
GetVersion	DMAC ドライバのバージョンを取得します	4.1.10

表 2-4 DMAC 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	DMA 転送を開始します
DMA_CMD_STOP	DMA 転送を停止します
DMA_CMD_SOFTWARE_TRIGGER	ソフトウェアトリガによる DMA 転送要求を発行します
DMA_CMD_AUTO_CLEAR_SOFT_REQ	ソフトウェアトリガによる DMA 転送要求自動クリア機能の有効、無効を設定します
DMA_CMD_ACT_SRC_DISABLE	DMA 転送を禁止にし、転送要因を解除します
DMA_CMD_REFRESH_DATA	DMA 転送設定（転送元アドレス、転送先アドレス、転送回数）を再設定します
DMA_CMD_REFRESH_EXTRA	DMA 転送設定（転送元アドレス、転送先アドレス、転送回数、オフセット値）を再設定します また、DMA 転送停止から DMA 転送再設定までの間に発生した DMAC 起動要求を保持するか選択できます

```
#include "r_dmac_api.h"

static void callback(void);

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* ノーマルモード、8 ビット、転送元インクリメント、転送先インクリメント */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                          /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                                    /* DMAC 設定起動要因 : ソフトウェアトリガ */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback);   /* DMA 転送完了割り込み許可 */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                  /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    /* すべての転送完了した際の処理を記述 */
}
```

図 2-3DMAC ドライバへのアクセス例（ノーマル転送）

```
#include "r_dmac_api.h"

static void callback(void);

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* リピート転送モード、8 ビット、転送元インクリメント、転送先インクリメント */
    config.mode = (DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;                              /* 転送サイズ 5 */
    config.block_size = 3;                                 /* リピートサイズ 3 */
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                        /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                              /* DMAC 設定起動要因 : ソフトウェアトリガ */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback);   /* DMA 転送完了割り込み許可 */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                 /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1);
}

/******************************************************************************************************
* callback function
******************************************************************************************************/
static void callback(void)
{
    /* すべての転送完了（転送サイズ(5)×リピートサイズ(3)回転送完了）した際の処理を記述 */
}
```

図 2-4 DMAC ドライバへのアクセス例（リピート転送）

```
#include "r_dmac_api.h"

static void callback(void);

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* 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 */
    config.mode = (DMA_MODE_BLOCK | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;                              /* 転送サイズ 5 */
    config.block_size = 3;                                 /* ブロックサイズ 3 */
    config.offset = 0;
    config.src_extended_repeat = 0;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                      /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                            /* DMAC 設定起動要因 : ソフトウェアトリガ */
    (void)dmac0Drv->InterruptEnable(DMA_INT_COMPLETE, callback);   /* DMA 転送完了割り込み許可 */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);               /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1);
}

/*****************************************************************************************************
* callback function
*****************************************************************************************************/
static void callback(void)
{
    /* すべての転送完了（ブロック数 5×転送回数 3 回）した際の処理を記述 */
}
```

図 2-5 DMAC ドライバへのアクセス例（ブロック転送）

2.3 DMAC 起動要因の設定

DMAC 起動要因は、Create 関数の第 1 引数で指定します。ソフトウェアトリガによる DMAC の起動を行う場合は、DMAC_TRANSFER_REQUEST_SOFTWARE を指定してください。任意の割り込み要求による DMAC の起動を行う場合、リンクするイベント信号の番号を指定します。詳細は「RE01 1500KB グループ　ユーザーズマニュアル　ハードウェア編(r01uh0796)」もしくは「RE01 256KB グループ ユーザーズマニュアル　ハードウェア編(r01uh0894)」の「DMAC イベントリンク設定レジスタ n（DELSRn）」を確認してください。また、起動要因の割り込みビットを許可に設定してください。

例）ADC140_ADI イベント（イベント番号 23H）で DMAC を起動させる場合
```
dmac0Drv->Create(0x23, &config);
```

2.4 DMA 割り込み

DMA 割り込みを使用する場合、InterruptEnable 関数にて使用する DMA 割り込みを許可にしてください。InterruptEnable 関数の第 1 引数に指定する DMA 転送割り込み要因定義一覧を表 2-5 に示します。

表 2-5 DMA 転送割り込み要因定義一覧

定義	内容
DMA_INT_COMPLETE	DMA 転送終了
DMA_INT_SRC_OVERFLOW	転送元アドレス拡張リピートエリアオーバフロー
DMA_INT_DEST_OVERFLOW	転送先アドレス拡張リピートエリアオーバフロー
DMA_INT_REPEAT_END	DMA 転送リピートサイズ終了
DMA_INT_ESCAPE_END	DMA 転送エスケープ終了

DMA 転送割り込み要因定義を OR 演算で組み合わせることで、複数の割り込み要因を許可にすることができます。

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

割り込み要求発生時にコールバック関数を呼び出す場合は、第 2 引数にコールバック関数を指定してください。コールバックを使用せず、GetStatus 関数で DMA 割り込み発生フラグをポーリングする場合、第 2 引数に NULL(0)を指定してください。

転送元アドレス拡張リピートエリアオーバフロー割り込み、転送先アドレス拡張リピートエリアオーバフロー割り込み、DMA 転送リピートサイズ終了割り込みによってコールバック関数を呼び出す場合は、同時に転送エスケープ終了割り込みを許可にする必要があります。転送エスケープ終了割り込みを許可にしていない場合、コールバック関数は呼び出されません。DMA 割り込み出力の概略論理図を図 2-6 に示します。

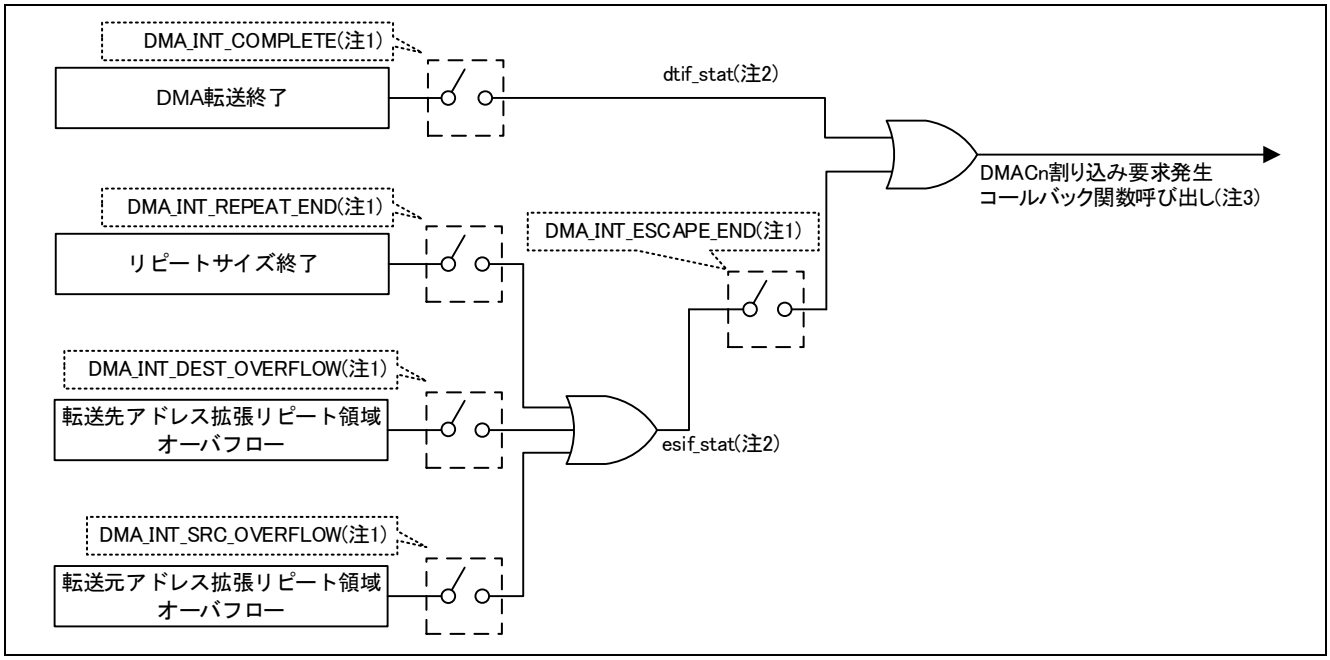

図 2-6 割り込み出力の概略論理図

注1. InterruptEnable 関数の引数に指定して割り込みを許可にした場合、割り込み要求が発生します。

注2. 該当の割り込みを許可状態にしていた場合、GetState 関数にて割り込み発生状況を取得できます。第 2 引数に指定した st_dma_state_t 型構造体変数の esif_state メンバに転送エスケープ終了割り込みの発生状況、dtif_stat メンバに転送終了割り込みの発生状況が格納されます。

注3. InterruptEnable 関数実行時、第 2 引数に NULL を指定していた場合、コールバックは発生しません。

InterruptEnable 関数にて、転送元アドレス拡張リピートエリアオーバフロー、転送先アドレス拡張リピートエリアオーバフロー、DMA 転送リピートサイズ終了割り込みを許可にした場合、割り込み要求が発生すると DMA 転送が停止(DMCNT.DTE = 0)します。DMA 転送を継続する場合は、割り込み要求発生後、Control 関数にて DMA_CMD_START コマンドを実行してください。

コールバック関数を使用する場合の DMA 割り込み使用例を図 2-7 に、コールバック関数を使用しない場合の DMA 割り込み使用例を図 2-8 に、複数の DMA 割り込みを許可にした場合の使用例を図 2-9 に示します。

```
#include "r_dmac_api.h"

static void callback(void);

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* ノーマルモード、8 ビット、転送元インクリメント、転送先インクリメント */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                          /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                  /* DMAC 設定起動要因 : ソフトウェアトリガ */
    /* 転送元アドレス拡張リピートエリアオーバフロー割り込み許可
       コールバックを発生させる場合は、必ず DMA_INT_ESCAPE_END を OR 演算で組み合わせて指定 */
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW  |  DMA_INT_ESCAPE_END, callback);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    /* 転送元アドレス拡張リピートエリアオーバフロー割り込み発生 */
    /* DMA 転送が停止(DMCNT.DTE = 0)しているため、転送を継続する場合は START コマンドを実行 */
    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* DMA 転送開始 */
}
```

図 2-7 コールバック関数を使用して DMA 転送を継続する場合の使用例

```
#include "r_dmac_api.h"

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* ノーマルモード、8 ビット、転送元インクリメント、転送先インクリメント */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                          /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                /* DMAC 設定起動要因 : ソフトウェアトリガ */
    /* 転送元アドレス拡張リピートエリアオーバフロー割り込み許可*/
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW , NULL);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                   /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1)
    {
        /* DMAC の状態を取得 */
        dmac0Drv->GetState(dmac0_state);

        /* 転送元アドレス拡張リピートエリアオーバフロー割り込み要求あり？ */
        if (1 == dmac0_state.esif_stat)
        {
            /* DMA 転送が停止(DMCNT.DTE = 0)しているため、
              転送を継続する場合は START コマンドを実行 */
            (void)dmac0Drv->Control(DMA_CMD_START, NULL);               /* DMA 転送開始 */
        }
    }
}
```

図 2-8 コールバック関数を使用せずに DMA 転送を継続する場合の使用例

```
#include "r_dmac_api.h"

static void callback(void);

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

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

main()
{
    st_dma_transfer_data_cfg_t config;

    /* ノーマルモード、8 ビット、転送元インクリメント、転送先インクリメント */
    config.mode = (DMA_MODE_NORMAL | DMA_SIZE_BYTE | DMA_SRC_INCR | DMA_DEST_INCR);
    config.src_addr = (uint32_t)(&tx_data [0]);
    config.dest_addr = (uint32_t)(&rx_data);
    config.transfer_count = 5;
    config.block_size = 0;
    config.offset = 0;
    config.src_extended_repeat = 1;
    config.dest_extended_repeat = 0;

    (void)dmac0Drv->Open();                                          /* DMAC ドライバオープン */
    (void)dmac0Drv->Create(DMAC_TRANSFER_REQUEST_SOFTWARE, &config);
                                                                    /* DMAC 設定起動要因 : ソフトウェアトリガ */
    /* 転送元アドレス拡張リピートエリアオーバフロー割り込みと DMA 転送終了割り込みを許可
      転送元アドレス拡張リピートエリアオーバフロー割り込みで
      コールバックを発生させる場合は、必ず DMA_INT_ESCAPE_END を OR 演算で組み合わせて指定 */
    (void)dmac0Drv->InterruptEnable(DMA_INT_SRC_OVERFLOW | DMA_INT_ESCAPE_END | DMA_INT_COMPLETE,
                                    callback);

    (void)dmac0Drv->Control(DMA_CMD_START, NULL);                    /* DMA 転送開始 */
    (void)dmac0Drv->Control(DMA_CMD_SOFTWARE_TRIGGER, NULL); /* ソフトウェアトリガ発行 */

    while(1);
}

/***********************************************************************************************
* callback function
***********************************************************************************************/
static void callback(void)
{
    /* DMAC の状態を取得 */
    dmac0Drv->GetState(dmac0_state);
    /* 転送元アドレス拡張リピート領域オーバフロー割り込み発生？） */
    if (1 == dmac0_state.esif_stat)
    {
        /* DMA 転送が停止(DMCNT.DTE = 0)しているため、
        転送を継続する場合は START コマンドを実行 */
        (void)dmac0Drv->Control(DMA_CMD_START, NULL);                /* DMA 転送開始 */
    }
    /* 転送終了割り込みの発生？ */
    if (1 == dmac0_state.dtif_stat)
    {
        /* DMA 転送が完了した場合の処理を記載 */
    }
}
```

図 2-9 複数の割り込みを許可にする場合の使用例

2.5 DMAC 割り込みの NVIC 登録

InterruptEnable 関数にて DMA 割り込みを許可にする場合、使用するチャネルの DMACn_INT(n=0～3)を、r_system_cfg.h にてネスト型ベクタ割り込みコントローラ（以下、NVIC）に登録する必要があります。詳細は「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」を参照してください。DMA 割り込みの登録例を図 2-10 に示します。

```
・・・

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

図 2-10 r_system_cfg.h での NVIC への割り込み登録例(DMAC0 使用時)

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

2.6.1 DMAC コントロールコード定義

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

表 2-6 DMAC コントロールコード一覧

定義	値	内容
DMA_CMD_START	(0x00)	DMA 転送開始
DMA_CMD_STOP	(0x01)	DMA 転送禁止
DMA_CMD_ACT_SRC_ENABLE	(0x02)	使用禁止(注)
DMA_CMD_ACT_SRC_DISABLE	(0x03)	DMAC 起動要因禁止
DMA_CMD_DATA_READ_SKIP_ENABLE	(0x04)	使用禁止(注)
DMA_CMD_CHAIN_TRANSFER_ABORT	(0x05)	使用禁止(注)
DMA_CMD_CHANGING_DATA_FORCIBLY_SET	(0x06)	使用禁止(注)
DMA_CMD_SOFTWARE_TRIGGER	(0x07)	ソフトウェアトリガによる DMA 転送要求
DMA_CMD_AUTO_CLEAR_SOFT_REQ	(0x08)	DMA ソフトウェア起動ビット自動クリア機能設定
DMA_CMD_REFRESH_DATA	(0x09)	DMA 転送再設定 (DMAC 起動要求保持選択、オフセット再設定なし)
DMA_CMD_REFRESH_EXTRA	(0x0A)	DMA 転送再設定（拡張機能） (DMAC 起動要求保持選択、オフセット再設定あり)

注　DTC 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。

2.6.2 DMA 転送モード定義

DMAC 転送モード定義は、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も設定してください。

例）リピート転送、8 ビットサイズ転送、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

```
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;
```

DMAC 転送モード定義の構成を図 2-11 に、各設定内容の詳細を表 2-7～表 2-11 に示します。

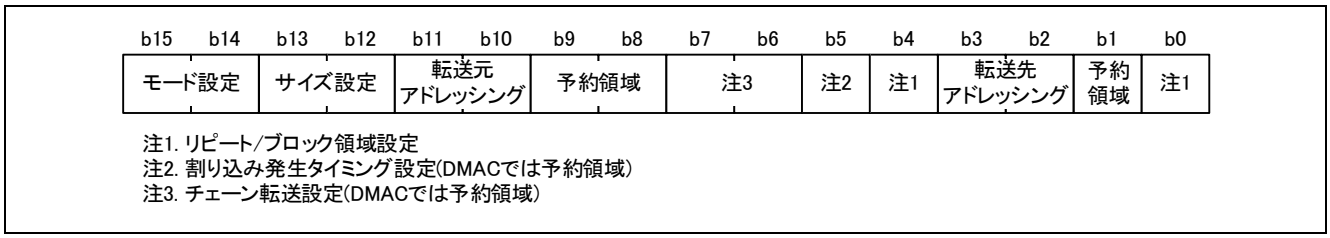

図 2-11 DMAC 転送モード定義の構成

表 2-7 DMA 転送モード定義一覧

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 2-8 DMA 転送サイズ定義一覧

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 2-9 DMA 転送リピート領域定義一覧

定義	値	内容
DMA_REPEAT_BLOCK_DEST	(0x0000U)	転送先領域をリピートする
DMA_REPEAT_BLOCK_SRC	(0x0010U)	転送元領域をリピートする
DMA_REPEAT_BLOCK_NONE	(0x0001U)	転送領域をリピートしない

表 2-10 DMA 転送元アドレスモード一覧

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)	転送元アドレスにオフセット値を設定する

表 2-11 DMA 転送先アドレスモード一覧

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)	転送先アドレスにオフセット値を設定する

2.6.3 DMA 転送割り込み要因定義

InterruptEnable 関数で指定する割り込み要因の定義です。

表 2-12 DMA 転送割り込み要因定義一覧

定義	値	内容
DMA_INT_COMPLETE	(1U<<0)	DMA 転送終了
DMA_INT_SRC_OVERFLOW	(1U<<1)	転送元アドレス拡張リピートエリアオーバフロー
DMA_INT_DEST_OVERFLOW	(1U<<2)	転送先アドレス拡張リピートエリアオーバフロー
DMA_INT_REPEAT_END	(1U<<3)	DMA 転送リピートサイズ終了
DMA_INT_ESCAPE_END	(1U<<4)	DMA 転送エスケープ終了

2.6.4 DMAC ステータスフラグクリア対象定義

ClearState 関数で指定する割り込みフラグおよび転送要求フラグの定義です。

表 2-13 DMAC 割り込みフラグおよび転送要求フラグクリア対象定義一覧

定義	値	内容
DMA_CLR_STATE_ESIF	(1U<<1)	転送エスケープ終了割り込みフラグ
DMA_CLR_STATE_DTIF	(1U<<2)	転送終了割り込みフラグ
DMA_CLR_STATE_SOFT_REQ	(1U<<3)	ソフトウェアトリガによる転送要求フラグ

2.7 構造体定義

DMAC 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。

2.7.1 st_dma_transfer_data_cfg_t 構造体

Create 関数で DMAC 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、リピート領域、転送元／転送先アドレスモードを、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	オフセット値を指定します
src_extended_repeat	uint8_t	転送元拡張リピート領域を指定します
dest_extended_repeat	uint8_t	転送先拡張リピート領域を指定します
*p_transfer_data	void	使用禁止(注 2)

注　DTC ドライバでのみ使用可。本要素は無視されます。

2.7.2 st_dma_refresh_data_t 構造体

Control 関数で DMA 転送再設定コマンド(DMA_CMD_REFRESH_DATA、DMA_CMD_REFRESH_EXTRA)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	使用禁止(注 3)
chain_transfer_nr	uint32_t	使用禁止(注 3)
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 (注 4)	bool	DMA 転送停止から DMA 再設定までに発生した DMAC 起動要求を保持するかを指定します 0：DMAC 起動要求を保持しない 1：DMAC 起動要求を保持する
offset(注 4)	int32_t	オフセット値を指定します

注　DTC ドライバでのみ使用可。本要素は無視されます。

注　DMA 転送再設定拡張コマンド(DMA_CMD_REFRESH_EXTRA)でのみ使用可。
DMA_CMD_REFRESH_DATA コマンドでは、本要素は無視されます。

2.7.3 st_dma_state_t 構造体

GetState 関数で取得したステータスを格納する構造体です。

表 2-16 st_dma_state_t 構造体

要素名	型	内容
in_progress	bool	DMA 転送のアクティブ状態を示します 0：非アクティブ 1：アクティブ
esif_stat	bool	転送エスケープ終了割り込みフラグを示します 0：割り込み発生なし 1：割り込み発生あり
dtif_stat	bool	転送終了割り込みフラグを示します 0：割り込み発生なし 1：割り込み発生あり
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 0 : DMA 転送要求フラグなし 1 : DMA 転送要求フラグあり

2.8 状態遷移

DMAC ドライバの状態遷移図を図 2-12 に、各状態でのイベント動作を表 2-17 に示します。

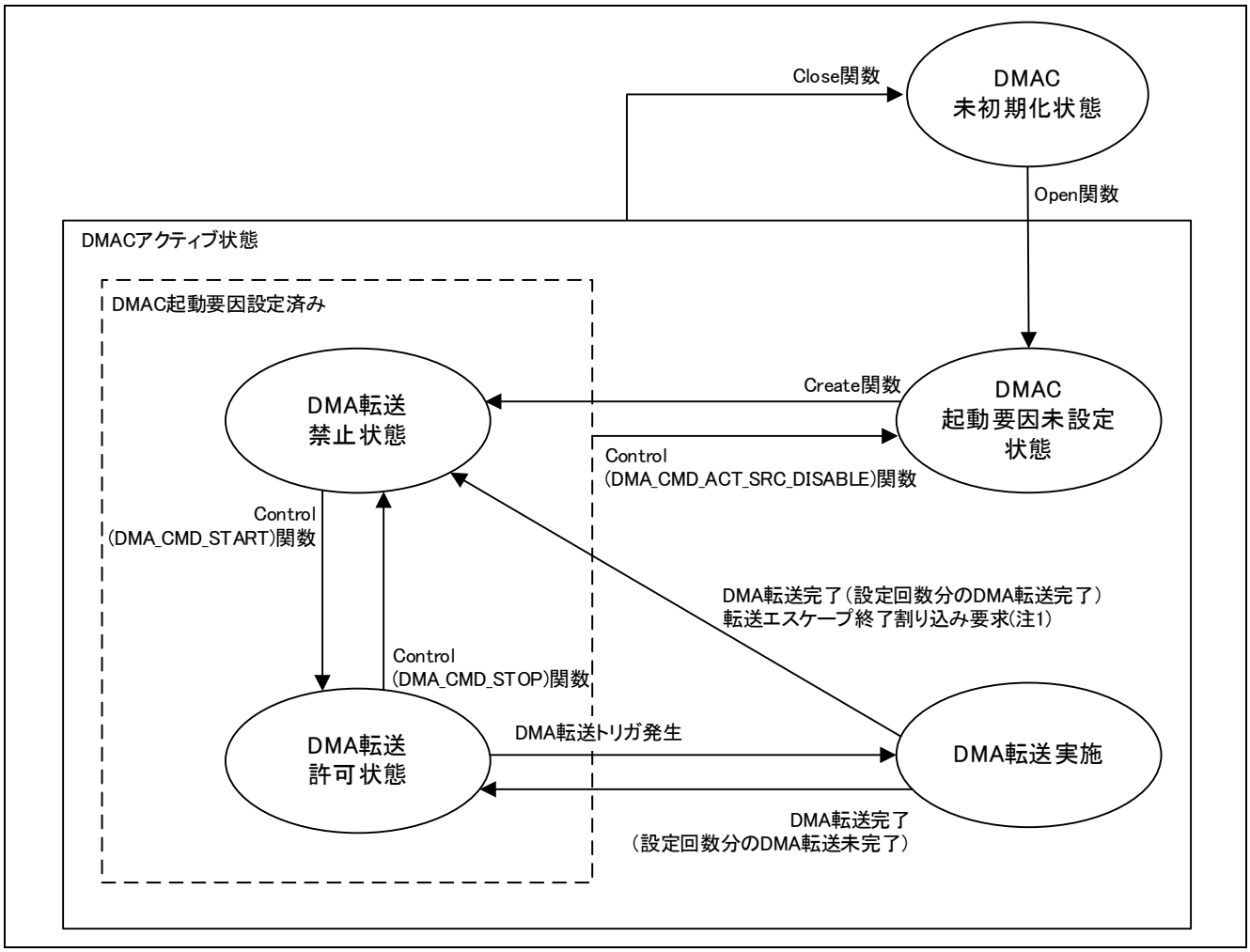

図 2-12 DMAC ドライバの状態遷移

注1. リピートサイズ終了割り込み、転送元アドレス拡張リピート領域オーバフロー割り込み、転送先アドレス拡張リピート領域オーバフロー割り込み要求のいずれかが発生した場合、DMA 転送禁止状態(DMCNT.DTE = 0)に遷移します。

表 2-17 DMAC ドライバ状態でのイベント動作(注)

状態	概要	イベント	アクション
DMAC 未初期化状態	リセット解除後の DMAC ドライバの状態です	Open 関数の実行	DMAC 起動要因未設定状態に遷移
DMAC 起動要因未設定状態	DMAC 動作許可 (DMAST.DMST = 1)、DMAC 起動要因が未設定の状態です	Create 関数の実行	DMA 転送禁止状態に遷移
DMA 転送禁止状態	DMA 転送禁止 (DMCNT.DTE = 0)状態、DMAC 起動要因設定済みの状態です	Control(DMA_CMD_ACT_SRC_DISABLE)関数の実行	DMAC 起動要因未設定状態に遷移します
		Control(DMA_CMD_START)関数の実行	DMA 転送許可状態に遷移します
		GetTransferByte 関数の実行	DMA 転送バイト数を取得します
DMA 転送許可状態	DMA 転送許可 (DMCNT.DTE = 1)状態、DMAC 起動要因設定済みの状態です	Control(DMA_CMD_ACT_SRC_DISABLE)関数の実行	DMAC 起動要因未設定状態に遷移します
		Control(DMA_CMD_STOP)関数の実行	DMA 転送禁止状態に遷移します
		DMAC 起動要因の発生	DMA 転送状態に遷移します
		GetTransferByte 関数の実行	DMA 転送バイト数を取得します
DMA 転送状態	DMA 転送実行中	DMA 転送完了 （指定回数分の DMA 転送未完了）	DMA 転送許可状態に遷移
		DMA 転送完了 （指定回数分の DMA 転送完了）	DMA 転送禁止状態に遷移
		転送エスケープ終了割り込み要求 （リピートサイズ終了割り込み、転送元アドレス拡張リピート領域オーバフロー割り込み、転送先アドレス拡張リピート領域オーバフロー割り込み要求のいずれかが発生）	DMA 転送禁止状態に遷移

注　GetVersion 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

DMAC ドライバは DMAC 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 0～65535 回まで設定できます。転送回数に 0 を設定すると、フリーランニングモードで動作します。ノーマル転送手順を図 3-1 に示します。

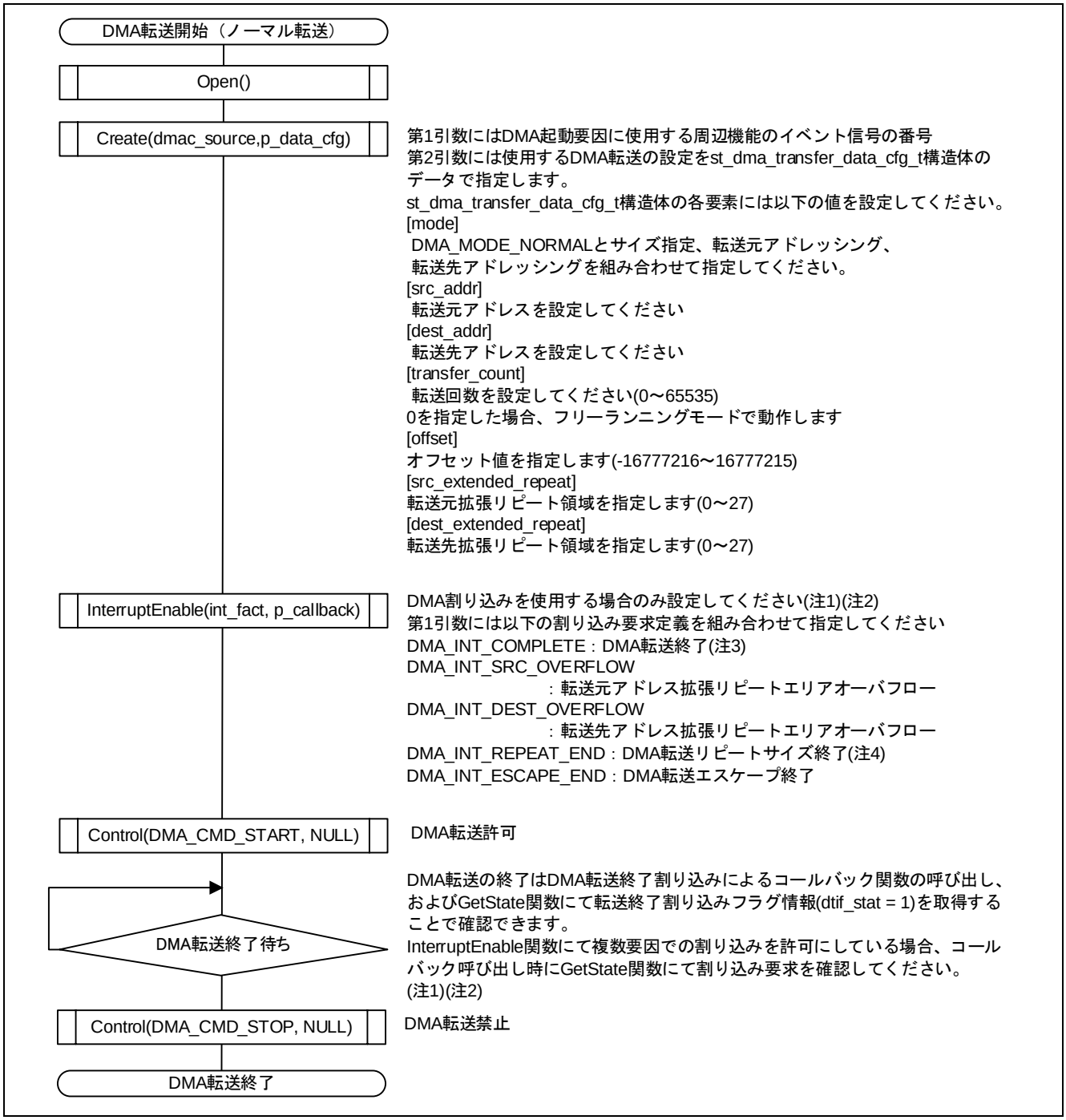

図 3-1 ノーマル転送手順

注1. 転送元アドレス拡張リピートエリアオーバフロー(DMA_INT_SRC_OVERFLOW)、転送先アドレス拡張リピートエリアオーバフロー(DMA_INT_DEST_OVERFLOW)割り込みでコールバック関数を呼び出す場合、InterruptEnable関数にて転送エスケープ終了割り込み(DMA_INT_ESCAPE_END)もあわせて許可にしてください。

注2. 転送元アドレス拡張リピートエリアオーバフロー割り込み要求、転送先アドレス拡張リピートエリアオーバフロー割り込み要求、DMA 転送リピートサイズ終了割り込み要求が発生すると、DMA 転送は停止(DMCNT.DTE = 0)します。DMA 転送を継続する場合、Control 関数にて DMA_CMD_START コマンドを実行してください。

注3. フリーランモードの場合、転送完了割り込みは発生しません。

注4. ノーマルモードではリピートサイズ終了割り込みは発生しません

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 1～65536 回、リピートサイズは 1～1024 回まで設定できます。リピート転送手順を図 3-2 に示します。

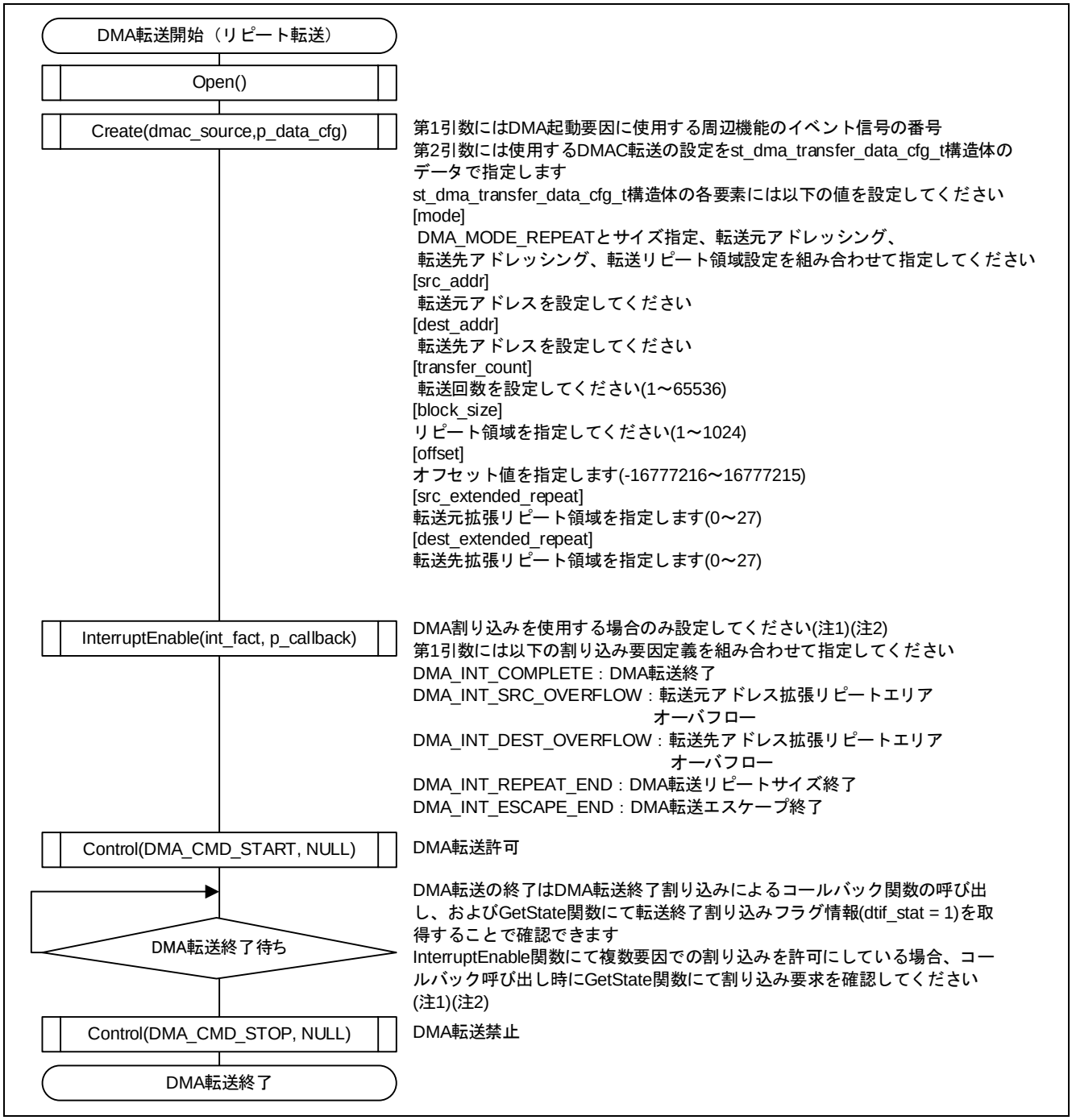

図 3-2 リピート転送手順

注1. 転送元アドレス拡張リピートエリアオーバフロー(DMA_INT_SRC_OVERFLOW)、転送先アドレス拡張リピートエリアオーバフロー(DMA_INT_DEST_OVERFLOW)、DMA 転送リピートサイズ終了(DMA_INT_REPEAT_END)割り込みでコールバック関数を呼び出す場合、InterruptEnable 関数にて転送エスケープ終了割り込み(DMA_INT_ESCAPE_END)もあわせて許可にしてください。

注2. 転送元アドレス拡張リピートエリアオーバフロー割り込み要求、転送先アドレス拡張リピートエリアオーバフロー割り込み要求、DMA 転送リピートサイズ終了割り込み要求が発生すると、DMA 転送は停止(DMCNT.DTE = 0)します。DMA 転送を継続する場合、Control 関数にて DMA_CMD_START コマンドを実行してください。

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 1～1024 バイト(8 ビットサイズ転送設定時)、2 バイト～2048 バイト（16 ビットサイズ転送設定時）、4 バイト～4096 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。

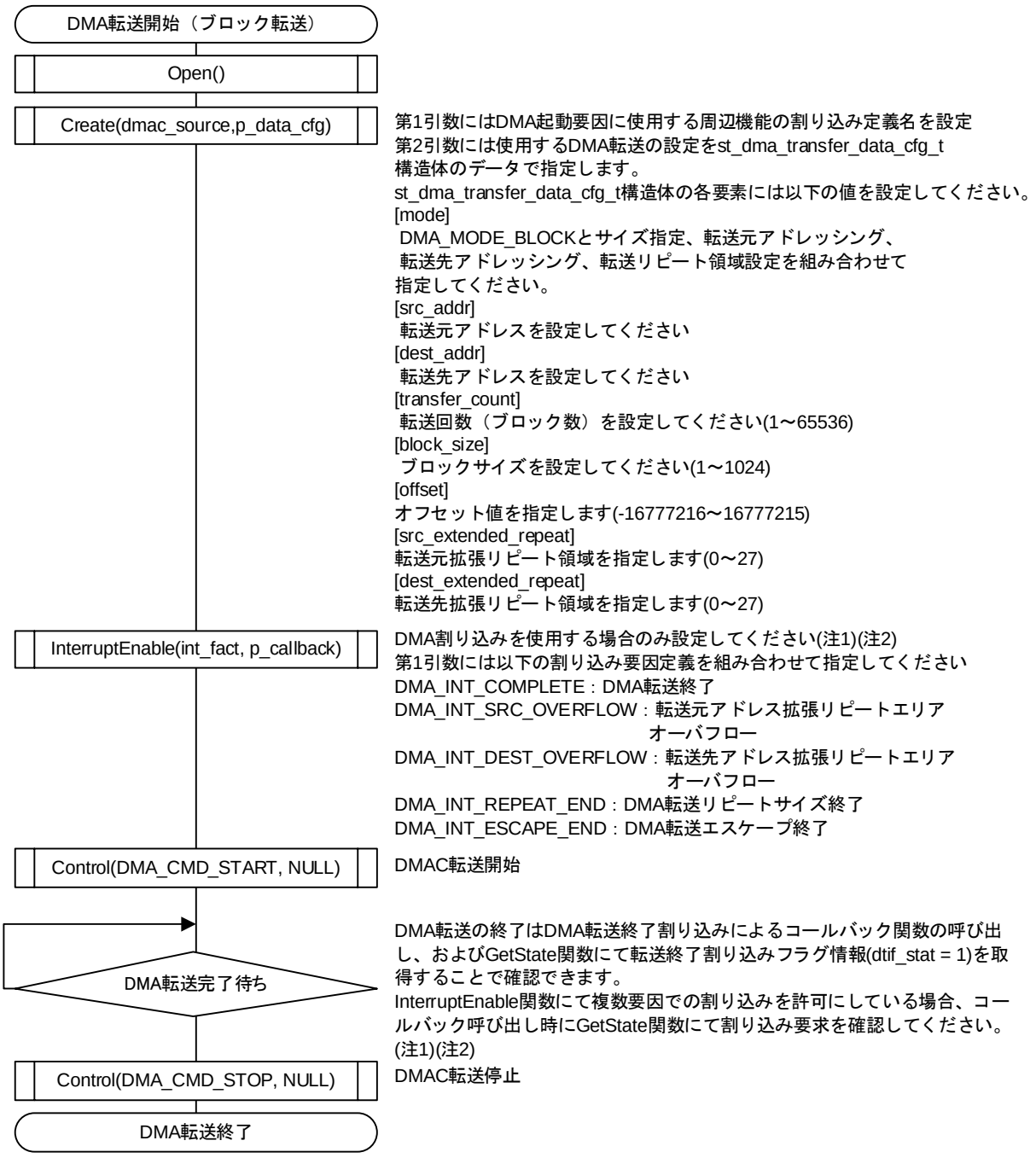

図 3-3 ブロック転送手順

注1. 転送元アドレス拡張リピートエリアオーバフロー(DMA_INT_SRC_OVERFLOW)、転送先アドレス拡張リピートエリアオーバフロー(DMA_INT_DEST_OVERFLOW)、DMA 転送リピートサイズ終了(DMA_INT_REPEAT_END)割り込みでコールバック関数を呼び出す場合、InterruptEnable 関数にて転送エスケープ終了割り込み(DMA_INT_ESCAPE_END)もあわせて許可にしてください。

注2. 転送元アドレス拡張リピートエリアオーバフロー割り込み要求、転送先アドレス拡張リピートエリアオーバフロー割り込み要求、DMA 転送リピートサイズ終了割り込み要求が発生すると、DMA 転送は停止(DMCNT.DTE = 0)します。DMA 転送を継続する場合、Control 関数にて DMA_CMD_START コマンドを実行してください。

3.4 Control 関数による DMAC 機能制御

Control 関数にて DMAC 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	DMACn 転送を開始します(注 2)
DMA_CMD_STOP	NULL	DMACn 転送を停止します(注 2)
DMA_CMD_ACT_SRC_DISABLE	NULL	DMACn 転送要因を禁止にします(注 2)
DMA_CMD_SOFTWARE_TRIGGER	NULL	ソフトウェアトリガによる DMA 転送要求を発行します
DMA_CMD_AUTO_CLEAR_SOFT_REQ	bool	DMA ソフトウェア起動ビット自動クリア機能の有効、無効を設定します true：自動クリア有効 false：自動クリア無効
DMA_CMD_REFRESH_DATA	st_dma_refresh_data_t 型のポインタ(注 1)	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
DMA_CMD_REFRESH_EXTRA	st_dma_refresh_data_t 型のポインタ(注 1)	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 あわせて、オフセット値の再設定および DMAC 起動要求保持選択を行います

注1. 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。

注2. n = 0～3

3.4.1 DMA 転送開始コマンド（DMA_CMD_START）

DMA 転送を許可状態(DMCNT.DTE = 1)にします。引数には NULL(0)を設定してください。DMA 転送開始コマンド使用例を図 3-4 に示します。

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

図 3-4 DMA 転送開始コマンド使用例

3.4.2 DMA 転送停止コマンド（DMA_CMD_STOP）

DMA 転送を禁止状態(DMCNT.DTE = 0)にします。引数には NULL(0)を設定してください。DMA 転送停止コマンド使用例を図 3-5 に示します。

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

図 3-5 DMA 転送停止コマンド使用例

3.4.3 DMAC 起動要因禁止コマンド（DMA_CMD_ACT_SRC_DISABLE）

使用チャネルの DELSRn.DELS(n = 0～3)を”0”（DMAC モジュールへの割り込み禁止）にし、DMA 転送を停止(DMCNT.DTE = 0)します。引数には NULL(0)を指定してください。DMAC 起動要因を禁止にした後、再度 DMA 転送を行う場合は Create 関数にて DMA 起動要因を設定する必要があります。DMAC 起動要因禁止コマンド使用例を図 3-6 に示します。

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

図 3-6 DMAC 起動要因禁止コマンド使用例

3.4.4 DMA ソフトウェア起動コマンド（DMA_CMD_SOFTWARE_TRIGGER）

DMREQ.SWREQ を”1”にし、ソフトウェアによる DMA 転送要求を発行します。DMA ソフトウェア起動コマンド使用例を図 3-7 に示します。

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

図 3-7 DMA ソフトウェア起動コマンド使用例

3.4.5 DMA ソフトウェア起動ビット自動クリア機能設定コマンド（DMA_CMD_AUTO_CLEAR_SOFT_REQ）

DMA ソフトウェア起動ビット自動クリア機能を有効もしくは無効に設定します。第 2 引数に”0”を設定すると、DMA ソフトウェア起動ビット自動クリア機能が無効に、”1”を設定すると有効になります。

ソフトウェア起動ビット自動クリア機能は、Open 関数にて DMAC を初期化したとき、デフォルト設定で有効状態になっています。ソフトウェア起動ビット自動クリア機能を無効にした場合は、ClearStatus（定義名）関数にてソフトウェア起動要求フラグをクリアしてください。DMA ソフトウェア起動ビット自動クリアコマンド使用例を図 3-8 に示します。

```
uint8_t arg = true;
  /* DMA ソフトウェア起動ビット自動クリア機能有効 */
  (void)dmac0Drv->Control(DMA_CMD_AUTO_CLEAR_SOFT_REQ, &arg);
```

図 3-8 DMA ソフトウェア起動ビット自動クリア機能設定コマンド使用例

3.4.6 DMA 転送再設定コマンド（DMA_CMD_REFRESH_DATA）

DMA 転送情報の再設定を行います。

DMA 転送が終了した後、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 DMA 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 DMA 転送許可コマンド（DMA_CMD_START）を使用して DMA 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。

本コマンドでは、chain_transfer_nr メンバ、keep_dma_req メンバおよび offset メンバは使用しません。”0”を設定してください。また、DMA 転送終了から DMA 再設定完了までの間に発生した DMAC 起動要求は無視されます（DELSRn.IR を 0 クリアします）。

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。

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

    /* DMAC0 の DMA 転送を再設定 */
    arg.act_src        = 0;                    /* DMAC では使用しないため、0 を設定 */
    arg.chain_transfer_nr = 0;                   /* DMAC では使用しないため、0 を設定 */
    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、DMA 転送許可 */
    (void)dmac0Drv->Control(DMA_CMD_REFRESH_DATA, &arg);
}
```

図 3-9 コールバック関数での DMA 転送再設定コマンド使用例

3.4.7 DMA 転送再設定拡張コマンド（DMA_CMD_REFRESH_EXTRA）

DMA 転送情報の再設定を行います。DMA_CMD_REFRESH コマンドの機能に加え、オフセット値の更新および DMAC 起動要求(注 1)保持選択が可能です。

DMA 転送が終了した後、モードは変更せずに、転送元アドレス（注 2）、転送先アドレス（注 2）、転送回数を更新します。auto_start メンバに”1”を設定すると、本コマンド実行後に DMA 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 DMA 転送許可コマンド（DMA_CMD_START）を使用して DMA 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。

拡張機能として、オフセット値の更新および DMAC 起動要求の保持選択を行えます。offset メンバには更新するオフセット値を設定してください。DMAC 起動要求の保持を行う場合、keep_dma_req メンバに”1”を設定してください。DMAC 起動要求の保持機能を有効にすると、DMA 転送終了から DMA 転送再設定中に発生した DMAC 起動要求が 1 回分保持されます。keep_dma_req メンバに”0”を設定した場合、DMA 転送終了から DMA 転送再設定中に発生した DMAC 起動要求を無視します（DELSRn.IR を 0 クリアします）。

chain_transfer_nr メンバは、DMAC では使用しません。”0”を設定してください。

注1. DMAC の起動要求ステータスフラグ（DELSRn.IR）を指します。

注2. 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。

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

    /* DMAC0 の DMA 転送を再設定 */
    arg.act_src        = 0;                    /* DMAC では使用しないため、0 を設定 */
    arg.chain_transfer_nr = 0;                   /* DMAC では使用しないため、0 を設定 */
    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、DMA 転送許可 */
    keep_dma_req     = true;                /* DMA 転送再設定中に発生した DMAC 起動要求保持を有効 */
    offset             = 2;                    /* オフセット値を更新 */
    (void)dmac0Drv->Control(DMA_CMD_REFRESH_EXTRA, &arg);
}
```

図 3-10 DMA 転送再設定（拡張）コマンド使用例

DMA 転送再設定コマンド（DMA_CMD_REFRESH_DATA）、DMA 転送再設定拡張コマンド（DMA_CMD_REFRESH_EXTRA）による DMA 転送再設定動作の例を図 3-11 と図 3-12 に示します。

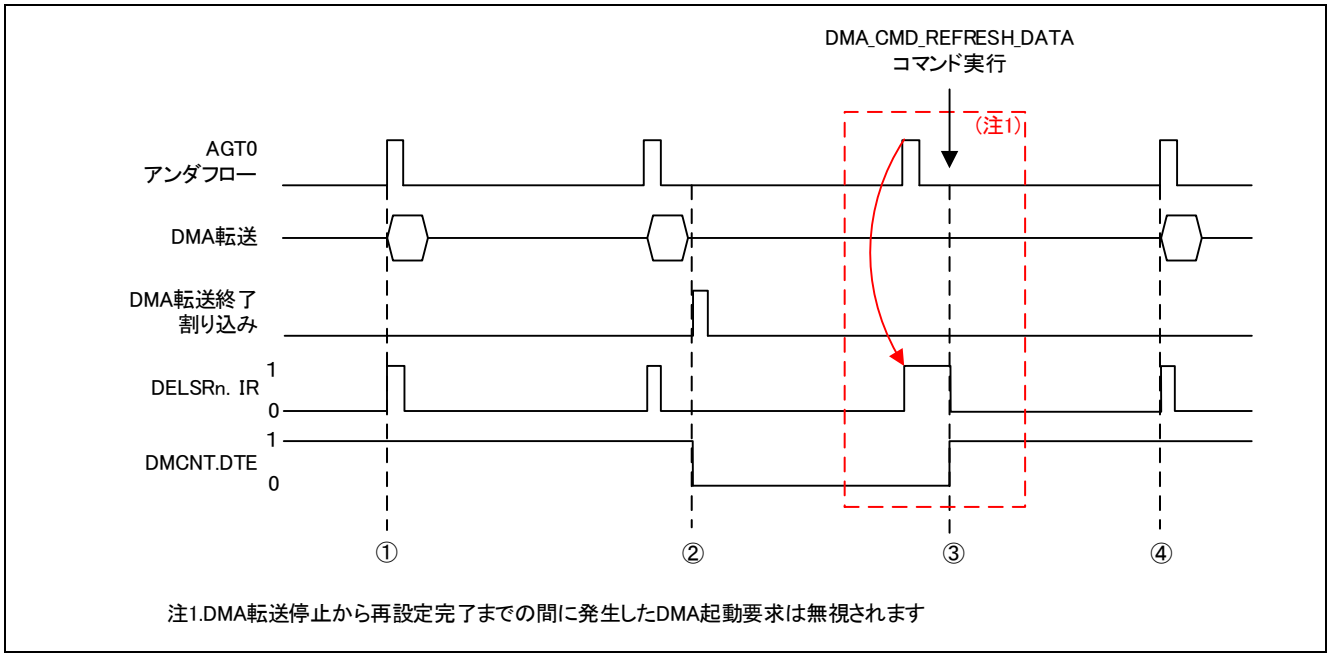

図 3-11 DMA_CMD_REFRESH_DATA コマンド動作例

① AGT0 アンダフローを起動要因として、1 データ分の DMA 転送が実行されます。

② 最終データの DMA 転送が実行されます。すべての DMA 転送が終了すると、DMA 転送終了割り込みが発生します。DMCNT.DTE が 0 になります。

③ Control 関数にて DMA_CMD_REFRESH_DATA コマンドを実行し、DMA 設定を再設定（転送回数 2 回、オートスタート有効）します。オートスタートを有効にしていた場合、DMCNT.DTE が 1 になります。DMA 転送終了から DMA 転送再設定完了までに発生した DMA 起動要求は無視されます (DELSRn.IR = 0)。

④ DMA 再設定後に発生した AGT0 アンダフローを起動要因として、DMA 転送が実行されます。

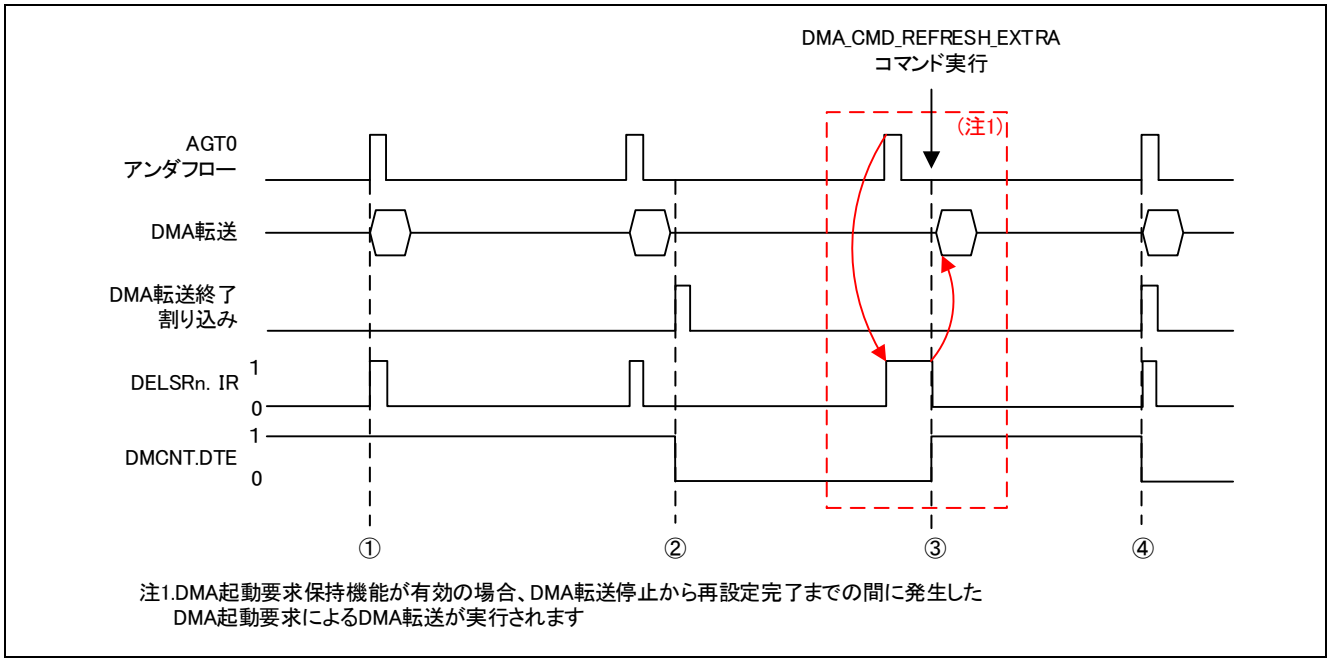

図 3-12 DMA_CMD_REFRESH_EXTRA コマンド（DMA 起動要求保持有効）動作例

① AGT0 アンダフローを起動要因として、1 データ分の DMA 転送が実行されます。

② 最終データの DMA 転送が実行されます。すべての DMA 転送が終了すると、DMA 転送終了割り込みが発生します。DMCNT.DTE が 0 になります。

③ Control 関数にて DMA_CMD_REFRESH_DATA コマンドを実行し、DMA 設定を再設定（転送回数 2 回、オートスタート有効、DMA 起動要求保持有効）します。オートスタートを有効にしていた場合、DMCNT.DTE が 1 になります。DMA 起動要求保持を有効にしていた場合、DMA 転送終了から DMA 転送再設定完了までに発生した DMA 起動要求によって DMA 転送が実行されます。

④ DMA 再設定後に発生した AGT0 アンダフローを起動要因として、DMA 転送が実行されます。すべての DMA 転送が終了し、DMA 転送終了割り込みが発生します。DMCNT.DTE が 0 になります。

3.5 DMA 転送バイト数の取得

GetTransferByte 関数にて、DMA 転送バイト数を取得できます。第 2 引数に取得結果を格納する変数を設定してください。第 1 引数は DMAC ドライバでは使用しません。

ノーマル転送モードで転送回数に 0(フリーランモード)を設定した場合、転送バイト数は取得できません。GetTransferByte 関数を実行すると、DMA_ERROR を返します。

GetTransferByte 関数使用例を図 3-13 に、ノーマル転送モードでの DMA 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 DMA 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 DMA 転送バイト数取得例を表 3-4 に示します。

```
uint32_t transfer_byte;
/* DMAC 転送バイト数を取得 */
(void)dmac0Drv->GetTransferByte(0, &transfer_byte);
```

図 3-13 GetTransferByte 関数使用例

表 3-2 ノーマル転送モードでの DMA 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
DMA 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 DMA 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
DMA 転送バイト数(注 1)	4	8	12	16	20	24	28

表 3-4 ブロック転送モードでの DMA 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
DMA 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 DMA 転送バイト数

注2. DMAC ドライバではリピートサイズ分の転送後も DMA 転送バイト数はカウントを継続します。
(0 に戻りません)

3.6 コンフィグレーション

DMAC 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_dmac_cfg.h ファイルに用意します。

3.6.1 パラメータチェック

DMAC 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：DMAC_CFG_PARAM_CHECKING_ENABLE

表 3-5 DMAC_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

3.6.2 DMACn_INT 割り込み優先レベル(n=0～3)

DMACn_INT 割り込みの優先レベルを設定します。

名称：DMACn_INT_PRIORITY

表 3-6 DMACn_INT_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

3.6.3 関数の RAM 配置

DMAC 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：DMAC_CFG_xxx

xxx には関数名をすべて大文字で記載

例） R_DMAC_Open 関数 → DMAC_CFG_R_DMAC_OPEN

表 3-7 DMAC_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_DMAC_GetVersion	
2	R_DMAC_Open	
3	R_DMAC_Close	
4	R_DMAC_Create	
5	R_DMAC_Control	
6	R_DMAC_InterruptEnable	
7	R_DMAC_InterruptDisable	
8	R_DMAC_GetState	
9	R_DMAC_ClearState	
10	R_DMAC_GetTransferByte	
11	dmac_interrupt_handler（DMACn_INT(n=0～3)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

DMAC 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_DMAC_Open 関数

表 4-1 R_DMAC_Open 関数仕様

書式	static e_dma_err_t R_DMAC_Open(st_dmac_resources_t * const p_dmac)
仕様説明	DMAC ドライバのオープン（モジュールストップの解除、使用する RAM の初期化）を行います
引数	st_dmac_resources_t * const p_dmac : DMAC のリソース オープンする DMAC のリソースを指定します。
戻り値	DMA_OK　DMAC オープン成功
	DMA_ERROR　DMAC オープン失敗 以下のいずれかの状態を検出するとオープン失敗となります ・使用する DMAC チャネルのリソースがすでにオープンされている場合 ・モジュールストップの遷移に失敗した場合（R_LPM_ModuleStart にてエラーが発生した場合）
	DMA_ERROR_LOCKED　DMAC リソースのロック失敗 使用する DMAC チャネルのリソースがロックされている場合ロック失敗となります (すでに R_SYS_ResourceLock 関数にて R_DMACn がロックされている場合)

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	　dmac0Drv->Open();
	}

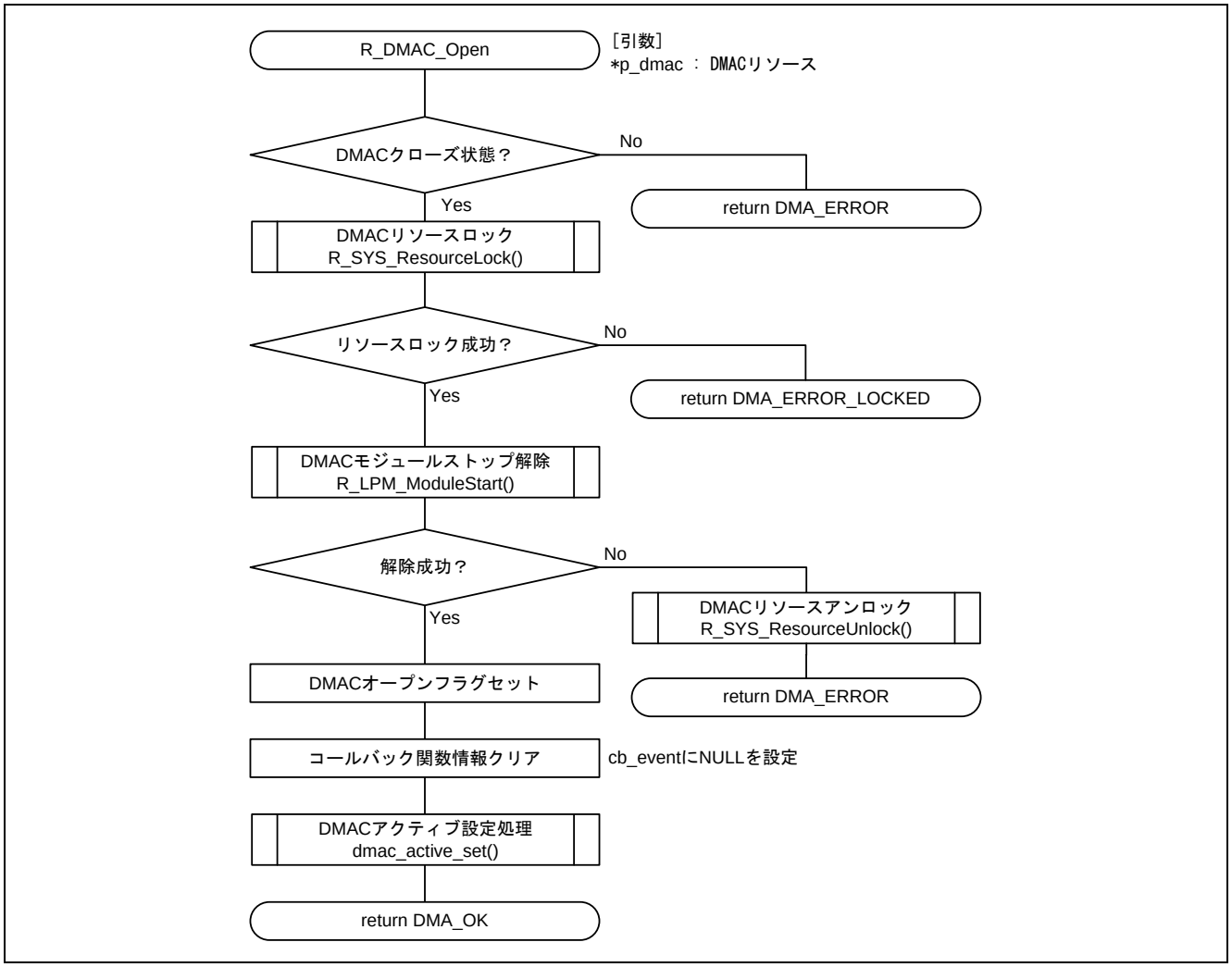

図 4-1 R_DMAC_Open 関数処理フロー

4.1.2 R_DMAC_Close 関数

表 4-2 R_DMAC_Close 関数仕様

書式	static e_dma_err_t R_DMAC_Close(st_dmac_resources_t * const p_dmac)
仕様説明	DMAC ドライバを解放します
引数	st_dmac_resources_t * const p_dmac : DMAC のリソース 解放する DMAC のリソースを指定します。
戻り値	DMA_OK DMAC の解放成功

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv->Close();
	}

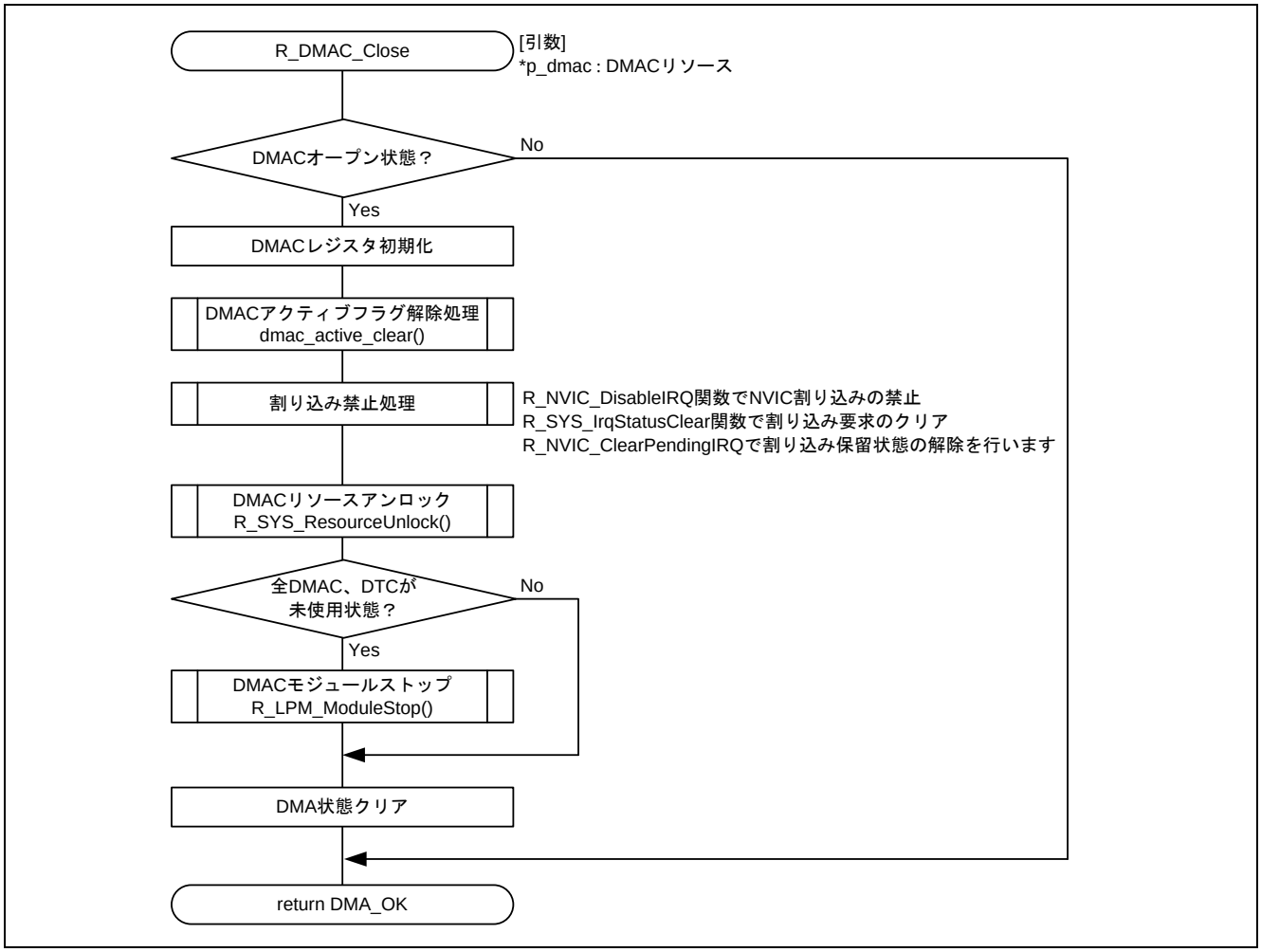

図 4-2 R_DMAC_Close 関数処理フロー

4.1.3 R_DMAC_Create 関数

表 4-3 R_DMAC_Create 関数仕様(1/2)

書式	static e_dma_err_t R_DMAC_Create(int16_t const dmac_source, st_dma_transfer_data_cfg_t * const p_data_cfg, st_dmac_resources_t * const p_dmac)
仕様説明	DMA 転送の設定を行います
引数	int16_t const dmac_source : DMAC 起動要因 DMAC の転送要因を指定します。 転送要因に周辺モジュールからの割り込み要求／外部割り込み入力端子からのトリガを使用する場合は、リンクするイベント信号の番号を指定してください。 イベント信号の番号については、「RE01 1500KB グループ　ユーザーズマニュアル　ハードウェア編(r01uh0796)」もしくは「RE01 256KB グループ ユーザーズマニュアル　ハードウェア編(r01uh0894)」の「16. 割り込みコントローラユニット（ICU）」を参照してください。　転送要因にソフトウェアトリガを使用する場合は、 DMAC_TRANSFER_REQUEST_SOFTWARE を指定してください。
	st_dma_transfer_data_cfg_t * const p_data_cfg : DMAC 設定情報 DMA 転送情報を指定します。 構造体内の設定については 2.7.1 を参照してください。
	st_dmac_resources_t * const p_dmac : DMAC のリソース 設定する DMAC のリソースを指定します。
戻り値	DMA_OK　　設定成功
	DMA_ERROR　　設定失敗 DMAC オープン前に実行した場合、設定失敗となります
	DMA_ERROR_MODE　　モード異常値による設定失敗 以下のいずれかの設定値異常を検出すると、モード異常値による設定失敗となります ・ノーマル、リピート、ブロック以外の転送方式設定 ・8bit、16bit、32bit 以外のデータサイズ設定 ・転送元リピート、転送先リピート、リピート領域なし以外のリピート領域設定 ・チェーン転送設定（DMAC では無効）
	DMA_ERROR_PARAMETER　　パラメータ異常による設定失敗 以下のいずれかの設定値異常を検出するとパラメータ異常による設定失敗となります ・DMAC 起動要因(dmac_source)に不正な値を設定した場合（設定可能範囲 ：0x01～0xAA） ・転送モード指定(p_data_cfg->mode)の未使用ビットに値を設定している場合 ・転送回数(p_data_cfg->transfer_count)に不正な値を設定した場合 （設定可能範囲については表 4-5 転送モード毎の転送回数の設定可能範囲を参照してください） ・リピート転送、もしくはブロック転送を指定した時に、ブロック転送サイズ(p_data_cfg->block_size)に不正な値を設定した場合 (設定可能範囲 ：1～1024) ・転送サイズ 16bit 時に、転送元アドレス(p_data_cfg->src_addr)もしくは転送先アドレス(p_data_cfg->dest_addr)の最下位ビットが 0 でない場合 ・転送サイズ 32bit 時に、転送元もしくは転送先アドレスの下位 2 ビットが 00 でない場合 ・転送元リピート設定時に、転送方式がノーマル転送でなく、かつ転送元拡張リピート領域(p_data_cfg->src_extended_repeat)に 0 以外を設定している場合 ・転送先リピート設定のときに、転送方式がノーマル転送でなく、かつ転送先拡張リピート領域(p_data_cfg->dest_extended_repeat)に 0 以外を設定している場合 ・転送元拡張リピート領域に不正な値を設定している場合（設定可能範囲 ：0～27） ・転送先拡張リピート領域に不正な値を設定している場合（設定可能範囲 ：0～27） ・転送元もしくは転送先にオフセットを設定する時に、オフセット値(p_data_cfg->offset)に不正な値を設定している場合（設定可能範囲 ：-16777216～16777215）

表 4-4 R_DMAC_Create 関数仕様(2/2)

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	const uint8_t tx_data[2] = {0x51,0xA2};
	uint8_t rx_data[2] = {0x00,0x00};
	
	main()
	{
	st_dma_transfer_data_cfg_t config;
	
	config.mode = (DMA_MODE_REPEAT | DMA_SIZE_BYTE | DMA_SRC_INCR |
	DMA_DEST_INCR | DMA_REPEAT_BLOCK_SRC);
	config.src_addr = &tx_data[0];
	config.dest_addr = & rx_data[0];
	config.transfer_count = 4;
	config.block_size = 4;
	config.offset = 0;
	config.src_extended_repeat = 0;
	config.dest_extended_repeat = 0;
	
	dmac0Drv->Create (5, &config);
	}

表 4-5 転送モード毎の転送回数の設定可能範囲

転送モード	設定可能範囲
ノーマル転送	0～65535
リピート転送	1～65536
ブロック転送	

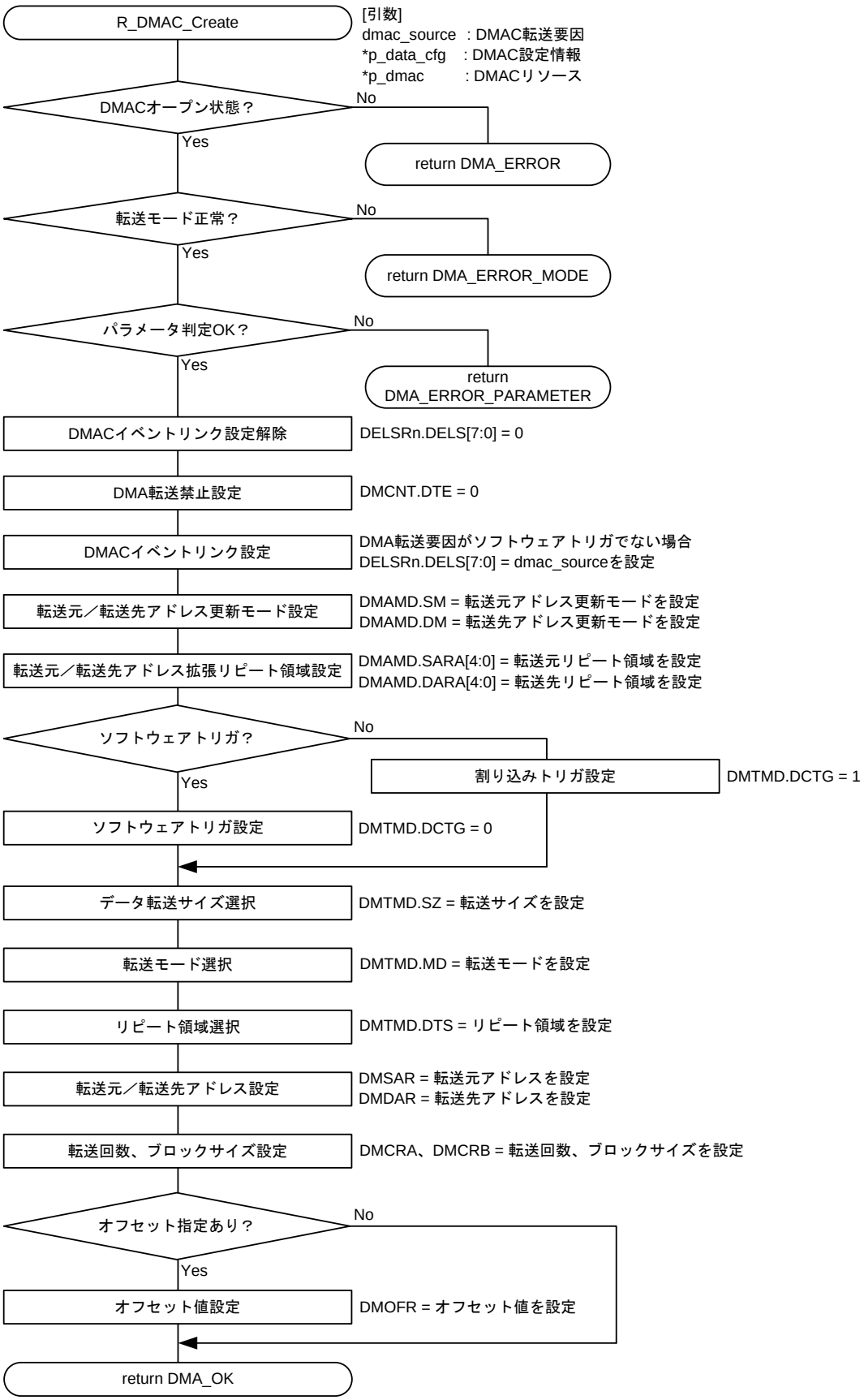

図 4-3 R_DMAC_Create 関数処理フロー

4.1.4 R_DMAC_Control 関数

表 4-6 R_DMAC_Control 関数仕様

書式	static e_dma_err_t R_DMAC_Control(e_dma_cmd_t cmd, void const * const p_arg, st_dmac_resources_t * const p_dmac)
仕様説明	DMAC の制御コマンドを実行します
引数	e_dma_cmd_t cmd : 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 : DMA 転送開始コマンド ・DMA_CMD_STOP : DMA 転送停止コマンド ・DMA_CMD_SOFTWARE_TRIGGER : ソフトウェアトリガによる DMA 転送要求コマンド ・DMA_CMD_AUTO_CLEAR_SOFT_REQ : DMA ソフトウェア起動ビット自動クリア設定コマンド ・DMA_CMD_ACT_SRC_DISABLE : DMAC 起動要因解除コマンド ・DMA_CMD_REFRESH_DATA : DMA 転送再設定コマンド ・DMA_CMD_REFRESH_EXTRA : DMA 転送再設定（拡張）コマンド
	void const * const p_arg : コマンド別引数（制御コマンドと引数の関係については表 4-7 参照）
	st_dmac_resources_t * const p_dmac : DMAC のリソース 制御対象の DMAC のリソースを指定します。
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 以下のいずれかの状態を検出すると制御コマンド実行失敗となります ・DMAC オープン前に実行した場合 ・ソフトウェアによる DMA 転送要求中に再度ソフトウェア要求コマンド (DMA_CMD_SOFTWARE_TRIGGER)を実行した場合 ・引数を要求するコマンド(DMA_CMD_AUTO_CLEAR_SOFT_REQ、 DMA_CMD_REFRESH_DATA)にて、引数に NULL(0)を指定した場合
	DMA_ERROR_MODE モード異常値による実行失敗 DMA 転送再設定コマンド(DMA_CMD_REFRESH_DATA、DMA_CMD_REFRESH_EXTRA)を実行する際、ノーマル、リピート、ブロック以外の転送方式を指定すると、モード異常値による実行失敗となります
	DMA_ERROR_PARAMETER パラメータ異常による実行失敗 以下の条件を検出した場合、パラメータ異常による実行失敗となります [DMA 転送再設定コマンド(DMA_CMD_REFRESH_DATA、DMA_CMD_REFRESH_EXTRA)] ・転送回数(p_data_cfg->transfer_count)に不正な値を設定した場合 （設定可能範囲については表 4-5 転送モード毎の転送回数の設定可能範囲を参照してください） ・リピート転送、もしくはブロック転送を指定した時に、 ブロック転送サイズ(p_data_cfg->block_size)に不正な値を設定した場合 (設定可能範囲 : 1～1024) [DMA 転送再設定（拡張）コマンド(DMA_CMD_REFRESH_EXTRA)] ・オフセット値に不正な値を設定すると DMA_ERROR_PARAMETER となります （設定可能範囲 : -16777216～16777215）。

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	main()
	{
	dmac0Drv-> Control (DMA_CMD_START, NULL);
	}

表 4-7 制御コマンドとコマンド別引数による動作

制御コマンド(cmd)	コマンド別引数(arg)	内容
DMA_CMD_START	NULL	DMA 転送を開始します。 R_DMAC_Create 関数で指定した転送要因をトリガとして転送を行います。
DMA_CMD_STOP	NULL	DMA 転送を終了します。
DMA_CMD_SOFTWARE_TRIGGER	NULL	ソフトウェアトリガによるDMA転送要求を発行します。 R_DMAC_Create 関数で転送要因に DMAC_TRANSFER_REQUEST_SOFTWARE を設定した場合のみ有効です。
DMA_CMD_AUTO_CLEAR_SOFT_REQ	true	ソフトウェアトリガによる DMA 転送開始後、転送要求をクリアします。
	false	ソフトウェアトリガによる DMA 転送開始後も、転送要求を維持します。
DMA_CMD_ACT_SRC_DISABLE	NULL	DMA 転送を禁止にし、転送要因を解除します。
DMA_CMD_REFRESH_DATA	st_dma_refresh_data_t 型の DMAC 設定情報を指定してください。 構造体の詳細については 2.7.2 を参照してください。	転送回数、転送ブロック数、転送元アドレス、転送先アドレスのみを更新します。 オートスタート ON で更新した場合、更新後 DMA 転送を再開します。
DMA_CMD_REFRESH_EXTRA		転送回数、転送ブロック数、転送元アドレス、転送先アドレスに加え、オフセット値を更新します。また、DMAC 起動要求保持選択を行えます。 DMA 要求保持 ON で更新した場合､DMA 転送再設定中に発生した DMA 要求を保持します。 オートスタート ON で更新した場合、更新後 DMA 転送を再開します。 DMAC 起動要求保持を有効にしていた場合、DMA 転送停止から DMA 再設定中に発生した DMAC 起動要求を保持します。

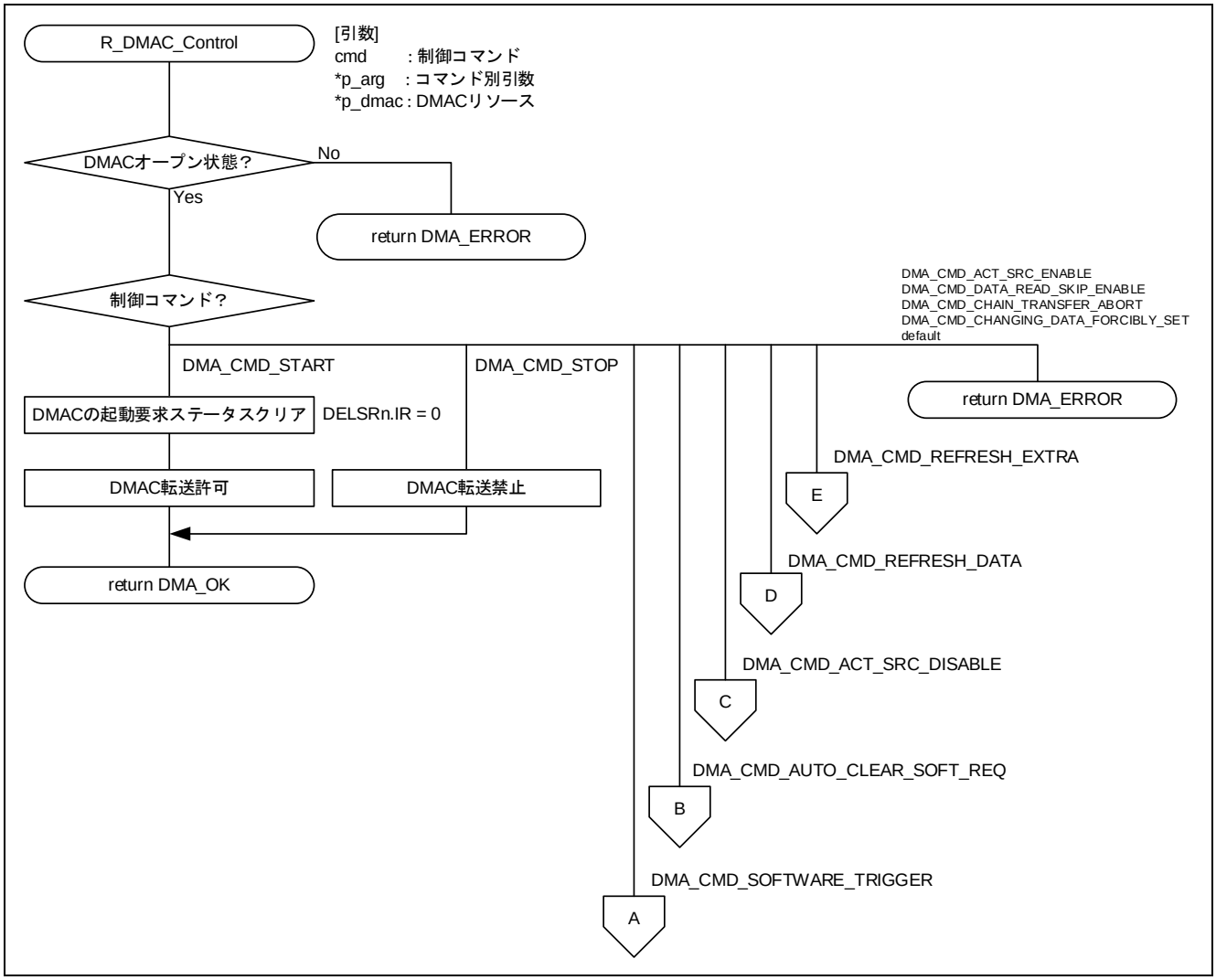

図 4-4 R_DMAC_Control 関数処理フロー(1/3)

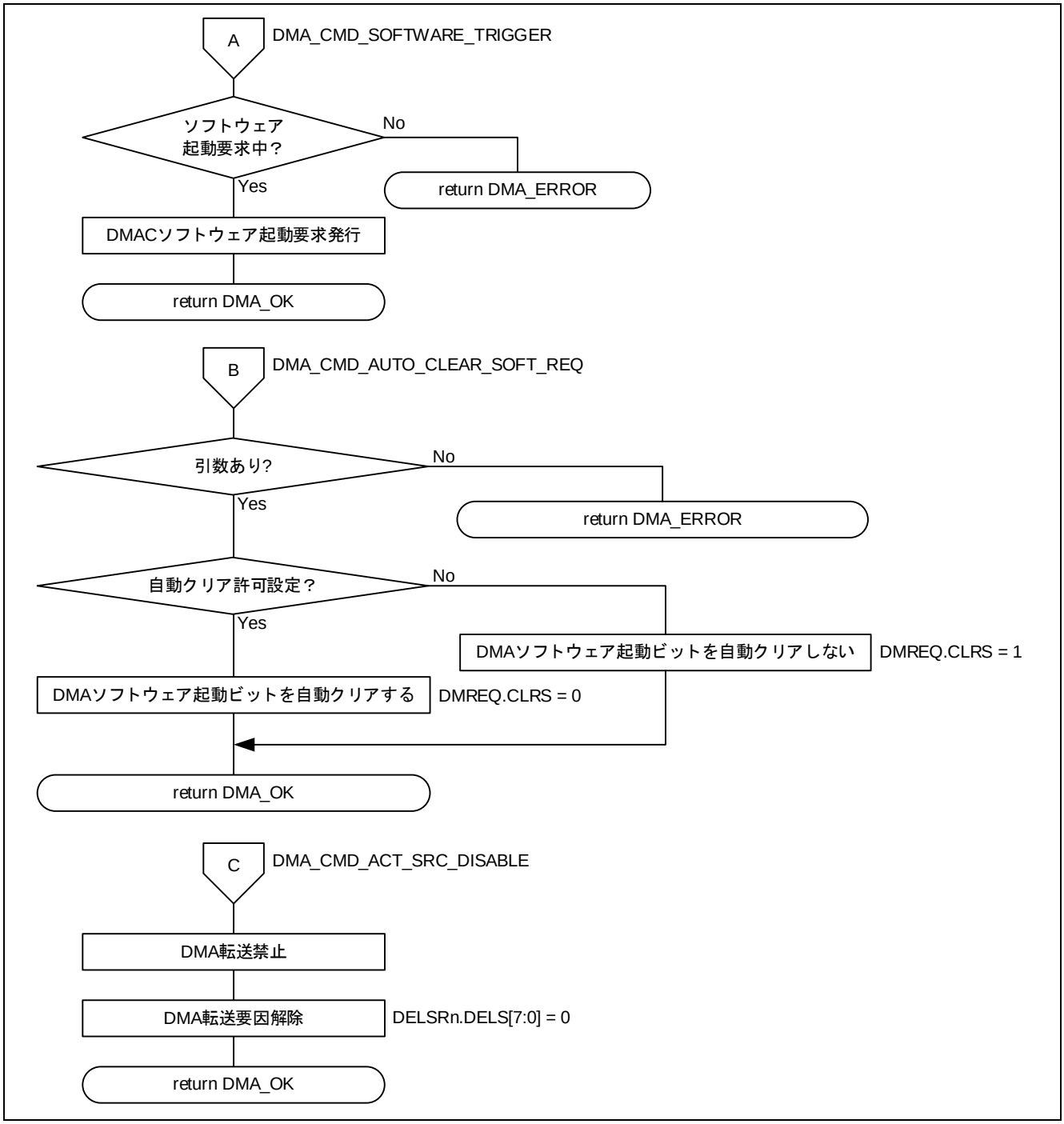

図 4-5 R_DMAC_Control 関数処理フロー(2/3)

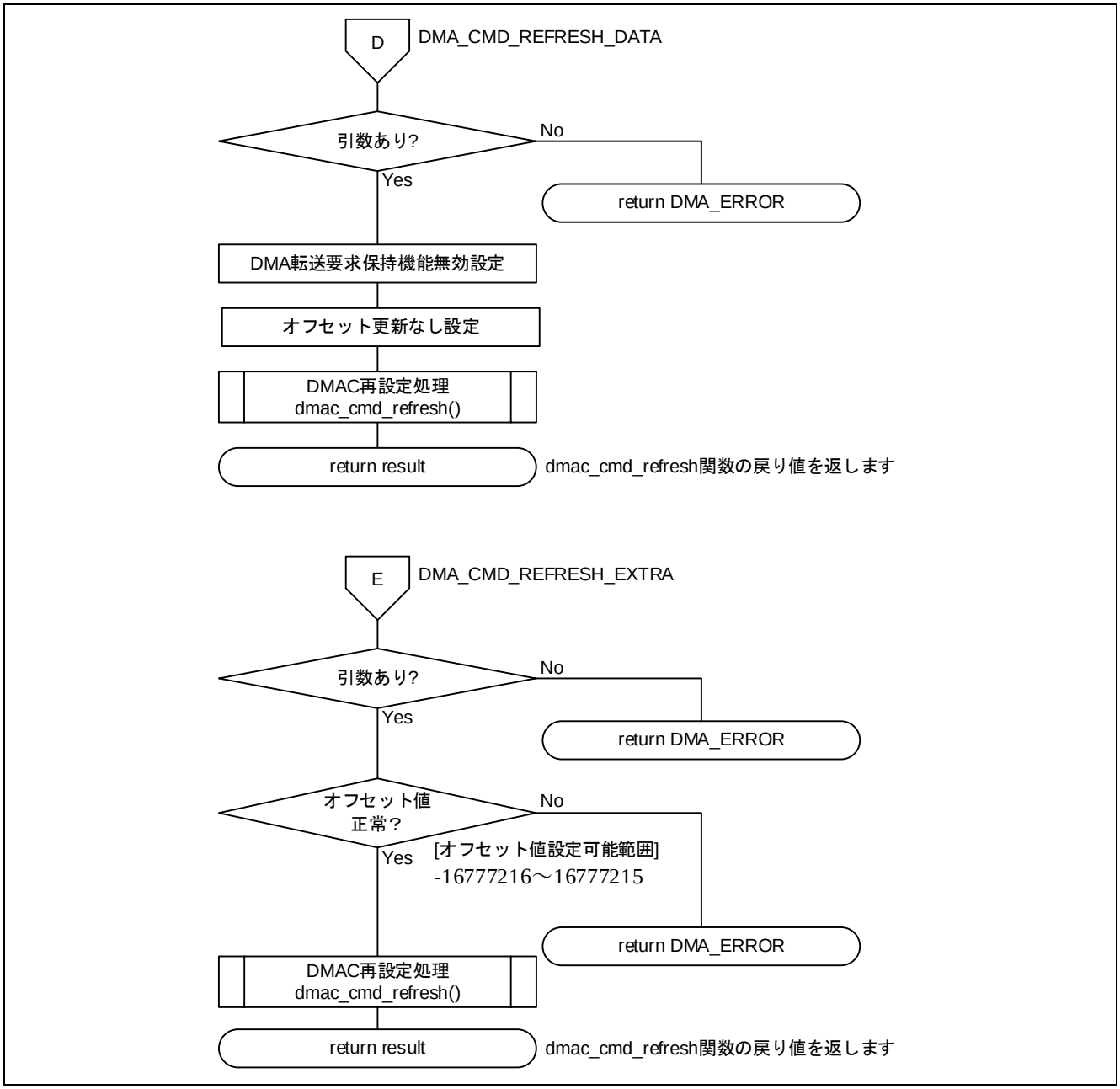

図 4-6 R_DMAC_Control 関数処理フロー(3/3)

4.1.5 R_DMAC_InterruptEnable 関数

表 4-8 R_DMAC_InterruptEnable 関数仕様

書式	static e_dma_err_t R_DMAC_InterruptEnable(uint16_t const int_fact, dma_cb_event_t const p_callback, st_dmac_resources_t * const p_dmac)
仕様説明	DMA 転送割り込みの設定を行います。
引数	uint16_t const int_fact : 割り込み要因 以下の割り込み要因を指定します。 0 を設定した場合は、DMA 転送完了割り込みの NVIC 登録のみ行います。 複数指定する場合は、OR 演算を使用して定義してください。 （例 : DMA_INT_COMPLETE | DMA_INT_SRC_OVERFLOW） ・DMA_INT_COMPLETE : DMA 転送終了 ・DMA_INT_SRC_OVERFLOW : 転送元アドレス拡張リピートエリアオーバフロー ・DMA_INT_DEST_OVERFLOW : 転送先アドレス拡張リピートエリアオーバフロー ・DMA_INT_REPEAT_END : DMA 転送リピートサイズ終了 ・DMA_INT_ESCAPE_END : DMA 転送エスケープ終了
	dma_cb_event_t const p_callback : コールバック関数 イベント発生時のコールバック関数を指定します。 NULL(0)を指定した場合、コールバック関数が実行されません。
	st_dmac_resources_t * const p_dmac : DMAC のリソース 割り込みを設定する DMAC のリソースを指定します。
戻り値	DMA_OK 設定成功
	DMA_ERROR 設定失敗 以下のいずれかの状態を検出すると設定失敗となります ・DMAC のオープン前に実行した場合 ・割り込み要因に不正な値が指定されている場合
	DMA_ERROR_SYSTEM_SETTING システム設定失敗 以下のいずれかの状態を検出するとシステム設定失敗となります。 ・r_system_cfg.h で DMAC 割り込みが未使用定義 (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)になっている場合 ・r_dmac_cfg.h で DMACn_INT_PRIORITY の設定が定義範囲を超えている場合

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	static void callback(void);
	
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> InterruptEnable (DMA_INT_SRC_OVERFLOW | DMA_INT_ESCAPE_END,
	callback);
	}

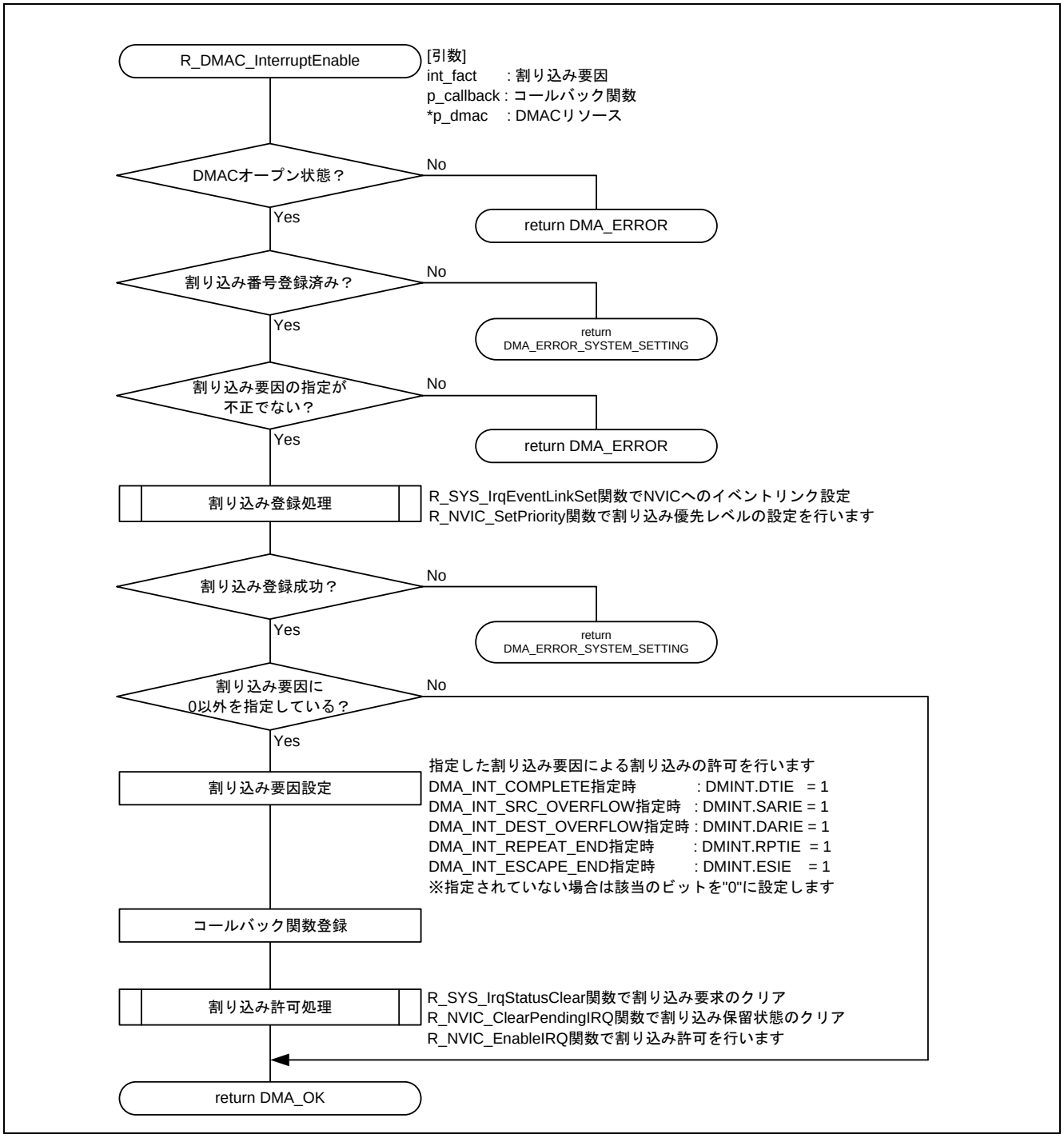

図 4-7 R_DMAC_InterruptEnable 関数処理フロー

4.1.6 R_DMAC_InterruptDisable 関数

表 4-9 R_DMAC_InterruptDisable 関数仕様

書式	static e_dma_err_t R_DMAC_InterruptDisable(st_dmac_resources_t * const p_dmac)
仕様説明	DMA 転送割り込みを禁止にします
引数	st_dmac_resources_t * const p_dmac : DMAC のリソース 割り込み禁止する DMAC リソースを指定します。
戻り値	DMA_OK 割り込み禁止成功
	DMA_ERROR 割り込み禁止失敗 DMAC オープン前に実行した場合、割り込み禁止失敗となります
	DMA_ERROR_SYSTEM_SETTING システム設定失敗 ・r_system_cfg.h で DMAC 割り込みが未使用定義(SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)になっている場合、システム設定失敗となります

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> InterruptDisable ();
	}

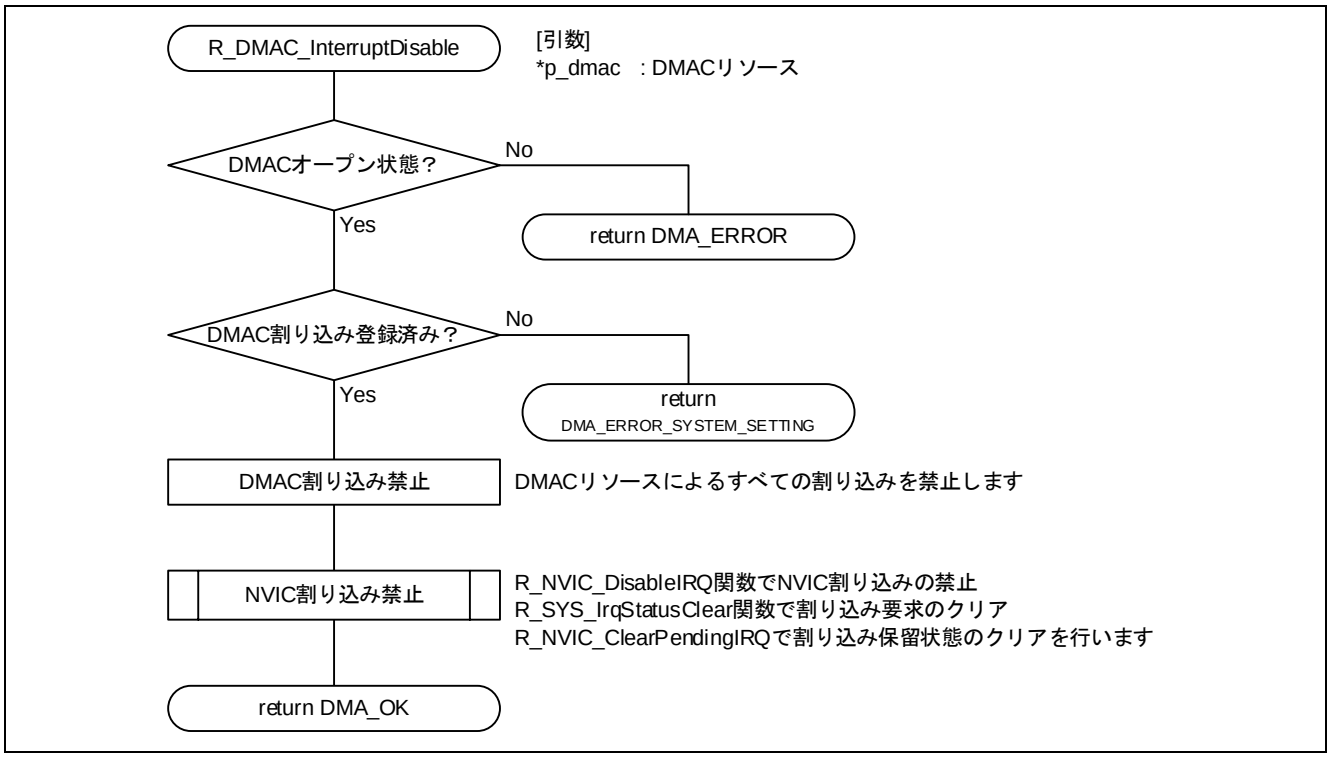

図 4-8 R_DMAC_InterruptDisable 関数処理フロー

4.1.7 R_DMAC_GetState 関数

表 4-10 R_DMAC_GetState 関数仕様

書式	static e_dma_err_t R_DMAC_GetState(st_dma_state_t * const state, st_dmac_resources_t * const p_dmac)
仕様説明	DMAC のステータスを取得します
引数	st_dma_state_t * const state : ステータス格納ポインタ 取得したステータスを格納するバッファ。
	st_dmac_resources_t * const p_dmac : DMAC のリソース ステータスを取得する DMAC のリソースを指定します。
戻り値	DMA_OK 取得成功
	DMA_ERROR 取得失敗 DMAC オープン前に実行した場合、取得失敗となります

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	st_dma_state_t dmac0_state;
	
	main()
	{
	dmac0Drv-> GetState(dmac0_state);
	}

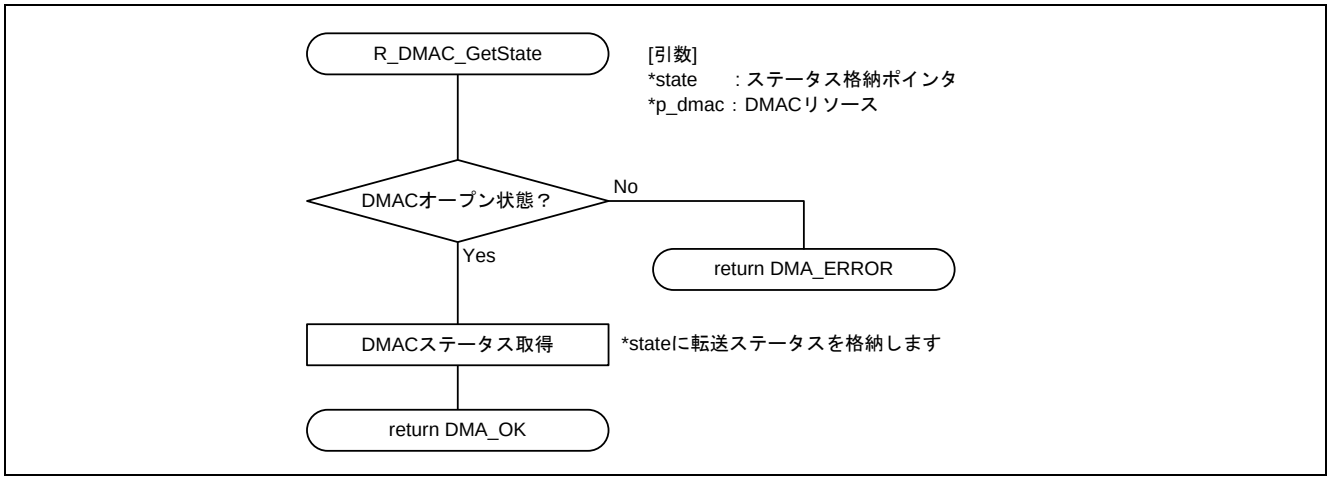

図 4-9 R_DMAC_GetState 関数処理フロー

4.1.8 R_DMAC_ClearState 関数

表 4-11 R_DMAC_ClearState 関数仕様

書式	static e_dma_err_t R_DMAC_ClearState(uint32_t clr_state, st_dmac_resources_t * const p_dmac)
仕様説明	指定した DMAC 割り込みフラグをクリアします
引数	uint32_t clr_state : クリア対象ステータス 以下の割り込みフラグを指定します。 複数指定する場合は、OR 演算を使用して定義してください。 （例 : DMA_CLR_STATE_ESIF | DMA_CLR_STATE_DTIF） ・DMA_CLR_STATE_ESIF : 転送エスケープ終了割り込みフラグ ・DMA_CLR_STATE_DTIF : 転送終了割り込みフラグ ・DMA_CLR_STATE_SOFT_REQ : ソフトウェアトリガによる転送要求フラグ
戻り値	DMA_OK クリア成功
	DMA_ERROR クリア失敗 DMA オープン前に実行した場合、クリア失敗となります

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	dmac0Drv-> ClearState(DMA_CLR_STATE_ESIF);
	}

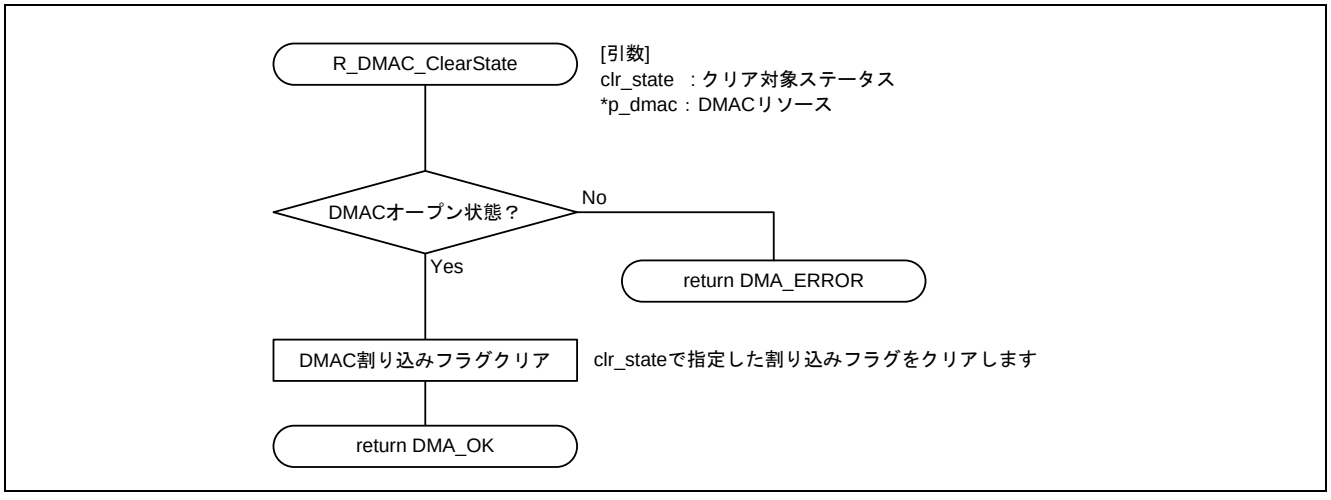

図 4-10 R_DMAC_ClearState 関数処理フロー

4.1.9 R_DMAC_GetTransferByte 関数

表 4-12 R_DMAC_GetTransferByte 関数仕様

書式	static e_dma_err_t R_DMAC_GetTransferByte(uint32_t * transfer_byte, st_dmac_resources_t * const p_dmac)
仕様説明	DMA 転送バイト数を取得します
引数	uint32_t *transfer_byte :DMA 転送バイト格納ポインタ DMA 転送バイトを格納するポインタを指定します
	st_dmac_resources_t * const p_dmac : DMAC のリソース ステータスを取得する DMAC のリソースを指定します。
戻り値	DMA_OK DMA 転送バイト数取得成功
	DMA_ERROR DMA 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 DMA 転送バイト取得失敗となります ・DMA オープン前に実行した場合 ・ノーマル転送のフリーランモードで実行した場合
	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アクセス時は、第 1 引数は無効(任意の値を設定)、第 2 引数に DMA 転送バイト格納ポインタを指定してください。
	DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	uint32_t byte;
	dmac0Drv-> GetTransferByte(0, &byte);
	}

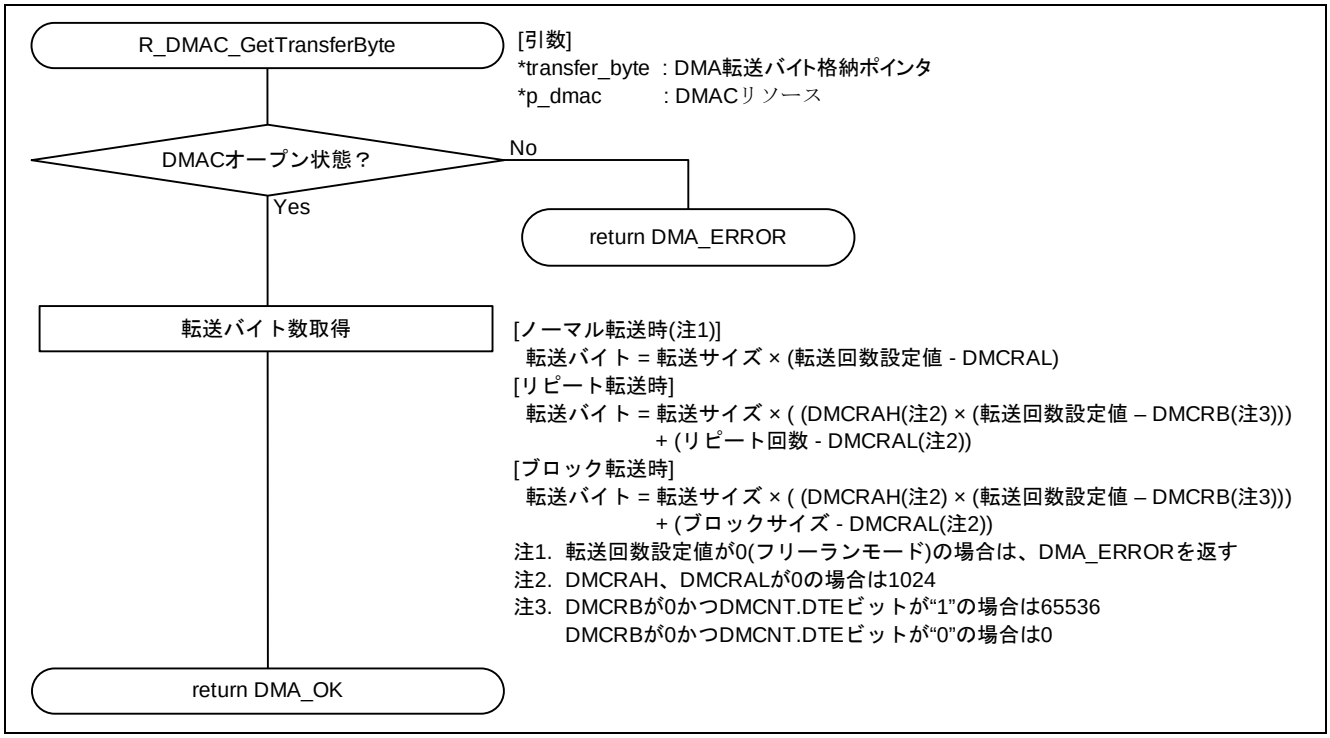

図 4-11 R_DMAC_GetTransferByte 関数処理フロー

4.1.10 R_DMAC_GetVersion 関数

表 4-13 R_DMAC_GetVersion 関数仕様

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

備考	インスタンスからのアクセス時は DMAC リソースの指定は不要です。
	
	[インスタンスからの関数呼び出し例]
	//DMAC driver instance(DMAC0)
	extern DRIVER_DMA Driver_DMAC0;
	DRIVER_DMA *dmac0Drv = &Driver_DMAC0;
	
	main()
	{
	uint32_t version;
	version = dmac0Drv-> GetVersion ();
	}

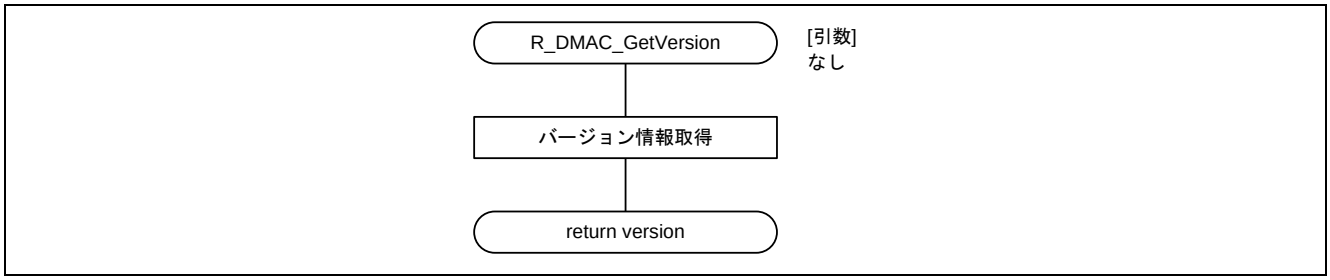

図 4-12 R_DMAC_GetVersion 関数処理フロー

4.1.11 dmac_active_set 関数

表 4-14 dmac_active_set 関数仕様

書式	static void dmac_active_set(uint16_t ch)
仕様説明	指定した DMAC チャネルをアクティブにします
引数	uint16_t ch : DMAC チャネル アクティブにするチャネル番号を指定します。
戻り値	なし
備考	-

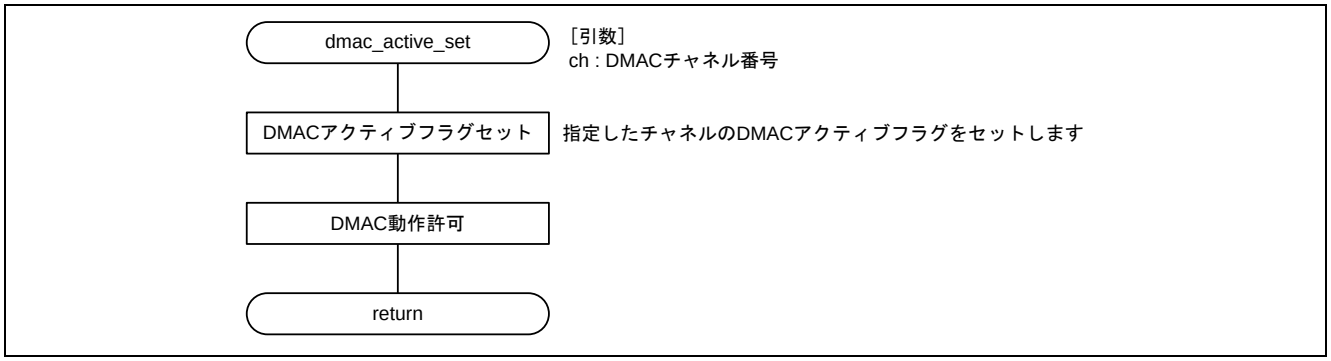

図 4-13 dmac_active_set 関数処理フロー

4.1.12 dmac_active_clear 関数

表 4-15 dmac_active_clear 関数仕様

書式	static void dmac_active_clear(uint16_t ch)
仕様説明	指定した DMAC チャネルのアクティブ状態を解除し、すべてのチャネルが非アクティブ状態になった場合、DMAC 動作を禁止にします
引数	uint16_t ch : DMAC チャネル アクティブ状態を解除するチャネルを指定します。
戻り値	なし
備考	-

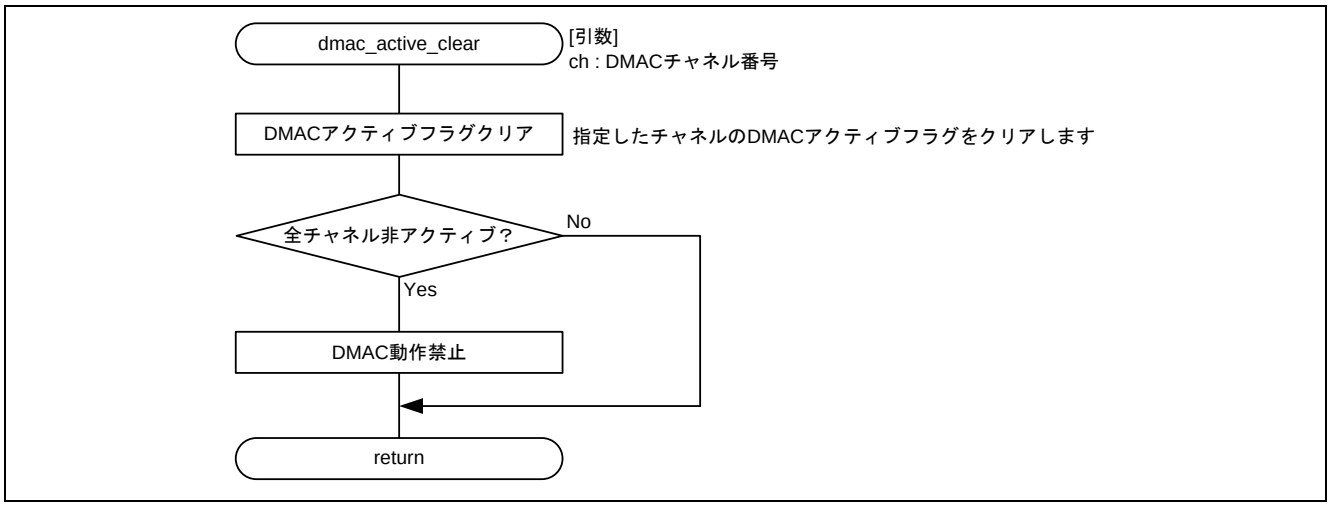

図 4-14 dmac_active_clear 関数処理フロー

4.1.13 dmac_cmd_refresh 関数

表 4-16 dmac_cmd_refresh 関数仕様

書式	static e_dma_err_t dmac_cmd_refresh(st_dma_refresh_data_t const * const p_data, st_dmac_resources_t * const p_dmac)
仕様説明	DMA 転送を再設定します
引数	st_dma_refresh_data_t const * const p_data : DMA 転送再設定データ 転送設定を指定します。
	st_dmac_resources_t * const p_dmac : DMAC のリソース 再設定する DMAC のリソースを指定します。
戻り値	DMA_OK 再設定成功
	DMA_ERROR_MODE モード異常値による再設定失敗 ノーマル、リピート、ブロック以外の転送方式設定を検出すると、モード異常値による再設定失敗となります
	DMA_ERROR_PARAMETER パラメータ異常による再設定失敗 以下のいずれかの設定値異常を検出するとパラメータ異常による再設定失敗となります ・転送回数(p_data_cfg->transfer_count)に不正な値を設定した場合 （設定可能範囲については表 4-5 転送モード毎の転送回数の設定可能範囲を参照してください） ・リピート転送、もしくはブロック転送を指定時に、 ブロック転送サイズ(p_data_cfg->block_size)に不正な値を設定した場合 (設定可能範囲 ：1～1024)
備考	-

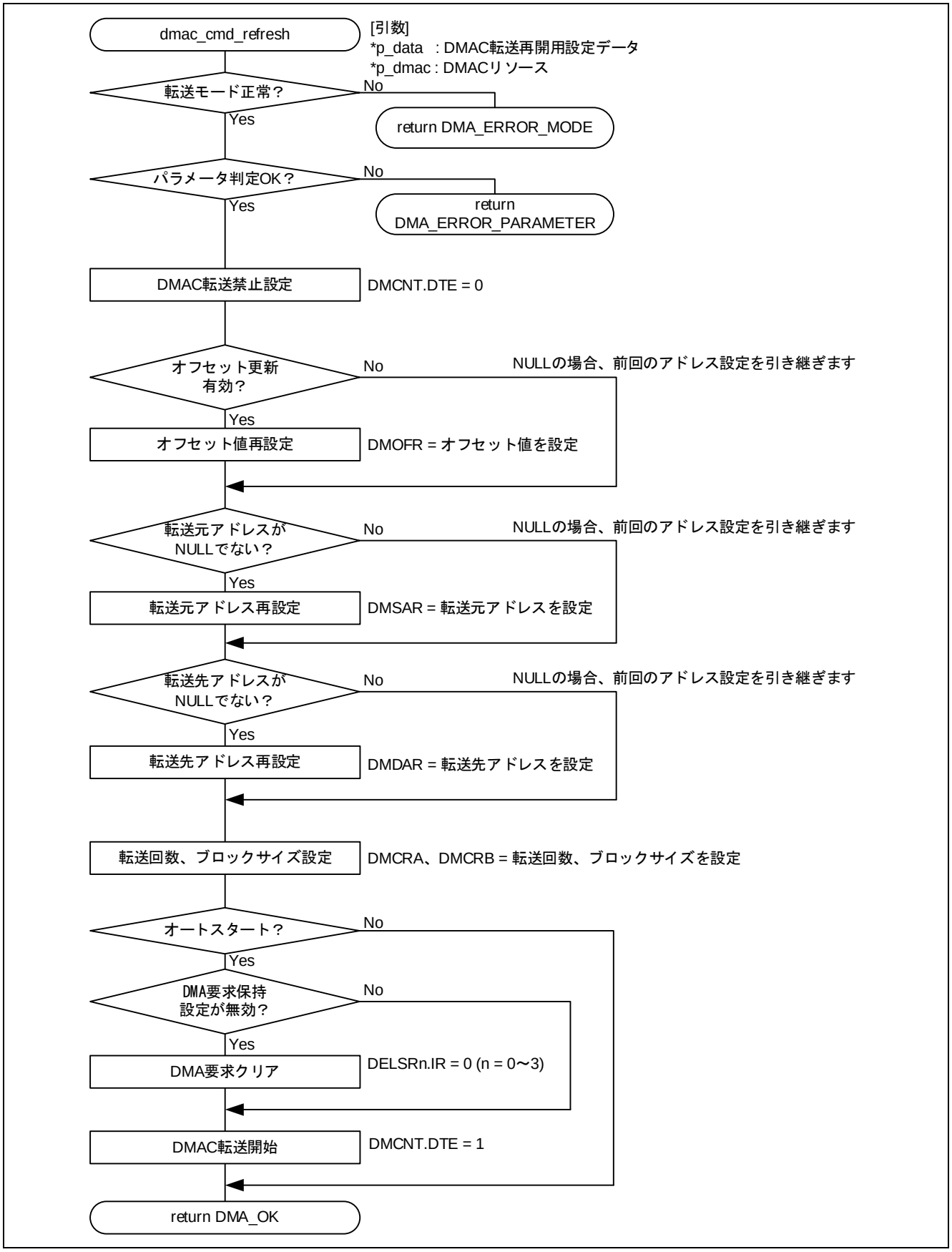

図 4-15 dmac_cmd_refresh 関数処理フロー

4.1.14 dmac_interrupt_handler 関数

表 4-17 dmac_interrupt_handler 関数仕様

書式	static void dmac_interrupt_handler(st_dmac_resources_t * const p_dmac)
仕様説明	DMAC 割り込み処理
引数	st_dmac_resources_t * const p_dmac : DMAC のリソース 転送する DMAC のリソースを指定します。
戻り値	なし
備考	-

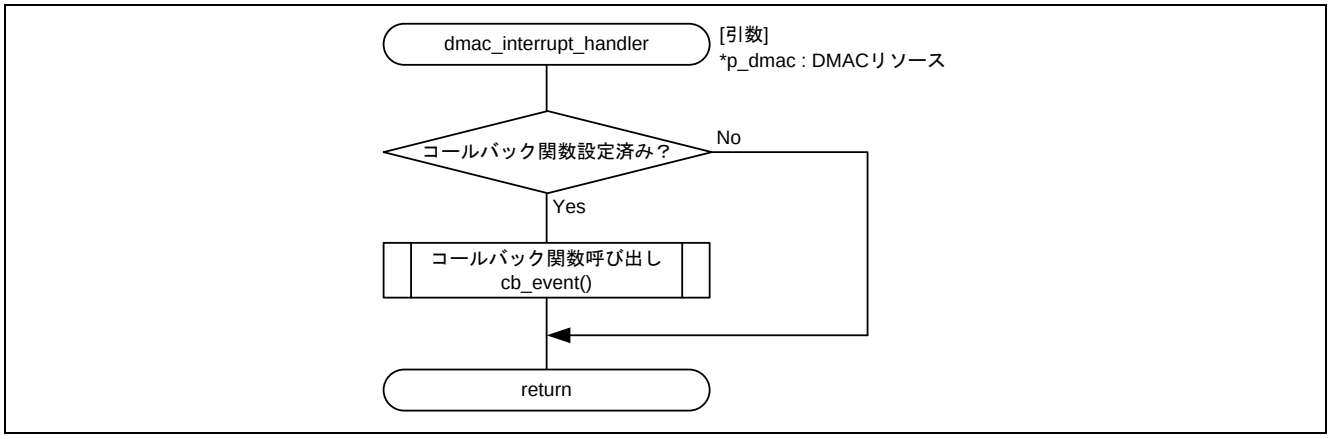

図 4-16 dmac_interrupt_handler 関数処理フロー

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

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

表 4-18 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
DMAC_VERSION_MAJOR	(注)	DMAC ドライバ メジャー番号
DMAC_VERSION_MINOR	(注)	DMAC ドライバ マイナー番号
DMA_ACTIVE_DMAC0	(0x0001)	DMAC0 アクティブフラグ
DMA_ACTIVE_DMAC1	(0x0002)	DMAC1 アクティブフラグ
DMA_ACTIVE_DMAC2	(0x0004)	DMAC2 アクティブフラグ
DMA_ACTIVE_DMAC3	(0x0008)	DMAC3 アクティブフラグ
DMA_ACTIVE_DTC	(0x8000)	DTC アクティブフラグ
DMA_CLR_STATE_ESIF	(1U<<1)	転送エスケープ終了割り込みフラグクリア用定義
DMA_CLR_STATE_DTIF	(1U<<2)	転送終了割り込みフラグクリア用定義
DMA_CLR_STATE_SOFT_REQ	(1U<<3)	DMA ソフトウェア起動フラグクリア用定義
DMAC_PRV_MIN_COUNT_VAL	(1)	DMA 転送回数下限値
DMAC_PRV_MAX_16BITS_COUNT_VAL	(65536)	DMAC16 ビット転送カウント指定時の上限値
DMAC_PRV_MAX_8BITS_COUNT_VAL	(256)	DMAC8 ビット転送カウント指定時の上限値
DMAC_PRV_MAX_10BITS_COUNT_VAL	(1024)	DMAC10 ビット転送カウント指定時の上限値
DMAC_PRV_EVENT_NUM_MIN	(0x01)	DMAC イベント番号下限値
DMAC_PRV_EVENT_NUM_MAX	(0xAA)	DMAC イベント番号上限値
DMAC_PRV_MAX_EXTRA_REP_AREA	(27)	アドレス拡張リピート領域サイズ上限値
DMAC_PRV_DELSR_IR_POS	(1U << 16)	DMAC 起動要求クリア用定義
DMAC_PRV_MIN_ADDR_OFFSET	(-16777216)	DMAC オフセットレジスタ下限値
DMAC_PRV_MAX_ADDR_OFFSET	(16777215)	DMAC オフセットレジスタ上限値
DMAC_PRV_FREERUN	(0xFFFFFFFF)	フリーランモード定義
DMAC_PRV_DMTMD_BYTE	(0)	転送サイズ 1 バイト判定定義
DMAC_PRV_DMTMD_WORD	(1)	転送サイズ 2 バイト判定定義
DMAC_PRV_DMTMD_LONG	(2)	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
DMAC_VERSION_MAJOR (1)
DMAC_VERSION_MINOR (01)

4.3 構造体定義

4.3.1 st_dmac_resources_t 構造体

DMAC のリソースを構成する構造体です。

表 4-19 st_dmac_resources_t 構造体

要素名	型	内容
*reg	volatile DMAC0_Type	対象の DMAC レジスタを示します
*delsr	volatile uint32_t	イベントリンク設定レジスタを示します
lock_id	e_system_mcu_lock_t	DMAC ロック ID
mstp_id	e_lpm_mstp_t	DMAC モジュールストップ ID
active_flag	uint16_t	DMAC アクティブフラグ
*info	st_dmac_mode_info_t	DMAC 状態情報
dmac_int_irq	IRQn_Type	DMAC 割り込みの割り当て番号
dmac_int_iesr_val	uint32_t	DMAC 割り込み用 IESR レジスタ設定値
dmac_int_priority	uint32_t	DMAC 割り込み優先レベル
dmac_int_callback	system_int_cb_t	DMAC 割り込みコールバック関数

4.3.2 st_dmac_mode_info_t 構造体

DMAC の状態を管理するための構造体です。

表 4-20 st_dmac_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
flags	uint8_t	ドライバ設定フラグ b0 :DMAC 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 :DMAC 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

DMAC ドライバ API から呼び出される外部関数を示します。

表 4-21 DMAC ドライバ API から呼び出す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/DMAC 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	設定エラーなし、かつ割り込み要因指定時
	R_NVIC_ClearPendingIRQ	
	R_NVIC_EnableIRQ	
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 DMA 割り込み設定について

DMA 割り込みの内、転送元アドレス拡張リピートエリアオーバフロー(DMA_INT_SRC_OVERFLOW)、転送先アドレス拡張リピートエリアオーバフロー(DMA_INT_DEST_OVERFLOW)、DMA 転送リピートサイズ終了(DMA_INT_REPEAT_END)割り込みでコールバック関数を使用する場合、InterruptEnable 関数実行時に OR 演算で転送エスケープ終了割り込み(DMA_INT_ESCAPE_END)を同時に許可する必要があります。
詳細は「2.4　DMA 割り込み」を参照してください。

5.3 DMA 割り込み要求発生時の動作について

転送元アドレス拡張リピートエリアオーバフロー(DMA_INT_SRC_OVERFLOW)、転送先アドレス拡張リピートエリアオーバフロー(DMA_INT_DEST_OVERFLOW)、DMA 転送リピートサイズ終了(DMA_INT_REPEAT_END)割り込み要求が発生すると、DMA 転送は停止(DMCNT.DTE = 0)します。割り込み要求発生後に DMA 転送を継続させる場合は、Control 関数にて DMA_CMD_START コマンドを実行してください。

5.4 NVIC への DMAC 割り込み登録

DMAC 割り込み(DMACn_INT(n=0～3))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 DMAC 割り込みが登録されていない場合、InterruptEnable 関数実行時に DMA_ERROR_SYSTEM_SETTING が戻ります。詳細は「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」を参照してください。

DMAC 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_PORT_IRQ0
                (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DMAC0_INT
                (SYSTEM_IRQ_EVENT_NUMBER0)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
#define SYSTEM_CFG_EVENT_NUMBER_DTC_COMPLETE
                (SYSTEM_IRQ_EVENT_NUMBER_NOT_USED)    /*!< Numbers 0/4/8/12/16/20/24/28 only */
・・・
```

図 5-1 r_system_cfg.h での NVIC への割り込み登録例

5.5 転送元アドレス、転送先アドレスの設定値の制限

Create 関数、DMA 転送再設定コマンド(DMA_CMD_REFRESH_DATA、DMA_CMD_REFRESH_EXTRA)、DMA 転送情報強制変更コマンド(DMA_CMD_CHANGING_DATA_FORCIBLY_SET)で転送元アドレス、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.10.2019	ー	初版
1.01	Dec.16.2019	ー プログラム (256KB)	256KB グループに対応 256KB の IO デファインにあわせて修正

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。