RX ファミリ

DMAC モジュール Firmware Integration Technology

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した DMAC モジュールについて説明します。本モジュールは DMAC を使用して、CPU を介さずにデータの転送を行います。以降、本モジュールを DMAC FIT モジュールと称します。

対象デバイス

- RX231 グループ、RX230 グループ
- RX23W グループ
- RX64M グループ、RX65N グループ、RX651 グループ
- RX66T グループ
- RX71M グループ
- RX72T グループ
- RX72M グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様に合わせて変更し、十分評価してください。

ターゲットコンパイラ

- ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

各コンパイラの動作確認環境に関する詳細な内容は、セクション「6.1 動作確認環境」を参照してください。

目次

1. 概要

1.1 DMACA FIT モジュールとは

本モジュールは API として、プロジェクトに組み込んで使用します。本モジュールの組み込み方については、「2.12 FITモジュールの追加方法」を参照してください。

1.2 DMACA FIT モジュールの概要

DMAC は、CPU を介さずにデータを転送します。DMAC は転送要求の発生により、転送元アドレスのデータを転送先アドレスへ転送します。

詳細は、ユーザーズマニュアル ハードウェア編の「DMA コントローラ」を参照してください。

(1) 転送モード

DMAC は、以下の転送モードをサポートします。

- ノーマル転送モード
- リピート転送モード
- ブロック転送モード

(2) 拡張リピートエリア機能

DMAC には転送元アドレス、転送先アドレスに拡張リピートエリアを設定する機能があります。拡張リピートエリアを設定すると、アドレスレジスタは拡張リピートエリアに指定した範囲のアドレス値を繰り返します。ただし、リピート領域またはブロック領域に指定したエリア（転送元または転送先）を拡張リピートエリアには指定しないでください。

(3) オフセットを使ったアドレス更新機能（DMAC0 のみ）

転送元アドレス、転送先アドレスの更新方法の種類として、固定／インクリメント／デクリメントの他にオフセット加算があります。オフセット加算では、1 データの転送を行うたびに DMAC オフセットレジスタに設定した値をアドレスに加算します。この機能により、途中のアドレスを飛ばしてデータ転送ができます。DMAC オフセットレジスタに 2 の補数で負の値を設定すると、オフセットによるアドレスの減算も可能です。ただし、オフセットには設定範囲の制限があるため、ご注意ください。

例えば、RX64M の場合、オフセットの設定範囲は、0byte ～ (16M-1) bytes (00000000h ~ 00FFFFFFh)、- 16M bytes ～ -1 byte (FF000000h ~ FFFFFFFFh)です。

(4) DMACA FIT モジュールの使用条件

使用条件は、以下です。

- r_bsp のデフォルトのロック機能を使用すること
- DMAC 用モジュールストップ設定ビットと DTC 用モジュールストップ設定ビットが共通であること

1.3 API の概要

表 1.1 に本モジュールに含まれる API 関数を示します。

表 1.1API 関数一覧

関数名	関数説明
R_DMACA_Init()	モジュール情報初期化処理
R_DMACA_Open()	チャネル別初期化処理
R_DMACA_Close()	チャネル別終了処理
R_DMACA_Create()	チャネル別レジスタと起動要因の設定処理
R_DMACA_Control()	動作設定処理
R_DMACA_Int_Callback()	チャネル別転送終了割り込み／転送エスケープ終了割り込み用コールバック関数の登録処理
R_DMACA_Int_Enable()	チャネル別転送終了割り込み／転送エスケープ終了割り込み許可処理
R_DMACA_Int_Disable()	チャネル別転送終了割り込み／転送エスケープ終了割り込み禁止処理
R_DMACA_GetVersion()	バージョン情報の取得処理

2. API 情報

本 FIT モジュールは、下記の条件で動作を確認しています。

2.1 ハードウェアの要求

ご使用になる MCU が以下の機能をサポートしている必要があります。

- DMAC(DMACA)
- ICU

2.2 ソフトウェアの要求

このドライバは以下の FIT モジュールに依存しています。

- ボードサポートパッケージ (r_bsp) v5.20 以上

2.3 サポートされているツールチェイン

本 FIT モジュールは「6.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

2.4　使用する割り込みベクタ

R_DMACA_Int_Enable()関数を実行すると、引数のチャネルと割り込み優先レベルに対応した転送終了割り込み、およびエスケープ転送終了割り込みが有効になります。

表 2-1 に本 FIT モジュールが使用する割り込みベクタを示します。

表 2-1 使用する割り込みベクター覧

デバイス	割り込みベクタ
RX230/RX231/RX23W	DMAC0I 割り込み[チャネル 0] （ベクタ番号：198）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：199）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：200）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：201）
RX64M	DMAC0I 割り込み[チャネル 0] （ベクタ番号：120）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：121）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：122）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：123）
	DMAC74I 割り込み[チャネル 4～7] （ベクタ番号：124）
RX65N/RX651	DMAC0I 割り込み[チャネル 0] （ベクタ番号：120）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：121）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：122）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：123）
	DMAC74I 割り込み[チャネル 4～7] （ベクタ番号：124）
RX66T	DMAC0I 割り込み [チャネル 0]（ベクタ番号：120）
	DMAC1I 割り込み [チャネル 1]（ベクタ番号：121）
	DMAC2I 割り込み [チャネル 2]（ベクタ番号：122）
	DMAC3I 割り込み [チャネル 3]（ベクタ番号：123）
	DMAC74I 割り込み [チャネル 4～7]（ベクタ番号：124）
RX71M	DMAC0I 割り込み[チャネル 0] （ベクタ番号：120）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：121）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：122）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：123）
	DMAC74I 割り込み[チャネル 4～7] （ベクタ番号：124）
RX72T	DMAC0I 割り込み[チャネル 0] （ベクタ番号：120）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：121）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：122）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：123）
	DMAC74I 割り込み[チャネル 4～7] （ベクタ番号：124）
RX72M	DMAC0I 割り込み[チャネル 0] （ベクタ番号：120）
	DMAC1I 割り込み[チャネル 1] （ベクタ番号：121）
	DMAC2I 割り込み[チャネル 2] （ベクタ番号：122）
	DMAC3I 割り込み[チャネル 3] （ベクタ番号：123）
	DMAC74I 割り込み[チャネル 4～7] （ベクタ番号：124）

2.5 ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_dmaca_rx_if.h に記載しています。

2.6 整数型

このプロジェクトは ANSI C99 を使用しています。これらの型は stdint.h で定義されています。

2.7 コンパイル時の設定

本モジュールのコンフィギュレーションオプションの設定は、r_dmaca_rx_config.h で行います。オプション名および設定値に関する説明を、下表に示します。

Configuration options in r_dmaca_rx_config.h	
DMACA_CFG_PARAM_CHECKING_ENABLE　1	パラメータチェック処理をコードに含めるか選択できます。 “0”の場合、パラメータチェック処理をコードから省略します。 “1”の場合、パラメータチェック処理をコードに含めます。 “0”を選択すると、パラメータチェック処理をコードから省略できるため、コードサイズが削減できます。
DMACA_CFG_USE_DTC_FIT_MODULE　0	DMACA FIT モジュールと共に DTC FIT モジュールを使用するかどうかを設定します。 “0”の場合、DTC FIT モジュールは使用しません。 “1”の場合、DMACA FIT モジュールと共に DTC FIT モジュールを使用します。

2.8　コードサイズ

本モジュールのコードサイズを下表に示します。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.7 コンパイル時の設定」のコンフィギュレーションオプションによって決まります。掲載した値は、「2.3 サポートされているツールチェイン」の C コンパイラでコンパイルオプションがデフォルト時の参考値です。コンパイルオプションのデフォルトは最適化レベル：2、最適化のタイプ：サイズ優先、データ・エンディアン：リトルエンディアンです。コードサイズは C コンパイラのバージョンやコンパイルオプションにより異なります。

ROM、RAM およびスタックのコードサイズ

デバイス		使用メモリ					
		ルネサス製コンパイラ		GCC		IAR コンパイラ	
		パラメータチェック処理あり	パラメータチェック処理なし	パラメータチェック処理あり	パラメータチェック処理なし	パラメータチェック処理あり	パラメータチェック処理なし
RX231	ROM	1598 バイト	1253 バイト	2840 バイト	2296 バイト	2860 バイト	2352 バイト
	RAM	36 バイト	36 バイト	120 バイト	20 バイト	42 バイト	42 バイト
	最大使用ユーザスタック	36 バイト	36 バイト	-	-	136 バイト	136 バイト
RX23W	ROM	1548 バイト	1203 バイト	-	-	-	-
	RAM	36 バイト	36 バイト	-	-	-	-
	最大のスタック使用量	72 バイト	72 バイト	-	-	-	-
RX65N	ROM	1764 バイト	1419 バイト	3352 バイト	2808 バイト	3461 バイト	2925 バイト
	RAM	72 バイト	72 バイト	40 バイト	40 バイト	76 バイト	76 バイト
	最大使用ユーザスタック	36 バイト	36 バイト	-	-	136 バイト	136 バイト
RX66T	ROM	1,732 バイト	1478 バイト	3376 バイト	2832 バイト	3439 バイト	2916 バイト
	RAM	72 バイト	72 バイト	40 バイト	40 バイト	76 バイト	76 バイト
	最大ユーザスタック	36 バイト	36 バイト	-	-	148 バイト	148 バイト
RX71M	ROM	1761 バイト	1416 バイト	3344 バイト	2800 バイト	3446 バイト	2925 バイト
	RAM	72 バイト	72 バイト	40 バイト	40 バイト	76 バイト	76 バイト
	最大使用ユーザスタック	36 バイト	36 バイト	-	-	148 バイト	148 バイト
RX72T	ROM	1773 バイト	1428 バイト	3312 バイト	2768 バイト	3446 バイト	2925 バイト
	RAM	72 バイト	72 バイト	40 バイト	40 バイト	76 バイト	76 バイト
	最大使用ユーザスタック	36 バイト	36 バイト	-	-	148 バイト	148 バイト
RX72M	ROM	1776 バイト	1431 バイト	3472 バイト	2920 バイト	3338 バイト	2817 バイト
	RAM	72 バイト	72 バイト	40 バイト	40 バイト	72 バイト	72 バイト
	最大使用ユーザスタック	80 バイト	80 バイト	-	-	156 バイト	156 バイト

注 1：表示のメモリサイズが適用されるのは、「Configuration Overview」に掲載したデフォルト設定を使用している場合です。選択した定義に応じて、メモリサイズは異なります。

注 2：動作確認条件で、以下の事項を記載済みです。

- r_dmaca_rx.c
- r_dmaca_rx_target.c

注 3：必要なメモリサイズは、C コンパイラのバージョンとコンパイル条件によって異なります。

注 4：表示のメモリサイズが適用されるのは、リトルエンディアンの場合です。同様に、エンディアンモードに応じて、上記のメモリサイズは異なります。

2.9 引数

API 関数の引数である構造体を示します。この構造体は API 関数のプロトタイプ宣言とともに r_dmaca_rx_if.h で記載されています。

```
typedef struct st_dmaca_transfer_data_cfg
{
    dmaca_transfer_mode_t     transfer_mode;          /* Transfer Mode  */
    dmaca_repeat_block_side_t repeat_block_side;
                              /* Repeat Area in Repeat or Block Transfer Mode */
    dmaca_data_size_t         data_size;  /* Transfer Data Size   */
    dmaca_activation_source_t act_source;       /* Activation Source    */
    dmaca_request_source_t    request_source;   /* Transfer Request Source    */
    dmaca_dti_t   dtie_request;     /* Transfer End Interrupt Request   */
    dmaca_esi_t   esie_request;     /* Transfer Escape End Interrupt Request  */
    dmaca_rpti_t  rptie_request;    /* Repeat Size End Interrupt Request      */
    dmaca_sari_t  sarie_request;    /* Source Address Extended Repeat Area
Overflow Interrupt Request    */
    dmaca_dari_t  darie_request;    /* Destination Address Extended Repeat Area
Overflow Interrupt Request    */
    dmaca_src_addr_mode_t     src_addr_mode;    /* Address Mode of Source     */
    dmaca_src_addr_repeat_area_t    src_addr_repeat_area;/* Source Address
Extended Repeat Area    */
    dmaca_des_addr_mode_t     des_addr_mode;/* Address Mode of Destination    */
    dmaca_des_addr_repeat_area_t    des_addr_repeat_area;   /* Destination
Address Extended Repeat Area  */
    uint32_t     offset_value;/* Offset value for DMA Offset Register (DMOFR) */
    dmaca_interrupt_select_t  interrupt_sel;    /* Configurable Options for
Interrupt Select  */
    void    *p_src_addr;                  /* Start Address of Source    */
    void    *p_des_addr;       /* Start Address of Destination    */
    uint32_t      transfer_count;                           /* Transfer Count */
    uint16_t      block_size;        /* Repeat Size or Block Size */
    uint8_t  rsv[2];
} dmaca_transfer_data_cfg_t;
```

```
typedef enum e_dmaca_command
{
    DMACA_CMD_ENABLE = 0,     /* Enables DMA transfer.      */
    DMACA_CMD_ALL_ENABLE,     /* Enables DMAC activation.   */
    DMACA_CMD_RESUME,   /* Resumes DMA transfer.      */
    DMACA_CMD_DISABLE,  /* Enables DMA transfer.      */
    DMACA_CMD_ALL_DISABLE,    /* Disables DMAC activation.  */
    DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ, /* SWREQ bit is cleared automatically
after DMA transfer.     */
    DMACA_CMD_SOFT_REQ_NOT_CLR_REQ,       /* SWREQ bit is not cleared after DMA
transfer.   */
    DMACA_CMD_SOFT_REQ_CLR,   /* Clears DMACA Software request flag.    */
    DMACA_CMD_STATUS_GET,     /* Gets the current status of DMACA.      */
    DMACA_CMD_ESIF_STATUS_CLR,      /* Clears Transfer Escape End Interrupt
Flag. */
    DMACA_CMD_DTIF_STATUS_CLR       /* Clears Transfer Interrupt Flag.  */
} dmaca_command_t;
```

2.10 戻り値

API 関数の戻り値を示します。この列挙型は API 関数のプロトタイプ宣言とともに r_dmaca_rx_if.h で記載されています。

```
typedef enum e_dmaca_return
{
    DMACA_SUCCESS_OTHER_CH_BUSY = 0,      /* Other DMAC channels are locked,  */
                                /*  so that cannot set to module stop state.    */
    DMACA_SUCCESS_DTC_BUSY,               /* DTC is locked, */
                                /* so that cannot set to module stop state.     */
    DMACA_SUCCESS,
    DMACA_ERR_INVALID_CH,           /* Channel is invalid.  */
    DMACA_ERR_INVALID_ARG,    /* Parameters are invalid.    */
    DMACA_ERR_INVALID_HANDLER_ADDR,       /* Invalid function address is set, */
                          /*  and any previous function has been unregistered.  */
    DMACA_ERR_INVALID_COMMAND,      /* Command is invalid.  */
    DMACA_ERR_NULL_PTR, /* Argument pointers are NULL.      */
    DMACA_ERR_BUSY,           /* Resource has been locked by other process.   */
    DMACA_ERR_SOFTWARE_REQUESTED,         /* DMA transfer request by software
has been generated already,   */
                                      /*  so that cannot execute command. */
    DMACA_ERR_SOFTWARE_REQUEST_DISABLED,  /* Transfer Request Source is not
Software.*/
    DMACA_ERR_INTERNAL                    /* DMACA driver internal error      */
 } dmaca_return_t;
```

2.11 コールバック関数

本モジュールでは、転送終了割り込み、およびエスケープ転送終了割り込みが発生したタイミングで、ユーザが設定したコールバック関数を呼び出します。

コールバック関数は、「2.9 引数」に記載された構造体メンバ“R_DMACA_Int_Callback()” に、ユーザの関数のアドレスを格納することで設定されます。コールバック関数が呼び出されるとき、定数が格納された変数が、引数として渡されます。

引数の型は void ポインタ型で渡されるため、コールバック関数の引数は下記の例を参考に void 型のポインタ変数としてください。

コールバック関数内部で値を使うときはキャストして値を使用してください。

2.12 FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1) e2 studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e2 studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e2 studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2) e2 studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e2 studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3) CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e2 studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4) CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.13 for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

```
while 文の例：
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized.*/
}

for 文の例：
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while 文の例：
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API 関数

R_DMACA_Init()

DMAC 内部情報を初期化する関数です。

Format

void R_DMACA_Init (void)

Parameters

なし

Return Values

なし

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

各 DMAC チャネル使用状況（内部情報）を初期化します。

また、各 DMAC 転送終了割り込み／転送エスケープ終了割り込み（DMAC0I、DMAC1I、DMAC2I、DMAC3I、DMAC74I）用のコールバック関数の登録を全て解除します。DMAC 転送終了割り込み／転送エスケープ終了割り込みを使用する場合は、事前に R_DMACA_Init()関数を実行後、後述の R_DMACA_Int_Callback()関数でコールバック関数を登録してください。

Example

```
#include "r_dmaca_rx_if.h"

/* DMACA driver を使用する場合は、最初に R_DMACA_Init()関数を実行してください */
R_DMACA_Init();
```

Special Notes:

使用する場合、最初に実行してください。ハードウェアセットアップ時に実行することを推奨します。

R_DMACA_Open()

DMACA FIT モジュールの API を使用する際、R_DMACA_Init()関数コール後に使用する関数です。

Format

```
dmaca_return_t      R_DMACA_Open (
        uint8_t channel
)
```

Parameters

uint8_t channel
DMAC チャネル番号

Return Values

[DMACA_SUCCESS]	*/* Successful operation*/*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*
[DMACA_ERR_BUSY]	*/* Resource has been locked by other process.*/*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

引数 channel で指定した DMAC チャネルをロック状態*1 に設定した後、初期設定します。

DMAC のモジュールストップを解除し、DMAC の起動を許可します。また、指定した DMAC チャネルの起動要因選択レジスタを初期化します。

Note 1：DMACA FIT モジュールは、r_bsp のデフォルトのロック機能を使用します。そのため、正常終了時には、指定した DMAC チャネルがロック状態になります。

Example

```
#include “r_dmaca_rx_if.h”
volatile dmaca_return_t  ret;

ret = R_DMACA_Open(DMACA_CH0);
```

Special Notes:

なし

R_DMACA_Close()

使用中の DMAC チャネルのリソースを開放する際に使用する関数です。

Format

```
dmaca_return_t        R_DMACA_Close (
        uint8_t      channel
)
```

Parameters

uint8_t channel
DMAC チャネル番号

Return Values

[DMACA_SUCCESS]	*/* Successful operation*/*
[DMACA_SUCCESS_OTHER_CH_BUSY]	*/* Successful operation.Other DMAC channels are locked.*/*
[DMACA_SUCCESS_DTC_BUSY]	*/* Successful operation.DTC is locked.*/*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*
[DMACA_ERR_INTERNAL]	*/* DMACA driver internal error */*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

引数 channel で指定した DMAC チャネルのロック*1 を解除し、指定した DMAC チャネルの DMA 転送許可(DTE)ビットをクリアし、DMA 転送を禁止させます。また、全チャネルの DMAC のロックが解除されている場合は、DMAC 動作許可(DMST)ビットをクリアし DMAC の起動を禁止させます。

さらに、DTC のロックが解除されている場合は、DMAC と DTC をモジュールストップ状態*2 に設定します。

Note 1：DMACA FIT モジュールは、r_bsp のデフォルトのロック機能を使用します。そのため、正常終了時には、指定した DMAC チャネルがロック解除状態になります。

Note 2：DMAC 用モジュールストップ設定ビットと DTC 用モジュールストップ設定ビットが共通であるため、DTC のロック状態も確認し、モジュールストップ状態に設定します。（詳細はユーザーズマニュアル ハードウェア編の「消費電力低減機能」を参照。）

なお、使用するモジュールの組み合わせによって、以下のとおり処理方法を変えてください。

DMAC 制御	DTC 制御	処理方法
DMACA FIT モジュール （ロック機能制御機能有、 DTC のロック状態確認機能 有）	DTC FIT モジュール （ロック機能制御機能有、 DMAC のロック状態確認機能 有）	Case 1 を参照
上記以外		Case 2 を参照

Case 1：r_bsp のデフォルトのロック機能を使用し、DTC を DTC FIT モジュール*1 で制御

r_bsp のデフォルトのロック機能を利用して DMAC の全チャネルのロックと DTC のロックが解除されていることを確認し、DMAC をモジュールストップします。

Note 1：DTC FIT モジュールが、DMAC のロック状態も確認し、モジュールストップ制御機能を持つことが条件です。

Case 2：上記以外の制御の場合

ユーザ自身で DMAC の全チャネルのロック解除状態と DTC のロック解除状態（使用されていないこと）を確認してください。DMACA FIT モジュールでは、そのための空関数を用意しています。

r_bsp のデフォルトのロック機能を使用しない場合は、r_dmaca_rx_target.c ファイル内の r_dmaca_check_DMACA_DTC_locking_byUSER()関数の/* do something */行の後に DMAC 全チャネルのロック状態と DTC のロック状態を確認するプログラムを記述してください。

r_bsp のデフォルトのロック機能を使用する場合であっても、DTC FIT モジュールを使用せずに DTC を制御している場合は、r_dmaca_rx_target.c ファイル内の r_dmaca_check_DTC_locking_byUSER()関数の/* do something */ 行の後に DTC のロック状態を確認するプログラムを記述してください。

なお、r_dmaca_check_DMACA_DTC_locking_byUSER()関数もしくは r_dmaca_check_DTC_locking_byUSER()関数の戻り値は、以下の dmaca_chk_looking_sw_t 型としてください。

dmaca_chk_locking_sw_t 型

```
DMACA_ALL_CH_UNLOCKED_AND_DTC_UNLOCKED
                                    /* All DMAC channels and DTC are unlocked.  */
DMACA_ALL_CH_UNLOCKED_BUT_DTC_LOCKED
                        /* All DMAC channels are unlocked, but DTC is locked.  */
DMACA_LOCKED_CH_EXIST   /* Other DMAC channels are locked.  */
```

Example

```
#include “r_dmaca_rx_if.h”
volatile dmaca_return_t  ret;

ret = R_DMACA_Close(DMACA_CH0);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

DTC FIT モジュールを使用せずに DTC を制御する場合、本関数コールによりモジュールストップ状態に設定されないように、DTC の使用状況を監視し、DTC のロックおよびロック解除を制御してください。DTC 転送設定時には、DTC が起動していない状態であってもロック状態を保持する必要があることに注意してください。

R_DMACA_Create()

DMAC のレジスタ設定と起動要因を設定する関数です。

Format

```
dmaca_return_t          R_DMACA_Create (
    uint8_t                     channel,
    damca_transfer_data_cfg_t   * p_data_cfg
)
```

Parameters

uint8_t channel
DMAC チャネル番号

*damca_transfer_data_cfg_t *p_data_cfg*
DMAC 転送情報 dmaca_transfer_data_cfg_t 構造体のポインタ

dmaca_transfer_data_cfg_t 構造体メンバと設定値（1/4）

構造体メンバ	概略	設定値	設定内容
transfer_mode	Transfer Mode	DMACA_TRANSFER_MODE_NORMAL	Normal transfer
		DMACA_TRANSFER_MODE_REPEAT	Repeat transfer
		DMACA_TRANSFER_MODE_BLOCK	Block transfer
repeat_block_side	Repeat Area in Repeat or Block Transfer Mode	DMACA_REPEAT_BLOCK_DESTINATION	The destination is specified as the repeat area or block area.
		DMACA_REPEAT_BLOCK_SOURCE	The source is specified as the repeat area or block area.
		DMACA_REPEAT_BLOCK_DISABLE	The repeat area or block area is not specified.
data_size	Transfer Data Size	DMACA_DATA_SIZE_BYTE	8-bit
		DMACA_DATA_SIZE_WORD	16-bit
		DMACA_DATA_SIZE_LWORD	32-bit
act_source	DMACA Activation Source	Iodefine.h ファイルの列挙型の定数リスト enum_ir のメンバ	DMAC 起動要因とする割り込みベクタ番号
request_source	DMACA Transfer Request Source	DMACA_TRANSFER_REQUEST_SOFTWARE	Software
		DMACA_TRANSFER_REQUEST_PERIPHERAL	Interrupts from peripheral modules or external interrupt input pins.
dtie_request	Transfer End Interrupt Request	DMACA_TRANSFER_END_INTERRUPT_DISABLE	Disables the transfer end interrupt request.
		DMACA_TRANSFER_END_INTERRUPT_ENABLE	Enables the transfer end interrupt request.

esie_request	Transfer Escape End Interrupt Request	DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE	Disables the transfer escape end interrupt request.
		DMACA_TRANSFER_ESCAPE_END_INTERRUPT_ENABLE	Enables the transfer escape end interrupt request.
rptie_request	Repeat Size End Interrupt Request	DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE	Disables the repeat size end interrupt request.
		DMACA_REPEAT_SIZE_END_INTERRUPT_ENABLE	Enables the repeat size end interrupt request.
sarie_request	Source Address Extended Repeat Area Overflow Interrupt Request	DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE	Disables an interrupt request for an extended repeat area overflow on the source address
		DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_ENABLE	Enables an interrupt request for an extended repeat area overflow on the source address
darie_request	Destination Address Extended Repeat Area Overflow Interrupt Request	DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE	Disables an interrupt request for an extended repeat area overflow on the destination address
		DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_ENABLE	Enables an interrupt request for an extended repeat area overflow on the destination address
src_addr_mode	Address Mode of Source	DMACA_SRC_ADDR_FIXED	Destination address is fixed.
		DMACA_SRC_ADDR_OFFSET	Offset addition
		DMACA_SRC_ADDR_INCR	Source address is incremented
		DMACA_SRC_ADDR_DECR	Source address is decremented
src_addr_repeat_area	Source Address Extended Repeat Area	DMACA_SRC_ADDR_EXT_REP_AREA_NONE	Not specified
		DMACA_SRC_ADDR_EXT_REP_AREA_2B	2 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4B	4 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8B	8 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16B	16 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32B	32 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_64B	64 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128B	128 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_256B	256 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_512B	512 bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_1KB	1K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_2KB	2K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_4KB	4K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_8KB	8K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_16KB	16K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_32KB	32K bytes 64K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_64KB	128K bytes 256K bytes
		DMACA_SRC_ADDR_EXT_REP_AREA_128K	512K bytes 1M bytes

		B DMACA_SRC_ADDR_EXT_REP_AREA_256K B DMACA_SRC_ADDR_EXT_REP_AREA_512K B DMACA_SRC_ADDR_EXT_REP_AREA_1MB DMACA_SRC_ADDR_EXT_REP_AREA_2MB DMACA_SRC_ADDR_EXT_REP_AREA_4MB DMACA_SRC_ADDR_EXT_REP_AREA_8MB DMACA_SRC_ADDR_EXT_REP_AREA_16M B DMACA_SRC_ADDR_EXT_REP_AREA_32M B DMACA_SRC_ADDR_EXT_REP_AREA_64M B DMACA_SRC_ADDR_EXT_REP_AREA_128 MB	2M bytes 4M bytes 8M bytes 16M bytes 32M bytes 64M bytes 128M bytes
des_addr_mode	Address Mode of Destination	DMACA_DES_ADDR_FIXED	Destination address is fixed.
		DMACA_DES_ADDR_OFFSET	Offset addition
		DMACA_DES_ADDR_INCR	Destination address is incremented.
		DMACA_DES_ADDR_DECR	Destination address is decremented.
des_addr_repeat_area	Destination Address Extended Repeat Area	DMACA_DES_ADDR_EXT_REP_AREA_NONE	Not specified
		DMACA_DES_ADDR_EXT_REP_AREA_2B DMACA_DES_ADDR_EXT_REP_AREA_4B DMACA_DES_ADDR_EXT_REP_AREA_8B DMACA_DES_ADDR_EXT_REP_AREA_16B DMACA_DES_ADDR_EXT_REP_AREA_32B DMACA_DES_ADDR_EXT_REP_AREA_64B DMACA_DES_ADDR_EXT_REP_AREA_128 DMACA_DES_ADDR_EXT_REP_AREA_256B DMACA_DES_ADDR_EXT_REP_AREA_512B DMACA_DES_ADDR_EXT_REP_AREA_1KB DMACA_DES_ADDR_EXT_REP_AREA_2KB DMACA_DES_ADDR_EXT_REP_AREA_4KB DMACA_DES_ADDR_EXT_REP_AREA_8KB DMACA_DES_ADDR_EXT_REP_AREA_16K B DMACA_DES_ADDR_EXT_REP_AREA_32K B DMACA_DES_ADDR_EXT_REP_AREA_64K B DMACA_DES_ADDR_EXT_REP_AREA_128K B DMACA_DES_ADDR_EXT_REP_AREA_256K B DMACA_DES_ADDR_EXT_REP_AREA_512K B DMACA_DES_ADDR_EXT_REP_AREA_1MB DMACA_DES_ADDR_EXT_REP_AREA_2MB DMACA_DES_ADDR_EXT_REP_AREA_4MB DMACA_DES_ADDR_EXT_REP_AREA_8MB	2 bytes 4 bytes 8 bytes 16 bytes 32 bytes 64 bytes 128 bytes 256 bytes 512 bytes 1K bytes 2K bytes 4K bytes 8K bytes 16K bytes 32K bytes 64K bytes 128K bytes 256K bytes 512K bytes 1M bytes 2M bytes 4M bytes 8M bytes 16M bytes 32M bytes 64M bytes 128M bytes

		DMACA_DES_ADDR_EXT_REP_AREA_16MB DMACA_DES_ADDR_EXT_REP_AREA_32MB DMACA_DES_ADDR_EXT_REP_AREA_64MB DMACA_DES_ADDR_EXT_REP_AREA_128MB	
offset_value	Offset value for DMA Offset Register (DMOFR)	32bit data 00000000h to 00FFFFFFh (0 bytes to (16M-1) bytes) FF000000h to FFFFFFFFh (-16M bytes to -1 byte) Note: Setting bits 31 to 25 is invalid.A value of bit 24 is extended to bits 31 to 25. Offset addition can be specified only for DMAC0. With R_DMACA_Create() function, setting this data is invalid except DMAC0.	Note: Offset subtraction can also be realized by setting a negative value. In this case, the negative value must be 2's complement.
interrupt_sel	Configurable Options for Interrupt Select	DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_ TRANSFER	At the beginning of transfer, clears the interrupt flag of the activation source to 0.
		DMACA_ISSUES_INTERRUPT_TO_CPU_END_OF_ TRANSFER	At the end of transfer, the interrupt flag of the activation source issues an interrupt to the CPU.
*p_src_addr	Start Address of Source	32bit data 00000000h to 0FFFFFFFh (256M bytes) F0000000h to FFFFFFFFh (256M bytes) Note: Setting bits 31 to 29 is invalid.A value of bit 28 is extended to bits 31 to 29.	Source address
*p_des_addr	Start Address of Destination		Destination address
transfer_count	Transfer Count	32bit data [Normal Transfer Mode] 00000001h to 0000FFFFh When the setting is 0000h, no specific number of transfer operations is set (free running mode) [Repeat Transfer Mode or Block Transfer Mode]. 00000001h to 00001000h	[Normal Transfer Mode] This data is set to DMCRAL register. [Repeat Transfer Mode or Block Transfer Mode] This data is set to DMCRB register.
block_size	Repeat Size or Block Size	16bit data [Normal Transfer Mode] Invalid [Repeat Transfer Mode or Block Transfer Mode]. 00000001h to 0000400h	[Normal Transfer Mode] Invalid [Repeat Transfer Mode or Block Transfer Mode] This data is set to DMCRAL register and DMCRAH register.

Return Values

[DMACA_SUCCESS]	*/* Successful operation */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*
[DMACA_ERR_INVALID_ARG]	*/* Parameters are invalid.*/*
[DMACA_ERR_NULL_PTR]	*/* Argument pointers are NULL.*/*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

引数の DMAC 転送情報 dmaca_transfer_data_cfg_t 構造体を参照し、指定した DMAC チャネルのレジスタを設定します。また、その DMAC チャネルに対する起動要因を設定します。

Example

Case1:ソフトウェアで DMAC 起動する場合

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_transfer_data_cfg_t td_cfg;
uint32_t src = 1234;
uint32_t des[3];

/* Operation – No Extended Repeat Area Function and No Offset Subtraction */
/* Source address is fixed
 * Transfer data size is 32-bit (long word).
 * DMAC transfer mode is Repeat mode & Source side is repeat area
 * At the beginning of transfer, clear the interrupt flag of the activation
source to 0.
 * Transfer Request source is software.*/

/* Set Transfer data configuration.*/
   td_cfg.transfer_mode             = DMACA_TRANSFER_MODE_REPEAT;
   td_cfg.repeat_block_side               = DMACA_REPEAT_BLOCK_SOURCE;
   td_cfg.data_size           = DMACA_DATA_SIZE_LWORD;
   td_cfg.act_source          = (dmaca_activation_source_t)0;
   td_cfg.request_source            = DMACA_TRANSFER_REQUEST_SOFTWARE;
   td_cfg.dtie_request        = DMACA_TRANSFER_END_INTERRUPT_DISABLE;
   td_cfg.esie_request  = DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE;
   td_cfg.rptie_request = DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE;
   td_cfg.sarie_request = DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.darie_request = DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg.src_addr_mode       = DMACA_SRC_ADDR_FIXED;
   td_cfg.src_addr_repeat_area            = DMACA_SRC_ADDR_EXT_REP_AREA_NONE;
   td_cfg.des_addr_mode       = DMACA_DES_ADDR_INCR;
   td_cfg.des_addr_repeat_area            = DMACA_DES_ADDR_EXT_REP_AREA_NONE;
   td_cfg.offset_value        = 0x00000000;
   td_cfg.interrupt_sel             =
DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_TRANSFER;
   td_cfg.p_src_addr    = (void *)&src;
   td_cfg.p_des_addr    = (void *)des;
   td_cfg.transfer_count            = 1;
   td_cfg.block_size    = 3;

/* Call R_DMACA_Create().*/
   ret = R_DMACA_Create(DMACA_CH0, &td_cfg);
```

Note: td_cfg.request_source が DMACA_TRANSFER_REQUEST_SOFTWARE の場合（DMAC への転送要求元をソフトウェアにしている場合）、R_DMACA_Create()関数は td_cfg.act_source の設定を無視します。

Case2:周辺モジュールを DMAC 起動要因とする場合(CMI1 割り込みを使用した例)

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_transfer_data_cfg_t td_cfg;
uint32_t src = 1234;
uint32_t des[3];

/* Operation – No Extended Repeat Area Function and No Offset Subtraction */
/* Source address is fixed.
 * Transfer data size is 32-bit (long word).
 * DMAC transfer mode is Repeat mode & Source side is repeat area
 * At the beginning of transfer, clear the interrupt flag of the activation
source to 0.
 * Transfer Request source is CMI1.*/

/* Set Transfer data configuration.*/
   td_cfg->transfer_mode            = DMACA_TRANSFER_MODE_REPEAT;
   td_cfg->repeat_block_side        = DMACA_REPEAT_BLOCK_SOURCE;
   td_cfg->data_size          = DMACA_DATA_SIZE_LWORD;
   td_cfg->act_source         = IR_CMT1_CMI1;
   td_cfg->request_source           = DMACA_TRANSFER_REQUEST_PERIPHERAL;
   td_cfg->dtie_request       = DMACA_TRANSFER_END_INTERRUPT_DISABLE;
   td_cfg->esie_request  = DMACA_TRANSFER_ESCAPE_END_INTERRUPT_DISABLE;
   td_cfg->rptie_request            = DMACA_REPEAT_SIZE_END_INTERRUPT_DISABLE;
   td_cfg->sarie_request       =
DMACA_SRC_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg->darie_request       =
DMACA_DES_ADDR_EXT_REP_AREA_OVER_INTERRUPT_DISABLE;
   td_cfg->src_addr_mode            = DMACA_SRC_ADDR_FIXED;
   td_cfg->src_addr_repeat_area     = DMACA_SRC_ADDR_EXT_REP_AREA_NONE;
   td_cfg->des_addr_mode      = DMACA_DES_ADDR_INCR;
   td_cfg->des_addr_repeat_area     = DMACA_DES_ADDR_EXT_REP_AREA_NONE;
   td_cfg->offset_addr  = 0;
   td_cfg->interrupt_sel       = DMACA_CLEAR_INTERRUPT_FLAG_BEGINNING_TRANSFER;
   td_cfg->p_src_addr   = (void *)&src;
   td_cfg->p_des_addr   = (void *)des;
   td_cfg->transfer_count     = 1;
   td_cfg->block_size   = 3;

/* Disable CMI1 interrupt request before calling R_DTC_Create().*/
IR(CMT1,CMI1) = 0;
IEN(CMT1,CMI1) = 0;

/* Call R_DMACA_Create().*/
   ret = R_DMACA_Create(DMACA_CH0, &td_cfg);
```

Special Notes:

なし

R_DMACA_Control()

DMAC の動作を制御する関数です。This function is run after calling R_DMACA_Open().

Format

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

Parameters

uint8_t channel

DMAC チャネル番号

dmaca_command_t command

DMAC 制御コマンド

Command	Description
DMACA_CMD_ENABLE	DMAC 転送を許可（チャネル単位で DMA 転送許可ビット制御）
DMACA_CMD_ALL_ENABLE	DMAC 起動を許可（DMAC 動作許可ビット制御）
DMACA_CMD_RESUME	DMAC 転送を再開（チャネル単位で DMA 転送許可ビット 制御）
DMACA_CMD_DISABLE	DMAC 転送を禁止（チャネル単位で DMA 転送許可ビット 制御）
DMACA_CMD_ALL_DISABLE	DMAC 起動を禁止（DMAC 動作許可ビット制御）
DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ	DMAC をソフトウェア起動し、ソフトウェア起動ビット自動クリア
DMACA_CMD_SOFT_REQ_NOT_CLR_REQ	DMAC をソフトウェア起動し、ソフトウェア起動ビット自動クリアしない
DMACA_CMD_SOFT_REQ_CLR	ソフトウェア起動ビットをクリア
DMACA_CMD_STATUS_GET	DMAC のステータス情報を取得
DMACA_CMD_ESIF_STATUS_CLR	転送エスケープ割り込みフラグ(ESIF)をクリア
DMACA_CMD_DTIF_STATUS_CLR	転送終了割り込みフラグ(DTIF)をクリア

*dmaca_stat_t *p_stat*

DMAC ステータス情報 dmaca_stat_t 構造体のポインタ

dmaca_stat_t 構造体メンバ

構造体メンバ	概略	設定値	設定内容
soft_req_stat	Software Request Status	false	A software transfer is not requested.
		true	A software transfer is requested.
esif_stat	Transfer Escape End Interrupt Status	false	A transfer escape end interrupt has not been generated.
		true	A transfer escape end interrupt has been generated.
dtif_stat	Transfer End Interrupt Status	false	A transfer end interrupt has not been generated.
		true	A transfer end interrupt has been generated.
act_stat	Active Flag of DMAC	false	DMAC operation is suspended.
		true	DMAC is operating.
transfer_count	Transfer Count	0000h - FFFFh	The number of normal transfer operations, block transfer operations or repeat transfer operations

Return Values

[DMACA_SUCCESS] */* Successful operation */*
[DMACA_ERR_INVALID_CH] */* Channel is invalid.*/*/*
[DMACA_ERR_INVALID_COMMAND] */* Command is invalid.*/*
[DMACA_ERR_NULL_PTR] */* Argument pointers are NULL.*/*
*[DMACA_ERR_SOFTWARE_REQUESTED*1]* */* DMA transfer request by software has been generated already. */*
*[DMACA_ERR_SOFTWARE_REQUEST_DISABLED*2]* */* Transfer Request Source is not Software.*/*

Note 1： DMA ソフトウェア起動ビット（以下、SWREQ bit と略す）を自動クリアする設定の状態で、既に SWREQ bit が”1”の場合に、DMACA_ERR_SOFTWARE_REQUESTED を返します。この戻り値が返る場合として、前回のソフトウェア起動要求をソフトウェア起動ビット自動クリア設定で実行したが、まだ要求が受け付けられていない場合等があります。

Note2 ： 転送要求を周辺モジュールに設定している状態で、ソフトウェア起動による DMA 転送を実行しようとした場合に、DMACA_ERR_SOFTWARE_REQUEST_DISABLED を返します。

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

<DMACA_CMD_ENABLE コマンド処理>

DMA 転送許可(DTE)ビットをセットし、指定した DMAC チャネルの転送を許可します。

<DMACA_CMD_ALL_ENABLE コマンド処理>

DMAC 動作許可(DMST)ビットをセットし、DMAC 起動を許可します。

<DMACA_CMD_RESUME コマンド処理>

DMA 転送許可(DTE)ビットをセットし、指定した DMAC チャネルの転送を再開します。

<DMACA_CMD_DISABLE コマンド処理>

DMA 転送許可(DTE)ビットをクリアし、指定した DMAC チャネルの転送を禁止します。

DMAC 転送を中止する場合や DMAC のレジスタ設定を変更する場合に使用します。

<DMACA_CMD_ALL_DISABLE コマンド処理>

DMAC 動作許可(DMST)ビットをクリアし、DMAC 起動を禁止します。

DMAC 転送を中止する場合や DMAC のレジスタ設定を変更する場合に使用します。

<DMACA_CMD_SOFT_REQ_WITH_AUTO_CLR_REQ コマンド処理>

SWREQ bit を自動クリアする設定(CLRS bit=0)にし、ソフトウェアによる DMA 転送要求が発生します。

<DMACA_CMD_SOFT_REQ_NOT_CLR_REQ コマンド処理>

SWREQ bit を自動クリアしない設定(CLRS bit=1)にし、ソフトウェアによる DMA 転送要求が発生します。

<DMACA_CMD_SOFT_REQ_CLR コマンド処理>

指定した DMAC チャネルの SWREQ bit をクリアします。

<DMACA_CMD_STATUS_GET コマンド処理>

指定した DMAC チャネルのステータス情報を引数の p_stat が示すアドレスへ書き込みます。

<DMACA_CMD_ESIF_STATUS_CLR コマンド処理>

指定した DMAC チャネルの転送エスケープ割り込みフラグ(ESIF)をクリアします。

<DMACA_CMD_DTIF_STATUS_CLR コマンド処理>

指定した DMAC チャネルの転送終了割り込みフラグ(DTIF)をクリアします。

Example

Case 1:ソフトウェアで DMAC 起動する場合

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;
dmaca_stat_t dmac_status;

/* Call R_DMACA_Control().
Enable DMAC transfer.*/
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ENABLE, &dmac_status);

/* Call R_DMACA_Control().
DMAC Software request flag set & request flag is cleared automatically.*/
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_SOFT_REQ_NOT_CLR_REQ, &dmac_status);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}

/* DMAC transfer end check */
do
{
      ret = R_DMACA_Control(DMACA_CN0, DMACA_CMD_STATUS_GET, &dmac_status);
      if (DMACA_SUCCESS != ret)
      {
          /* do something */
      }
}while( false == (dmac_status.dtif_stat));
```

Case 2:周辺モジュールを DMAC 起動要因とする場合（CMI1 割り込みを使用した例）

```
#include "r_dmaca_rx_if.h"
```

```
dmaca_return_t ret;
dmaca_stat_t dmac_status;

/* Disable CMI1 interrupt request before calling R_DTC_Control().*/
IR(CMT1,CMI1) = 0;
IEN(CMT1,CMI1) = 0;

/* Call R_DMACA_Control().
Enable DMAC transfer.*/
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ENABLE, &dmac_status);

/* Enable CMI1 interrupt request before calling R_DTC_Create().*/
IEN(CMT1,CMI1) = 1;

/* DMAC transfer end check */
do
{
      ret = R_DMACA_Control(DMACA_CN0,  DMACA_CMD_STATUS_GET,  &dmac_status);
      if (DMACA_SUCCESS != ret)
      {
          /* do something */
      }
}while( false == (dmac_status.dtif_stat));
```

Case 3:上記の Case1 や Case2 の処理に続いて DMAC 転送を継続または再開する場合

```
/* 必要であれば各レジスタ設定値を変更(R_DMACA_Create()関数参照) */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_RESUME, &dmac_status);
```

Case 4:上記の Case1 や Case2 の処理後に DMAC 転送を終了する場合

```
/* 転送終了割り込みフラグクリア */
ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_DTIF_STATUS_CLR, &dmac_status);
/* なお、転送エスケープ終了割り込みを有効にしていた場合は DMACA_CMD_ESIF_STATUS_CLR コマンドで転送エスケープ終了割り込みフラグもクリアする。 */
/* ret = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ESIF_STATUS_CLR, &dmac_status); */
```

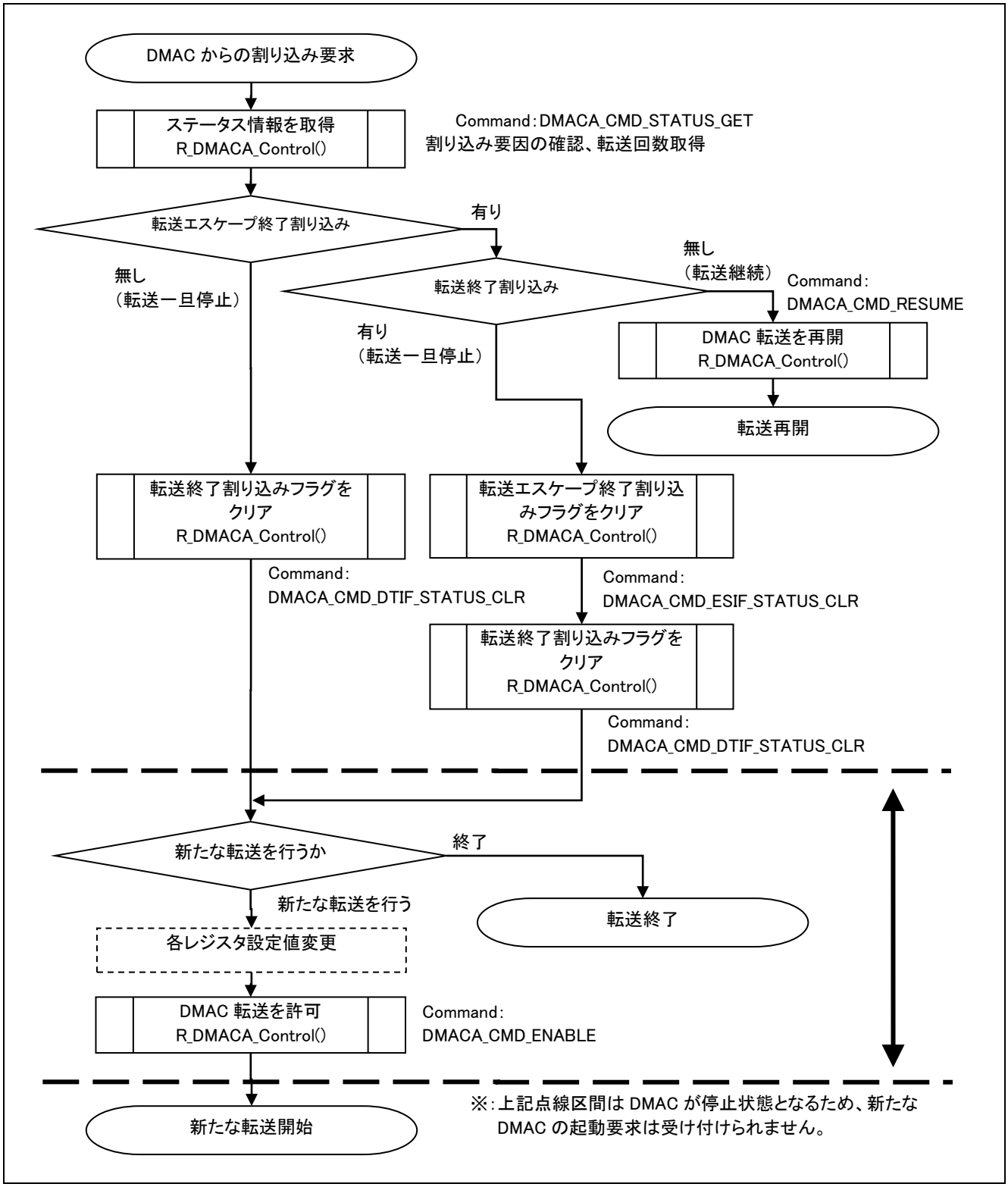

図 3.1 DMAC 転送終了または継続時の処理例

Special Notes:

DMAC チャネル 4～7 を使用し、かつ、割り込みで転送終了待ちを行う場合、転送終了割り込み／転送エスケープ終了割り込み用コールバック関数を用いて、転送エスケープ割り込みフラグ(ESIF)もしくは転送終了割り込みフラグ(DTIF)をクリアしてください。

R_DMACA_Int_Callback()

DMAC 転送終了割り込み／転送エスケープ終了割り込み用コールバック関数を登録する関数です。

Format

```
dmaca_return_t          R_DMACA_Int_Callback (
    uint8_t         channel,
    void            * p_callback
)
```

Parameters

uint8_t channel
DMAC チャネル番号

*void *p_callback*
DMAC 転送終了割り込み／転送エスケープ終了割り込み発生時にコールされる関数へのポインタ

Return Values

[DMACA_SUCCESS]	*/* Successful operation */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*
[DMACA_ERR_INVALID_HANDLER_ADDR]	*/* Invalid function address is set.*/*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

指定したチャネルの DMAC 転送終了割り込み／転送エスケープ終了割り込み用にコールバック関数を登録します。FIT_NO_FUNC や NULL がコールバックの引数として渡された場合、登録済のコールバック関数は登録が解除されます。

また、DMACA_ERR_INVALID_HANDLER_ADDR が返った場合、登録済のコールバック関数は登録が解除されます。

Note：コールバック関数の引数、戻り値のどちらも void 型にしてください。

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/* DMACA driver を使用する場合は、最初に 1 度だけ R_DMACA_Init()関数を実行してください */
R_DMACA_Init();

/* DMAC0I 割り込みのコールバック関数(例：関数名を dmac0i_callback とした場合)を登録する
ret = R_DMACA_Int_Callback(DMACA_CH0,(void *)dmac0i_callback);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

なし

R_DMACA_Int_Enable()

DMAC 転送終了割り込み／転送エスケープ終了割り込みを許可する関数です。

Format

```
dmaca_return_t          R_DMACA_Int_Enable (
    uint8_t         channel,
    uint8_t         priority
)
```

Parameters

uint8_t channel
DMAC チャネル番号

uint8_t priority
DMAC 転送終了割り込み／転送エスケープ終了割り込みの割り込み優先レベル

Return Values

[DMACA_SUCCESS] */* Successful operation */*
[DMACA_ERR_INVALID_CH] */* Channel is invalid. */*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

指定したチャネルの DMAC 転送終了割り込み／転送エスケープ終了割り込みを許可します。

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/*チャネル 0 の DMAC 転送終了割り込み／転送エスケープ終了割り込み(DMAC0I)を割り込み優先レベル
10 として許可 */
ret = R_DMACA_Int_Enable(DMACA_CH0,10);
if (DMAC_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

なし

R_DMACA_Init_Disable()

DMAC 転送終了割り込み／転送エスケープ終了割り込みを禁止する関数です。

Format

```
dmaca_return_t          R_DMACA_Int_Disable (
    uint8_t         channel,
)
```

Parameters

uint8_t channel

DMAC チャネル番号

Return Values

[DMACA_SUCCESS]	*/* Successful operation */*
[DMACA_ERR_INVALID_CH]	*/* Channel is invalid.*/*

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

指定したチャネルの DMAC 転送終了割り込み／転送エスケープ終了割り込みを禁止します。

Example

```
#include "r_dmaca_rx_if.h"

dmaca_return_t ret;

/*チャネル 0 の DMAC 転送終了割り込み／転送エスケープ終了割り込み(DMAC0I)を禁止 */
ret = R_DMACA_Int_Disable(DMACA_CH0);
if (DMACA_SUCCESS != ret)
{
    /* do something */
}
```

Special Notes:

なし

R_DMACA_GetVersion()

ドライバのバージョン情報を取得する際に使用する関数です。

Format

uint32_t R_DMACA_GetVersion (void)

Parameters

なし

Return Values

バージョン番号

上位 2 バイト：メジャーバージョン、下位 2 バイト：マイナーバージョン

Properties

r_dmaca_rx_if.h にプロトタイプ宣言されています。

Description

バージョン情報を返します。

Example

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

Special Notes:

なし

4. 端子設定

DMACA FIT モジュールはピン設定を使用しません。

5. デモプロジェクト

デモプロジェクトには、FIT モジュールとそのモジュールが依存するモジュール（例：r_bsp）を使用する main()関数が含まれます。本 FIT モジュールには以下のデモプロジェクトが含まれます。

5.1 dma_demo_rskrx231

プログラム dma_demo_rskrx231 は、リピート転送モードに設定した DMAC で AD 変換結果を転送します。プログラムを実行すると、DMAC が AD 変換結果を 32 バイトのバッファに順次保存します。

5.2 dma_demo_rskrx65n_2m

プログラム dma_demo_rskrx65n_2m は、リピート転送モードに設定した DMAC で AD 変換結果を転送します。プログラムを実行すると、DMAC が AD 変換結果を 32 バイトのバッファに順次保存します。

5.3 ワークスペースにデモを追加する

デモプロジェクトは、本アプリケーションノートで提供されるファイルの FITDemos サブディレクトリにあります。ワークスペースにデモプロジェクトを追加するには、「ファイル」 >> 「インポート」を選択し、「インポート」ダイアログから「一般」の「既存プロジェクトをワークスペースへ」を選択して「次へ」ボタンをクリックします。「インポート」ダイアログで「アーカイブ・ファイルの選択」ラジオボタンを選択し、「参照」ボタンをクリックして FITDemos サブディレクトリを開き、使用するデモの zip ファイルを選択して「終了」をクリックします。

5.4 デモのダウンロード方法

デモプロジェクトは、RX Driver Package には同梱されていません。デモプロジェクトを使用する場合は、個別に各 FIT モジュールをダウンロードする必要があります。「スマートブラウザ」の「アプリケーションノート」タブから、本アプリケーションノートを右クリックして「サンプル・コード（ダウンロード）」を選択することにより、ダウンロードできます。

6. 付録

6.1　動作確認環境

本 FIT モジュールの動作確認環境を以下に示します。

表 6.1 動作確認環境 (Rev.2.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V.7.5.0 IAR Embedded Workbench for Renesas RX 4.12.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.8.4.201902 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加 -WI,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 4.12.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのリビジョン	Rev.2.20
使用ボード	Renesas Starter Kit+ for RX72M（型名：RTK5572Mxxxxxxxxxx）

表 6.2 動作確認環境 (Rev.2.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V.7.5.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのリビジョン	Rev.2.10
使用ボード	Renesas Solution Starter Kit for RX23W（型名：RTK5523Wxxxxxxxxx）

表 6.3 動作確認環境 (Rev.2.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e2 studio V.7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.01.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 4.8.4.201803 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std=gnu99 リンクオプション：「Optimize size (サイズ最適化) (-Os)」を使用する場合、統合開発環境のデフォルト設定に以下のオプションを追加 -Wl,--no-gc-sections これは、FIT 周辺機器モジュール内で宣言されている割り込み関数をリンカが誤って破棄（discard）することを回避（work around）するための対策です。
	IAR C/C++ Compiler for Renesas RX version 4.10.1 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのリビジョン	Rev.2.00
使用ボード	Renesas Starter Kit+ for RX65N-2MB（型名：RTK50565Nxxxxxxxxx）

表 6.4 動作確認環境 (Rev.1.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio V.7.3.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.01.00
	コンパイルオプション：統合開発環境のデフォルト設定を使用し、以下のオプションを追加。 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのリビジョン	Rev.1.20
使用ボード	Renesas Starter Kit for RX72T（型名：RTK5572Txxxxxxxxxx）

表 6.5 動作確認環境 (Rev.1.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e² studio V.7.0.0
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler Package for RX Family V3.00.00
	コンパイルオプション：統合開発環境のデフォルト設定を使用し、以下のオプションを追加。 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのリビジョン	Rev.1.10
使用ボード	Renesas Starter Kit for RX231（型名：R0K505231SxxxBE） Renesas Starter Kit for RX64M（型名：R0K50564MSxxxBE） Renesas Starter Kit for RX65N（型名：RTK500565NSxxxxxBE） Renesas Starter Kit for RX65N-2MB（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit for RX66T（型名：RTK50566T0SxxxxxBE） Renesas Starter Kit for RX71M（型名：R0K50571MSxxxBE）

表 6.6 動作確認環境 (Rev.1.05)

項目	内容
統合開発環境	ルネサス エレクトロニクス製 e² studio V6.0.0
Cコンパイラ	ルネサス エレクトロニクス製 C/C++ compiler for RX family V.2.07.00)
	コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
エンディアン	ビッグエンディアン／リトルエンディアン
モジュールのバージョン	Ver.1.05
使用ボード	Renesas Starter Kit for RX231（型名：R0K505231SxxxBE） Renesas Starter Kit for RX64M（型名：R0K50564MSxxxBE） Renesas Starter Kit for RX65N（型名：RTK500565NSxxxxxBE） Renesas Starter Kit for RX65N-2M（型名：RTK50565N2SxxxxxBE） Renesas Starter Kit for RX71M（型名：R0K50571MSxxxBE）

6.2 トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e² studio を使用している場合
  アプリケーションノート RX ファミリ e² studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「This MCU is not supported by the current r_dmaca_rx module.」エラーが発生します。

A：追加した FIT モジュールがユーザプロジェクトのターゲットデバイスに対応していない可能性があります。追加した FIT モジュールの対象デバイスを確認してください。

7. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

(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。)

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。)

テクニカルアップデートの対応について

該当のテクニカルアップデートはありません。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2014.07.31	—	初版発行
1.01	2014.08.29	6	1.3 関連アプリケーションノート　を追加
1.02	2015.01.15	1	対象デバイス　に、RX71M を追加。
		1	FIT 関連ドキュメント　に、「CS+に組み込む方法(R01AN1826JJ)」を追加。
		4	1.2.1 API の概要 表 1-1 API 関数　R_DMACA_Init()を表の先頭に移動。
		4	1.2.1 API の概要 表 1-1 API 関数　R_DMACA_Int_Callback()、R_DMACA_Int_Enable()、R_DMACA_Int_Disable()　の説明欄 「転送終了割り込み／転送エスケープ終了割り込み」　元は、「転送終了割り込み」であった。
		5	1.2.2 動作環境とメモリサイズ　(1)RX64M の場合　表 1-3 動作確認条件 使用ボード　型名　元は、Renesas Starter Kit for RX64M であった。
		6	1.2.2 動作環境とメモリサイズ　(2)RX71M の場合　を追加。
		7	1.3 関連アプリケーションノート　に、SCIFA クロック同期式シングルマスタ制御モジュール Firmware Integration Technology（R01AN2280JJ）を追加。
		12	3. API 関数　R_DMACA_Init()を 3.1 に移動。元は 3.5 であった。
		12	3.1 R_DMACA_Init() Description　にて、 「各 DMAC チャネル」　元は、「各 DMA チャネル」であった。 「転送終了割り込み／転送エスケープ終了割り込み」　元は、「転送終了割り込み」であった。
		12	3.1 R_DMACA_Init() Special Notes　を変更した。元は「なし」であった。
		13	3.2 R_DMACA_Open()　「R_DMACA_Init()関数コール後に使用する関数です。」　元は、「最初に使用する関数です。」であった。
		22	3.5 R_DMACA_Control() Parameter Command 表にて、 DMACA_CMD_ESIF_STATUS_CLR の内容欄 「転送エスケープ割り込みフラグ(ESIF)をクリア」　元は、「転送エスケープ割り込みフラグをクリア」であった。 DMACA_CMD_DTIF_STATUS_CLR の内容欄 「転送終了割り込みフラグ(DTIF)をクリア」　元は、「転送終了割り込みフラグをクリア」であった。
		23	3.5 R_DMACA_Control() Description にて、 <DMACA_CMD_ESIF_STATUS_CLR コマンド処理> 「転送エスケープ割り込みフラグ(ESIF)をクリア」　元は、「転送エスケープ割り込みフラグをクリア」であった。 <DMACA_CMD_DTIF_STATUS_CLR コマンド処理> 「転送終了割り込みフラグ(DTIF)をクリア」　元は、「転送終了割り込みフラグをクリア」であった。
		25	3.5 R_DMACA_Control() Example Case4 にて 「転送エスケープ終了割り込み」　元は、「転送エスケープ割り込み」であった。
		26	3.5 R_DMACA_Control() Example 図 3-1 にて 「転送エスケープ終了割り込み」　元は、「転送エスケープ割り込み」であった。

		26	3.5 R_DMACA_Control() Example Special Notes にて 元は、「なし」であった。
		27	3.6 R_DMACA_Int_Callback() にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
		27	3.6 R_DMACA_Int_Callback() Parematers、Description にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
		29	3.7 R_DMACA_Int_Enable() にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
		29	3.7 R_DMACA_Int_Enable() Parematers、Description、Example にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
		30	3.8 R_DMACA_Int_Disable() にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
		30	3.8 R_DMACA_Int_Disable() Description、Example にて 「転送終了割り込み／転送エスケープ終了割り込み」 元は、「転送終了割り込み」であった。
1.03	2015.06.15	1	対象デバイス に、RX230、RX231 を追加。
		7	1.2.2 動作環境とメモリサイズ (3)RX231 の場合 を追加。
		8	1.3 関連アプリケーションノート DTC モジュールのタイトルを更新
1.04	2016.09.30	1	対象デバイス に、RX65N を追加。
		8	1.2.2 動作環境とメモリサイズ (4)RX65N の場合 を追加。
		11	2.7 引数 uint8_t rsv[2];を追加
		13	2.9 モジュールの追加方法 を更新。
		24	3.5 R_DMACA_Control() 表 dmaca_stat_t 構造体メンバ transfer_count を追加。
		28	図 3-1. ステータス情報を取得 「転送回数取得」を追加。
1.05	2017.07.31	-	下記の章のタイトルを変更した。 ・1.1 DMACA FIT モジュールとは：元は 1.1 DMACA FIT モジュールであった。 下記の章の本文を移動した。 ・1.2 DMACA FIT モジュールの概要：元は 1. DMACA FIT モジュールの概要であった。 下記の章番号を変更した。 ・5.1 動作確認環境：元は 1.2.2 動作環境とメモリサイズであった。 ・5. 付録：元は 4. 付録であった。 ・6. 参考ドキュメント：元は 5.参考ドキュメントであった。 下記の章を追加した。 ・2.4 使用する割り込みベクタ ・2.8 コードサイズ ・2.12 FIT モジュールの追加方法 ・4. 端子設定 ・5.2 トラブルシューティング
		1	対象デバイスに、RX651 を追加。
		4	2.2 ソフトウェアの要求 「r_cgc_rx」を削除。

1.10	2018.09.28	1 6 8 33	RX66T のサポートを追加。 RX66T に対応する割り込みベクタ番号を追加。 RX66T に対応するコードサイズを追加。 「5.1 動作確認環境」：Rev.1.10 に対応する表を追加。
1.11	2018.11.16	33	Renesas Starter Kit for RX66T 製品番号変更
1.20	2019.02.01	— 6 8 14-32 38 39 39	RX72T グループのサポートを追加。 RX72T グループに対応する割り込みベクタ番号を追加。 RX72T グループに対応するコードサイズを追加。 各 API 関数で「Reentrant」の説明を削除。 「5. デモプロジェクト」を追加。 Renesas Starter Kit+ for RX66T の型名を変更。 「6.1 動作確認環境」Rev.1.20 に対応する表を追加。
2.00	2019.05.20	— 1 5 8 34 37 プログラム	以下のコンパイラをサポート。 - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX 「ターゲットコンパイラ」のセクションを追加。 関連ドキュメントを削除。 「2.2 ソフトウェアの要求」r_bsp v5.20 以上が必要 「2.8 コードサイズ」セクションを更新。 表 5.1「動作確認環境」： Rev.2.00 に対応する表を追加。 「Web サイトおよびサポート」のセクションを削除。 GCC と IAR コンパイラに関して、以下を変更。 「evenaccess」を、BSP のマクロ定義で置き換えた。 割り込み関数の宣言を、BSP のマクロ定義で置き換えた。
2.10	2019.06.28	1, 6 8 34 35	RX23W のサポートを追加。 RX23W に対応するコードサイズを追加。 「5. デモプロジェクト」を追加。 「6.1 動作確認環境」： Rev.2.10 に対応する表を追加。
		プログラム	RX23W のサポートを追加。 デモプロジェクトを追加。
2.20	2019.08.15	1 8 35 プログラム	RX72M のサポートを追加。 RX72M に対応するコードサイズを追加。 「6.1 動作確認環境」： Rev.2.20 に対応する表を追加。 表 6.2：RX23W ボード名変更。 RX72M のサポートを追加。

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。