RX ファミリ

DTC モジュール Firmware Integration Technology

R01AN1819JJ0220
Rev.2.20
2019.02.01

要旨

本アプリケーションノートは、Firmware Integration Technology (FIT)を使用した DTC モジュールについて説明します。本モジュールは DTC ソフトウェアモジュールを使用して、データ転送処理の制御を行います。以降、本モジュールを DTC FIT モジュールと称します。

対象デバイス

- RX110 グループ、RX111 グループ、RX113 グループ、RX130 グループ
- RX230 グループ、RX231 グループ、RX23T グループ、RX24T グループ、RX24U 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

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あわせて変更し、十分評価してください。

関連ドキュメント

Firmware Integration Technology ユーザーズマニュアル (R01AN1833)
ボードサポートパッケージモジュール Firmware Integration Technology (R01AN1685)
e²studio に組み込む方法 Firmware Integration Technology (R01AN1723)
CS+に組み込む方法 Firmware Integration Technology (R01AN1826)
Renesas e² studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)
RX ファミリ DMA コントローラ DMACA 制御モジュール Firmware Integration Technology（R01AN2063）
RX ファミリ DTC モジュールを使用するシーケンス転送サンプルプログラム Firmware Integration Technology（R01AN3434）

目次

1. **概要** 3
1.1 DTC FIT モジュールとは 3
1.2 DTC FIT モジュールの概要 3
1.3 API の概要 5
1.4 DTC IP バージョン 5

2. API 情報 6
2.1 ハードウェアの要求 6
2.2 ソフトウェアの要求 6
2.3 サポートされているツールチェーン 6
2.4 使用する割り込みベクタ 6
2.5 ヘッダファイル 6
2.6 整数型 6
2.7 コンパイル時の設定 7
2.8 コードサイズ 8
2.9 引数 11
2.10 戻り値 13
2.11 コールバック関数 13
2.12 FIT モジュールの追加方法 14
2.13 for 文、while 文、do while 文について 15

3. API 関数 16
R_DTC_Open() 16
R_DTC_Close() 17
R_DTC_Create() 19
R_DTC_CreateSeq() 26
R_DTC_Control() 34
R_DTC_GetVersion() 39

4. 端子設定 40

5. 付録 41
5.1 動作確認環境 41
5.2 トラブルシューティング 42

6. 参考ドキュメント 43

1. 概要

1.1 DTC 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 DTC FIT モジュールの概要

DTC FIT モジュールは、以下の 3 つの転送モードをサポートしています。

- 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

各モードでチェーン転送機能、および、シーケンス転送の許可／禁止を設定できます。詳細はユーザーズマニュアル ハードウェア編の「データトランスファコントローラ」章をご覧ください。

DTC は割り込み要因の割り込み要求によって起動されます。ユーザは各起動要因に対する 1 個の転送情報、またはチェーン転送機能を使用する場合、連続する複数の転送情報を作成する必要があります。

転送情報には転送元と転送先の先頭アドレスと、DTC がデータを転送元から転送先にどのように転送するかを指定する設定情報が含まれます。DTC が起動すると、該当の割り込みに対応する転送情報を読み込み、その情報に従ってデータ転送を開始します。

DTC は指定された割り込み要因に対応する転送情報の先頭アドレスを DTC ベクタテーブルから読み込みます。ベクタテーブルは 4 バイトアドレスの配列で、各割り込み要因に対応する転送情報（n）の先頭アドレスが、ベクタ番号（n）に従ってテーブルのアドレス（4×n）の位置に配列要素として格納されています。

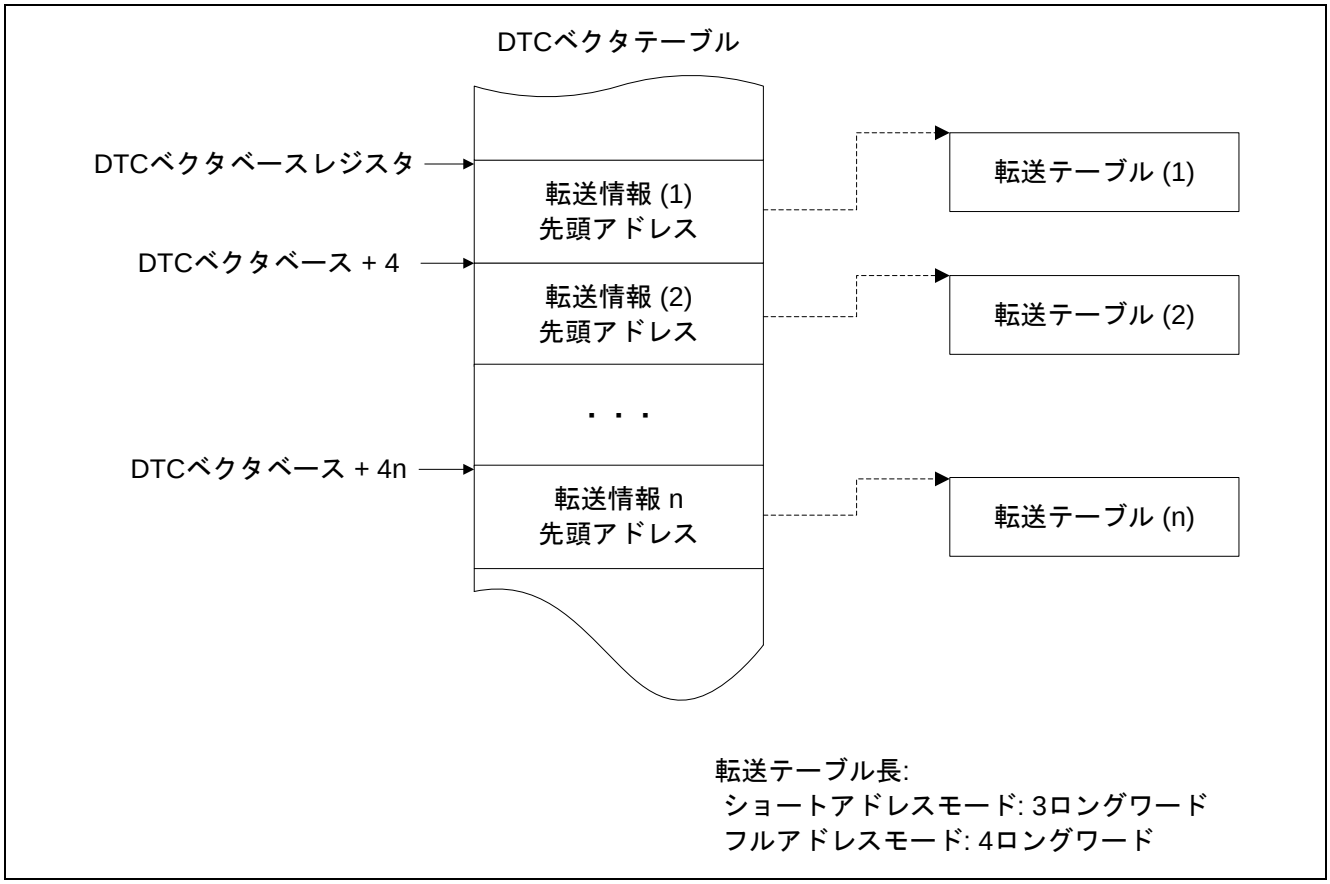

図 1.1　DTC ベクタと転送情報

DTC を使用する前に、RAM 領域に DTC ベクタテーブル用にメモリ領域を割り当てる必要があります。メモリはバイト単位で割り当てられ、メモリサイズは DTC で対応可能な割り込み要因の最大ベクタ番号に依存します。メモリサイズは、targets フォルダの各 MCU フォルダにある r_dtc_rx_target.h ファイルで定義される DTC_VECTOR_TABLE_SIZE_BYTES の値によって指定されます。メモリサイズのデフォルト値は割り込みベクタテーブルで定義できるすべての起動要因に対応可能な値となります。例えば、RX111 の場合、デフォルト値は 0x3E4 (0x3E4 = 249×4)で、RX64M の場合は 0x400 (0x400 = 256×4)となります。DTC ベクタテーブルの先頭アドレスは 1K バイト単位であることが必要です。また、ベクタテーブルはコンパイル時にリンカを使って配置することもできます。

DTC モジュールはショートアドレスモードとフルアドレスモードの 2 つのアドレスモードで動作することができます。ショートアドレスモードでは、1 つの転送情報のサイズは 3 ロングワード（12 バイト）で、DTC は 0x00000000～0x007FFFFF と 0xFF800000～0xFFFFFFFF の 16M バイトのメモリ空間にアクセスできます。フルアドレスモードでは、転送情報のサイズは 4 ロングワード（16 バイト）で、DTC は 0x00000000～0xFFFFFFFF の 4G バイトのメモリ空間にアクセスできます。

デフォルトでは、DTC は起動割り込みが発生するたびに転送情報をリードします。1 つの起動要因から 2 回もしくは連続して何回もの起動が発生する場合、前の起動動作で転送情報が既に DTC 内に存在するため、2 回目以降のリードをスキップして DTC の転送効率を向上させることができます。転送情報リードスキップを許可するには、初期化時に R_DTC_Open()で設定するか、または R_DTC_Control()で DTC_CMD_ENABLE_READ_SKIP コマンドを使用します。

DTC モジュールを初期化するには、R_DTC_Open()を呼び出します。この関数は、DTC にクロックの供給を開始し、DTC ベクタテーブルの先頭アドレスを DTC ベクタベースレジスタ（DTCVBR）に書き込みます。シーケンス転送を使用する場合は、DTC インデックステーブの先頭アドレスを DTC インデックステーブルベースレジスタ（DTCIBR）に書き込みます。また、r_dtc_rx_config.h のユーザ設定に従って転送情報リードスキップ、DTC アドレスモード、および DTCER レジスタの設定を初期化します。

R_DTC_Create()関数にユーザが選択した設定内容を渡して、各割り込み要因に対応する転送情報を作成します。転送情報には転送元と転送先の先頭アドレス、および DTC がどのようにデータを転送元から転送先に転送するかを指示する情報が含まれます。R_DTC_Create()では、転送情報の先頭アドレスを DTC ベクタテーブルの指定されたベクタ番号の位置に格納します。

R_DTC_CreateSeq()関数はシーケンス転送を行うための転送情報を作成し、転送情報の先頭アドレスを DTC インデックステーブルの指定されたシーケンス番号の位置に格納します。

R_DTC_Control()を使って、DTC 起動因となる割り込みの選択と解除、DTC に供給するクロックの起動と停止、転送情報リードスキップ機能の許可／禁止、処理中のチェーン転送の中断、シーケンス転送の許可／禁止／中断を行います。

起動要因により割り込みが発生すると、DTC が起動されます。DTC は起動割り込みのベクタ番号に対応する転送情報を読み込んで設定を行い、データを転送します。R_DTC_Control()を使用して、DTC の動作状態や現処理の起動割り込みのベクタ番号などの DTC のステータスを取得できます。また、R_DTC_Control()関数を使用して実行中のチェーン転送処理を中断する機能やシーケンス転送処理を中断する機能もサポートしています。

DTC FIT モジュールの使用条件

以下に本モジュールの使用条件を示します。

- r_bsp でデフォルトのロック関数を使用する必要があります。
- DMAC と DTC のモジュールストップ設定ビットには共通のビットを使用する必要があります。

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 1-1　API 関数一覧

関数	関数説明
R_DTC_Open()	初期化処理
R_DTC_Close()	終了処理
R_DTC_Create()	レジスタおよび起動要因設定処理
R_DTC_CreateSeq()	シーケンス転送用のレジスタおよび起動要因設定処理
R_DTC_Control()	動作設定処理
R_DTC_GetVersion()	バージョン情報取得処理

1.4　DTC IP バージョン

表 1-2 に DTC IP バージョンと対象デバイスの関係について示します。

DTC IP バージョンの違いにより、R_DTC_Create()関数と R_DTC_CreateSeq()関数の引数仕様が異なります。詳細は「**エラー! 参照元が見つかりません。 エラー! 参照元が見つかりません。**」を参照してください。

表 1-2　DTC IP バージョン一覧

DTC IP バージョン	対象デバイス
DTCa	● RX110 グループ、RX111 グループ、RX113 グループ、RX130 グループ ● RX230 グループ、RX231 グループ、RX23T グループ、RX24T グループ、RX24U グループ ● RX64M グループ、RX66T グループ ● RX71M グループ、RX72T グループ
DTCb	● RX65N グループ

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。

- DTC（DTCa または DTCb）
- 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)

2.3 サポートされているツールチェーン

本モジュールは「5.1 動作確認環境」で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

DTC FIT モジュールは R_DTC_Create()関数、または R_DTC_CreateSeq()関数の引数 p_data_cfg->response_interrupt を設定することで表 **2.1** に示す割り込みが有効なります。

表 2.1 使用する割り込みベクター覧

関数名	引数	設定値	割り込み発生タイミング
R_DTC_Create() R_DTC_CreateSeq()	p_data_cfg->response_interrupt	DTC_INTERRUPT_AFTER_ALL_COMPLETE	指定した回数のデータ転送が終了したとき、CPU へ割り込み要求が発生
		DTC_INTERRUPT_PER_SINGLE_TRANSFER	データ転送のたびに、CPU への割り込み要求が発生

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_dtc_rx_if.h に記載しています。

“DTC_VECTOR_TABLE_SIZE_BYTES” 定義を使って RAM 領域に DTC ベクタテーブル用のメモリを割り当てるときは、r_dtc_rx_target.h ファイルも、同様にインクルードされなければなりません。

2.6 整数型

このドライバは 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_dtc_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_dtc_rx_config.h	
#define DTC_CFG_PARAM_CHECKING_ENABLE ※デフォルト値は r_bsp_config.h ファイルで定義される“BSP_CFG_PARAM_CHECKING_ENABLE”の値となります。	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：パラメータチェック処理をコードから省略します。 • 1：パラメータチェック処理をコードに含めます。 システムのデフォルト設定を再使用するために、デフォルト値を“BSP_CFG_PARAM_CHECKING_ENABLE”に設定します。
#define DTC_CFG_DISABLE_ALL_ACT_SOURCE ※デフォルト値は“DTC_ENABLE”	R_DTC_OPEN()で DTCER レジスタをクリアするかどうかを設定します。 • DTC_DISABLE：処理なし。 • DTC_ENABLE：R_DTC_OPEN()ですべての DTCER レジスタをクリアします。
#define DTC_CFG_SHORT_ADDRESS_MODE ※デフォルト値は“DTC_DISABLE”	DTC でサポートするアドレスモードを設定します。 • DTC_DISABLE：フルアドレスモードを選択します。 • DTC_ENABLE：ショートアドレスモードを選択します。
#define DTC_CFG_TRANSFER_DATA_READ_SKIP_EN ※デフォルト値は“DTC_ENABLE”	転送情報リードスキップを許可するかどうかを設定します。 • DTC_DISABLE：転送情報リードスキップを禁止します。 • DTC_ENABLE：転送情報リードスキップを許可します。
#define DTC_CFG_USE_DMAC_FIT_MODULE ※デフォルト値は“DTC_ENABLE”	DTC FIT モジュールと一緒に DMAC FIT モジュールを使用するかどうかを設定します。 • DTC_DISABLE：DMAC FIT モジュールを使用しない。 • DTC_ENABLE：DMAC FIT モジュールを使用する。 DMAC FIT モジュールを使用しないときに“DTC_ENABLE”を設定すると、コンパイルエラーが発生します。
#define DTC_CFG_USE_SEQUENCE_TRANSFER ※デフォルト値は“DTC_DISABLE”	シーケンス転送を使用するかどうか設定します。 • DTC_DISABLE：シーケンス転送を使用しない。 • DTC_ENABLE：シーケンス転送を使用する。 本定義を“DTC_ENABLE”とした場合、DTC_CFG_SHORT_ADDRESS_MODE は“DTC_DISABLE”に設定してください。本定義と DTC_CFG_SHORT_ADDRESS_MODE の定義を共に“DTC_ENABLE”にした場合、コンパイルエラーが発生します。また、シーケンス転送未対応の MCU に対して本定義を“DTC_ENABLE”にした場合、コンパイルエラーが発生します。

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ーン」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, and Stack Code Sizes				
デバイス	分類	使用メモリ		備考
RX111	ROM	998 バイト		
	RAM	9 バイト ＋2,024 バイト（注５、注６）		
	最大使用ユーザスタック	52 バイト		
	最大使用割り込みスタック	-		
RX231	ROM	1,236 バイト		
	RAM	9 バイト ＋2,024 バイト（注５、注６）		
	最大使用ユーザスタック	52 バイト		
	最大使用割り込みスタック	-		
RX65N	ROM	1,740 バイト（注６）	1,920 バイト（注７）	
	RAM	9 バイト ＋2,048 バイト （注５、注６）	9 バイト ＋3,072 バイト （注５、注７）	
	最大使用ユーザスタック	56 バイト	56 バイト	
	最大使用割り込みスタック	-	-	
RX66T	ROM	1536 バイト (注 6)		
	RAM	9 バイト +2,048 バイト (注 5、6)		
	最大ユーザスタック	52 バイト		
	最大割り込みスタック	-		
RX71M	ROM	1,664 バイト		
	RAM	9 バイト ＋2,048 バイト（注５、注６）		
	最大使用ユーザスタック	52 バイト		
	最大使用割り込みスタック	-		
RX72T	ROM	1,515 バイト		
	RAM	9 バイト ＋2,048 バイト（注５、注６）		
	最大使用ユーザスタック	60 バイト		
	最大使用割り込みスタック	-		

注１：「2.7 コンパイル時の設定」のデフォルト設定を選択した場合の値です。選択する定義により、コードサイズは異なります。
注２：動作条件は以下のとおりです。

- r_dtc_rx.c
- r_dtc_rx_target.c

注３：必要メモリサイズは、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。
注４：リトルエンディアン時の値です。エンディアンにより、上記のメモリサイズは、異なります。
注５：DTC FIT モジュールは、malloc()関数を使用し、DTC ベクタテーブル、および、DTC インデックステーブルに必要なメモリを確保します。このメモリサイズは、ターゲット MCU の r_dtc_rx_target.h にある“#define DTC_VECTOR_TABLE_SIZE_BYTES”により決まります。
注６：DTC_CFG_USE_SEQUENCE_TRANSFER が DTC_DISABLE の場合
注７：DTC_CFG_USE_SEQUENCE_TRANSFER が DTC_ENABLE の場合

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_dtc_rx_if.h に記載されています。

2.9.1 r_dtc_rx_if.h

```
/* Short-address mode */
typedef struct st_transfer_data { /* 3 long words */
    uint32_t lw1;
    uint32_t lw2;
    uint32_t lw3;
} dtc_transfer_data_t;

/* Full-address mode */
typedef struct st_transfer_data { /* 4 long words */
    uint32_t lw1;
    uint32_t lw2;
    uint32_t lw3;
    uint32_t lw4;
} dtc_transfer_data_t;

/* Transfer data configuration */
/* Moved struct dtc_transfer_data_cfg_t to r_dtc_rx_target_if.h */

typedef enum e_dtc_command {
    DTC_CMD_DTC_START,        /* DTC will accept activation requests.         */
    DTC_CMD_DTC_STOP,         /* DTC will not accept new activation request.  */
    DTC_CMD_ACT_SRC_ENABLE,
                  /* Enable an activation source specified by vector number.  */
    DTC_CMD_ACT_SRC_DISABLE,
                  /* Disable an activation source specified by vector number. */
    DTC_CMD_DATA_READ_SKIP_ENABLE,  /* Enable Transfer Data Read Skip.        */
    DTC_CMD_DATA_READ_SKIP_DISABLE, /* Disable Transfer Data Read Skip.       */
    DTC_CMD_STATUS_GET,             /* Get the current status of DTC.         */
    DTC_CMD_CHAIN_TRANSFER_ABORT
                              /* Abort the current Chain transfer process.    */
    DTC_CMD_SEQUENCE_TRANSFER_ENABLE      /* Enable sequence transfer         */
    DTC_CMD_SEQUENCE_TRANSFER_DISABLE     /* Disable Sequence transfer        */
    DTC_CMD_SEQUENCE_TRANSFER_ABORT       /* Abort sequence transfer          */
} dtc_command_t;
```

2.9.2 r_dtc_rx_target_if.h

dtc_transfer_data_cfg_t は DTC の IP Version により定義が異なります。

1. DTCa の場合

```
typedef struct st_dtc_transfer_data_cfg {
      dtc_transfer_mode_t     transfer_mode;    /* DTC transfer mode          */
      dtc_data_size_t         data_size;        /* Size of data               */
      dtc_src_addr_mode_t     src_addr_mode;    /* Address mode of source     */
      dtc_chain_transfer_t    chain_transfer_enable;
                                        /* Chain transfer is enabled or not  */
      dtc_chain_transfer_mode_t chain_transfer_mode;
                                       /* How chain transfer is performed    */
      dtc_interrupt_t         response_interrupt;
                                       /* How response interrupt is raised   */
      dtc_repeat_block_side_t repeat_block_side;/* Side being repeat or block */
      dtc_dest_addr_mode_t   dest_addr_mode;    /* Address mode of destination*/
      uint32_t                source_addr;      /* Start address of source    */
      uint32_t                dest_addr;  /* Start address of destination     */
      uint32_t                transfer_count;   /* Transfer count             */
      uint16_t                block_size;
                                 /* Size of a block in block transfer mode */
      uint16_t                rsv;              /* Reserve bit                */
} dtc_transfer_data_cfg_t;
```

2. DTCb の場合

```
typedef struct st_dtc_transfer_data_cfg {
      dtc_transfer_mode_t     transfer_mode;    /* DTC transfer mode          */
      dtc_data_size_t         data_size;        /* Size of data               */
      dtc_src_addr_mode_t     src_addr_mode;    /* Address mode of source     */
      dtc_chain_transfer_t    chain_transfer_enable;
                                        /* Chain transfer is enabled or not  */
      dtc_chain_transfer_mode_t chain_transfer_mode;
                                        /* How chain transfer is performed   */
      dtc_interrupt_t         response_interrupt;
                                        /* How response interrupt is raised  */
      dtc_repeat_block_side_t repeat_block_side;/* Side being repeat or block */
      dtc_dest_addr_mode_t    dest_addr_mode;   /* Address mode of destination*/
      uint32_t                source_addr;      /* Start address of source    */
      uint32_t                dest_addr;  /* Start address of destination     */
      uint32_t                transfer_count;   /* Transfer count       */
      uint16_t                block_size;
                                 /* Size of a block in block transfer mode */
      uint16_t                rsv;              /* Reserve bit                */
      dtc_write_back_t        writeback_disable;
                          /* Transfer information writeback is enabled or not  */
      dtc_sequence_end_t      sequence_end;
                            /* Sequence transfer is continued or end      */
      dtc_refer_index_table_t refer_index_table_enable;
                            /* Index table reference is enabled or not    */
      dtc_disp_add_t          disp_add_enable;
                  /* Displacement value is added to the source address or not */
} dtc_transfer_data_cfg_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_dtc_rx_if.h で記載されています。

```
typedef enum e_dtc_err       /* DTC API error codes */
{
    DTC_SUCCESS_DMAC_BUSY = 0,
            /* One or some DMAC resources are locked by another process.      */
    DTC_SUCCESS,
    DTC_ERR_OPENED,                 /* DTC was initialized already.           */
    DTC_ERR_NOT_OPEN,               /* DTC module is not initialized yet.     */
    DTC_ERR_INVALID_ARG,            /* Arguments are invalid.                 */
    DTC_ERR_INVALID_COMMAND,        /* Command parameters are invalid.        */
    DTC_ERR_NULL_PTR,               /* Argument pointers are NULL.            */
    DTC_ERR_BUSY        /* The DTC resources are locked by another process.   */
    DTC_ERR_ACT                     /* Data transfer is in progress           */
} dtc_err_t;
```

2.11 コールバック関数

DTC FIT モジュールではコールバック関数を使用しません。

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_DTC_Open()

DTC FIT モジュールの API 使用時に、最初に実行される関数です。

Format

dtc_err_t R_DTC_Open (void)

Parameters

なし

Return Values

[DTC_SUCCESS] */* 正常終了 */*
[DTC_ERR_OPENED] */* DTC は既に初期化されています。 */*
[DTC_ERR_BUSY] */* リソースは他のプロセスによってロックされています。 */*

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

この関数は、DTC をロックし（注 1）、DTC への電源供給を開始し、DTC ベクタテーブル、アドレスモード、転送情報リードスキップ機能の設定を初期化します。また、r_dtc_rx_config.h 内で DTC_CFG_DISABLE_ALL_ACT_SOURCE を DTC_ENABLE に設定した場合、すべての DTCER レジスタをクリアします。DTC_CFG_USE_SEQUENCE_TRANSFER を DTC_ENABLE に設定した場合、DTC インデックステーブルで使用する領域を確保します。

注1. DTC 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ています。そのため、処理が正常に終了すると、DTC はロックされた状態になります。

Example

```
dtc_err_t ret;
/* Call R_DTC_Open() */
ret = R_DTC_Open();
```

Special Notes:

r_bsp_config.h の#define BSP_CFG_HEAP_BYTES には、r_dtc_rx_target.h の#define DTC_VECTOR_TABLE_SIZE_BYTES より大きい値を設定してください。
これは、DTC FIT モジュールで malloc()関数を使用して、DTC ベクタテーブルの領域を確実に確保するためです。

R_DTC_Close()

この関数は、DTC のリソースを開放します。

Format

dtc_err_t R_DTC_Close (void)

Parameters

なし

Return Values

[DTC_SUCCESS]	*/* 正常終了 */*
[DTC_SUCCESS_DMAC_BUSY]	*/* 正常終了。 1 つ以上の DMAC のリソースがロックされています。 */*

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

この関数は、DTC のロックを解除し（注 1）、DTC 起動許可レジスタ（DTCERn）をクリアして、すべての DTC 起動要因を禁止にします。DTC へのクロック供給を停止し、DTC はモジュールストップ状態へ遷移します。

さらに、DMAC のすべてのチャネルのロックが解除されていた場合、本関数は DMAC と DTC をモジュールストップ状態に遷移させます（注 2）。

注1. DTC 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ています。そのため、処理が正常に終了すると、DTC はロック解除された状態になります。

注2. DMAC および DTC のモジュールストップ設定ビットとして共用ビットが使用されるため、本関数では、モジュールストップ状態を設定する前に、すべての DMAC チャネルのロックが解除されていることを確認します。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」章をご覧ください。

以下を参照し、使用するモジュールの組み合わせに応じて処理方法を変更してください。

DMAC コントロール	DTC コントロール	処理方法
DMACA FIT モジュール （ロック機能コントロール関数および DTC ロック状態確認関数がある）	DTC FIT モジュール （ロック機能コントロール関数および DMAC ロック状態確認関数がある）	処理 1
上記以外		処理 2

処理 1: r_bsp のデフォルトのロック関数を使用し、DMAC FIT モジュール（注 1）で DMAC を制御する

関数は、r_bsp のデフォルトのロック関数を使用して、DMAC の全チャネルおよび DTC のロックが解除されていることを確認し、DTC をモジュールストップ状態に遷移させます。

注1. DMAC FIT モジュールがモジュールストップコントロール関数（DTC がロック状態であることを確認する関数）を備えていることが、この処理の必要条件となります。

処理 2:上記以外の方法による制御

ユーザは、DMAC の全チャネルのロックが解除されていること、および DTC のロックが解除されている（使用中でない）ことを確認するためのコードを提供する必要があります。DTC FIT モジュールには、この処理用に空関数が用意されています。

r_bsp のデフォルトのロック機能を使用しない場合、r_dtc_rx_target.c ファイルの r_dtc_check_DMAC_locking_byUSER()関数で“/* do something */”とコメントが入っている行の後に、DMAC の全チャネルおよび DTC のロック／ロック解除を確認するためのプログラムコードを挿入してください。

なお、r_dtc_check_DMAC_locking_byUSER()関数の戻り値には、以下に示すブール型を使用してください。

r_dtc_check_DMAC_locking_byUSER()の戻り値

[true] */* DMAC の全チャネルのロックが解除されています。*/*
[false] */* 1 つ以上の DMAC のチャネルがロックされています。*/*

Example

```
dtc_err_t ret;
ret = R_DTC_Close();
```

Special Notes:

DMAC FIT モジュールを使用せずに DMAC を制御する場合は、本関数の呼び出しによって DMAC がモジュールストップ状態に遷移されないように、DMAC の使用状態を監視し、DMAC のロック／ロック解除を制御してください。DMAC 転送設定を行わない時は、DMAC が動作中でなくても、DMAC はロックされている必要があります。

R_DTC_Create()

この関数は、DTC レジスタの設定と起動要因の設定を行います。

Format

```
dtc_err_t       R_DTC_Create (
        dtc_activation_source_t     act_source,
        dtc_transfer_data_t         *p_transfer_data,
        dtc_transfer_data_cfg_t     *p_data_cfg,
        uint32_t                    chain_transfer_nr
)
```

Parameters

dtc_activation_source_t act_source
起動要因

*dtc_transfer_data_t *p_transfer_data*
RAM の転送情報領域の開始アドレスへのポインタ

*dtc_transfer_data_cfg_t *p_data_cfg*
転送情報設定へのポインタ

DTCb の場合、以下の構造体メンバへの設定は無効であり、本関数内で以下の値を設定します。

```
p_data_cfg->writeback_disable = DTC_WRITEBACK_ENABLE;
p_data_cfg->sequence_end = DTC_SEQUENCE_TRANSFER_CONTINUE;
p_data_cfg->refer_index_table_enable = DTC_REFER_INDEX_TABLE_DISABLE;
p_data_cfg->disp_add_enable = DTC_SRC_ADDR_DISP_ADD_DISABLE;
```

uint32_t chain_transfer_nr
チェーン転送数

転送情報数とそれに対応する設定情報は“チェーン転送数 + 1”になります。例えば、chain_transfer_nr = 1 のとき、連続する転送情報が 2 つ、それに対応する設定情報が 2 つあることになり、最初の設定情報でチェーン転送が有効になります。

転送情報（*p_transfer_data*）の型定義はアドレスモードに依存します（詳細は以下参照）。ユーザはこのデータ型を使って転送情報を正しくメモリに配置します。

```
#if (1 == DTC_CFG_SHORT_ADDRESS_MODE) /* Short address mode */
typedef struct st_transfer_data { /* 3 long words */
    uint32_t lw1;
    uint32_t lw2;
    uint32_t lw3;
} dtc_transfer_data_t;
#else /* Full-address mode */
typedef struct st_transfer_data { /* 4 long words */
    uint32_t lw1;
    uint32_t lw2;
    uint32_t lw3;
    uint32_t lw4;
} dtc_transfer_data_t;
#endif
```

「転送情報設定へのポインタ（* p_data_cf）」の型は、DTC IP バージョンにより異なります。設定情報のデータ構造体を以下に示します。

1.DTCa の場合

```
typedef struct st_dtc_transfer_data_cfg {
      dtc_transfer_mode_t     transfer_mode;    /* DTC transfer mode          */
      dtc_data_size_t         data_size;        /* Size of data               */
      dtc_src_addr_mode_t     src_addr_mode;    /* Address mode of source     */
      dtc_chain_transfer_t    chain_transfer_enable;
                                        /* Chain transfer is enabled or not  */
      dtc_chain_transfer_mode_t chain_transfer_mode;
                                       /* How chain transfer is performed    */
      dtc_interrupt_t         response_interrupt;
                                       /* How response interrupt is raised   */
      dtc_repeat_block_side_t repeat_block_side;/* Side being repeat or block */
      dtc_dest_addr_mode_t   dest_addr_mode;    /* Address mode of destination*/
      uint32_t                source_addr;      /* Start address of source    */
      uint32_t                dest_addr;  /* Start address of destination     */
      uint32_t                transfer_count;   /* Transfer count             */
      uint16_t                block_size;
                                 /* Size of a block in block transfer mode */
      uint16_t                rsv;              /* Reserve bit                */
} dtc_transfer_data_cfg_t;
```

2.DTCb の場合

```
typedef struct st_dtc_transfer_data_cfg {
      dtc_transfer_mode_t     transfer_mode;    /* DTC transfer mode          */
      dtc_data_size_t         data_size;        /* Size of data               */
      dtc_src_addr_mode_t     src_addr_mode;    /* Address mode of source     */
      dtc_chain_transfer_t    chain_transfer_enable;
                                        /* Chain transfer is enabled or not  */
      dtc_chain_transfer_mode_t chain_transfer_mode;
                                        /* How chain transfer is performed   */
      dtc_interrupt_t         response_interrupt;
                                        /* How response interrupt is raised  */
      dtc_repeat_block_side_t repeat_block_side;/* Side being repeat or block */
      dtc_dest_addr_mode_t    dest_addr_mode;   /* Address mode of destination*/
      uint32_t                source_addr;      /* Start address of source    */
      uint32_t                dest_addr;  /* Start address of destination     */
      uint32_t                transfer_count;   /* Transfer count             */
      uint16_t                block_size;
                                 /* Size of a block in block transfer mode */
      uint16_t                rsv;              /* Reserve bit                */
      dtc_write_back_t        writeback_disable;
                          /* Transfer information writeback is enabled or not  */
      dtc_sequence_end_t      sequence_end;
                            /* Sequence transfer is continued or end      */
      dtc_refer_index_table_t refer_index_table_enable;
                            /* Index table reference is enabled or not    */
      dtc_disp_add_t          disp_add_enable;
                  /* Displacement value is added to the source address or not */
} dtc_transfer_data_cfg_t;
```

以下の列挙型の定義で、上記構造体の設定可能なオプションを示します。

```
/* Configurable options for DTC Transfer mode */
typedef enum e_dtc_transfer_mode
{
    DTC_TRANSFER_MODE_NORMAL = (0),        /* = (0 << 6):Normal mode */
    DTC_TRANSFER_MODE_REPEAT = (1 << 6),   /* Repeat mode */
    DTC_TRANSFER_MODE_BLOCK  = (2 << 6),   /* Block mode */
} dtc_transfer_mode_t;

/* Configurable options for DTC Data transfer size */
typedef enum e_dtc_data_size
{
    DTC_DATA_SIZE_BYTE  = (0),         /* = (0 << 4):8-bit (byte) data */
    DTC_DATA_SIZE_WORD  = (1 << 4),    /* 16-bit (word) data */
    DTC_DATA_SIZE_LWORD = (2 << 4),    /* 32-bit (long word) data */
} dtc_data_size_t;

/* Configurable options for Source address addressing mode */
typedef enum e_dtc_src_addr_mode
{
    DTC_SRC_ADDR_FIXED =  (0),      /* = (0 << 2):Source address is fixed.*/
    DTC_SRC_ADDR_INCR  =  (2 << 2),
                        /* Source address is incremented after each transfer.*/
    DTC_SRC_ADDR_DECR  =  (3 << 2),
                        /* Source address is decremented after each transfer.*/
} dtc_src_addr_mode_t;

/* Configurable options for Chain transfer */
typedef enum e_dtc_chain_transfer
{
    DTC_CHAIN_TRANSFER_DISABLE      = (0),       /* Disable Chain transfer.*/
    DTC_CHAIN_TRANSFER_ENABLE       = (1 << 7),  /* Enable Chain transfer.*/
} dtc_chain_transfer_t;

/* Configurable options for how chain transfer is performed */
typedef enum e_dtc_chain_transfer_mode
{
    DTC_CHAIN_TRANSFER_CONTINUOUSLY = (0),
                  /* = (0 << 6):Chain transfer is performed continuously.*/
    DTC_CHAIN_TRANSFER_NORMAL       = (1 << 6)
/* Chain transfer is performed only when the counter is changed to 0 or CRAH.*/
} dtc_chain_transfer_mode_t;

/* Configurable options for Interrupt */
typedef enum e_dtc_interrupt
{
    DTC_INTERRUPT_AFTER_ALL_COMPLETE  = (0),
      /* Interrupt is generated when specified data transfer is completed.*/
    DTC_INTERRUPT_PER_SINGLE_TRANSFER = (1 << 5)
      /* Interrupt is generated when each transfer time is completed.*/
} dtc_interrupt_t;
/* Configurable options for Side to be repeat or block */
typedef enum e_dtc_repeat_block_side
{
    DTC_REPEAT_BLOCK_DESTINATION = (0),
      /* = (0 << 4):Destination is repeat or block area.*/
    DTC_REPEAT_BLOCK_SOURCE  = (1 << 4)
      /* Source is repeat or block area.*/
} dtc_repeat_block_side_t;
```

```
/* Configurable options for Destination address addressing mode */
typedef enum e_dtc_dest_addr_mode
{
    DTC_DES_ADDR_FIXED = (1 << 2),  /* Destination address is fixed.*/
    DTC_DES_ADDR_INCR  = (2 << 2),
        /* Destination address is incremented after each transfer.*/
    DTC_DES_ADDR_DECR  = (3 << 2)
        /* Destination address is decremented after each transfer.*/
} dtc_dest_addr_mode_t;

/* Configurable options to write back transfer information */
typedef enum e_dtc_write_back
{
      DTC_WRITEBACK_ENABLE  = (0),  /* Writeback is enabled */
      DTC_WRITEBACK_DISABLE = (1)   /* Writeback is disabled */
} dtc_write_back_t;

/* Configurable option to continue/end sequence transfer */
typedef enum e_dtc_sequence_end
{
      DTC_SEQUENCE_TRANSFER_CONTINUE = (0), /* Sequence transfer is continued */
      DTC_SEQUENCE_TRANSFER_END      = (1)  /* Sequence transfer is ended */
} dtc_sequence_end_t;

/* Configurable options for index table reference */
typedef enum e_dtc_refer_index_table
{
      DTC_REFER_INDEX_TABLE_DISABLE = (0),    /* Index table is not referred */
      DTC_REFER_INDEX_TABLE_ENABLE  = (1 << 1)  /* Index table is referred */
} dtc_refer_index_table_t;

/* Configurable options to add/not to add Displacement value to the destination
address */
typedef enum e_dtc_disp_add
{
      DTC_SRC_ADDR_DISP_ADD_DISABLE = (0),
                  /* Displacement value is not added to the source address */
      DTC_SRC_ADDR_DISP_ADD_ENABLE  = (1)
                  /* Displacement value is added to the source address */
} dtc_disp_add_t;
```

p_data_cfg->transfer_count には、ノーマル転送モードとブロック転送モードでは 1～65536 の値が、リピート転送モードでは 1～256 の値が設定されます。

p_data_cfg->block_size には、ブロック転送モードで 1～256 の値が設定されます。

ショートアドレスモードでは、転送情報の開始アドレス（第 2 引数）、転送元領域、転送先領域は 0x00000000～0x007FFFFF および 0xFF800000～0xFFFFFFFF の範囲内で設定されます。

Return Values

[DTC_SUCCESS]	*/* 正常終了 */*
[DTC_ERR_NOT_OPEN]	*/* DTC は未初期化状態です。 */*
[DTC_ERR_INVALID_ARG]	*/* 引数は無効な値です。 */*
[DTC_ERR_NULL_PTR]	*/*引数のポインタが NULL です。 */*

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

転送情報に設定情報を書き込みます。

割り込み番号に対応する転送情報の先頭アドレスを DTC ベクタテーブルに書き込みます。

Example

処理 1: チェーン転送を行わない場合

```
dtc_transfer_data_cfg_t td_cfg;
dtc_activation_source_t act_src = DTCE_ICU_SWINT; /* activation source is
Software Interrupt */

dtc_transfer_data_t  transfer_data;  /* assume that DTC address mode is full
mode */

dtc_err_t ret;
uint32_t src = 1234;
uint32_t des[3];
uint8_t  ien_bk;

/* create the configuration – no chain transfer */
/* Source address addressing mode is FIXED
 * Data size is 32 bits (long word)
 * DTC transfer mode is Repeat mode & Source side is repeat area
 * Interrupt is raised after each single transfer
 * Chain transfer is disabled
*/
td_cfg.src_addr_mode         = DTC_SRC_ADDR_FIXED;
td_cfg.data_size             = DTC_DATA_SIZE_LWORD;
td_cfg.transfer_mode         = DTC_TRANSFER_MODE_REPEAT;
td_cfg.dest_addr_mode        = DTC_DES_ADDR_INCR;
td_cfg.repeat_block_side     = DTC_REPEAT_BLOCK_SOURCE;
td_cfg.response_interrupt    = DTC_INTERRUPT_PER_SINGLE_TRANSFER;
td_cfg.chain_transfer_enable = DTC_CHAIN_TRANSFER_DISABLE;
td_cfg.chain_transfer_mode   = (dtc_chain_transfer_mode_t)0;

td_cfg.source_addr           = (uint32_t)&src;
td_cfg.dest_addr             = (uint32_t)des;
td_cfg.transfer_count        = 1;
td_cfg.block_size            = 3;

/* Disable Software interrupt request before calling R_DTC_Create() */
ien_bk = ICU.IER[3].BIT.IEN3 ; /* store old setting */
ICU.IER[3].BIT.IEN3 = 0;

/* Calling to R_DTC_Create() */
```

```
ret = R_DTC_Create(act_src, &transfer_data, &td_cfg, 0);

/* Restore the setting for Software interrupt request */
ICU.IER[3].BIT.IEN3 = ien_bk;
```

処理 2: チェーン転送を 1 回行う場合

```
dtc_transfer_data_cfg_t td_cfg[2]; /* need 2 configuration sets */
dtc_activation_source_t act_src = DTCE_ICU_SWINT;
                          /* activation source is Software Interrupt */
uint32_t transfer_data[8];
          /* for 2 Transfer data; assume that DTC address mode is full mode */
dtc_err_t ret;
uint32_t src = 1234;
uint32_t des[3];              /* The destination for first Transfer data */
uint32_t des2[3];             /* The destination for second Transfer data */
uint8_t  ien_bk;

/* create the configuration 1 – support chain transfer */
/* Source address addressing mode is FIXED
 * Destination address addressing mode is INCREMENTED
 * Data size is 32 bits (long word)
 * DTC transfer mode is Normal mode
 * Interrupt is raised after each single transfer
 * Chain transfer is enabled
 * Chain transfer is performed after when transfer counter is set to 0
*/
td_cfg[0].src_addr_mode         = DTC_SRC_ADDR_FIXED;
td_cfg[0].data_size             = DTC_DATA_SIZE_LWORD;
td_cfg[0].transfer_mode         = DTC_TRANSFER_MODE_NORMAL;
td_cfg[0].dest_addr_mode        = DTC_DES_ADDR_INCR;
td_cfg[0].repeat_block_side     = DTC_REPEAT_BLOCK_SOURCE;
td_cfg[0].response_interrupt    = DTC_INTERRUPT_PER_SINGLE_TRANSFER;
td_cfg[0].chain_transfer_enable = DTC_CHAIN_TRANSFER_ENABLE;
td_cfg[0].chain_transfer_mode   = DTC_CHAIN_TRANSFER_NORMAL;

td_cfg[0].source_addr           = (uint32_t)&src;
td_cfg[0].dest_addr       = (uint32_t)des; /* transfer from source to des 1 */
td_cfg[0].transfer_count        = 1;
td_cfg[0].block_size            = 3;

/* create the configuration 2 – no chain transfer */
/* Source address addressing mode is FIXED
 * Destination address addressing mode is INCREMENTED
 * Data size is 32 bits (long word)
 * DTC transfer mode is Normal mode
 * Interrupt is raised after each single transfer
 * Chain transfer is disabled*/
td_cfg[1].src_addr_mode         = DTC_SRC_ADDR_FIXED;
td_cfg[1].data_size             = DTC_DATA_SIZE_LWORD;
td_cfg[1].transfer_mode         = DTC_TRANSFER_MODE_NORMAL;
td_cfg[1].dest_addr_mode        = DTC_DES_ADDR_INCR;
td_cfg[1].repeat_block_side     = DTC_REPEAT_BLOCK_SOURCE;
td_cfg[1].response_interrupt    = DTC_INTERRUPT_PER_SINGLE_TRANSFER;
td_cfg[1].chain_transfer_enable = DTC_CHAIN_TRANSFER_DISABLE;
td_cfg[1].chain_transfer_mode   = (dtc_chain_transfer_mode_t)0;

td_cfg[1].source_addr           = (uint32_t)&src;
td_cfg[1].dest_addr       = (uint32_t)des2; /* transfer from source to des 2*/
td_cfg[1].transfer_count        = 1;
td_cfg[1].block_size            = 3;
```

```
/* Disable Software interrupt request before calling R_DTC_Create() */
ien_bk = ICU.IER[3].BIT.IEN3 ; /* store old setting */
ICU.IER[3].BIT.IEN3 = 0;

/* Call R_DTC_Create() */
ret = R_DTC_Create(act_src, transfer_data , td_cfg, 1); /* The fourth argument
indicates that there's one chain transfer enabled in first Transfer data */

/* Restore the setting for Software interrupt request */
ICU.IER[3].BIT.IEN3 = ien_bk;
```

処理 3: 複数要因の登録を行う場合

```
dtc_transfer_data_cfg_t td_cfg_sw;
dtc_transfer_data_cfg_t td_cfg_cmt;
dtc_activation_source_t act_src_sw = DTCE_ICU_SWINT;
                                /* activation source is Software Interrupt */
dtc_activation_source_t act_src_cmt = DTCE_CMT0_CMI0;
                                /* activation source is CMT Interrupt */
dtc_transfer_data_t transfer_data_sw;
                                /* assume that DTC address mode is full mode */
dtc_transfer_data_t transfer_data_cmt;
                                /* assume that DTC address mode is full mode */

dtc_err_t ret;
uint32_t src_sw = 1234;
uint32_t src_cmt = 5678;
uint32_t des_sw[3];
uint32_t des_cmt[3];
uint8_t  ien_bk;

/* create the configuration – no chain transfer */
/* Source address addressing mode is FIXED
* Data size is 32 bits (long word)
* DTC transfer mode is Repeat mode & Source side is repeat area
* Interrupt is raised after each single transfer
* Chain transfer is disabled
*/
td_cfg_sw.src_addr_mode = DTC_SRC_ADDR_FIXED;
td_cfg_sw.data_size = DTC_DATA_SIZE_LWORD;
td_cfg_sw.transfer_mode = DTC_TRANSFER_MODE_REPEAT;
td_cfg_sw.dest_addr_mode = DTC_DES_ADDR_INCR;
td_cfg_sw.repeat_block_side = DTC_REPEAT_BLOCK_SOURCE;
td_cfg_sw.response_interrupt = DTC_INTERRUPT_PER_SINGLE_TRANSFER;
td_cfg_sw.chain_transfer_enable = DTC_CHAIN_TRANSFER_DISABLE;
td_cfg_sw.chain_transfer_mode = (dtc_chain_transfer_mode_t)0;

td_cfg_sw.source_addr = (uint32_t)&src_sw;
td_cfg_sw.dest_addr = (uint32_t)des_sw;
td_cfg_sw.transfer_count = 1;
td_cfg_sw.block_size = 3;

/* Disable Software interrupt request before calling R_DTC_Create() */
ien_bk = ICU.IER[3].BIT.IEN3 ; /* store old setting */
ICU.IER[3].BIT.IEN3 = 0;

/* Calling to R_DTC_Create() */
ret = R_DTC_Create(act_src_sw, &transfer_data_sw, &td_cfg_sw, 0);
/* Restore the setting for Software interrupt request */
ICU.IER[3].BIT.IEN3 = ien_bk;
```

```
/* create the configuration – no chain transfer */
/* Source address addressing mode is FIXED
* Data size is 32 bits (long word)
* DTC transfer mode is Repeat mode & Source side is repeat area
* Interrupt is raised after each single transfer
* Chain transfer is disabled
*/
td_cfg_cmt.src_addr_mode = DTC_SRC_ADDR_FIXED;
td_cfg_cmt.data_size = DTC_DATA_SIZE_LWORD;
td_cfg_cmt.transfer_mode = DTC_TRANSFER_MODE_REPEAT;
td_cfg_cmt.dest_addr_mode = DTC_DES_ADDR_INCR;
td_cfg_cmt.repeat_block_side = DTC_REPEAT_BLOCK_SOURCE;
td_cfg_cmt.response_interrupt = DTC_INTERRUPT_PER_SINGLE_TRANSFER;
td_cfg_cmt.chain_transfer_enable = DTC_CHAIN_TRANSFER_DISABLE;
td_cfg_cmt.chain_transfer_mode = (dtc_chain_transfer_mode_t)0;

td_cfg_cmt.source_addr = (uint32_t)&src_cmt;
td_cfg_cmt.dest_addr = (uint32_t)des_cmt;
td_cfg_cmt.transfer_count = 1;
td_cfg_cmt.block_size = 3;

/* Calling to R_DTC_Create() */
ret = R_DTC_Create(act_src_cmt, &transfer_data_cmt, &td_cfg_cmt, 0);

R_CMT_CreateOneShot(10000, &cmt_callback, &cmt_channel);
```

Special Notes:

R_DTC_Create()を呼び出す前に、ユーザは割り込み要求許可ビット（IERm.IENj）をクリアし、処理対象の割り込み要求を禁止にする必要があります（割り込み要因は R_DTC_Create()に渡されます）。

```
ICU.IER[m].BIT.IENj = 0;
```

R_DTC_Create()の処理終了後に、禁止にした割り込み要求を許可します。

IERm.IENj ビットと割り込み要因の対応については、ユーザーズマニュアル ハードウェア編の割り込みコントローラ（ICU）章の「割り込みベクタテーブル」をご覧ください。

R_DTC_CreateSeq()

この関数は、シーケンス転送で使用する DTC レジスタと起動要因の設定を行います。

Format

```
dtc_err_t           R_DTC_CreateSeq(
                    dtc_activation_source_t     act_source,
                    dtc_transfer_data_t         *p_transfer_data,
                    dtc_transfer_data_cfg_t     *p_data_cfg,
                    uint32_t                    sequence_transfer_nr,
                    uint8_t                     sequence_no
)
```

Parameters

dtc_activation_source_t *act_source*

起動要因

dtc_transfer_data_t * *p_transfer_data*

RAM の転送情報領域の開始アドレスへのポインタ

dtc_transfer_data_cfg_t * *p_data_cfg*

転送情報設定へのポインタ

以下の構造体メンバも設定してください。

```
p_data_cfg->writeback_disable
p_data_cfg->sequence_end
p_data_cfg->refer_index_table_enable
p_data_cfg->disp_add_enable
```

uint32_t *sequence_transfer_nr*

1 シーケンス転送の転送情報数（0 - 4294967295）

sequence_transfer_nr	説明
0	指定したシーケンス番号（sequence_no）の転送要求が発生した場合、シーケンスを開始せずに CPU 割り込み要求を出力するための設定を行います。
1 - 4294967295	指定したシーケンス番号（sequence_no）の転送要求が発生した場合、シーケンス転送を行うための転送情報を設定します。 事前に sequence_transfer_nr に指定する数の転送情報を準備し、転送情報の先頭アドレスを*p_data_cfg に設定してください。

uint8_t *sequence_no*

シーケンス番号（0 - 255）

転送情報の型定義、データ構造体は R_DTC_Create()と同じです。全 256 とおりのシーケンス転送情報を設定することができます。

Return Values

DTC_SUCCESS	*/* 正常終了 */*
DTC_ERR_NOT_OPEN	*/* DTC は未初期化状態です。 */*
DTC_ERR_INVALID_ARG	/* 引数は無効な値です。 */
DTC_ERR_NULL_PTR	*/*引数のポインタが NULL です。 */*

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

転送情報に設定情報を書き込みます。

シーケンス番号に対応する転送情報の先頭アドレスを DTC インデックステーブルに書き込みます。

Example

受信 FIFO フル割り込み(以下、RXI)を DTC 起動要因として、シーケンス転送による調歩同期式シリアル受信を行う例を以下に説明します。使用する SCI はチャネル 10 です。外部通信デバイスから最初に受信した 1 バイトデータ（cmnd）に応じて自動的にシーケンス転送を開始します。

処理 1:

外部通信デバイスから cmnd=“00h”を受信後、SCI10 受信 FIFO しきい値を 4 バイトに変更し、外部通信デバイスから出力される 4 バイトのデータを受信し、DTC 転送によって RAM へ格納させる。

表 3-1 処理 1 で設定する転送情報

メンバ	転送情報 1	転送情報 2	転送情報 3
transfer_mode	ノーマル転送	ブロック転送	ノーマル転送
data_size	8 ビット	16 ビット	8 ビット
src_addr_mode	ソースアドレス固定	ソースアドレス固定	ソースアドレス固定
chain_transfer_enable	チェーン転送禁止	チェーン転送許可	チェーン転送禁止
chain_transfer_mode	チェーン転送を連続で実行（設定無効）	チェーン転送を連続で実行	チェーン転送を連続で実行（設定無効）
response_interrupt	指定したデータ転送が完了したら、割り込みを生成	指定したデータ転送が完了したら、割り込みを生成	指定したデータ転送が完了したら、割り込みを生成
repeat_block_side	転送先はリピートまたはブロック領域（設定無効）	転送先はリピートまたはブロック領域	転送先はリピートまたはブロック領域（設定無効）
dest_addr_mode	転送先アドレスは固定	転送ごとに、転送先アドレスをインクリメント	転送先アドレスは固定
source_addr	ROM の dtc_fcrh_data[0]のアドレス	SCI10.FRDR レジスタアドレス	ROM の g_dtc_fcrh_cmnd のアドレス
dest_addr	SCI10.FCR.H レジスタアドレス	RAM の g_dtc_rx_buf0[0]のアドレス	SCI10.FCR.H レジスタアドレス
transfer_count	1	1	1
block_size	（設定無効）	4	（設定無効）
writeback_disable	ライトバックしない	ライトバックしない	ライトバックしない
sequence_end	シーケンス転送を継続	シーケンス転送を継続	シーケンス転送を終了
refer_index_table_enable	インデックステーブルを参照しない	インデックステーブルを参照しない	インデックステーブルを参照しない
disp_add_enable	転送元アドレスにディスプレースメント値を加算しない	転送元アドレスにディスプレースメント値を加算しない	転送元アドレスにディスプレースメント値を加算しない

```
#include "platform.h"
#include "r_dtc_rx_if.h"

#define CMND0_RCV_NUM (4)
#define CMND0_RCV_FIFO_TRG (4)
#define CMND0_FCRH_DATA ((uint8_t)(0xF0 | CMND0_RCV_FIFO_TRG))
#define CMND0_INFO_NUM (3)

dtc_transfer_data_cfg_t g_dtc_pre_seqinfo_cmnd0[CMND0_INFO_NUM];
dtc_transfer_data_t g_dtc_seqinfo_cmnd0[CMND0_INFO_NUM];
uint16_t g_dtc_rx_buf0[CMND0_RCV_NUM];
const uint8_t g_dtc_fcrh_cmnd = 0xF1;
static const uint8_t dtc_fcrh_data[] =
{
      CMND0_FCRH_DATA,
      CMND1_FCRH_DATA,
      CMND2_FCRH_DATA,
      CMND3_FCRH_DATA
};

void dtc_pre_seqinfo_cmnd0_init(void);

void main(void)
{
      dtc_err_t ret;
      dtc_activation_source_t act_source;
      uint32_t sequence_transfer_nr;
      uint8_t  sequence_no;
      uint8_t ien_bk;

      …

      /* ---- DTC sequence transfer information for Cmnd0 ---- */
      dtc_pre_seqinfo_cmnd0_init();
      act_source = DTCE_SCI10_RXI10;
      sequence_transfer_nr = CMND0_INFO_NUM;
      sequence_no = 0;
      ien_bk  = IEN(SCI10,RXI10); /* IEN(x,x)->ICU.IER[z].BIT.IENz;*/
      IEN(SCI10,RXI10) = 0;
      ret = R_DTC_CreateSeq(act_source,
                            &g_dtc_seqinfo_cmnd0[0],
                            &g_dtc_pre_seqinfo_cmnd0[0],
                            sequence_transfer_nr,
                            sequence_no);
      IEN(SCI10,RXI10)= ien_bk;

      …
}

void dtc_pre_seqinfo_cmnd0_init(void)
{
      /* [1st] Sequence transfer information –
               Changing the SCI10 Receive FIFO trigger */
      /* MRA */
      g_dtc_pre_seqinfo_cmnd0[0].transfer_mode = DTC_TRANSFER_MODE_NORMAL;
      g_dtc_pre_seqinfo_cmnd0[0].data_size = DTC_DATA_SIZE_BYTE;
      g_dtc_pre_seqinfo_cmnd0[0].src_addr_mode = DTC_SRC_ADDR_FIXED;
      g_dtc_pre_seqinfo_cmnd0[0].writeback_disable = DTC_WRITEBACK_DISABLE;
      /* MRB */
      g_dtc_pre_seqinfo_cmnd0[0].chain_transfer_enable =
```

```
                                        DTC_CHAIN_TRANSFER_DISABLE;
    g_dtc_pre_seqinfo_cmnd0[0].chain_transfer_mode =
                                        DTC_CHAIN_TRANSFER_CONTINUOUSLY;
    g_dtc_pre_seqinfo_cmnd0[0].response_interrupt =
                                        DTC_INTERRUPT_AFTER_ALL_COMPLETE;
    g_dtc_pre_seqinfo_cmnd0[0].repeat_block_side =
                                        DTC_REPEAT_BLOCK_DESTINATION;
    g_dtc_pre_seqinfo_cmnd0[0].dest_addr_mode = DTC_DES_ADDR_FIXED;
    g_dtc_pre_seqinfo_cmnd0[0].refer_index_table_enable =
                                        DTC_REFER_INDEX_TABLE_DISABLE;
    g_dtc_pre_seqinfo_cmnd0[0].sequence_end =
                                        DTC_SEQUENCE_TRANSFER_CONTINUE;
    /* MRC */
    g_dtc_pre_seqinfo_cmnd0[0].disp_add_enable =
                                        DTC_SRC_ADDR_DISP_ADD_DISABLE;
    /* SAR */
    g_dtc_pre_seqinfo_cmnd0[0].source_addr = (uint32_t)&dtc_fcrh_data[0];
    /* DAR */
    g_dtc_pre_seqinfo_cmnd0[0].dest_addr = (uint32_t)&SCI10.FCR.BYTE.H;
    /* CRA, CRB */
    g_dtc_pre_seqinfo_cmnd0[0].transfer_count = 1;


    /* [2nd] Sequence transfer information –
             transfers the received data from SCI10.FRDR to RAM */
    /* MRA */
    g_dtc_pre_seqinfo_cmnd0[1].transfer_mode = DTC_TRANSFER_MODE_BLOCK;
    g_dtc_pre_seqinfo_cmnd0[1].data_size = DTC_DATA_SIZE_WORD;
    g_dtc_pre_seqinfo_cmnd0[1].src_addr_mode = DTC_SRC_ADDR_FIXED;
    g_dtc_pre_seqinfo_cmnd0[1].writeback_disable = DTC_WRITEBACK_DISABLE;
    /* MRB */
    g_dtc_pre_seqinfo_cmnd0[1].chain_transfer_enable =
                                         DTC_CHAIN_TRANSFER_ENABLE;
    g_dtc_pre_seqinfo_cmnd0[1].chain_transfer_mode =
                                         DTC_CHAIN_TRANSFER_CONTINUOUSLY;
    g_dtc_pre_seqinfo_cmnd0[1].response_interrupt =
                                         DTC_INTERRUPT_AFTER_ALL_COMPLETE;
    g_dtc_pre_seqinfo_cmnd0[1].repeat_block_side =
                                         DTC_REPEAT_BLOCK_DESTINATION;
    g_dtc_pre_seqinfo_cmnd0[1].dest_addr_mode = DTC_DES_ADDR_INCR;
    g_dtc_pre_seqinfo_cmnd0[1].refer_index_table_enable =
                                         DTC_REFER_INDEX_TABLE_DISABLE;
    g_dtc_pre_seqinfo_cmnd0[1].sequence_end =
                                         DTC_SEQUENCE_TRANSFER_CONTINUE;
    /* MRC */
    g_dtc_pre_seqinfo_cmnd0[1].disp_add_enable =
DTC_SRC_ADDR_DISP_ADD_DISABLE;
    /* SAR */
    g_dtc_pre_seqinfo_cmnd0[1].source_addr = (uint32_t)&SCI10.FRDR.WORD;
    /* DAR */
    g_dtc_pre_seqinfo_cmnd0[1].dest_addr = (uint32_t)&g_dtc_rx_buf0[0];
    /* CRA, CRB */
    g_dtc_pre_seqinfo_cmnd0[1].transfer_count = 1;
    g_dtc_pre_seqinfo_cmnd0[1].block_size = CMND0_RCV_FIFO_TRG;

    /* [3rd] Sequence transfer information –
             Changing the SCI10 Receive FIFO trigger to 1 */
    /* MRA */
    g_dtc_pre_seqinfo_cmnd0[2].transfer_mode = DTC_TRANSFER_MODE_NORMAL;
    g_dtc_pre_seqinfo_cmnd0[2].data_size = DTC_DATA_SIZE_BYTE;
    g_dtc_pre_seqinfo_cmnd0[2].src_addr_mode = DTC_SRC_ADDR_FIXED;
    g_dtc_pre_seqinfo_cmnd0[2].writeback_disable = DTC_WRITEBACK_DISABLE;
```

```
      /* MRB */
      g_dtc_pre_seqinfo_cmnd0[2].chain_transfer_enable =
                                                DTC_CHAIN_TRANSFER_DISABLE;
      g_dtc_pre_seqinfo_cmnd0[2].chain_transfer_mode =
                                                DTC_CHAIN_TRANSFER_CONTINUOUSLY;
      g_dtc_pre_seqinfo_cmnd0[2].response_interrupt =
                                                DTC_INTERRUPT_AFTER_ALL_COMPLETE;
      g_dtc_pre_seqinfo_cmnd0[2].repeat_block_side =
                                                DTC_REPEAT_BLOCK_DESTINATION;
      g_dtc_pre_seqinfo_cmnd0[2].dest_addr_mode = DTC_DES_ADDR_FIXED;
      g_dtc_pre_seqinfo_cmnd0[2].refer_index_table_enable=
                                                DTC_REFER_INDEX_TABLE_DISABLE;
      g_dtc_pre_seqinfo_cmnd0[2].sequence_end = DTC_SEQUENCE_TRANSFER_END;
      /* MRC */
      g_dtc_pre_seqinfo_cmnd0[2].disp_add_enable =
DTC_SRC_ADDR_DISP_ADD_DISABLE;
      /* SAR */
      g_dtc_pre_seqinfo_cmnd0[2].source_addr = (uint32_t)&g_dtc_fcrh_cmnd;
      /* DAR */
      g_dtc_pre_seqinfo_cmnd0[2].dest_addr = (uint32_t)&SCI10.FCR.BYTE.H;
      /* CRA, CRB */
      g_dtc_pre_seqinfo_cmnd0[2].transfer_count = 1;
}
```

処理 2:

外部通信デバイスから cmnd≧“04h”を受信した場合、シーケンス転送せずに CPU への割り込み要求を発生させる。

```
#include "platform.h"
#include "r_dtc_rx_if.h"

void main(void)
{
      dtc_err_t ret;
      dtc_activation_source_t act_source;
      uint32_t sequence_transfer_nr;
      uint8_t  sequence_no;
      uint8_t ien_bk;
      uint16_t i;

      …

      /* ---- DTC sequence transfer information for Cmnd4-Cmnd255 ---- */
      for (i = 4; i < 256; i++)
      {
            act_source = DTCE_SCI10_RXI10;
            sequence_transfer_nr = 0;
            sequence_no = i;
            ien_bk  = IEN(SCI10,RXI10); /* IEN(x,x)->ICU.IER[z].BIT.IENz;*/
            IEN(SCI10,RXI10) = 0;
            ret = R_DTC_CreateSeq(act_source,
                                  NULL,
                                  NULL,
                                  sequence_transfer_nr,
                                  sequence_no);
            IEN(SCI10,RXI10) = ien_bk;
      }

      …
```

Special Notes:

R_DTC_CreateSeq()を呼び出す前に、ユーザは割り込み要求許可ビット（IERm.IENj）をクリアし、処理対象の割り込み要求を禁止にする必要があります（割り込み要因は R_DTC_CreateSeq()に渡されます）。

```
ICU.IER[m].BIT.IENj = 0;
```

R_DTC_CreateSeq()の処理終了後に、禁止にした割り込み要求を許可します。

IERm.IENj ビットと割り込み要因の対応については、ユーザーズマニュアル ハードウェア編の割り込みコントローラ（ICU）章の「割り込みベクタテーブル」をご覧ください。

R_DTC_Control()

この関数は DTC の動作を制御します。

Format

```
dtc_err_t       R_DTC_Control (
                dtc_command_t       command,
                dtc_stat_t          * p_stat,
                dtc_cmd_arg_t       * p_args
)
```

Parameters

dtc_command_t command

DTC の制御コマンド。

*dtc_stat_t * p_stat*

コマンドが DTC_CMD_STATUS_GET の場合、ステータスのポインタ。

dtc_stat_t 構造体のメンバ

メンバ	内容	設定値	説明
vect_nr	DTC 起動ベクタ番号	ベクタ番号監視	この数値は DTC 転送が処理中のときにのみ有効です（DTC アクティブフラグ= 1）
in_progress	DTC アクティブフラグ	- false - true	- DTC 転送動作中ではない - DTC 転送動作中

*dtc_cmd_arg_t * p_args*

コマンドが *DTC_CMD_ACT_SRC_ENABLE*、DTC_CMD_ACT_SRC_DISABLE、DTC_CMD_CHAIN_TRANSFER_ABORT、DTC_CMD_SEQUENCE_TRANSFER_ENABLE、または DTC_CMD_CHANGING_DATA_FORCIBLY_SET の場合、引数の構造体のポインタ。

dtc_cmd_arg_t 構造体のメンバ

メンバ	内容	説明
act_src	DTC 起動ベクタ番号	この数値はコマンドが DTC_CMD_ACT_SRC_ENABLE または DTC_CMD_ACT_SRC_DISABLE または DTC_CMD_SEQUENCE_TRANSFER_ENABLE または DTC_CMD_CHANGING_DATA_FORCIBLY_SET の場合のみ有効
chain_transfer_nr	チェーン転送数（注 1）	この数値はコマンドが DTC_CMD_CHAIN_TRANSFER_ABORT または DTC_CMD_CHANGING_DATA_FORCIBLY_SET の場合のみ有効
*p_transfer_data	RAM の転送情報領域の開始アドレスへのポインタ	この数値はコマンドが DTC_CMD_CHANGING_DATA_FORCIBLY_SET の場合のみ有効
*p_data_cfg	転送情報設定へのポインタ	この数値はコマンドが DTC_CMD_CHANGING_DATA_FORCIBLY_SET の場合のみ有効

注1. ユーザが R_DTC_Create()を呼び出した時の引数“chain_transfer_nr”と同じ値を設定してください。

Return Values

[DTC_SUCCESS] */* 正常終了 */*
[DTC_ERR_NOT_OPEN] */* DTC は未初期化状態です。*/*
[DTC_ERR_INVALID_COMMAND] */* コマンドのパラメータが無効です。もしくは、DTC_CMD_CHANGING_DATA_FORCIBLY_SET コマンドのエラー。 */*
[DTC_ERR_NULL_PTR] */* 引数のポインタが NULL です。*/*
[DTC_ERR_ACT] */* データ転送実行中 */*

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

コマンドより異なる処理を実行します。

コマンド	引数 dtc_stat_t *	引数 dtc_cmd_arg_t *	説明
DTC_CMD_DTC_START	NULL	NULL	DTC モジュール起動ビット（DTCST）を使って DTC モジュールを起動します。
DTC_CMD_DTC_STOP	NULL	NULL	DTC モジュール起動ビット（DTCST）を使って DTC モジュールを停止します。
DTC_CMD_DATA_READ_SKIP_ENABLE	NULL	NULL	DTC 転送情報リードスキップ許可ビット（RRS）を使って、転送情報リードスキップを許可します。
DTC_CMD_DATA_READ_SKIP_DISABLE	NULL	NULL	DTC 転送情報リードスキップ許可ビット（RRS）を使って、転送情報リードスキップを禁止します。
DTC_CMD_ACT_SRC_ENABLE	NULL	p_args->act_src	DTC 起動許可ビット（DTCE）を 1 使って、DTC 起動要因をセットします。
DTC_CMD_ACT_SRC_DISABLE	NULL	p_args->act_src	DTC 起動許可ビット（DTCE）を使って、DTC 起動要因をクリアします。
DTC_CMD_STATUS_GET	p_stat->in_progress p_stat->vect_nr	NULL	DTC ステータスレジスタ（DTCSTS）を使って DTC のアクティブフラグ（ACT）とデータ転送実行中のベクタ番号（VECN[7:0]）を取得します。
DTC_CMD_CHAIN_TRANSFER_ABORT	NULL	p_args->chain_transfer_nr	処理中のチェーン転送を中止します。
DTC_CMD_SEQUENCE_TRANSFER_ENABLE	NULL	p_args->act_src	DTC シーケンス転送許可レジスタ（DTCSEQ）を使って、シーケンス転送ベクタ番号指定とシーケンス転送を許可します。
DTC_CMD_SEQUENCE_TRANSFER_DISABLE	NULL	NULL	DTC シーケンス転送許可レジスタ（DTCSEQ）を使って、シーケンス転送を禁止します。
DTC_CMD_SEQUENCE_TRANSFER_ABORT	NULL	NULL	シーケンス転送終了ビット（SQTFRL）を使って、シーケンス転送を強制的に終了します。
DTC_CMD_CHANGING_DATA_FORCIBLY_SET	NULL	p_args->act_src p_args->chain_transfer_nr p_args->p_transfer_data p_args->p_data_cfg	R_DTC_Create()によって設定された値を変更します。R_DTC_Create()によって強制的に設定されたパラメータ（注 1）を変更するのに有効な処理です。

注 1 ：writeback_disable、sequence_end、refer_index_table_enable、および disp_add_enable

Example

処理 1: DTC モジュールを起動する。

```
dtc_err_t ret;

/* Start DTC module */
ret = R_DTC_Control(DTC_CMD_DTC_START, NULL, NULL);
```

処理 2 : DTC モジュールを停止する。

```
dtc_err_t ret;

/* Stop DTC module */
ret = R_DTC_Control(DTC_CMD_DTC_STOP, NULL, NULL);
```

処理 3 : 転送情報リードスキップを許可する。

```
dtc_err_t ret;

/* Enable transfer information read skip */
ret = R_DTC_Control(DTC_CMD_DATA_READ_SKIP_ENABLE, NULL, NULL);
```

処理 4 : 転送情報リードスキップを禁止する。

```
dtc_err_t ret;

/* Disable transfer information read skip */
ret = R_DTC_Control(DTC_CMD_DATA_READ_SKIP_DISABLE, NULL, NULL);
```

処理 5 : DTCE を使用し、DTC 起動要因をセットする。

```
dtc_err_t ret;
dtc_cmd_arg_t args;

/* Disable DTC transfer request by SCI10 receive data full interrupt */
IEN(SCI10, RXI10) = 0;

/* Set SCI10 receive data full interrupt as DTC activation source*/
args.act_src = DTCE_SCI10_RXI10;

/* Set the interrupt used for DTC activation source */
ret = R_DTC_Control(DTC_CMD_ACT_SRC_ENABLE, NULL, &args);
```

処理 6 : DTCE を使用し、DTC 起動要因をクリアする。

```
dtc_err_t ret;
dtc_cmd_arg_t args;

/* Disable DTC trasnfer request by SCI10 receive data full interrupt */
IEN(SCI10, RXI10) = 0;

/* Set SCI10 receive data full interrupt as DTC activation source */
args.act_src = DTCE_SCI10_RXI10;

/* Delete the interrupt used for DTC activation source */
ret = R_DTC_Control(DTC_CMD_ACT_SRC_DISABLE, NULL, &args);
```

処理 7：DTC のアクティブフラグ（ACT）とデータ転送実行中のベクタ番号（VECN[7:0]）を取得する。

```
dtc_err_t ret;
dtc_stat_t stat;
uint8_t interrupt_number;

/* Get DTC Active Flag (ACT) and Vector number(VECN[7:0])in progress */
ret = R_DTC_Control(DTC_CMD_STATUS_GET, stat, NULL);

if (true == stat.in_progress)
{
/* Vector number is valid */
interrupt_number = stat.vect_nr;
}
else
{
      /* Vector number is inbalid */
}
```

処理 8：処理中のチェーン転送を中止する。

```
dtc_err_t ret;
dtc_cmd_arg_t args;

/* No. Of chain transfer = 5 */
args. chain_transfer_nr = 5;

/* Abort the chain transfer in process */
ret = R_DTC_Control(DTC_CMD_STATUS_GET, NULL, &args);
```

処理 9：シーケンス転送を許可する。

```
dtc_err_t ret;
dtc_cmd_arg_t args;

/* Set SCI10 receive data full interrupt as sequence transfger activation source
*/
args.act_src = DTCE_SCI10_RXI10;

/* Enable sequence transfer */
ret = R_DTC_Control(DTC_CMD_SEQUENCE_TRANSFER_ENABLE, NULL, &args);
```

処理 10：シーケンス転送を禁止する。

```
dtc_err_t ret;

/* Disable sequence transfer */
ret = R_DTC_Control(DTC_CMD_SEQUENCE_TRANSFER_DISABLE, NULL, NULL);
```

処理 11：シーケンス転送を強制的に終了する。

```
dtc_err_t ret;

/* Disable DTC transfer request by SCI10 receive data full interrupt */
IEN(SCI10, RXI10) = 0;

/* Issue command repeatedly until sequence transfer can be aborted */
do
{
ret = R_DTC_Control(DTC_CMD_SEQUENCE_TRANSFER_ABORT, NULL, NULL);
} while (DTC_ERR_ACT == ret);
```

処理 12：R_DTC_Create()によって設定された値を変更する。

```
dtc_activation_source_t act_source;
uint32_t chain_transfer_nr;

act_source = DTCE_SCI10_RXI10;
chain_transfer_nr = 0;
if (R_DTC_Create(act_source,
                    &g_dtc_info_sqnum,
                    &g_dtc_pre_info_sqnum,
                    chain_transfer_nr) != DTC_SUCCESS)
{
    /* Error */
}

g_dtc_pre_info_sqnum.writeback_disable       = DTC_WRITEBACK_DISABLE;
g_dtc_pre_info_sqnum.sequence_end             = DTC_SEQUENCE_TRANSFER_CONTINUE;
g_dtc_pre_info_sqnum.refer_index_table_enable = DTC_REFER_INDEX_TABLE_ENABLE;
g_dtc_pre_info_sqnum.disp_add_enable          = DTC_SRC_ADDR_DISP_ADD_DISABLE;
args.act_src = DTCE_SCI10_RXI10;
args.chain_transfer_nr = 0;
args.p_transfer_data = &g_dtc_info_sqnum;
args.p_data_cfg = &g_dtc_pre_info_sqnum;
if (R_DTC_Control(DTC_CMD_CHANGING_DATA_FORCIBLY_SET, NULL, &args) !=
DTC_SUCCESS)
{
    /* Error */
}
```

Special Notes:

コマンドが DTC_CMD_GET_STATUS の場合、DTC が処理中（p_stat->in_progress が true）の場合にのみベクタ番号は有効です。

コマンドが DTC_CMD_ENABLE_ACT_SRC、DTC_CMD_DISABLE_ACT_SRC もしくは DTC_CMD_SEQUENCE_TRANSFER_ABORT の場合、R_DTC_Control()関数を呼び出す前に、ユーザは割り込み要求許可ビット（IERm.IENj）をクリアし、処理対象の割り込み要求を禁止にする必要があります（割り込み要因は R_DTC_Control()に渡されます）。

```
ICU.IER[m].BIT.IENj = 0;
```

R_DTC_Control()の処理終了後に、禁止にした割り込み要求を許可します。

IERm.IENj ビットと割り込み要因の対応については、ユーザーズマニュアル ハードウェア編の割り込みコントローラ（ICU）章の「割り込みベクタテーブル」をご覧ください。

Abort 処理では元の転送情報は壊れてしまうため、転送中断後に再度チェーン転送情報を作成する必要があります。

DTC_CMD_CHANGING_DATA_FORCIBLY_SET で無効な値を設定しようとした場合、R_DTC_Control()は DTC_ERR_INVALID_COMMAND を返します。

無効な値を検出する前に、R_DTC_Control()は既にいくつかのレジスタを更新した可能性があります。この動作が発生するのは、ユーザが無効な値を指定して FORCIBLY DTC を変更しようとした場合のみです。

R_DTC_GetVersion()

この関数は、本モジュールのバージョン番号を返します。

Format

uint32_t R_DTC_GetVersion (void)

Parameters

なし

Return Values

バージョン番号

最上位の 2 バイトがメジャーバージョン番号、最下位の 2 バイトがマイナーバージョン番号

Properties

ファイル r_dtc_rx_if.h にプロトタイプ宣言されています。

Description

この関数は本モジュールのバージョンを返します。

Example

```
uint32_t version;
version = R_DTC_GetVersion();
```

Special Notes:

なし

4. 端子設定

DTC FIT モジュールはピン設定を使用しません。

5. 付録

5.1　動作確認環境

DTC FIT モジュールの動作確認環境を以下に示します。

表 5.1　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 V7.3.0
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.3.01.00
	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
エンディアンの順序	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.2.20
使用ボード	Renesas Starter Kit for RX72T（型名：RTK5572TKCCxxxxxBE）

表 5.2　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 e2 studio V7.0.0
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.3.00.00
	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
エンディアンの順序	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.2.10
使用ボード	Renesas Starter Kit for RX111 （型名：R0K505111SxxxBE） Renesas Starter Kit for RX113 （型名：R0K505113SxxxBE） Renesas Starter Kit for RX130 （型名：RTK5005130SxxxxxBE） Renesas Starter Kit for RX130-512KB （型名：RTK5051308SxxxxxBE） Renesas Starter Kit for RX231 （型名：R0K505231SxxxBE） Renesas Starter Kit for RX23T （型名：RTK500523TSxxxxxBE） Renesas Starter Kit for RX24T （型名：RTK500524TSxxxxxBE） Renesas Starter Kit for RX24U （型名：RTK500524USxxxxxBE） Renesas Starter Kit for RX64M （型名：R0K50564MSxxxBE） Renesas Starter Kit for RX71M （型名：R0K50571MSxxxBE） 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）

表 5.3　動作確認環境 (Rev.2.08)

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.2.08
使用ボード	Renesas Starter Kit for RX111 （型名：R0K505111SxxxBE） Renesas Starter Kit for RX113 （型名：R0K505113SxxxBE） Renesas Starter Kit for RX130 （型名：RTK5005130SxxxxxBE） Renesas Starter Kit for RX130-512KB （型名：RTK5051308SxxxxxBE） Renesas Starter Kit for RX231 （型名：R0K505231SxxxBE） Renesas Starter Kit for RX23T （型名：RTK500523TSxxxxxBE） Renesas Starter Kit for RX24T （型名：RTK500524TSxxxxxBE） Renesas Starter Kit for RX24U （型名：RTK500524USxxxxxBE） Renesas Starter Kit for RX64M （型名：R0K50564MSxxxBE） Renesas Starter Kit for RX71M （型名：R0K50571MSxxxBE） Renesas Starter Kit for RX65N （型名：RTK500565NSxxxxxBE） Renesas Starter Kit for RX65N-2MB （型名：RTK50565N2SxxxxxBE）

5.2　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_dtc_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。

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

（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

本モジュールは該当するテクニカルアップデートはありません。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://japan.renesas.com

お問合せ先

http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.02	2015.04.01	—	初版発行
2.03	2015.06.15	1	対象デバイスに、RX230 と RX231 を追加
		10	1.2.2 動作環境とメモリサイズ　(5)RX231 の場合を追加。
		17	3.2 R_DTC_Close() Description において、“DMAC のすべてのチャネルのロックが解除される場合”を“DMAC のすべてのチャネルのロックが解除されていた場合”に変更
		25	3.3 R_DTC_Create() Example　処理 3: 複数要因の登録を行う場合　を追加
2.04	2015.12.29	1	対象デバイスに、RX130、RX23T、RX24T を追加
		3	1. 概要 以下の内容を変更した。 「DTC は割り込み要因の　~　を作成する必要があります。」
		13	2.6 コンパイル時の設定　#define DTC_CFG_SHORT_ADDRESS_MODE 元は”ADDRRESS”であった。
		14	2.7 引数 「/* Short-address mode */」、「/* Full-address mode */」を累加
		15	2.9 FIT モジュールの追加方法　を更新した。
		19	3.3 R_DTC_Create()　Parameters #if (1 == DTC_CFG_SHORT_ADDRESS_MODE) 元は”ADDRRESS”であった。
		23	3.3 R_DTC_Create()　Example　処理 1 「uint8_t ien_bk;」を追加 「des_addr」　元は「dest_addr」であった。
		23	3.3 R_DTC_Create()　Example　処理 2 「uint32_t transfer_data[8]」　元は「uint32 transfer_data[8]」であった。 「uint8_t ien_bk;」を追加
		24	3.3 R_DTC_Create()　Example　処理 2 「des_addr」　元は「dest_addr」であった。（2 箇所）
		25	3.3 R_DTC_Create()　Example　処理 3 「uint8_t ien_bk;」を追加 「des_addr」　元は「dest_addr」であった。
		25	3.3 R_DTC_Create()　Example　処理 3 「des_addr」　元は「dest_addr」であった。
		29	3.4 R_DTC_Control()　Example 「uint8_t　interrupt_number;」を追加
2.05	2016.09.30	1	対象デバイスに、RX65N を追加
		3-4	1. 概要　シーケンス転送の内容を追加
		5	1.2.1 API の概要 表 1.1　「R_DTC_CreateSeq()関数」を追加
		11	1.2.2 動作環境とメモリサイズ　(6)RX65N の場合を追加
		13	2.1 ハードウェアの要求　「DTCb」を追加

		14 15 15-16 15 17 18 22 24 30 – 35 36 37 38 -40	2.6 コンパイル事の設定 表　「#define DTC_CFG_USE_SEQUENCE_TRANSFER」を追加 2.7 引数　「r_dtc_rx_target_if.h」を追加 2.7.1 r_dtc_rx_if.h、2.7.2 r_dtc_rx_target_if.h 元は 2.7 引数の内容であった。 2.7.1 r_dtc_rx_if.h　構造体 dtc_command_t に以下を追加 DTC_CMD_SEQUENCE_TRANSFER_ENABLE DTC_CMD_SEQUENCE_TRANSFER_DISABLE DTC_CMD_SEQUENCE_TRANSFER_ABORT 2.8 戻り値　「DTC_ERR_ACT」を追加した 3.1 R_DTC_Open()　Description　DTC インデックステーブルの内容を追加 3.3 R_DTC_Create()　データ構造体 dtc_transfer_data_cfg_t に DTCb の内容を追加 3.3 R_DTC_Create()　以下のデータ構造体を追加 dtc_write_back_t、dtc_sequence_end_t、 dtc_refer_index_table_t、dtc_disp_add_t 3.4 R_DTC_CreateSeq()　新規追加 3.5 R_DTC_Control() Return Values　DTC_ERR_ACT を追加 3.5 R_DTC_Control() Description　表を追加 3.5 R_DTC_Control() Example　内容を見直した
2.06	2017.01.31	11 21 - 22 30	1.2.2 動作環境とメモリサイズ　「(6)RX65N の場合」の表 1-12 と表 1-13 の情報を更新した。 3.3 R_DTC_Create() Parameters 説明を追加した。 3.4 R_DTC_CreateSeq() Parameters 説明を追加した。
2.07	2017.03.31	- 1 5 6 38	下記の章番号を変更した。 • 2.3 動作確認環境、2.8 コードサイズ、4.1 動作確認環境詳細：元は 1.2.2 章であった。 対象デバイスに、RX24U を追加 1.3 DTC IP バージョン　新規追加 1.4 関連アプリケーションノート　内容を見直した 4. 付録　新規追加
2.08	2017.07.31	— 1 7 32 – 36	下記の章のタイトルを変更した。 • 1.1 DTC FIT モジュールとは：元は 1.1 DTC FIT モジュールであった。 下記の章の本文を移動した。 • 1.2 DTC FIT モジュールの概要：元は 1. 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 2.3 動作確認環境であった。 • 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。 • 2.12 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 対象デバイスに、RX651 を追加 2.2 ソフトウェアの要求　「r_cgc_rx」を削除。 3.5 R_DTC_Control()　新規コマンド「DTC_CMD_CHANGING_DATA_FORCIBLY_SET」を追加
2.10	2018.09.28	1、5 8 40	RX66T のサポートを追加。 RX66T に対応するコードサイズを追加。 「5.1 動作確認環境」：Rev.2.10 に対応する表を追加。
2.11	2018.11.16	40	Renesas Starter Kit for RX66T 製品番号変更

2.20	2019.02.01	—	RX72T グループのサポートを追加。
		1	対象デバイスに RX72T グループを追加。
		5	「DTC IP バージョン」セクションに RX72T グループを追加。
		9	RX72T に対応するコードサイズを追加。
		16-39	各 API 関数で「Reentrant」の説明を削除。
		38	R_DTC_Control()関数の「Special Notes」を更新。
		46	「5. デモプロジェクト」を追加。
		47	Renesas Starter Kit+ for RX66T の型名を変更。。
		47	「6.1 動作確認環境」Rev.2.20 に対応する表を追加。

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. 未使用端子の処理

   【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

   【注意】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. リザーブアドレス（予約領域）のアクセス禁止

   【注意】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、将来の機能拡張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。

4. クロックについて

   【注意】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。

5. 製品間の相違について

   【注意】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、内部 ROM、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、ノイズ輻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)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口　http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

Colophon 6.0