RX Family

DTC Module Using Firmware Integration Technology

R01AN1819EJ0206
Rev.2.06
Jan 31, 2017

Introduction

This application note explains how to use the control software module for Data Transfer Controller (DTC) on RX Family MCUs. The module is the DTC control module using Firmware Integration Technology (FIT) and is referred to below as the DTC FIT module.

In systems where the DTC FIT module is used simultaneously with the DMA controller (DMAC), it is necessary to ensure that the DMAC control software does not enable the module stop state while the DTC is operating, because a shared bit is used as both the DMAC module stop setting bit and the DTC module stop setting bit.

Target Device

Supported microcontrollers

RX110 Group, RX111 Group, RX113 Group, RX130 Group
RX230 Group, RX231 Group, RX23T Group, RX24T Group
RX64M Group
RX71M Group

When applying the information in this application note to a microcontroller other than the above, modifications should be made as appropriate to match the specification of the microcontroller and careful evaluation performed.

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Module Using Firmware Integration Technology (R01AN1685EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826EJ)

Contents

1. Overview

The DTC FIT module supports 3 transfer modes:

- Normal transfer mode
- Repeat transfer mode
- Block transfer mode

Each mode can enable Chain transfer and Sequence transfer functionality or not. For additional details, see the "Data Transfer Controller" section of the User's Manual: Hardware.

The DTC is activated by interrupt requests from interrupt sources. The user should create transfer information corresponding to each activation source or many consecutive information elements in the case of chain transfers. Transfer information consists of a start address for source and destination and, configuration information controlling how the DTC will transfer the data. When the DTC in activated, it will read the corresponding Transfer information and start the transfer.

DTC reads start address of a Transfer data that belongs to a specified interrupt source in DTC Vector table. This Vector table is an array of 4 byte addresses and start address of Transfer data (n) that belong to interrupt source with vector number (n) will be stored at the row of table (element of array) having index (4 * n).

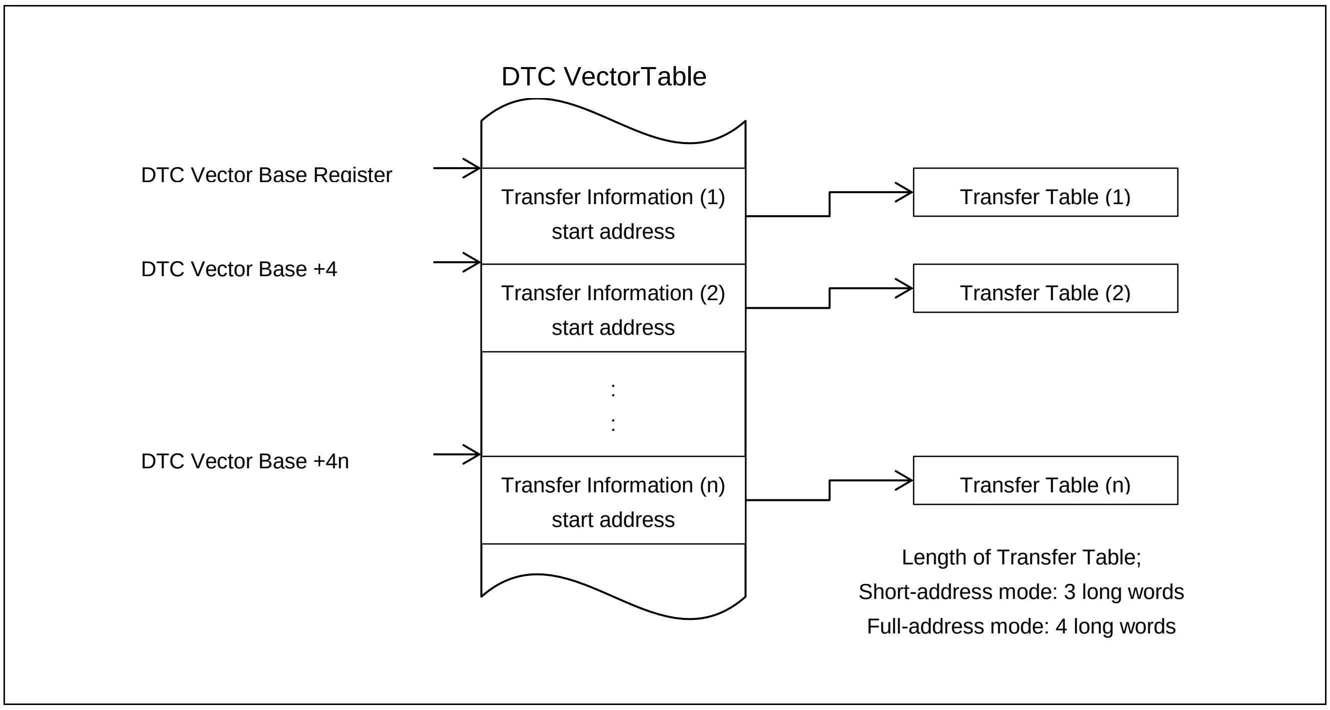
DTC VectorTable
DTC Vector Base Register
DTC Vector Base +4
DTC Vector Base +4n
Transfer Information (1) start address
Transfer Information (2) start address
:
:
Transfer Information (n) start address
Transfer Table (1)
Transfer Table (2)
Transfer Table (n)
Length of Transfer Table;
Short-address mode: 3 long words
Full-address mode: 4 long words

Figure 1.1 DTC Vector and Transfer Information

The user must allocate a memory space for DTC Vector table on RAM area before using DTC and the size (in byte units) of allocated memory depends on the maximum vector number value of interrupt sources supported by DTC and it is specified by equate DTC_VECTOR_TABLE_SIZE_BYTES defined in file r_dtc_rx_target.h for each MCU in "targets" folder; this default value is a value which supports all available activation source define in Interrupt Vector Table (For example, if it is RX111, it is 0x3E4 (0x3E4 = 249 * 4). if it is RX64M, it is 0x400 (0x400 = 256 *4).). The start address of DTC Vector table must be in 1-Kbyte units and user may also use the Linker to allocate Vector table at compilation time.

The DTC can work on 2 address modes: short mode and full mode. In short mode, the size of one Transfer data is 3 long words (12 bytes) and DTC can access to a 16-Mbyte memory space in the range 0x00000000 to 0x007FFFFF and 0xFF800000 to 0xFFFFFFFF. In full mode, the size of one Transfer data is 4 long words (16 bytes) and DTC can access to a 4-Gbyte memory space (0x00000000 to 0xFFFFFFFF).

By default, DTC will read Transfer data whenever an activation interrupt is raised. When there are 2 or many continuous active times just caused by an activation source, the user can skip the read process from the moment of second activation time to increase the performance of DTC because the content of Transfer data is already existed in DTC from the previous active time. To enable the Transfer Data Read Skip, the user can configure at initialization time by R_DTC_Open() or can use R_DTC_Control() with command DTC_CMD_DATA_READ_SKIP_ENABLE.

To initialize DTC, the R_DTC_Open() is called. This function will start supplying clock to DTC, and write the start address of DTC vector table to DTC Vector Base Register (DTCVBR). When using the sequence transfer, DTC index table address is written to DTC index table base register (DTCIBR). Also the function initializes the settings for Transfer Data Read Skip, DTC address mode and the DTCER registers corresponding to the configuration selections of user in r_dtc_rx_config.h.

The users shall provide configuration selections to R_DTC_Create() function to create Transfer data corresponding to a specific interrupt source. A Transfer data contains start address of source and destination and configuration information about how DTC will transfer data content from source to destination area. In R_DTC_Create(), the start address of Transfer data is stored in DTC vector table at the row according with the input vector number.

R_DTC_CreateSeq() creates the transfer information for sequence transfer and stores the start address of the transfer information at the specified location of the sequence number in DTC index table.

The R_DTC_Control() is used to select (or deselect) an interrupt as a DTC activation source, start or stop supplying clock to DTC, enable or disable Transfer Data Read Skip, abort the current chain transfer process, and enable or disable or abort the sequence transfer.

DTC is active when the activation source raises an interrupt. It will read the Transfer data corresponding to the vector number of activation interrupt to self-configure, and then transfer the data. Users can also use R_DTC_Control() to get the current status of DTC: whether DTC is in progress, the vector number of current active interrupt. The driver also support aborting the current Chain transfer process and sequence transfer process via R_DTC_Control() function.

Usage Conditions of DTC FIT Module

The usage conditions of the module are as follows.

- The r_bsp default lock function must be used.
- A single common bit must be used as the DMAC module stop setting bit and the DTC module stop setting bit.

1.1 DTC FIT Module

The DTC FIT module can be combined with other FIT modules for easy integration into the target system.

The functions of DTC FIT module can be incorporated into software programs by means of APIs. For information on incorporating the DTC FIT module into projects, see 2.9, "Adding Driver to Your Project".

1.2 Overview and Memory Size of APIs

1.2.1 Overview of APIs

Table 1.1 lists the API functions of DTC FIT module.

Table 1.1 API Functions

Function Name	Description
R_DTC_Open()	Initialization Processing
R_DTC_Close()	End Processing
R_DTC_Create()	Register and Activation Source Setting Processing
R_DTC_CreateSeq()	Register and Activation Source Setting Processing for sequence transfer
R_DTC_Control()	Operation Setting Processing
R_DTC_GetVersion()	Version Information Acquisition Processing

1.2.2 Operating Environment and Memory Size

The confirmed operating conditions and the required memory sizes for DTC FIT Module are list.

The memory sizes listed apply when the default settings listed in 2.6 “Compile Settings”, are used. The memory sizes differ according to the definitions selected.

(1) RX64M

Table 1.2 Operation Confirmation Conditions

Item	Contents
MCU used	RX64M Group (program ROM: 4 MB, RAM: 512 KB)
Operating frequency	ICLK: 120 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.0.1.08
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.00
Board used	R0K50564MSxxxBE (Renesas Starter Kit for RX64M)

Table 1.3 Required Memory Sizes

Memory	Size	Remarks
ROM	1,657 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	52 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table using the malloc() function.

(2) RX111

Table 1.4 Operation Confirmation Conditions

Item	Contents
MCU used	RX111 Group (program ROM: 128 KB, RAM: 16 KB)
Operating frequency	ICLK: 32 MHz, PCLKB: 32 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.0.1.08
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.00
Board used	R0K505111SxxxBE (Renesas Starter Kit for RX111)

Table 1.5 Required Memory Sizes

Memory	Size	Remarks
ROM	938 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	48 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table using the malloc() function.

(3) RX113

Table 1.6 Operation Confirmation Conditions

Item	Contents
MCU used	RX113 Group (program ROM: 512 KB, RAM: 64 KB)
Operating frequency	ICLK: 32 MHz, PCLKB: 32 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.1.1.08
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.01
Board used	R0K505113SxxxBE (Renesas Starter Kit for RX113)

Table 1.7 Required Memory Sizes

Memory	Size	Remarks
ROM	1,084 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	52 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table using the malloc() function.

(4) RX71M

Table 1.8 Operation Confirmation Conditions

Item	Contents
MCU used	RX71M Group (program ROM: 4 MB, RAM: 512 KB)
Operating frequency	ICLK: 240 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.1.2.09
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.01.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.02
Board used	R0K50571MSxxxBE (Renesas Starter Kit for RX71M)

Table 1.9 Required Memory Sizes

Memory	Size	Remarks
ROM	1,648 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	52 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table using the malloc() function.

(5) RX231

Table 1.10 Operation Confirmation Conditions

Item	Contents
MCU used	RX231 Group (program ROM: 512 KB, RAM: 64 KB)
Operating frequency	ICLK: 32 MHz, PCLKB: 16 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V3.1.3.06
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.02.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.03
Board used	R0K505231CxxxBE (Renesas Starter Kit for RX231)

Table 1.11 Required Memory Sizes

Memory	Size	Remarks
ROM	1,212 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	52 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table using the malloc() function.

(6) RX65N

Table 1.12 Operation Confirmation Conditions

Item	Contents
MCU used	RX65N Group (program ROM: 1 MB, RAM: 256 KB)
Operating frequency	ICLK: 120 MHz, PCLKB: 60 MHz
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics e^2 studio V5.2.0.020
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.05.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big endian/Little endian
Module version	Ver. 2.06
Board used	RTK500565NSxxxxxBE (Renesas Starter Kit for RX65N)

Table 1.13 Required Memory Sizes (Note 1)

Memory	Size	Remarks
ROM	1,759 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting
RAM	9 bytes (Little endian)	- Under confirmation conditions listed above - r_dtc_rx_config.h : default setting - Excluding a Vector Table (Note 2)
Max. user stack	56 bytes	
Max. interrupt stack	—	

Note 1: The required memory sizes differ according to the C compiler version and the compile conditions. The above memory sizes also differ according to endian mode.

Note 2: For Vector Table Size, refer to #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h. The DTC FIT module secures the memory required for the DTC Vector table and the DTC index table using the malloc() function. Note, however, that for #define DTC_CFG_USE_SEQUENCE_TRANSFER (DTC_DISABLE), the memory required for DTC index table is not secured.

1.3 Related Application Note

The application note that is related to the DTC FIT module is listed below. Reference should also be made to this application note.

- RX Family DMA Controller DMACA Control Module Using Firmware Integration Technology (R01AN2063EJ)

As for usage of DTC FIT module, refer to sample program in the following application note:

- RX Family DTC Module Sequence Transfer Application Example Firmware Integration Technology (R01AN3434EJ)
- RX Family RSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology (R01AN1914EJ)
- RX Family QSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology (R01AN1940EJ)
- RX Family SCIFA Clock Synchronous Single Master Control Module Using Firmware Integration Technology (R01AN2280EJ)

2. API Information

The names of the APIs of the DTC FIT module follow the Renesas API naming standard.

2.1 Hardware Requirements

The microcontroller used must support the following functionality.

- DTC (DTCa or DTCb)
- ICU
  The DTC FIT module shall modify the DTCER registers in ICU module to select an Interrupt source as a DTC activation source.

2.2 Software Requirements

The DTC FIT module is dependent on the following packages.

- r_bsp
- r_cgc_rx (only if CGC is necessary)

2.3 Supported Toolchain

The operation of the DTC FIT module has been confirmed with the toolchain listed in 1.2.2.

2.4 Header Files

All the API calls and interface definitions used are listed in r_dtc_rx_if.h.

Compile time configurable options are located in r_dtc_rx_config.h. Both of these files should be included by the User's application. And r_dtc_rx_target.h file should be included by User's application, when allocating a memory space for DTC Vector table on RAM area using DTC_VECTOR_TABLE_SIZE_BYTES definition.

2.5 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.6 Compile Settings

The configuration option settings for the DTC FIT module are specified in r_dtc_rx_config.h.

The option names and setting values are described below.

Configuration options in *r_dtc_rx_config.h*	
#define DTC_CFG_PARAM_CHECKING_ENABLE Note: The default value is the value of BSP_CFG_PARAM_CHECKING_ENABLE in the r_bsp_config.h file.	SPECIFY WHETHER TO INCLUDE CODE FOR API PARAMETER CHECKING 0: Compiles out parameter checking. 1: Includes parameter checking. Default value is set to BSP_CFG_PARAM_CHECKING_ENABLE to re-use the system default setting.
#define DTC_CFG_DISABLE_ALL_ACT_SOURCE Note: The default value is "DTC_ENABLE".	SPECIFY WHETHER THE DTCER REGISTERS WILL BE CLEARED IN R_DTC_OPEN() DTC_DISABLE: Do nothing. DTC_ENABLE: Clear all DTCER registers in R_DTC_Open().
#define DTC_CFG_SHORT_ADDRESS_MODE Note: The default value is "DTC_DISABLE".	SPECIFY WHICH ADDRESS MODE IS SUPPORTED BY DTC DTC_DISABLE: Select the Full-address mode. DTC_ENABLE: Select the Short-address mode.
#define DTC_CFG_TRANSFER_DATA_READ_SKIP_EN Note: The default value is "DTC_ENABLE".	SPECIFY WHETHER THE TRANSFER DATA READ SKIP IS ENABLED DTC_DISABLE: Disable Transfer Data Read Skip. DTC_ENABLE: Enable Transfer Data Read Skip.
#define DTC_CFG_USE_DMAC_FIT_MODULE Note: The default value is "DTC_ENABLE".	SPECIFY WHETHER THE DMAC FIT MODULE IS USED WITH DTC FIT MODULE DTC_DISABLE: DMAC FIT module is not used with DTC FIT module. DTC_ENABLE: DMAC FIT module is used with DTC FIT module. When DMAC FIT module is not used and "DTC_ENABLE" is set, the compiling error will be generated.
#define DTC_CFG_USE_SEQUENCE_TRANSFER Note: The default value is "DTC_DISABLE".	SPECIFY WHETHER THE SEQUENCE TRANSFER IS USED. DTC_DISABLE: SEQUENCE TRANSFER is not used. DTC_ENABLE: SEQUENCE TRANSFER is used. When defined as "DTC_ENABLE", set DTC_CFG_SHORT_ADDRESS_MODE to "DTC_DISABLE". When defined both this definition and DTC_CFG_SHORT_ADDRESS_MODE as "DTC_ENABLE", the compiling error will be generated. When defined as "DTC_ENABLE" for the MCU not supporting sequence transfer, the compiling error will be generated as well.

2.7 Arguments

The structure for the arguments of the API functions is shown below. This structure is listed in r_dtc_rx_if.h, and r_dtc_rx_target_if.h along with the prototype declarations of the API functions.

2.7.1 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;
```

2.7.2 r_dtc_rx_target_if.h

dtc_transfer_data_cfg_t has different definition according to DTC IP Version.

1. DTCa

```
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
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;
```

2.8 Return Values

The API function return values are shown below. This enumerated type is listed in r_dtc_rx_if.h, along with the prototype declarations of the API functions.

```
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;
```

2.9 Adding FIT Module to Your Project

This module must be added to each project in the e^2 Studio.

There are two methods for adding to a project: using the FIT plug-in and adding manually.

When the FIT plug-in is used, FIT modules can be added to projects easily and the include file path will be updated automatically. Therefore we recommend using the FIT plug-in when adding FIT modules to a project.

There are the following methods to add FIT module using FIT plug in.

1. Use "FIT Configurator".
   This is the latest method that the plug-in function such as Lib file path automatic setting is enhanced, which we recommend the use.

   For the procedure, refer to “4.3.2 Install the FIT Modules with the FIT Plugin.” in “RX64M/RX71M Group RX Driver Package Ver.1.02 (R01AN2606EJ)” application note.

2. Use the existing ”FIT plug-in”.
   For the procedure, refer to “3. Adding FIT Modules to e2 studio Projects Using FIT Plug-In” in “Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)” application note.

3. API Functions

3.1 R_DTC_Open()

This function is run first when using the APIs of the DTC FIT module.

Format

```
dtc_err_t   R_DTC_Open(
    void
)
```

Parameters

None

Return Values

```
DTC_SUCCESS:                /* Successful operation */
DTC_ERR_OPENED:             /* DTC has been initialized already. */
DTC_ERR_BUSY:               /* Resource has been locked by other process. */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

Locks*[1] the DTC and starts supplying clock to DTC, then initializes DTC vector table, address mode, Data Transfer Read Skip. When setting DTC_CFG_DISABLE_ALL_ACT_SOURCE to DTC_ENABLE in r_dtc_rx_config.h, all DTCER registers are cleared. When setting DTC_CFG_USE_SEQUENCE_TRANSFER to DTC_ENABLE, the area used in DTC index table is secured.

Note: 1. The DTC FIT module uses the r_bsp default lock function. As a result, the DTC is in the locked state after a successful end.

Reentrant

Function shall protect the code accessing to global variables and DTC registers by hardware lock BSP_LOCK_DTC supported by BSP module.

Example

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

Special Notes:

Set #define BSP_CFG_HEAP_BYTES in r_bsp_config.h to the value greater than #define DTC_VECTOR_TABLE_SIZE_BYTES in r_dtc_rx_target.h.

This is to secure the DTC Vector table area using the malloc() function in the DTC FIT module.

3.2 R_DTC_Close()

This function is used to release the resources of the DTC.

Format

```
dtc_err_t R_DTC_Close(
    void
)
```

Parameters

None

Return Values

```
DTC_SUCCESS:                /* Successful operation */
DTC_SUCCESS_DMAC_BUSY       /* Successful operation.
                                   One or some DMAC resources are locked. */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

Unlocks*[1] the DTC and disable all DTC activation source by clearing the DTC Activation Enable Register DTCERn; stop supplying clock to DTC and put it to Module stop state.

If in addition all DMAC channels have been unlocked, the function sets the DMAC and DTC to the module stop state.*[2]

Note: 1. The DTC FIT module uses the r_bsp default lock function. As a result, the DTC is in the unlocked state after a successful end.
2. Because a shared bit is used as both the DMAC module stop setting bit and the DTC module stop setting bit, the function confirms that all DMAC channels are unlocked before making the module stop setting. (For details, see the "Low Power Consumption" section in the User's Manual: Hardware.

Change the processing method to match the combination of modules used, as shown below.

DMAC Control	DTC Control	Processing Method
DMACA FIT module (lock function control function present, DTC lock state checking function present)	DTC FIT module (lock function control function present, DMAC lock state checking function present)	See case 1.
Other than the above		See case 2.

Case 1: Using the r_bsp Default Lock Function and Controlling the DMAC with the DMAC FIT Module*1

The function uses the r_bsp default lock function to confirm that all DMAC channels are unlocked and that the DTC is unlocked, then puts the DTC into the module stop state.

Note: 1. A necessary condition is that the DMAC FIT module has a module stop control function that confirms the locked state of the DTC.

Case 2: Control Other Than the Above

The user must provide code to confirm that all DMAC channels are unlocked and that the DTC is unlocked (not in use). The DTC FIT module includes an empty function for this purpose.

If the r_bsp default lock function is not used, insert the program code for checking the locked/unlocked state of all the DMAC channels and the DTC after the line marked /* do something */ in the r_dtc_check_DMAC_locking_byUSER() function in the file r_dtc_rx_target.c.

Note that bool type shown below should be used for the return value of the r_dtc_check_DMAC_locking_byUSER() function.

Returns value of r_dtc_check_DMAC_locking_byUSER()

```
 true                          /* All DMAC channels are unlocked. */
 false                         /* One or some DMAC channels are locked. */
```

Reentrant

Function shall protect the code accessing to global variables and DTC registers by hardware lock BSP_LOCK_DTC supported by BSP module.

Example

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

Special Notes:

When controlling the DMAC without using the DMAC FIT module, make sure to monitor the usage of the DMAC and control locking and unlocking of the DMAC so that calling this function does not set the DMAC to the module stop state. Note that even if the DMAC has not been activated, it is necessary to keep it in the locked state when not making DMAC transfer settings.

3.3 R_DTC_Create()

This function is used to make DTC register settings and to specify the activation source.

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

act_source

Activation source

** p_transfer_data*

Pointer to start address of Transfer data area on RAM

** p_data_cfg*

Pointer to settings for Transfer data

In the case of DTCb, the setting to the following structure members is invalid. This function sets the following values.

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;

chain_transfer_nr

Number of chain transfer

The number of Transfer data and corresponding configurations is (number of chain transfer + 1). Example: if chain_transfer_nr = 1, it means that there are 2 continuous Transfer data and 2 corresponding configurations and the first configuration enable the chain transfer.

The type definition of a Transfer data (* p_transfer_data) depends on the address mode and the details are shown as below and the users will use this data type to allocate memory for Transfer data exactly:

```
#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
```

The type of “Pointer to settings for Transfer data(* p_data_cfg)” is different by the DTC IP version. The data structure of configuration is below:

1. DTCa

```
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

```
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;
```

The following enumerate definitions indicate configurable options for above structures:

```
/* 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;
```

The transfer_count is set from 1 to 65536 in Normal transfer mode and Block transfer mode, from 1 to 256 in Repeat transfer mode.

The block_size value is set from 1 to 256 in Block transfer mode.

In short address mode, the start address of Transfer data (second argument), source area and destination area is in range (0x00000000 to 0x007FFFFF and 0xFF800000 to 0xFFFFFFFF).

Return Values

```
DTC_SUCCESS                 /* Successful operation */
DTC_ERR_NOT_OPEN            /* DTC is not initialized yet. */
DTC_ERR_INVALID_ARG         /* Parameters are invalid. */
DTC_ERR_NULL_PTR            /* Argument pointers are NULL. */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

Writes the configuration to Transfer data.

Writes the start address of Transfer data corresponding to interrupt number into DTC vector table.

Reentrant

Function shall protect the code accessing to global variables and DTC registers by hardware lock BSP_LOCK_DTC supported by BSP module.

Example

Case 1: In the case of No chain transfer

```
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;
```

Case 2: In the case of ONE chain transfer

```
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;
```

Case 3: In the case of multiple source registration

```
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:

Before calling R_DTC_Create(), user must disable the current interrupt request (the interrupt source is passed to R_DTC_Create()) by clearing Interrupt Request Enable bit IERm.IENj:

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

Then, enable the interrupt request disabled after R_DTC_Create() is ended.

The correspondence between IERm.IENj bit and interrupt source is described in Interrupt Vector Table, chapter Interrupt Controller (ICU) of User's Manual: Hardware.

3.4 R_DTC_CreateSeq()

This function performs the setting of the DTC register used in the sequence transfer and the activation source.

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

act_source

Activation source

** p_transfer_data*

Pointer to the start address in the transfer information area in RAM.

** p_data_cfg*

Pointer to the transfer information setting

Set the following structure members.

p_data_cfg->writeback_disable

p_data_cfg->sequence_end

p_data_cfg->refer_index_table_enable

p_data_cfg->disp_add_enable

sequence_transfer_nr

Transfer information counts per sequence transfer (0 - 4294967295)

sequence_transfer_nr	Description
0	When transfer request for the sequence number (sequence_no) specified is generated, the setting is made to output CPU interrupt request without starting the sequence.
1 - 4294967295	When transfer request for the sequence number (sequence_no) specified is generated, the transfer information for the sequence transfer is set. Prepare transfer information about the number to be specified sequence_transfer_nr in advance, and set the start address of the transfer information to *p_data_cfg.

sequence_no

Sequence number (0 - 255)

The type definition of the transfer information and the data structure are the same as R_DTC_Create(). Total of 256 ways of the sequence information can be set.

Return Values

```
DTC_SUCCESS                 /* Successful operation */
DTC_ERR_NOT_OPEN            /* DTC is not initialized yet. */
DTC_ERR_INVALID_ARG         /* Arguments are invalid. */
DTC_ERR_NULL_PTR            /* Argument pointers are NULL. */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

This function writes the setting information to the transfer information.

Start address of the transfer information for the sequence number is written to DTC index table.

Reentrant

Function shall protect the code accessing to global variables and DTC registers by hardware lock BSP_LOCK_DTC supported by BSP module.

Example

Examples of asynchronous serial receiving by the sequence transfer based on the Receive FIFO Full Interrupt (RXI) as DTC activation source is explained as follows. SCI used is Channel 10. Sequence transfer is automatically started according to 1 bit data (cmnd) received first from external communication device.

Case 1:

After receiving cmnd= ”00h” from external communication device, SCI10 receive FIFO threshold is changed to 4 bytes, then, 4 bytes data output from external communication device is received, and is stored in the RAM by DTC transfer.

Table 3-1 Transfer information specified in Case 1

Member	Transfer information 1	Transfer information 2	Transfer information 3
transfer_mode	Normal transfer	Block transfer	Normal transfer
data_size	8 bits	16 bits	8 bits
src_addr_mode	Fix source address	Fix source address	Fix source address
chain_transfer_enable	Disable chain transfer	Enable chain transfer	Disable chain transfer
chain_transfer_mode	Perform chain transfer continuously (setting disabled)	Perform chain transfer continuously	Perform chain transfer continuously (setting disabled)
response_interrupt	Generate interrupt after the specified data transfer is complete.	Generate interrupt after the specified data transfer is complete.	Generate interrupt after the specified data transfer is complete.
repeat_block_side	Destination is repeat or block area (setting disabled)	Destination is repeat or block area	Destination is repeat or block area (setting disabled)
dest_addr_mode	Fix destination address	Increment destination Address per transfer	Fix destination address
source_addr	ROM dtc_fcrh_data[0] Address	SCI10.FRDR register address	ROM g_dtc_fcrh_cmnd address
dest_addr	SCI10.FCR.H register Address	RAM g_dtc_rx_buf0[0] address	SCI10.FCR.H register address
transfer_count	1	1	1
block_size	(Setting disabled)	4	(Setting disabled)
writeback_disable	No write back	No write back	No write back
sequence_end	Continue sequence transfer	Continue sequence transfer	End sequence transfer
refer_index_table_enable	Not refer to index table	Not refer to index table	Not refer to index table
disp_add_enable	Not add Displacement value to the source address	Not add Displacement value to the source address	Not add Displacement value to the source address

```
#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;
}
```

Case 2 :

When receiving cmnd >= “04h” from external communication device, generate the interrupt to CPU without sequence transfer.

```
#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:

Before calling R_DTC_CreateSeq(), user must disable the current interrupt request (the interrupt source is passed to R_DTC_CreateSeq()) by clearing Interrupt Request Enable bit (IERm.IENj):

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

Then, enable the interrupt request disabled after R_DTC_CreateSeq() is ended.

The correspondence between IERm.IENj bit and interrupt source is described in Interrupt Vector Table, chapter Interrupt Controller (ICU) of User’s Manual: Hardware.

3.5 R_DTC_Control()

This function controls the operation of the DTC.

Format

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

Parameters

command	*DTC control command*
** p_stat*	*Pointer to the status when command is DTC_CMD_STATUS_GET.*

Member of dtc_stat_t Structure

Member	Short Description	Setting Value	Setting Details
vect_nr	DTC-Activating Vector Number	Vector Number Monitoring	The value is only valid when DTC transfer is in progress (the value of the DTC Active Flag is 1).
in_progress	DTC Active Flag	- false - true	- DTC transfer operation is not in progress. - DTC transfer operation is in progress.

** p_args*	*Pointer to the argument structure when command is DTC_CMD_ACT_SRC_ENABLE, DTC_CMD_ACT_SRC_DISABLE, DTC_CMD_CHAIN_TRANSFER_ABORT, or DTC_CMD_SEQUENCE_TRANSFER_ENABLE.*

Members of dtc_cmd_arg_t Structure

Member	Short Description	Setting Details
act_src	DTC-Activating Vector Number	The value is only valid when command is DTC_CMD_ACT_SRC_ENABLE or DTC_CMD_ACT_SRC_DISABLE or DTC_CMD_SEQUENCE_TRANSFER_ENABLE
chain_transfer_nr	Number of chain transfer (Note)	The value is only valid when command is DTC_CMD_CHAIN_TRANSFER_ABORT.

Note: Set the value as same as the argument chain_transfer_nr when user call R_DTC_Create() before.

Return Values

```
DTC_SUCCESS                   /* Successful operation */
DTC_ERR_NOT_OPEN              /* DTC is not initialized yet. */
DTC_ERR_INVALID_COMMAND       /* Command parameters are invalid. */
DTC_ERR_NULL_PTR              /* Argument pointers are NULL. */
DTC_ERR_ACT                   /* Data transfer is in progress.  */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

Processing is performed depending on the command.

Command	**Arguments** dtc_stat_t *	**Arguments** dtc_cmd_arg_t *	**Description**
DTC_CMD_DTC_START	NULL	NULL	Starts DTC module using DTC Module Start (DTCST) bit.
DTC_CMD_DTC_STOP	NULL	NULL	Stops DTC module using DTC Module Start (DTCST) bit.
DTC_CMD_DATA_READ_SKIP_ENABLE	NULL	NULL	Enables Transfer Data Read Skip using DTC Transfer Information Read Skip Enable (RRS) bit.
DTC_CMD_DATA_READ_SKIP_DISABLE	NULL	NULL	Disables Transfer Data Read Skip using DTC Transfer Information Read Skip Enable (RRS) bit.
DTC_CMD_ACT_SRC_ENABLE	NULL	p_args->act_src	Sets an interrupt source using DTC Start Enable (DTCE) bit.
DTC_CMD_ACT_SRC_DISABLE	NULL	p_args->act_src	Clears an interrupt source using DTC Start Enable (DTCE) bit.
DTC_CMD_STATUS_GET	p_stat->in_progress p_stat->vect_nr	NULL	Gets a DTC Active Flag (ACT) and vector number (VECN[7:0]) using DTC Status Register (DTCSTS).
DTC_CMD_CHAIN_TRANSFER_ABORT	NULL	p_args-> chain_transfer_nr	Aborts the current active chain transfer.
DTC_CMD_SEQUENCE_TRANSFER_ENABLE	NULL	p_args->act_src	Specifies Sequence Transfer Vector number and enables Sequence transfer using DTC Sequence Transfer Enable Register (DTCSEQ).
DTC_CMD_SEQUENCE_TRANSFER_DISABLE	NULL	NULL	Disables Sequence Transfer using DTC Sequence Transfer Enable Register (DTCSEQ).
DTC_CMD_SEQUENCE_TRANSFER_ABORT	NULL	NULL	Aborts Sequence Transfer using Sequence Transfer End bit (SQTFRL).

Reentrant

Function shall protect the code accessing to global variables and DTC registers by hardware lock BSP_LOCK_DTC supported by BSP module.

Example

Case 1: Start DTC module

```
dtc_err_t ret;

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

Case 2: Stop DTC module

```
dtc_err_t ret;

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

Case 3 : Enable transfer information read skip

```
dtc_err_t ret;

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

Case 4: Disable transfer information read skip

```
dtc_err_t ret;

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

Case 5 : Using the DTCE, set the interrupt used for DTC activation source

```
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);
```

Case 6 : Using the DTCE, clear the interrupt used for DTC activation source

```
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);
```

Case 7 : Get DTC Active Flag (ACT) and Vector number (VECN[7:0]) in progress

```
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 */
}
```

Case 8 : Abort the chain transfer in process

```
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);
```

Case 9 : Enable the sequence transfer

```
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);
```

Case 10: Disable the sequence transfer

```
dtc_err_t ret;

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

Case 11: Abort the sequence transfer

```
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);
```

Special Notes:

When the command is DTC_CMD_GET_STATUS, the vector number is valid if only the DTC is in the progress (p_stat->in_progress is true).

With command DTC_CMD_ENABLE_ACT_SRC, DTC_CMD_DISABLE_ACT_SRC or DTC_CMD_SEQUENCE_TRANSFER_ABORT, before calling R_DTC_Control(), user must disable the current interrupt request (the interrupt source is passed to R_DTC_Control()) by clearing Interrupt Request Enable bit (IERm.IENj);

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

After processing of R_DTC_Control() is ended, the interrupt request disabled is enabled.

The correspondence between IERm.IENj bit and interrupt source is described in Interrupt Vector Table, chapter Interrupt Controller (ICU) of User’s Manual: Hardware.

With abort processing, user must re-create the Chain transfer data after the transfer is aborted because the old Transfer data are destroyed.

3.6 R_DTC_GetVersion()

This function is used to get the driver version information.

Format

```
uint32_t R_DTC_GetVersion(void)
```

Parameters

None

Return Values

Version number

Upper 2 bytes: major version, lower 2 bytes: minor version

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

Returns the version information.

Reentrant

Reentrant is possible.

Example

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

Special Notes:

None.

4. Reference Documents

User's Manual: Hardware
Technical Update/Technical News
User's Manual: Development Tools

The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

Revision History

Rev.	Date	Description	
		Page	Summary
2.02	2015.04.01	—	First edition issued
2.03	2015.06.15	1	-Added RX230 Group and RX231 Group In Target Device
		9	1.2.2 Operating Environment and Memory Size -Added (5)RX231
		18	3.2 R_DTC_Close() Description -Changed "If all DMAC channels are unlocked," to "If all DMAC channels have been unlocked,"
		27	3.3 R_DTC_Create() -Added Case 3: In the case of multiple source registration to Example
2.04	2015.12.29	1	-Added RX130 Group, RX23T Group, and RX24T Group In Target Device
		2	- Changed the explanation of 1. Overviews; "The DTC is activated by interrupt……start the transfer."
		13	2.6 Compile Settings #define DTC_CFG_SHORT_ADDRESS_MODE -Changed from "ADDRRESS"
		14	2.7 Arguments -Added /* Short-address mode */ and /* Full-address mode */
		16	-Updated 2.9 Adding Driver to Your Project
		20	3.3 R_DTC_Create() Parameters #if (1 == DTC_CFG_SHORT_ADDRESS_MODE) -Changed from "ADDRRESS"
		25	3.3 R_DTC_Create() Example Case 1 -Added uint8_t ien_bk; -Changed from dest_addr to des_addr
		26	3.3 R_DTC_Create() Example Case 2 -Changed from uint32 transfer_data[8] to uint32_t transfer_data[8] -Added uint8_t ien_bk;
		26	3.3 R_DTC_Create() Example Case 2 -Changed from dest_addr to des_addr (2 places)
		27	3.3 R_DTC_Create() Example Case 3 -Added uint8_t ien_bk; -Changed from dest_addr to des_addr
		27	3.3 R_DTC_Create() Example Case 3 -Changed from dest_addr to des_addr :
		30	3.4 R_DTC_Control() Example -Added uint8_t interrupt_number;
2.05	2016.09.30	1	-Added RX65N Group In Target Device
		2-3	-Added the contents of sequence transfer to 1. Overview
		4	1.2.1 Overview of API -Added "R_DTC_CreateSeq()" to Table 1.1
		10	1.2.2 Operating Environment and Memory Size -Added (6)RX65N
		12	2.1 Hardware Requirements -Added DTCb

		13	2.6 Compile Settings -Added “#define DTC_CFG_USE_SEQUENCE_TRANSFER to the table
		14	2.7 Arguments -Added r_dtc_rx_target_if.h
		14-15	- Divided the contents of 2.7 Arguments into 2.7.1 r_dtc_rx_if.h and 2.7.2 r_dtc_rx_target_if.h
		15	2.7.1 r_dtc_rx_if.h -Added Structure dtc_command_t to the followings; DTC_CMD_SEQUENCE_TRANSFER_ENABLE DTC_CMD_SEQUENCE_TRANSFER_DISABLE DTC_CMD_SEQUENCE_TRANSFER_ABORT
		16	2.8 Return Values - Added DTC_ERR_ACT
		16	2.9 Adding FIT Module to Your Project -Changed the title from Adding Driver to Your Project
		17	3.1 R_DTC_Open() -Added the contents of Description　DTC Index table
		21	3.3 R_DTC_Create() -Added the contents of DTCb to Data structure dtc_transfer_data_cfg_t
		23	3.3 R_DTC_Create() -Added the following data structure; dtc_write_back_t, dtc_sequence_end_t, dtc_refer_index_table_t, dtc_disp_add_t
		29 - 34	-Added 3.4 R_DTC_CreateSeq()
		35	3.5 R_DTC_Control() Return Values -Added DTC_ERR_ACT
		36	3.5 R_DTC_Control() Description -Added the table
		37 -39	3.5 R_DTC_Control() -Revised the contents of Example
2.06	2017.01.31	10	1.2.2 Operating Environment and Memory Size -Updated Table 1.12 and Table 1.13.
		20 - 21	3.3 R_DTC_Create() Parameters -Added the explanation.
		29	3.4 R_DTC_CreateSeq() Parameters -Added the explanation.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different type number, confirm that the change will not lead to problems.
   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

RENESAS

SALES OFFICES

Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0