RX Family

DTC Module Using Firmware Integration Technology

R01AN1819EJ0202
Rev.2.02
Dec 12, 2014

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
- 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 function or not. For details, see the “Data Transfer Controller” section of the User’s Manual: Hardware.

The DTC is active by interrupt signals from interrupt sources and for each specified interrupt source, user should create a Transfer data (or many continuous Transfer data elements for Chain transfer function) belong to it. A Transfer data contains start address of source and destination and configuration information about how DTC transfers data content from source to destination area. When DTC is active, it will read a Transfer data corresponding to that interrupt 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).

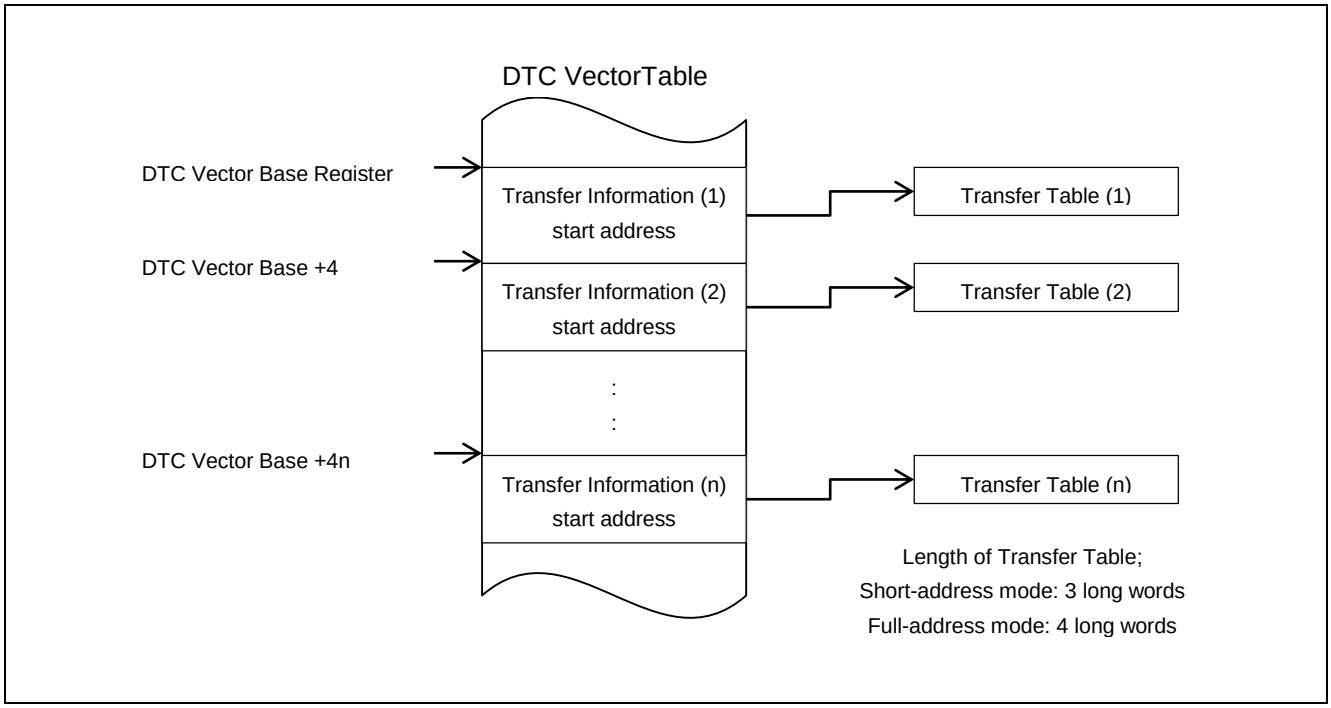

Figure 1.1 DTC Vector and Transfer Information

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, write the start address of DTC vector table to DTC Vector Base Register (DTCVBR), and initialize 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.

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 and abort the current chain transfer process.

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 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_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.0

(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 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 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 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 DTC Vector table using the malloc() function.

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 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)
- 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_reference.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_ADDRRESS_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.

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, along with the prototype declarations of the API functions.

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

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



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;
         /* The side being repeat or block in repeat / block transfer mode. */
    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 */
} dtc_transfer_data_cfg_t;



typedef enum e_dtc_command {
    DTC_CMD_DTC_START,          /* DTC will can 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_command_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_t;
```

2.9 Adding Driver to Your Project

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

You can use the FIT plug-in to add the FIT module to your project, or the module can be added manually.

It is recommended to use the FIT plug-in as you can add the module to your project easily and also it will automatically update the include file paths for you.

To add the FIT module using the plug-in, refer to chapter 2. “Adding FIT Modules to e^2 studio Projects Using FIT Plug-In” in the “Adding Firmware Integration Technology Modules to Projects” application note (R01AN1723EU).

To add the FIT module manually, refer to the following 2.9.1. “Adding the DTC FIT module (when not using the plug-in)”.

When using the FIT module, the BSP FIT module also needs to be added. For details on the BSP FIT module, refer to the “Board Support Package Module Using Firmware Integration Technology” application note (R01AN1685EU).

2.9.1 Adding the DTC FIT module (when not using the plug-in)

1. This application note is distributed with a zip file package that includes the DTC FIT module in its own folder r_dtc_rx.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the r_dtc_rx folder.
4. Open your e^{2}Studio workspace.
5. In the e^{2}Studio project explorer window, select the project that you want to add the DTC FIT module to.
6. Drag and drop the r_dtc_rx folder from the browser window into your e^{2}Studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
   a. Navigate to the "Add directory path" control:
      i. ‘project name’ → properties → C/C++ Build → Settings → Compiler → Source -Add (green + icon)
   b. Add the following paths:
      i. "${workspace_loc:/${ProjName}/r_dtc_rx}"
   Whether you used the plug-in or manually added the package to your project, it is necessary to configure the module for your application.
8. Locate the r_dtc_rx_config_reference.h file in the r_dtc_rx/ref source folder in your project and copy it to the r_config folder in your project.
9. Change the name of the copy in the r_config folder to r_dtc_rx_config.h.
10. Make the required configuration settings by editing the r_dtc_rx_config.h file. Refer to 2.6 “Compile Settings” for details.

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. Clears all DTCER register if sets the equate DTC_CFG_DISABLE_ALL_ACT_SOURCE to 1 in r_dtc_rx_config.h.

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.

Because the DTC FIT module secures DTC Vector table using the malloc() function.

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 are 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

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 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_ADDRRESS_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 data structure of configuration is below:

```
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 */
} 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;
```

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];

/* 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.des_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 No 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 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 */

/* 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].des_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].des_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;
```

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;
```

And restore the setting value for it 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_Control()

This function is used to control 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

Command	Description
DTC_CMD_DTC_START	Starts DTC module using DTC Module Start (DTCST) bit.
DTC_CMD_DTC_STOP	Stops DTC module using DTC Module Start (DTCST) bit.
DTC_CMD_DATA_READ_SKIP_ENABLE	Enables Transfer Data Read Skip using DTC Transfer Information Read Skip Enable (RRS) bit.
DTC_CMD_DATA_READ_SKIP_DISABLE	Disables Transfer Data Read Skip using DTC Transfer Information Read Skip Enable (RRS) bit.
DTC_CMD_ACT_SRC_ENABLE	Selects one interrupt as a DTC activation source using DTC Start Enable (DTCE) bit.
DTC_CMD_ACT_SRC_DISABLE	Removes one interrupt as a DTC activation source using DTC Start Enable (DTCE) bit.
DTC_CMD_STATUS_GET	Gets a DTC Active Flag and vector number using DTC Status Register (DTCSTS).
DTC_CMD_CHAIN_TRANSFER_ABORT	Aborts the current active chain transfer.

** p_stat*

Pointer to the status when command is DTC_CMD_STATUS_GET

Members 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	DTC transfer operation is not in progress.
		true	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 or DTC_CMD_CHAIN_TRANSFER_ABORT.

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.
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_BUSY:               /* Resource has been locked by other process. */
```

Properties

Prototype declarations are contained in r_dtc_rx_if.h.

Description

DTC_CMD_DTC_START command processing

Sets the DTC Module Start (DTCST) bit in order to start DTC hardware module.

DTC_CMD_DTC_STOP command processing

Clears the DTC Module Start (DTCST) bit in order to stop DTC hardware module.

DTC_CMD_DATA_READ_SKIP_ENABLE command processing

Sets the DTC Transfer Information Read Skip Enable (RRS) bit in order to enable Transfer Data Read Skip.

DTC_CMD_DATA_READ_SKIP_DISABLE command processing

Clears the DTC Transfer Information Read Skip Enable (RRS) bit in order to disable Transfer Data Read Skip.

DTC_CMD_ACT_SRC_ENABLE command processing

Selects one interrupt as a DTC activation source using DTC Start Enable (DTCE) bit in DTC Start Enable Register (ICU.DTCER).

DTC_CMD_ACT_SRC_DISABLE command processing

Removes one interrupt as a DTC activation source using DTC Start Enable (DTCE) bit in DTC Start Enable Register (ICU.DTCER).

DTC_CMD_STATUS_GET command processing

Gets a DTC Active Flag and vector number from DTC Status Register (DTCSTS).

DTC_CMD_CHAIN_TRANSFER_ABORT command processing

Aborts the current active chain transfer.

Reentrant

This function is used to start or stop supplying to DTC, clear a specified DTC Activation Enable Register n, enable or disable Transfer Data Read Skip, abort the current Chain transfer process and get current status of DTC.

Example

```
dtc_err_t ret;
dtc_stat_t dtc_status;

/* Stop supplying to DTC. */
ret = R_DTC_Control(DTC_CMD_DTC_STOP, 0, NULL, 0); /* Only the first argument
is used */

/* Get the current status */
ret = R_DTC_Control(DTC_CMD_STATUS_GET, 0, &dtc_status, 0);

if (true == dtc_status.in_progress)
{
/* the vector number is valid now. */
interrupt_number = dtc_status.vect_nr;
}
else /* The DTC is not in progress*/
{
/* do something */
}
/* Abort the current active chain transfer
 * The number of chain transfer is 5.
 */
ret = R_DTC_Control(DTC_CMD_CHAIN_TRANSFER_ABORT, 0, NULL, 5);
```

Special Notes:

When the command is DTC_CMD_GET_STATUS, the vector number is valid if only the DTC is in the progress (in_progress is true).
With command DTC_CMD_ENABLE_ACT_SRC and DTC_CMD_DISABLE_ACT_SRC, 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;
```

And restore the setting value for it after R_DTC_Control() 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.

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

3.5 R_DTC_GetVersion()

This function is used to fetch 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

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

Technical Update/Technical News

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

User's Manual: Development Tools

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: RX Build Rev.1.00 (R20UT02747EJ)

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: RX Coding Rev.1.00 (R20UT02748EJ)

[e^2 studio] RX Family C/C++ Compiler Package V.2.01.00 User's Manual: Message Rev.1.00 (R20UT02749EJ)

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

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Nov 15, 2013	—	First edition issued
2.00	Aug 29, 2014	1	-Added RX110 Group and RX64M Group In Target Device. -Added DMAC control software in Introduction. -Added ‘Related Documents’.
		2	-Changed Figure 1.1. -Added an example of RX64M for Vector Table size.
		3	-Added ‘Usage Conditions of DTC FIT module’.
		3-5	-Added ‘1.2 Overview and Memory Size of APIs’.
		6	-Added ‘1.3 Related Application Note’.
		7	-Removes ‘2.2 Hardware Resource Requirement’ and move the contents to ‘2.1 Hardware Requirement’. -Changed ‘DTC’ to ‘DTC (DTCa)’ in 2.1 Hardware Requirement. -Removed version number on r_bsp_package in 2.2 Software Requirements. -Added ‘r_cgc_rx (only if CGC is necessary) in 2.2 Software Requirements. -Removes ‘2.4 Limitations’. -Updated toolchain number in 2.3 Supported Toolchain. -Updated 2.4 Header Files and 2.5 Integer Types.
		8	-Updated 2.6 Compile Setting and added define ‘DTC_CFG_USE_DMAC_FIT_MODULE’. -Removed ‘2.9 API Data Structure’
		9	-Added ‘2.7 Arguments’.
		10	-Modified ‘2.9 Added Driver to Your Project’. -Added 2.9.1 Adding to DTC FIT module (When not using the plug-in)
		11	-Move Return Values form ‘3.2 Return Values’ to ‘2.8 Return Values’.
		12	-Removed ‘3.1 Summary’.
		12	3.1 R_DTC_Open() -Changed argument ‘dtc_vector_table’ to ‘void’ in R_DTC_Open(). -Changed Return value, Properties, Description and Example.
		13-14	3.2 R_DTC_Close() -Changed Return value, Properties, Description, Example and Special Notes.
		17-20	3.3 R_DTC_Create() -Changed Return value, Properties, Example and Special Notes.
		21-23	3.4 R_DTC_Control() -Changed Parameters, Return value, Properties, Descriptions, Reentrant, Example and Special Notes.
		24	3.5 R_DTC_GetVersion() -Changed Return value, Properties, Descriptions, Reentrant, Example and Special Notes.
		25	Added ‘4. Reference Documents’.
		—	As for DTC FIT Module Software, -Added RX110 Group, RX64M Group -Added targets folder and MCU folder. -Changed usage of r_dtc_acquire_hw_lock() and r_dtc_release_hw_lock() as follows.

			Hardware lock : R_DTC_Open() Hardware unlock : R_DTC_Close() -Changed argument ‘dtc_vector_table’ to ‘p_dtc_vector_table. in R_DTC_Open(). -Changed to get dtc_vector_table using malloc() in R_DTC_Open(). -Changed argument transfer_data to*p_transfer_data void in R_DTC_Create(). -Changed to enable module stop state after checking the hardware lock state of DMAC. -Modified Return Values of All functions.
2.01	Nov 28, 2014	1	-Added RX113 Group In Target Device.
		4	-Added Note 2 in Table 1.3.
		5	-Added Note 2 in Table 1.5.
		6	-Added (3) RX113 In 1.2.2 Operating Environment and Memory Size.
2.02	Dec 12, 2014	1	-Added RX71M Group In Target Device.
		1	-Added an application note (R01AN1826EJ) in Related Documents.
		4	-Changed type name of Board used in Table 1.2, (1) RX64M, 1.2.2 Operating Environment and Memory Size.
		5	-Changed type name of Board used in Table 1.4, (2) RX111, 1.2.2 Operating Environment and Memory Size.
		6	-Changed type name of Board used in Table 1.6, (3) RX113, 1.2.2 Operating Environment and Memory Size.
		7	-Added (4) RX71M In 1.2.2 Operating Environment and Memory Size.

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
3. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application for which it is not intended. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
11. This document may not be reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

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
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

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 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

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 906, 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 Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0