RX Family

CMT Module Using Firmware Integration Technology

Introduction

This application note describes the Compare Match Timer (CMT) module which uses Firmware Integration Technology (FIT). This module uses CMT to generate repetitive events and fixed time intervals. In this document, this module is referred to as the CMT FIT module.

This CMT FIT module supports the use of any CMT channel as the RTOS system timer (some devices only).

Target Devices

- RX110 Group
- RX111 Group
- RX113 Group
- RX130 Group
- RX230 Group
- RX231 Group
- RX23T Group
- RX24T Group
- RX24U Group
- RX64M Group (RTOS system timer supported)
- RX65N, RX651 Groups (RTOS system timer supported)
- RX66T Group
- RX71M Group (RTOS system timer supported)
- RX72T Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to “6.1 Confirmed Operation Environment".

Contents

1. Overview

1.1 CMT FIT Module

The CMT FIT module can be used by being implemented in a project as an API. See section 2.12, Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the CMT FIT Module

This software module provides a simple interface to the RX Compare Match Timer (CMT) peripheral. The CMT is a two-channel, 16-bit timer. Each channel contains a free-running counter with a prescaler and a 16-bit compare register. Interrupts can be generated when the free-running counter matches the compare register. The counter is automatically reset and restarted on a compare event making this an ideal timer for pacing repetitive software events like RTOS schedulers. A CMT unit contains two channels; RX MCUs contain either one or two CMT units resulting in either two or four independent CMT channels.

This driver provides functions for creating and starting a CMT channel, pausing and restarting a channel, and shutting down a channel. User application code can be called via a callback function.

1.3 Using the FIT CMT module

The primary use of the CMT module is to make it easy to generate repetitive events and fixed time intervals.

After adding the CMT module to your project you will need to modify the *r_cmt_rx_config.h* file to configure the software for your installation.

Use the functions R_CMT_CreatePeriodic and R_CMT_CreateOneShot to start a timer with unused CMT channels and default interrupt priority level. Use the functions R_CMT_CreatePeriodicAssignChannelPriority and R_CMT_CreateOneShotAssignChannelPriority to start a timer with desired CMT channels and desired interrupt priority level.

Provide a pointer to your callback function as an argument and your callback will be called when the timer expires. Be aware that during execution of your callback, interrupts will be disabled by default, since it is executing from within the context of the ISR. Therefore it is recommended to keep callback functions small so that they complete quickly.

In theory, the CMT timer maximum clocking speed is limited to PCLK/8. When using the periodic timer function to generate a clock, be aware that interrupt and callback processing takes some time. So this will limit the maximum frequency that can be generated.

To use a CMT channel as a RTOS system timer, configure #define BSP_CFG_RTOS_USED and #define BSP_CFG_RTOS_SYSTEM_TIMER in r_bsp_config.h. These two macros provide protection mechanism to the CMT channel so that it is used exclusively by the RTOS system timer.

1.4 API Overview

Table 1.1 lists the API functions included in this module.

Table 1.1 API Functions

Function	Description
R_CMT_CreatePeriodic()	Finds an unused CMT channel, configures the timer for the desired periodic frequency by setting the appropriate prescaler, associates a user callback function with the timer's interrupt, and starts the timer. The timer continues to run, generating an interrupt and calling the callback at the desired frequency, until the user shuts it down.
R_CMT_CreateOneShot()	Similar to R_CMT_CreatePeriodic; however, the timer is shut down after the first interrupt and callback.
R_CMT_CreatePeriodicAssignChannelPriority()	Similar to R_CMT_CreatePeriodic; however, the timer is configured with desired CMT channel & interrupt priority
R_CMT_CreateOneShotAssignChannelPriority()	Similar to R_CMT_CreateOneShot; however, the timer is configured with desired CMT channel & interrupt priority level
R_CMT_Control()	Commands the timer to pause, restart, change interrupt priority or report status.
R_CMT_Stop()	Turns off a CMT channel, disables interrupts, and powers down the CMT unit if it is not in use.
R_CMT_GetVersion()	Returns the driver version number at runtime.

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- CMT

2.2 Software Requirements

This driver is dependent upon the following FIT modules:

- Renesas Board Support Package (r_bsp) v5.20 or higher
- The peripheral clock must be initialized before starting the CMT.

2.3 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 6.1, Confirmed Operation Environment.

2.4 Interrupt Vector

The CMT interrupt is enabled by execution **R_CMT_CreatePeriodic** function, **R_CMT_CreateOneShot** function, **R_CMT_CreatePeriodicAssignChannelPriority** function and **R_CMT_CreateOneShotAssignChannelPriority** function.

Table 2.1 lists the interrupt vector used in the CMT FIT Module.

Table 2.1 Interrupt Vector Used in the CMT FIT Module

Device	Interrupt Vector
RX110[*1] RX111[*1] RX113 RX130[*1] RX230 RX231 RX23T RX24T RX24U RX66T RX72T	CMI0 interrupt[channel 0] (vector no.: 28) CMI1 interrupt[channel 1] (vector no.: 29) CMI2 interrupt[channel 2] (vector no.: 30) CMI3 interrupt[channel 3] (vector no.: 31)
RX64M RX651 RX65N RX71M	CMI0 interrupt[channel 0] (vector no.: 28) CMI1 interrupt[channel 1] (vector no.: 29) CMI2 interrupt[channel 2] (vector no.: 128) [*2] CMI3 interrupt[channel 3] (vector no.: 129) [*2]

Note 1. Only have 2 channels (CMT0, CMT1).

Note 2. The interrupt vector numbers for software configurable interrupt show the default values specified in the board support package FIT module (BSP module).

2.5 Header Files

All API calls and their supporting interface definitions are located in “r_cmt_rx_if.h”.

2.6 Integer Types

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

2.7 Configuration Overview

The configuration option settings of this module are located in r_cmt_rx_config.h. The option names and setting values are listed in the table below:

Configuration options in r_cmt_rx_config.h	
CMT_RX_CFG_IPR 5	Interrupt priority level used for CMT interrupts

2.8 Code Size

Typical code sizes associated with this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview. The table lists reference values when the C compiler’s compile options are set to their default values, as described in 2.3, Supported Toolchain. The compile option default values are optimization level: 2, optimization type: for size, and data endianness: little-endian. The code size varies depending on the C compiler version and compile options.

ROM, RAM and Stack Code Sizes								
Device	Category	Memory Used						Remarks
		Renesas Compiler		GCC		IAR Compiler		
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	
2 channel parts RX110, RX111, and RX130	ROM	1290 bytes	1290 bytes	2288 bytes	2288 bytes	2155 bytes	2155 bytes	
	RAM	16 bytes	16 bytes	16 bytes	16 bytes	8 bytes	8 bytes	
	STACK	68 bytes	68 bytes	-	-	188 bytes	188 bytes	
4 channel parts RX113, RX230, RX231, RX23T, RX24T, RX24U, RX64M, RX651, RX65N, RX66T, RX71M	ROM	1805 bytes	1805 bytes	3160 bytes	3160 bytes	2980 bytes	2980 bytes	
	RAM	10 bytes	10 bytes	32 bytes	32 bytes	20 bytes	20 bytes	
	STACK	68 bytes	68 bytes	-	-	200 bytes	200 bytes	
4 channel parts RX72T	ROM	1815 bytes	1815 bytes	3152 bytes	3152 bytes	2918 bytes	2979 bytes	
	RAM	32 bytes	32 bytes	32 bytes	32 bytes	20 bytes	20 bytes	
	STACK	68 bytes	68 bytes	-	-	200 bytes	200 bytes	Maximum interrupt stack

2.9 Parameters

This section describes the parameter structure used by the API functions in this module. The structure is located in r_cmt_rx_if.h as are the prototype declarations of API functions.

2.10 Return Values

This section describes return values of API functions. This enumeration is located in r_cmt_rx_if.h as are the prototype declarations of API functions.

All CMT functions return a Boolean value that indicates success or failure of the call.

2.11 Callback Function

In this module, the callback function specified by the user is called when the CMT interrupt occurs.

The callback function is specified by storing the address of the user function in the “void (* callback)(void * pdata)” structure member (see 2.9, Parameters). When the callback function is called, the variable which stores the channel number is passed as the argument.

The argument is passed as void type. Thus, the argument of the callback function is cast to a void pointer. See examples below as reference.

When using a value in the callback function, type cast the value.

```
void my_cmt_callback(void * pdata)
{
    uint32_t  cmt_event_channel_number;

    cmt_event_channel_number = *((uint32_t *)pdata); //cast pointer to uint32_t
    ...
}
```

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.13 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

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

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

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

3. API Functions

R_CMT_CreatePeriodic()

This function finds an unused CMT channel, configures it for the requested frequency, associates a user callback function with the timer's interrupt, and powers up and starts the timer.

Format

```
bool    R_CMT_CreatePeriodic (
        uint32_t                frequency_hz
        void (* callback) (void *pdata),
        uint32_t                *channel
)
```

Parameters

uint32_t frequency_hz

Desired frequency in Hz [note 1]. The range and resolution of the timer is determined by settings of the peripheral clock. The best pre-scaler for the CMT channel is chosen by the driver.

callback

Pointer to the user's callback function. It should receive a single void * argument.

*uint32_t *channel*

The CMT FIT module finds the first CMT channel that is not in use and assigns it to the caller. This allows multiple drivers to use the CMT driver without having to pre-assign all timer channels. This argument provides a way for the driver to indicate back to the caller which channel has been assigned.

Return Values

```
[true]      /* Successful; CMT initialized                          */
[false]     /* No free CMT channels available, or invalid settings  */
```

Properties

Prototyped in file "r_cmt_rx_if.h"

Description

The R_CMT_CreatePeriodic function finds an unused CMT channel, assigns it to the caller, and registers a user callback function to be called upon compare match events. The CMT is configured to generate compare matches at the frequency specified in the call.

Example

This example sets up 10 Hz (100 ms) compare match operation with callback.
The example shows a user provided callback function cb that will be called to notify the user each time compare match event occurs.

```
uint32_t   ch;
bool       ret;

ret = R_CMT_CreatePeriodic(10, &cb, &ch);

if (true != ret)
{
```

```
    /* Handle the error */
}
```

Special Notes:

1. Maximum periodic frequency
   In hardware, the CMT timer maximum clocking speed is limited to PCLK/8. However, when using the periodic timer function to generate a clock, be aware that interrupt and callback processing takes some time. As requested frequency rises, interrupt and callback processing will take an increasing percentage of the processor's time. At some point, too much time is consumed to leave any time for other useful work. So this will limit the maximum frequency that can be generated. The maximum practical frequency will depend on your system design, but in general, frequencies up to a few kilohertz are reasonable.

R_CMT_CreatePeriodicAssignChannelPriority()

This function configures desired CMT channel for the requested frequency and desired interrupt priority level, associates a user callback function with the timer's interrupt, powers up and starts the timer.

Format

```
bool    R_CMT_CreatePeriodicAssignChannelPriority (
        uint32_t                frequency_hz
        void (* callback) (void *pdata),
        uint32_t                channel,
        cmt_priority_t  priority
)
```

Parameters

uint32_t frequency_hz

Desired frequency in Hz [note 1]. The range and resolution of the timer is determined by settings of the peripheral clock. The best pre-scaler for the CMT channel is chosen by the driver.

callback

Pointer to the user's callback function. It should receive a single void * argument.

uint32_t channel

Desired CMT channel that is used to configure.

cmt_priority_t priority

Desired priority level of timer's interrupt:

CMT_PRIORITY_0: Interrupt is disabled
CMT_PRIORITY_1: Lowest interrupt priority
CMT_PRIORITY_2
CMT_PRIORITY_3
CMT_PRIORITY_4
CMT_PRIORITY_5
CMT_PRIORITY_6
CMT_PRIORITY_7
CMT_PRIORITY_8
CMT_PRIORITY_9
CMT_PRIORITY_10
CMT_PRIORITY_11

CMT_PRIORITY_12
CMT_PRIORITY_13
CMT_PRIORITY_14
CMT_PRIORITY_15: Highest interrupt priority

Return Values

[true]	*/* Successful; CMT initialized*	**/*
[false]	*/* No free CMT channels available, or invalid settings* Note 2	**/*

Properties

Prototyped in file “r_cmt_rx_if.h”

Description

The R_CMT_CreatePeriodicAssignChannelPriority assigns desired CMT channel to the caller with desired interrupt priority level, and registers a user callback function to be called upon compare match events. The CMT is configured to generate compare matches at the frequency specified in the call.

Example

This example sets up 10 Hz (100 ms) compare match operation with CMT channel 0, interrupt priority level 7 and callback.
The example shows a user provided callback function cb that will be called to notify the user each time compare match event occurs.

```
uint32_t   ch = 0;
bool       ret;
cmt_priority_t priority = CMT_PRIORITY_7;

ret = R_CMT_CreatePeriodicAssignChannelPriority(10, &cb, ch, priority);

if (true != ret)
{
    /* Handle the error */
}
```

Special Notes:

1. Maximum periodic frequency
   In hardware, the CMT timer maximum clocking speed is limited to PCLK/8. However, when using the periodic timer function to generate a clock, be aware that interrupt and callback processing takes some time. As requested frequency rises, interrupt and callback processing will take an increasing percentage of the processor’s time. At some point, too much time is consumed to leave any time for other useful work. So this will limit the maximum frequency that can be generated. The maximum practical frequency will depend on your system design, but in general, frequencies up to a few kilohertz are reasonable.
2. Invalid settings
   The function will return false if one of the following invalid settings occurs: invalid channel, invalid priority, channel was in used, or frequency could not be used.

R_CMT_CreateOneShot()

This function finds an unused CMT channel, configures it for the requested period, associates a user callback function with the timer's interrupt, and powers up and starts the timer.

Format

```
bool    R_CMT_CreateOneShot (
        uint32_t                period_us,
        void (* callback)(void *pdata),
        uint32_t                *channel
)
```

Parameters

uint32_t period_us

Desired period in microseconds. The range and resolution of the timer is determined by settings of the peripheral clock. The best pre-scaler for the CMT channel is chosen by the driver.

callback

Pointer to the user's callback function. It should data a single void * argument.

uint32_t channel

The CMT FIT module finds the first CMT channel that is not in use and assigns it to the caller. This allows multiple drivers to use the CMT driver without having to pre-assign all timer channels. This argument provides a way for the driver to indicate back to the caller which channel has been assigned.

Return Values

[true] / Successful; CMT initialized */*
[false] / No free CMT channels available, or invalid settings */*

Properties

Prototyped in file "r_cmt_rx_if.h"

Description

The R_CMT_CreateOneShot function finds an unused CMT channel, assigns it to the caller, and registers a user callback function to be called upon the compare match event. The CMT is configured to generate a compare match after the period specified in the call. The timer is shut down after a single compare match event.

Example

This example sets up 100 ms compare match operation with callback.

```
uint32_t   ch;
bool       ret;

ret = R_CMT_CreateOneShot(100000, &cb, &ch);

if (true != ret)
{
    /* Handle the error */
}
```

Special Notes:
None.

R_CMT_CreateOneShotAssignChannelPriority()

This function configures the desired CMT channel for the requested period with desired interrupt priority level, associates a user callback function with the timer's interrupt, powers up and starts the timer.

Format

```
bool    R_CMT_CreateOneShotAssignChannelPriority (
            uint32_t                period_us,
            void (* callback)(void *pdata),
            uint32_t                channel,
            cmt_priority_t    priority
)
```

Parameters

uint32_t period_us
Desired period in microseconds. The range and resolution of the timer is determined by settings of the peripheral clock. The best pre-scaler for the CMT channel is chosen by the driver.

callback
Pointer to the user's callback function. It should data a single void * argument.

uint32_t channel
Desired CMT channel that is used to configure.

cmt_priority_t priority
Desired priority level of timer's interrupt:

CMT_PRIORITY_0: Interrupt is disabled
CMT_PRIORITY_1: Lowest interrupt priority
CMT_PRIORITY_2
CMT_PRIORITY_3
CMT_PRIORITY_4
CMT_PRIORITY_5
CMT_PRIORITY_6
CMT_PRIORITY_7
CMT_PRIORITY_8
CMT_PRIORITY_9
CMT_PRIORITY_10
CMT_PRIORITY_11
CMT_PRIORITY_12
CMT_PRIORITY_13
CMT_PRIORITY_14
CMT_PRIORITY_15: Highest interrupt priority

Return Values

[true]	*/* Successful; CMT initialized*	**/*
[false]	*/* No free CMT channels available, or invalid settings* Note 1	**/*

Properties
Prototyped in file “r_cmt_rx_if.h”

Description
The R_CMT_CreateOneShotAssignChannelPriority assigns the desired CMT channel to the caller with desired interrupt priority level, and registers a user callback function to be called upon the compare match event. The CMT is configured to generate a compare match after the period specified in the call. The timer is shut down after a single compare match event.

Example
This example sets up 100 ms compare match operation with with CMT channel 0, interrupt priority level 7 & callback.
```
uint32_t   ch = 0;
bool       ret;
cmt_priority_t priority = CMT_PRIORITY_7;

ret = R_CMT_CreateOneShotAssignChannelPriority(100000, &cb, ch, priority);

if (true != ret)
{
    /* Handle the error */
}
```

Special Notes:
1. Invalid settings
   The function will return false if one of the following invalid settings occurs: invalid channel, invalid priority, channel was in used, or frequency could not be used.

R_CMT_Stop()

Stops a CMT channel and powers down the CMT unit if possible.

Format

```
bool    R_CMT_Stop (
        uint32_t                channel
)
```

Parameters

uint32_t channel
The CMT timer channel to stop

Return Values

[true] */* Successful; CMT closed */*
[false] */* Invalid settings */*

Properties

Prototyped in file “r_cmt_rx_if.h”.

Description

This function frees the CMT channel by clearing its assignment and disabling the associated interrupt. The CMT channel cannot be used again until it has been reopened with either the R_CMT_CreatePeriodic or the R_CMT_CreateOneShot function.

If the CMT channel is already used as RTOS system timer, a call to this function with this CMT channel as channel, will result in FALSE being returned.

Example

This example stops CMT timer channel.

```
uint32_t   ch;
bool       ret;

/* Open and start the timer */
ret = R_CMT_CreatePeriodic(10, &cb, &ch);

/* Stop the timer */
ret = R_CMT_Stop(ch);

if (true != ret)
{
    /* Handle the error */
}
```

Special Notes:

None.

R_CMT_Control()

This function provides various ways to control and monitor a CMT channel.

Format

```
bool    R_CMT_Control (
        uint32_t                channel,
        cmt_commands_t          command,
        void                    *pdata
)
```

Parameters

uint32_t channel

CMT channel number to control.

cmt_commands_t command

Command to execute:

CMT_RX_CMD_IS_CHANNEL_COUNTING
CMT_RX_CMD_PAUSE
CMT_RX_CMD_RESUME
CMT_RX_CMD_RESTART
CMT_RX_CMD_GET_NUM_CHANNELS
CMT_RX_CMD_SET_PRIORITY
CMT_RX_CMD_GET_PRIORITY

Return Values

[true] / The command completed properly. Check pdata */*
[false] / The command did not complete properly */*

Properties

Prototyped in file “r_cmt_rx_if.h”.

Description

This function provides a number of commands:

CMT_RX_CMD_IS_CHANNEL_COUNTING tells if a CMT channel is currently running. Check *pdata.
CMT_RX_CMD_PAUSE pauses a timer without closing it (without powering it off).
CMT_RX_CMD_RESUME restarts a paused timer without resetting the counter to zero
CMT_RX_CMD_RESTART restarts a paused timer after resetting the counter to zero
CMT_RX_CMD_GET_NUM_CHANNELS returns the total number of channels available
CMT_RX_CMD_SET_PRIORITY sets the interrupt priority of the CMT channel.
CMT_RX_CMD_GET_PRIORITY gets the interrupt priority of the CMT channel.

If the CMT channel is already used as RTOS system timer, a call to this function with this CMT channel as channel, and any of *CMT_RX_CMD_IS_CHANNEL_COUNTING*, *CMT_RX_CMD_PAUSE*, *CMT_RX_CMD_RESUME*, *CMT_RX_CMD_RESTART* as command, will result in FALSE being returned.

Example 1

This example pauses a timer and restarts a paused timer.

```
uint32_t   ch;
bool       ret;

/* Open and Start the timer */
ret = R_CMT_CreatePeriodic(10, &cb, &ch);

if (true != ret)
{
    /* Handle the error */
}

/* Pause the timer */
ret = R_CMT_Control(ch, CMT_RX_CMD_PAUSE, NULL);

if (true != ret)
{
    /* Handle the error */
}

/* Restart the timer after resetting the counter to zero */
ret = R_CMT_Control(ch_info, CMT_RX_CMD_RESTART, NULL);

if (true != ret)
{
    /* Handle the error */
}
```

Example 2

This example checks state of channel and get total number of channels available.

```
uint32_t   ch;
uint32_t   ch_num;
bool       ret;
bool       data;

/* Open and Start the timer */
ret = R_CMT_CreatePeriodic(10, &cb, &ch);

if (true != ret)
{
    /* Handle the error */
}

/* Check state of channel */
ret = R_CMT_Control(ch, CMT_RX_CMD_IS_CHANNEL_COUNTING, (void*)&data);

if (true != ret)
{
    /* Handle the error */
}

/* Get available of channel */
ret = R_CMT_Control(ch, CMT_RX_CMD_GET_NUM_CHANNELS, (void*)&ch_num);

if (true != ret)
{
    /* Handle the error */
}
```

Special Notes:

None.

R_CMT_GetVersion()

This function returns the driver version number at runtime.

Format
uint32_t R_CMT_GetVersion (void)

Parameters
None.

Return Values
Version number with major and minor version digits packed into a single 32-bit value.

Properties
Prototyped in file “r_cmt_rx_if.h”

Description
The function returns the version of this module. The version number is encoded such that the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Example
Example showing this function being used.

```
/* Retrieve the version number and convert it to a string. */

uint32_t   version, version_high, version_low;
char       version_str[9];

version = R_CMT_GetVersion();

version_high = (version >> 16)&0xf;
version_low  =  version & 0xff;

sprint(version_str, “CMT v%1.1hu.%2.2hu”, version_high, version_low);
```

Special Notes:
None.

4. Pin Setting

CMT FIT module don’t use pin setting.

5. Demo Projects

Demo projects include function main() that utilizes the FIT module and its dependent modules (e.g. r_bsp). This FIT module includes the following demo projects.

5.1 cmt_demo_rskrx113

The cmt_demo_rskrx113 program demonstrates how to create a timer tick using a CMT channel, how to set up a callback function to handle CMT interrupts and how to de-reference the channel information in the callback argument. As the program runs, the CMT callback function toggles LED0 at a 2 Hz rate.

5.2 cmt_demo_rskrx231

The cmt_demo_rskrx231 program is identical to cmt_demo_rskrx113.

5.3 cmt_demo_rskrx64M

The cmt_demo_rskrx64M program is identical to cmt_demo_rskrx113.

5.4 cmt_demo_rskrx71m

The cmt_demo_rskrx71m program is identical to cmt_demo_rskrx113.

5.5 cmt_demo_rskrx65n

The cmt_demo_rskrx65n program demonstrates how to create a timer tick using a CMT channel, how to set up a callback function to handle CMT interrupts and how to de-reference the channel information in the callback argument. As the program runs, the CMT callback function toggles LED0 and LED1 at a 2 Hz rate.

5.6 cmt_demo_rskrx65n_2m

The cmt_demo_rskrx65n_2m program is identical to cmt_demo_rskrx65n.

5.7 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo

project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.8 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Browser* >> *Application Notes* tab.

6. Appendices

6.1 Confirmed Operation Environment

This section describes confirmed operation environment for the CMT FIT module.

Table 6.1 Confirmed Operation Environment (Rev.4.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.4.0 IAR Embedded Workbench for Renesas RX 4.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.4.00
Board used	Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxx)

Table 6.2 Confirmed Operation Environment (Rev.3.40)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.40
Board used	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 6.3 Confirmed Operation Environment (Rev.3.31)

Item	Contents
Integrated development environment	Renesas Electronics e² studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.31
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX65N-2M (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.4 Confirmed Operation Environment (Rev.3.30)

Item	Contents
Integrated development environment	Renesas Electronics e² studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.30
Board used	Renesas Starter Kit for RX66T (product No.: RTK50566T0SxxxxxBE) Renesas Starter Kit+ for RX65N-2M (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.5 Operation Confirmation Environment (Rev.3.21)

Item	Contents
Integrated development environment	Renesas Electronics e² studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.21
Board used	Renesas Starter Kit+ for RX65N-2M (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

Table 6.6 Operation Confirmation Environment (Rev.3.20)

Item	Contents
Integrated development environment	Renesas Electronics e² studio Version 6.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.07.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.3.20
Board used	Renesas Starter Kit+ for RX65N-2M (product No.: RTK50565N2CxxxxxBR) Renesas Starter Kit+ for RX130-512KB (product No.: RTK5051308CxxxxxBR)

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”

- Using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “Board Support Package Module Using Firmware Integration Technology (R01AN1685)”.

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_cmt_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

(3) Q: I have added the FIT module to the project and built it. Then I got an error for when the configuration setting is wrong.

A: The setting in the file “r_cmt_rx_config.h” may be wrong. Check the file “r_cmt_rx_config.h”. If there is a wrong setting, set the correct value for that. Refer to 2.7, Configuration Overview for details.

7. 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
RX Family Compiler CC-RX User's Manual (R20UT3248)
The latest versions can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

None

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
2.00	Nov 6, 2013	—	First GSCE Release
2.10	Nov 15, 2013	—	Formula for CMCOR value corrected.
2.30	Apr 12, 2014	1, 2, 7	Updated to indicate support for additional MCUs Added section 1.2 on callback functions Added notes on maximum periodic frequency
2.40	Nov 14, 2014	—	Added support for the RX113 Group.
2.41	Dec 4, 2014	5	Added Code Size section.
2.50	Mar 10, 2015	—	Added support for the RX71M Group.
2.51	Mar 10, 2015	3	Fixed a bug in cmt_isr_common which passed the CMT chnl number instead of a pointer to the CMT chnl number to the callback function. Updated section 1.2.2 regarding pointer cast as uint32_t.
2.60	June 30, 2015	—	Added support for the RX231 Group.
2.70	Sep 30, 2015	—	Added support for the RX23T Group.
		5	Updated the ROM size for 4 channels in 2.9 Code Size.
2.80	Oct 1, 2015	—	Added support for the RX130 Group.
		5	Updated the ROM size for 2 channels in 2.9 Code Size.
2.90	Dec 1, 2015	—	Added support for the RX230 and the RX24T Groups.
		1, 5	Changed the document number for the “Board Support Package Firmware Integration Technology Module” application note.
		4	Changed the description in section 2.
		5	Updated the Code Size table for the RX230 and the RX24T Groups.
		11	Changed description for “false” in Return Values. In Description, deleted the description regarding a call for a CMT channel not in operation.
		12	Added “4. Demo Projects”.
2.91	June 15, 2016	12	Added RSKRX64M to “4. Demo Projects”.
		13	Added “Related Technical Updates”.
3.00	Oct 1, 2016	—	Added support for the RX65N Group
		5	Changed the tabular format of Code Size. Updated the Code Size table for the RX65N Group.
		7, 9, 10, 11,12	Added a description of API function sample code.
3.10	Feb 28, 2017	—	Added support for the RX24T (including ROM 512 KB version) and RX24U Groups.
		—	Corrected some descriptions.
		4	Added RXC v2.06.00 to “2.5 Supported Toolchains”.
		Program	Modified the timing to stop the timer to fix the following issue: The compare match counter is operating alone even if the CMT is stopped if an interrupt occurs at an unexpected timing during one-shot timer operation Modified the register setting sequence when stopping the timer with the R_CMT_Stop function. Modified the register setting sequence when disabling an interrupt with the R_CMT_Stop function. Fixed the following issue:

			When all channels are used, an error occurs if attempting to respecify the one-shot timer in the callback function for the one-shot timer.
3.20	July 21, 2017	—	Added support for the RX130-512KB and RX65N-2MB.
		4	Added RXC v2.07.00 to “2.5 Supported Toolchains”.
		5	Added “2.6 Interrupt Vector”.
		7	Updated “2.13 Adding the FIT Module to Your Project”.
3.21	Oct 31, 2017	15	Added RSKRX65N, RSKRX65N-2M to “4. Demo Projects”. Added 4.8 Downloading Demo Projects
		16	Added 5. Appendices
3.30	Sep 28, 2018	1, 3	Added support for RTOS.
		1, 4	Added support for RX66T
		5	Added code size corresponding to RX66T
		12	R_CMT_Stop(): Modified Description
		13	R_CMT_Control(): Modified Description
		18	6.1 Confirmed Operation Environment: Added table for Rev.3.30
3.31	Nov 16, 2018	—	Added document number in XML
		18	Changed Renesas Starter Kit Product No for RX66T. Added table for Rev.3.31
3.40	Feb 01, 2019	Program	Added support for RX72T.
		1, 4	Added support for RX72T.
		5	Added code size corresponding to RX72T
		9-16	Removed ‘Reentrant’ description in each API function.
		18	6.1 Confirmed Operation Environment: Added Table for Rev 3.40
3.50	May 01, 2019	3,4,5, 11, 12,14,15,	Added new API: R_CMT_CreatePeriodicAssignChannelPriority(); R_CMT_CreateOneShotAssignChannelPriority()
		18	R_CMT_Control(): Added new commands to get/set interrupt priority of CMT channel
4.00	May.20.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Deleted the RX210, RX631, and RX63N in Target Devices for end of update these devices. Added the section of Target compilers. Deleted related documents.
		7	Updated the section of 2.8 Code Size
		6	2.3 Software Requirements Requires r_bsp v5.20 or higher 2.4 Interrupt Vector: Deleted the description of RX210, RX631, RX63N
		22	Table 6.1 Confirmed Operation Environment: Added table for Rev.4.00
		26	Deleted the section of Website and Support.
		Program	Changed bellow for support GCC and IAR compiler: 1. Deleted the inline expansion of the R_CMT_GetVersion function. 2. Replaced evenaccess with the macro definition of BSP. 3. Replaced the declaration of interrupt functions with the macro definition of BSP.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

   Handle unused pins in accordance 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 the 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.

5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.

6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.