# RX Family

CMTW Module Using Firmware Integration Technology

## Introduction

The RX Family MCUs supported by this module have two CMTW units. Throughout this document these units are referred as channels. Each channel has one compare match (CM), two output compares (OC0 and OC1), and two input captures (IC0 and IC1). Compare match is a general purpose timer that can be used to generate timer ticks. Output compare and input capture are timers but also have MCU pins associated with them. They are used to generate waveforms based on the timer settings or detect external events and capture the time of the event. The CMTW timers are 32-bit wide allowing from hundreds of nanoseconds to hours of time events to be specified.

This document describes the CMTW FIT Module API for the supported RX class MCUs. Software architecture, system interfaces and usage details are provided here to facilitate integration of the CMTW FIT module into a user's application.

## Target Device

- RX64M Group
- RX651, RX65N Groups
- RX66N Group
- RX71M Group
- RX72M Group
- RX72N Group

## 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 Operation Confirmation Environment".

## Contents

# 1. Overview

This software provides a unified, abstracted interface for setting up the CMTW peripheral for the supported RX class MCUs. The software also provides support for interrupt handling for the CMTW peripheral and a notification mechanism to the user's application through a callback function.

There are two CMTW channels and therefore a means to identify a particular CMTW channel in each API function is provided. The setup operations are performed in the R_CMTW_Open() API function. This API powers on the specified CMTW channel, initializes it based on the user supplied configuration parameters and enables the interrupts if needed. A pointer to a function called a "callback function" can be provided through this API so that the user is notified when a timer event occurs. This notification operation take place in the CMTW interrupt handlers. Therefore, it is advised that user supplied callback functions complete quickly to allow time to other processes in the system. The API also supports one-shot operations where the requested action is executed only once.

After initialization is complete, the user can start, stop, resume, or restart the timers. This is done by R_CMTW_Control() API. When the CMTW operations are no longer needed, the channel can be closed by R_CMTW_Close() API. This API disables the CMTW channel and powers it off to reduce power consumption.

Figure 1 below shows an example of a high level project view using CMTW FIT module.

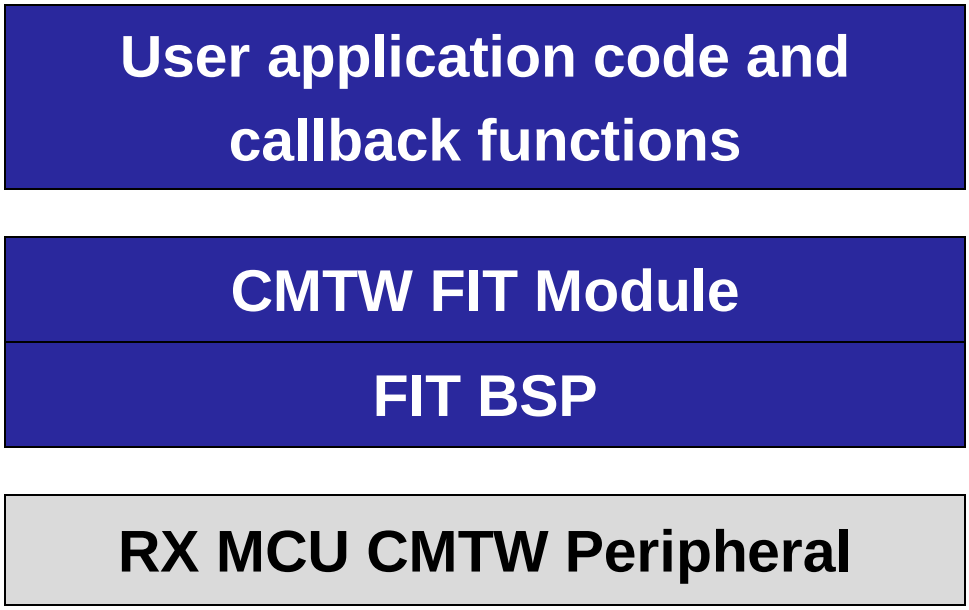


**Figure 1 - Example Figure Showing Project Layers**

## 1.1 Using the CMTW Firmware Integration Technology (FIT) module

The primary use of the CMTW module is to make it easy to setup and control the CMTW timers. The user's application can arbitrarily assign a callback function that will execute when a timer event occurs.

After adding the CMTW module to your project you will need to modify the r_cmtw_rx_config.h file to configure the software for your installation. See Section 2.9 for details on configuration options.

Output and input pins used by the CMTW peripheral must be correctly set up before the use of the CMTW module. The module does not provide any means to initialize the pin registers, which needs to be done externally prior to calling CMTW API functions. Typically this would be accomplished at system startup time as part of a general pin initialization routine. GPIO and MPC FIT modules can be used to setup the pins. Table 1 shows the output and input pin assignments for the CMTW peripheral.

| Output and Input Pin Assignments for CMTW | | |
|---|---|---|
| **Channel 0** | Output compare 0 (TOC0) | PC7 |
| | Output compare 1 (TOC1) | PE7 |
| | Input capture 0 (TIC0) | PC6 |
| | Input capture 1 (TIC1) | PE6 |
| **Channel 1** | Output compare 0 (TOC2) | PD3 |
| | Output compare 1 (TOC3) | PE3 |
| | Input capture 0 (TIC2) | PD2 |
| | Input capture 1 (TIC3) | PE2 |

**Table 1 – Pin Configurations for the CMTW**

## 1.2 Interrupts

There is no need to set up interrupt vectors for CMTW as this software provides the interrupt handlers for all CMTW interrupts. Depending on the user supplied configuration parameters to the R_CMTW_Open() API, interrupt only or callback actions can be taken. Interrupt only action is useful if CMTW is used with another peripheral e.g. DTC or DMAC. However, this software does not setup other peripherals that can be used with CMTW.

Callback action is used to notify the user about the CMTW timer events. If requested by the user, the CMTW ISR calls the user supplied callback function. It is within the callback that user code is run in response to the ISR. Since callbacks are being processed within the context of the interrupt and interrupts are disabled at this time, it is strongly recommended that callback function completes as quickly as possible to avoid missing other interrupts that might occur in the system.

In case no interrupts are needed, timer only action can be used. This is useful for producing output compare waveforms without generating interrupts and consuming CPU time.

## 1.3 Callback Functions

### 1.3.1 Example callback function prototype declaration.

```
void my_cmtw_callback(void *pdata);
```

### 1.3.2 Dereferencing of pdata argument.

The ISR code uses a pointer to a structure defined in the *r_cmtw_rx_if.h* file to pass the event information to the user callback function. This structure is of type `cmtw_callback_data_t`. Since FIT callback functions take a void pointer, this pointer must be type-casted to `(cmtw_callback_data_t *)` before it can be dereferenced and access the information provided by the CMTW interrupt handler. The interrupt data structure and types are defined in section 2.11.1.

Example:

```
void my_cmtw_callback(void *pdata)
{
    cmtw_callback_data_t    *p_cb_data = (cmtw_callback_data_t *)pdata;
    ....
    cb_data.channel = p_cb_data->channel;
    cb_data.event = p_cb_data->event;
    cb_data.count = p_cb_data->count;
    ...
}
```

# 2. API Information

This driver API follows the Renesas API naming standards.

## 2.1 Hardware Requirements

This driver requires that your MCU supports the following peripheral(s):

- CMTW

## 2.2 Hardware Resource Requirements

This section details the hardware peripherals that this driver requires. Unless explicitly stated, these resources must be reserved for the driver, and the user cannot use them.

None.

## 2.3 Software Requirements

This driver is dependent upon the following FIT packages:

- Renesas Board Support Package (r_bsp) v5.20 or higher.

This driver assumes that the related I/O pins have been correctly initialized elsewhere prior to calling this software's API functions.

## 2.4 Limitations

This driver is applicable only to CMTW operations. If another peripheral is linked with CMTW, it must be setup externally. Please see section 2.9 for limitations on the usage of locks and dependency to the BSP.

### 2.4.1 RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.

The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.

In the case of the CCRX project (e2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of the GCC project (e2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.

The default settings of the section may be changed due to the IDE version upgrade. Please check the section settings when using the latest IDE.

## 2.5 Supported Toolchains

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

## 2.6 Interrupt Vector

CMTW interrupt is enabled by execution **R_CMTW_Open()** function.

**Table 2.1 Interrupt Vector Used in the CMTW FIT Module** lists the interrupt vector used in the CMTW FIT Module.

**Table 2.1 Interrupt Vector Used in the CMTW FIT Module**

| Device | Interrupt Vector |
|---|---|
| RX64M<br>RX651, RX65N<br>RX66N<br>RX71M, RX72M<br>RX72N | ● CMWI0 interrupt[channel 0] (vector no.: 30)<br>● CMWI1 interrupt[channel 1] (vector no.: 31)<br>● IC0I0 interrupt[channel 0] (vector no.: 168)*1<br>● IC1I0 interrupt[channel 0] (vector no.: 169) *1<br>● OC0I0 interrupt[channel 0] (vector no.: 170) *1<br>● OC1I0 interrupt[channel 0] (vector no.: 171) *1<br>● IC0I1 interrupt[channel 1] (vector no.: 172) *1<br>● IC1I1 interrupt[channel 1] (vector no.: 173) *1<br>● OC0I1 interrupt[channel 1] (vector no.: 174) *1<br>● OC1I1 interrupt[channel 1] (vector no.: 175) *1 |

Note 1. The interrupt vector numbers for software configurable interrupt B show the default values specified in the board support package FIT module (BSP module).

## 2.7 Header Files

All API calls and their supporting interface definitions are located in "*r_cmtw_rx_if.h* ".

Build-time configuration options are selected or defined in the file "*r_cmtw_rx_config.h*".

Both of these files should be included by the user's application.

## 2.8 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

## 2.9 Configuration Overview

Some features or behavior of the software are determined at build-time by configuration options that the user must select.

| Configuration options in r_cmtw_rx_config.h | |
|---|---|
| **CMTW_CFG_PARAM_CHECKING_ENABLE** | This macro is used to enable parameter checking for CMTW API functions. Disabling the parameter checking is provided for systems that absolutely require faster and smaller code.<br>By default the module is configured to use the setting of the system-wide BSP_CFG_PARAM_CHECKING_ENABLE macro. This can be locally overridden for the CMTW module by redefining the macro. To control parameter checking locally, set the macro to 1 to enable or to 0 to disable the checking. |
| **CMTW_CGF_REQUIRE_LOCK** | This macro is used to lock the API functions while they are being used to prevent simultaneous calls to them. This prevents multiple accesses to the API function within an RTOS environment.<br>By default the module is configured to enable the locking. If the system is not multitasked and there is no possibility of running multiple instances of the API functions, this macro can be set to 0 to disable the locking to save code space. |
| **CWTW_CFG_IPR_CM_CH0** | This macro defines the priority level for compare match interrupt for channel 0. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_OC0_CH0** | This macro defines the priority level for output compare 0 interrupt for channel 0. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_OC1_CH0** | This macro defines the priority level for output compare 1 interrupt for channel 0. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_IC0_CH0** | This macro defines the priority level for input capture 0 interrupt for channel 0. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_IC1_CH0** | This macro defines the priority level for input capture 1 interrupt for channel 0. Valid values are 1 to 15. |
| **CWTW_CFG_IPR_CM_CH1** | This macro defines the priority level for compare match interrupt for channel 1. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_OC0_CH1** | This macro defines the priority level for output compare 0 interrupt for channel 1. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_OC1_CH1** | This macro defines the priority level for output compare 1 interrupt for channel 1. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_IC0_CH1** | This macro defines the priority level for input capture 0 interrupt for channel 1. Valid values are 1 to 15. |
| **CMTW_CFG_IPR_IC1_CH1** | This macro defines the priority level for input capture 1 interrupt for channel 1. Valid values are 1 to 15. |

**Table 2 - List of CMTW module configuration options**

## 2.10 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 2.9, Configuration Overview. The table lists reference values when the C compiler's compile options are set to their default values, as described in 2.5, Supported Toolchains. 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**

| Devices | Category | | Memory Used | | | | | | Remarks |
|---|---|---|---|---|---|---|---|---|---|
| | | | Renesas Compiler | | GCC | | IAR Compiler | | |
| | | | With Parameter Checking and Require Locking | Without Parameter Checking and Require Locking | With Parameter Checking and Require Locking | Without Parameter Checking and Require Locking | With Parameter Checking and Require Locking | Without Parameter Checking and Require Locking | |
| RX64M | ROM | 1 channel used | 2025 bytes | 1798 bytes | 3036 bytes | 2708 bytes | 4746 bytes | 4439 bytes | |
| | | 2 channels used | 2522 bytes | 2295 bytes | 3632 bytes | 3304 bytes | 5199 bytes | 4892 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 0 bytes | 0 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 0 bytes | 0 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 64 bytes | 64 bytes | - | - | 176 bytes | 176 bytes | |
| RX65N | ROM | 1 channel used | 2025 bytes | 1798 bytes | 3036 bytes | 2708 bytes | 4746 bytes | 4439 bytes | |
| | | 2 channels used | 2522 bytes | 2295 bytes | 3632 bytes | 3304 bytes | 5199 bytes | 4892 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 0 bytes | 0 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 0 bytes | 0 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 64 bytes | 64 bytes | - | - | 176 bytes | 176 bytes | |
| RX71M | ROM | 1 channel used | 2025 bytes | 1664 bytes | 3036 bytes | 2708 bytes | 4746 bytes | 4439 bytes | |
| | | 2 channels used | 2522 bytes | 2161 bytes | 3632 bytes | 3304 bytes | 5199 bytes | 4892 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 0 bytes | 0 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 0 bytes | 0 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 64 bytes | 64 bytes | - | - | 176 bytes | 176 bytes | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| RX72M | ROM | 1 channel used | 2033 bytes | 1806 bytes | 3356 bytes | 2972 bytes | 2917 bytes | 2606 bytes | |
| | | 2 channels used | 2530 bytes | 2303 bytes | 3952 bytes | 3568 bytes | 3371 bytes | 3060 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 0 bytes | 0 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 0 bytes | 0 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 120 bytes | 120 bytes | - | - | 220 bytes | 220 bytes | |
| RX72N | ROM | 1 channel used | 2089 bytes | 1862 bytes | 3401 bytes | 3020 bytes | 4599 bytes | 4288 bytes | |
| | | 2 channels used | 2586 bytes | 2359 bytes | 3997 bytes | 3616 bytes | 5048 bytes | 4737 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 32 bytes | 32 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 64 bytes | 64 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 80 bytes | 80 bytes | - | - | 136 bytes | 136 bytes | |
| RX66N | ROM | 1 channel used | 2089 bytes | 1862 bytes | 3448 bytes | 2996 bytes | 4591 bytes | 4292 bytes | |
| | | 2 channels used | 2586 bytes | 2359 bytes | 4044 bytes | 3592 bytes | 5040 bytes | 4741 bytes | |
| | RAM | 1 channel used | 32 bytes | 32 bytes | 32 bytes | 32 bytes | 16 bytes | 16 bytes | |
| | | 2 channels used | 64 bytes | 64 bytes | 64 bytes | 64 bytes | 32 bytes | 32 bytes | |
| | Maximum stack usage | | 80 bytes | 80 bytes | - | - | 136 bytes | 136 bytes | |

## 2.11 API Data Types

This section details the data structures that are used with the driver's API functions.

### 2.11.1 Special Data Types

To provide strong type checking and reduce errors, many parameters used in API functions require arguments to be passed using the provided type definitions. Allowable values are defined in the public interface file *r_cmtw_rx_if.h*. The following special types have been defined:

```
/* Channel numbers */
typedef enum
{
    CMTW_CHANNEL_0 = 0,
    CMTW_CHANNEL_1,
    CMTW_CHANNEL_MAX,
} cmtw_channel_t;

/* Time base */
typedef enum
{
    CMTW_TIME_NSEC = 0,
```

```
    CMTW_TIME_USEC,
    CMTW_TIME_MSEC,
    CMTW_TIME_SEC,
    CMTW_TIME_MAX,
} cmtw_time_unit_t;

/* PCLK divisor */
typedef enum
{
    CMTW_CLK_DIV_8 = 0,         // PCLK/8
    CMTW_CLK_DIV_32,            // PCLK/32
    CMTW_CLK_DIV_128,           // PCLK/128
    CMTW_CLK_DIV_512,           // PCLK/512
    CMTW_CLK_DIV_MAX,
} cmtw_clock_divisor_t;

/* Counter clearing source */
typedef enum
{
    CMTW_CLR_CMT        = 0,
    CMTW_CLR_DISABLED   = 1,
    CMTW_CLR_IC0        = 4,
    CMTW_CLR_IC1        = 5,
    CMTW_CLR_OC0        = 6,
    CMTW_CLR_OC1        = 7,
    CMTW_CLR_MAX,
} cmtw_clear_source_t;

/* Actions to take */
typedef enum
{
    CMTW_ACTION_NONE        = 0x00,     // Do nothing with this timer
    CMTW_ACTION_TIMER       = 0x01,     // Run timer only no interrupt
    CMTW_ACTION_INTERRUPT   = 0x02,     // Generate interrupt request
    CMTW_ACTION_CALLBACK    = 0x04,     // Generate interrupt request and
                                        //   execute user-defined callback
    CMTW_ACTION_ONESHOT     = 0x08,     // Generate interrupt, act only once,
                                        //   and turn off
} cmtw_actions_t;

/* Output pin states  */
typedef enum
{
    CMTW_OUTPUT_RETAIN      = 0,        // Do not change the pin state
    CMTW_OUTPUT_LO_TOGGLE,              // Output low initially and toggle
    CMTW_OUTPUT_HI_TOGGLE,              // Output hi  initially and toggle
    CMTW_OUTPUT_MAX,
} cmtw_output_states_t;

/* Input pin edges to capture */
typedef enum
{
    CMTW_EDGE_RISING        = 0,        // Capture rising edge
    CMTW_EDGE_FALLING,                  // Capture falling edge
    CMTW_EDGE_ANY,                      // Capture both rising and falling edges
    CMTW_EDGE_MAX,
} cmtw_edge_states_t;

/* Control function command codes. */
typedef enum
{
```

```
    CMTW_CMD_START,             // Activate clocking
    CMTW_CMD_RESUME,            // Same as start
    CMTW_CMD_STOP,              // Pause clocking
    CMTW_CMD_RESTART,           // Zero the timer counter then activate clocking
    CMTW_CMD_MAX,               // Not a valid command.
} cmtw_cmd_t;

/* Open function CM settings */
typedef struct
{
    uint32_t                time;
    cmtw_actions_t          actions;
} cmtw_cm_settings_t;

/* Open function OC settings */
typedef struct
{
    uint32_t                time;
    cmtw_actions_t          actions;
    cmtw_output_states_t    output;
} cmtw_oc_settings_t;

/* Open function IC settings */
typedef struct
{
    cmtw_actions_t          actions;
    cmtw_edge_states_t      edge;
} cmtw_ic_settings_t;

/* Open function channel settings */
typedef struct
{
    cmtw_time_unit_t            time_unit;
    cmtw_clock_divisor_t        clock_divisor;
    cmtw_clear_source_t         clear_source;
    cmtw_cm_settings_t          cm_timer;
    cmtw_oc_settings_t          oc_timer_0;
    cmtw_oc_settings_t          oc_timer_1;
    cmtw_ic_settings_t          ic_timer_0;
    cmtw_ic_settings_t          ic_timer_1;
} cmtw_channel_settings_t;

/* Callback function events */
typedef enum
{
    CMTW_EVENT_CM = 0,        //compare match
    CMTW_EVENT_IC0,           //input capture 0
    CMTW_EVENT_IC1,           //input capture 1
    CMTW_EVENT_OC0,           //output compare 0
    CMTW_EVENT_OC1,           //output compare 1
} cmtw_event_t;

/* Callback function data structure */
typedef struct
{
    cmtw_channel_t      channel;    //event channel number
    cmtw_event_t        event;      //type of event
    uint32_t            count;      //timer counter at event
} cmtw_callback_data_t;
```

## 2.12 Return Values

This shows the different values API functions can return. This return type is defined in “r_cmtw_rx_if.h”.

```
/* CMTW function return codes */
typedef enum
{
    CMTW_SUCCESS = 0,
    CMTW_ERR_BAD_CHAN,          // Invalid channel number
    CMTW_ERR_CH_NOT_ENABLED,    // Channel is disabled by user configuration
    CMTW_ERR_CH_NOT_OPENED,     // Channel not yet opened
    CMTW_ERR_CH_NOT_CLOSED,     // Channel still open from previous open
    CMTW_ERR_CH_NOT_RUNNIG,     // Channel open and received stop command
    CMTW_ERR_CH_NOT_STOPPED,    // Channel running and received start command
    CMTW_ERR_UNKNOWN_CMD,       // Control command is not recognized
    CMTW_ERR_INVALID_ARG,       // Argument is not valid for parameter
    CMTW_ERR_NULL_PTR,          // Rcvd null pointer; missing required arg
    CMTW_ERR_LOCK,              // The lock procedure failed
    CMTW_ERR_OUT_OF_RANGE,      // Calculated count value is not in range
} cmtw_err_t;
```

## 2.13 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.

## 2.14 “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_CMTW_Open()

This function powers up the specified CMTW channel, initializes the associated CMTW registers and enables interrupts if requested. Takes a callback function pointer for responding to interrupt events. After successful completion, the channel moves to open state. This function must be called before calling any other API functions.

### Format

```
cmtw_err_t  R_CMTW_Open(cmtw_channel_t           channel,
                        cmtw_channel_settings_t *pconfig,
                        void (* const pcallback)(void *pdata));
```

### Parameters

*channel*

Number of the CMTW channel to initialize

*pconfig*

Pointer to the CMTW channel settings data structure

*pcallback*

Pointer to the user function to call from the interrupt

### Return Values

| | |
|---|---|
| CMTW_SUCCESS | Successful, channel is initialized |
| CMTW_ERR_BAD_CHAN | Invalid channel number |
| CMTW_ERR_CH_NOT_ENABLED | Channel is disabled by user configuration |
| CMTW_ERR_CH_NOT_CLOSED | Channel currently in operation, perform R_CMTW_Close() first |
| CMTW_ERR_INVALID_ARG | An element of the pconfig structure contains an invalid value |
| CMTW_ERR_OUT_OF_RANGE | Calculated count value is not in range |
| CMTW_ERR_NULL_PTR | Either pconfig or pcallback is null |
| CMTW_ERR_LOCK | The lock could not be acquired, the channel is busy |

### Properties

Prototyped in file “r_cmtw_rx_if.h”

### Description

This function sets up a CMTW channel. After completion of the open function the CMTW channel will be initialized and ready to be started by calling R_CMTW_Control(). This function must be called once prior to calling any other CMTW API functions. Once successfully completed, the status of the selected CMTW channel will be set to "open". After that this function should not be called again for the same CMTW channel without first performing a "close" by calling R_CMTW_Close().

### Example 1

This example sets up channel 0 for 500 ms compare match operation with callback. Note that the clear source is set to CMTW_CLR_CMT to generate 500 ms timer tick. There is only one clear source that can be set for a given channel among the available clear source events. If an incorrect clear source is selected, the timer counter will not be cleared until it rolls over which may take a long time or never happen. The example shows a user provided callback function cb that will be called to notify the user each time compare match event occurs.

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;

/* Clear all fields of the cmtw_channel_settings_t structure */
memset(&ch, 0, sizeof(ch));
```

```
/* Setup for 500 ms compare match with callback */
ch.time_unit = CMTW_TIME_MSEC;
ch.clock_divisor = CMTW_CLK_DIV_8;
ch.clear_source = CMTW_CLR_CMT;
ch.cm_timer.time = 500;
ch.cm_timer.actions = CMTW_ACTION_CALLBACK;
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}
```

### Example 2

This example sets up channel 0 for output compare 0 (OC0) and output compare 1 (OC1) operation. OC0 action is set to CMTW_ACTION_TIMER so no interrupt will be generated for that event. If the PC7 pin is setup for TOC0, it will toggle at a rate of 50 ms. The reason for this is that the clear source is CMTW_CLR_OC1. This means that the timer counter will clear at every OC1 event which is set to 50 ms. 10 ms into this time, OC0 event will happen thus generating 2 events that have the same period with a phase shift. To toggle the PE7 pin, it must be setup for TOC1. Since OC1 action is setup for callback, the user provided callback function cb will be called from the OC1 ISR for channel 0.

In this example if the clear source is set to CMTW_CLR_OC0, it will result in TOC0 toggling at 10 ms rate. Since the timer counter will clear at 10 ms, it will not reach 50 ms and OC1 event will not occur.

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;

/* Clear all fields of the cmtw_channel_settings_t structure */
memset(&ch, 0, sizeof(ch));

/* Setup 50 ms output compare with TOC0 and TOC1 toggle */
ch.time_unit = CMTW_TIME_MSEC;
ch.clock_divisor = CMTW_CLK_DIV_8;
ch.clear_source = CMTW_CLR_OC1;
ch.oc_timer_0.time = 10;
ch.oc_timer_0.actions = CMTW_ACTION_TIMER;
ch.oc_timer_0.output = CMTW_OUTPUT_HI_TOGGLE;
ch.oc_timer_1.time = 50;
ch.oc_timer_1.actions = CMTW_ACTION_CALLBACK;
ch.oc_timer_1.output = CMTW_OUTPUT_HI_TOGGLE;
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}
```

### Example 3

This example sets up channel 1 for input capture 0 (IC0) operation. The PD2 pin must be assigned to TIC2 prior to calling CMTW API functions for correct behavior. The clear source is set to CMTW_CLR_DISABLED and therefore timer counter is not cleared when a TIC2 event, which is set to any edge, occurs. The user provided callback function cb will be called from the IC0 ISR for channel 1 when an edge change is detected. Channel number, ISR event, and the timer value (in timer tick count) at which the event happened are passed to the callback function. If the number of counts between the two TIC2 events are needed, it can be calculated by subtracting the current count from the previous one in the callback function.

```
cmtw_err_t rc;
cmtw_channel_settings_t ch;
```

```
/* Clear all fields of the cmtw_channel_settings_t structure */
memset(&ch, 0, sizeof(ch));

/* Setup input capture with no clear */
ch.time_unit = CMTW_TIME_USEC;
ch.clock_divisor = CMTW_CLK_DIV_128;
ch.clear_source = CMTW_CLR_DISABLED;
ch.ic_timer_0.actions = CMTW_ACTION_CALLBACK;
ch.ic_timer_0.edge = CMTW_EDGE_ANY;
rc = R_CMTW_Open(CMTW_CHANNEL_1, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}
```

**Example 4**

This is an example of a callback function for example 3. The void pointer pdata is type-casted to (cmtw_callback_data_t *) to properly access the data. The data type information is available in the r_cmtw_rx_if.h file. Previous and current count values are saved in the global variables. The time between two TIC2 events is simply the difference of these values. Since the timer tick information is available to the user (e.g. the peripheral clock and CMTW clock divisor), actual time can be calculated easily.

```
/* Global variables */
uint32_t   g_previous_count;
uint32_t   g_current_count;
uint32_t   g_delta_count;

void cb(void *pdata)
{
    cmtw_callback_data_t cb_data;
    cmtw_callback_data_t *p_cb_data = (cmtw_callback_data_t *)pdata;

    cb_data.channel = p_cb_data->channel;
    cb_data.event = p_cb_data->event;
    cb_data.count = p_cb_data->count;

    g_previous_count = g_current_count;
    g_current_count  = cb_data.count;
    g_delta_count    = g_current_count - g_previous_count;

}
```

**Special Notes:**

None

## R_CMTW_Control()

This function starts, stops, resumes, or restarts a CMTW channel that is in the open state.

### Format

```
cmtw_err_t  R_CMTW_Control(cmtw_channel_t   channel,
                           cmtw_cmd_t       cmd);
```

### Parameters

*channel*

Number of the CMTW channel to control

*cmd*

Enumerated command codes:

| | |
|---|---|
| CMTW_CMD_START | Activate the timer |
| CMTW_CMD_RESUME | Same as start |
| CMTW_CMD_STOP | Pause the timer |
| CMTW_CMD_RESTART | Zero the counter then activate the timer |

### Return Values

| | |
|---|---|
| CMTW_SUCCESS | Successful, the command is executed |
| CMTW_ERR_BAD_CHAN | Invalid channel number |
| CMTW_ERR_CH_NOT_ENABLED | Channel is disabled by user configuration |
| CMTW_ERR_CH_NOT_OPENED | Channel currently closed, perform R_CMTW_Open() first |
| CMTW_ERR_CH_NOT_RUNNIG | Channel currently not started - perform R_CMTW_Control() to start |
| CMTW_ERR_CH_NOT_STOPPED | Channel currently running - perform R_CMTW_Control() to stop |
| CMTW_ERR_UNKNOWN_CMD | Invalid command |
| CMTW_ERR_LOCK | The lock could not be acquired, the channel is busy |

### Properties

Prototyped in file “r_cmtw_rx_if.h”

### Description

This function starts or resumes a CMTW channel that is in the open state. Once successfully completed, the status of the selected CMTW channel will be set to "running". Stop command pauses a channel that is in a running state. In this state the timer registers retain their values immediately before executing the stop command. Timer operations can be continued by start or resume commands. The restart command clears the timer counter and resumes the operations.

### Example 1

This example starts channel 0 that is in the open state.

```
/* Open the timer */
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}

/* And start it */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_START);

if (CMTW_SUCCESS != rc)
{
```

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

**Example 2**

This example stops channel 0 that is in the running state.

```
/* Open the timer */
rc = R_CMTW_Open(CMTW_CHANNEL_0, &ch, &cb);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}

/* And start it */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_START);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}

/* Now stop it */
rc = R_CMTW_Control(CMTW_CHANNEL_0, CMTW_CMD_STOP);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}
```

**Special Notes:**

None

## R_CMTW_Close()

Disables the specified CMTW channel and powers it down. The channel moves to the closed state.

### Format

```
cmtw_err_t  R_CMTW_Close(cmtw_channel_t channel);
```

### Parameters

*channel*

Number of the CMTW channel to close

### Return Values

| | |
|---|---|
| CMTW_SUCCESS | Successful, channel is closed |
| CMTW_ERR_BAD_CHAN | Invalid channel number |
| CMTW_ERR_CH_NOT_ENABLED | Channel is disabled by user configuration |
| CMTW_ERR_CH_NOT_OPENED | Channel currently closed, perform R_CMTW_Open() first |
| CMTW_ERR_LOCK | The lock could not be acquired, the channel is busy |

### Properties

Prototyped in file “r_cmtw_rx_if.h”

### Description

This function stops and disables a CMTW channel that is in the open or running state. Once successfully completed, the status of the selected CMTW channel will be set to "stopped" and the channel is powered down to reduce power consumption. The channel cannot be used again until it has been reopened with the R_CMTW_Open() function.

### Example 1

This example closes channel 0.

```
/* Close the timer */
rc = R_CMTW_Close(CMTW_CHANNEL_0);

if (CMTW_SUCCESS != rc)
{
/* Handle the error */
}
```

### Special Notes:

None

## R_CMTW_GetVersion()

This function returns the driver version number at runtime.

### Format

```
uint32_t    R_CMTW_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_cmtw_rx_if.h”

### Description

This 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

This example shows how this function may be used.

```
/* Retrieve the version number and convert it to a string. */
uint32_t version, major, minor;
char version_str[11];

version = R_CMTW_GetVersion();

major = (version >> 16)&0xf;
minor =  version & 0xff;

sprintf(version_str, "CMTW v%1.0hu.%2.2hu", major, minor);
```

### Special Notes:

None

# 4. Pin Setting

To use the CMTW FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC). The pin assignment is referred to as the “Pin Setting” in this document. Please perform the pin setting after calling the R_CMTW_Open() function.

When performing the Pin Setting in the e$^2$ studio, the Pin Setting feature of the FIT configurator or the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the FIT configurator or the Smart Configurator. Pins are configured by calling the function defined in the source file.

# 5. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g.. r_bsp). The standard naming convention for the demo project is <module>_demo_<board> where <module> is the peripheral acronym (e.g. s12ad, cmt, sci) and the <board> is the standard RSK (e.g. rskrx113). For example, s12ad FIT module demo project for RSKRX113 will be named as s12ad_demo_rskrx113. Similarly the exported .zip file will be <module>_demo_<board>.zip. For the same example, the zipped export/import file will be named as s12ad_demo_rskrx113.zip

## 5.1 cmtw_demo_rskrx64m

### Description

This is a simple demo of the RX64M Compare Match Timer Wide (CMTW) for the RSKRX64M starter kit (FIT module "r_cmtw_rx"). The demo configures channel 0 for a 500 ms compare match operation with interrupt callback, and channel 1 is configured for a 1 sec compare match operation with interrupt callback. When a compare match interrupt is generated by channel 0, the interrupt callback function is called and LED 0 is toggled (every 500 ms). When a compare match interrupt is generated by channel 1, the interrupt callback function is called and LED 1 is toggled (every 1 sec).

### Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If PC stops at Main, press F8 to resume.
3. Set breakpoints and watch global variables

### Boards Supported

RSKRX64M

## 5.2 cmtw_demo_rskrx71m

### Description

This is a simple demo of the RX71M Compare Match Timer Wide (CMTW) for the RSKRX71M starter kit (FIT module "r_cmtw_rx"). The demo configures channel 0 for a 500 ms compare match operation with interrupt callback, and channel 1 is configured for a 1 sec compare match operation with interrupt callback. When a compare match interrupt is generated by channel 0, the interrupt callback function is called and LED 0 is toggled (every 500 ms). When a compare match interrupt is generated by channel 1, the interrupt callback function is called and LED 1 is toggled (every 1 sec).

### Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If PC stops at Main, press F8 to resume.
3. Set breakpoints and watch global variables

### Boards Supported

RSKRX71M

## 5.3 cmtw_demo_rskrx65n

### Description

This is a simple demo of the RX65N Compare Match Timer Wide (CMTW) for the RSKRX65N starter kit (FIT module "r_cmtw_rx"). The demo configures channel 0 for a 500 ms compare match operation with interrupt callback, and channel 1 is configured for a 1 sec compare match operation with interrupt callback. When a compare match interrupt is generated by channel 0, the interrupt callback function is called and LED 0 is toggled (every 500 ms). When a compare match interrupt is generated by channel 1, the interrupt callback function is called and LED 1 is toggled (every 1 sec).

### Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If PC stops at Main, press F8 to resume.
3. Set breakpoints and watch global variables

### Boards Supported

RSKRX65N

## 5.4 cmtw_demo_rskrx65n_2m

### Description

This is a simple demo of the RX65N-2MB Compare Match Timer Wide (CMTW) for the RSKRX65N-2MB starter kit (FIT module "r_cmtw_rx"). The demo configures channel 0 for a 500 ms compare match operation with interrupt callback, and channel 1 is configured for a 1 sec compare match operation with interrupt callback. When a compare match interrupt is generated by channel 0, the interrupt callback function is called and LED 0 is toggled (every 500 ms). When a compare match interrupt is generated by channel 1, the interrupt callback function is called and LED 1 is toggled (every 1 sec).

### Setup and Execution

1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If PC stops at Main, press F8 to resume.
3. Set breakpoints and watch global variables

### Boards Supported

RSKRX65N-2MB

## 5.5 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”.

## 5.6 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 Brower >> Application Notes* tab.

# 6. Appendices

## 6.1 Operation Confirmation Environment

This section describes operation confirmation environment for the CMTW FIT module.

**Table 6.1 Confirmed Operation Environment (Rev.2.20)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 7.7.0<br>IAR Embedded Workbench for Renesas RX 4.12.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 4.8.4.201902<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99<br>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:<br>-Wl,--no-gc-sections<br>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 4.12.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.20 |
| Board used | Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx) |

**Table 6.2 Confirmed Operation Environment (Rev.2.10)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 7.5.0<br>IAR Embedded Workbench for Renesas RX 4.12.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 4.8.4.201902<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99<br>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:<br>-Wl,--no-gc-sections<br>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 4.12.1<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.10 |
| Board used | Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx) |

**Table 6.3 Confirmed Operation Environment (Rev.2.00)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ studio Version 7.4.0<br>IAR Embedded Workbench for Renesas RX 4.10.1 |
| C compiler | Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-lang = c99 |
| | GCC for Renesas RX 4.8.4.201803<br>Compiler option: The following option is added to the default settings of the integrated development environment.<br>-std=gnu99<br>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:<br>-Wl,--no-gc-sections<br>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<br>Compiler option: The default settings of the integrated development environment. |
| Endian | Big endian/little endian |
| Revision of the module | Rev.2.00 |
| Board used | Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565Nxxxxxxxxx) |

**Table 6.4 Operation Confirmation Environment (Rev.1.32)**

| 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.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.1.32 |
| Board used | Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2CxxxxxBR) |

**Table 6.5 Operation Confirmation Environment (Rev.1.31)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ 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.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.1.31 |
| Board used | Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2CxxxxxBR) |

**Table 6.6 Operation Confirmation Environment (Rev.1.30)**

| Item | Contents |
|---|---|
| Integrated development environment | Renesas Electronics e$^2$ 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.<br>-lang = c99 |
| Endian | Big endian/little endian |
| Revision of the module | Rev.1.30 |
| Board used | Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2CxxxxxBR) |

## 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 a 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_cmtw_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_cmtw_rx_config.h" may be wrong. Check the file "r_cmtw_rx_config.h". If there is a wrong setting, set the correct value for that. Refer to 2.9 Configuration Overview for details.

(4) Q: The expected waveform is not output.

A: The pin setting may not be performed correctly. When using this FIT module, the pin setting must be performed. Refer to 4. Pin Setting for details.

# Related Technical Updates

This module reflects the content of the following technical updates.

None

# Revision History

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Sept 3, 2014 | -- | First edition issued |
| 1.10 | Mar 2, 2015 | -- | Added support for the RX71M Group |
| 1.20 | Oct 1, 2016 | -- | Added support for the RX65N Group |
| | | 8 | Changed the tabular format of Code Size.<br>Updated the Code Size table for the RX65N Group. |
| | | 21 | Added “4. Pin Setting”. |
| | | 23 | Added “Related Technical Updates”. |
| 1.30 | Jul 21, 2017 | -- | Added support for the RX65N-2MB. |
| | | 6 | Added RXC v2.06.00, RXC v2.07.00 to “2.5 Supported Toolchains” |
| | | 7 | Added “2.6 Interrupt Vector” |
| | | 13 | Updated “2.13 Adding the FIT Module to Your Project” |
| | | 22 | Updated “4. Pin Setting”. |
| 1.31 | Oct 31, 2017 | 24 | Added “5.3 cmtw_demo_rskrx65n” |
| | | 24 | Added “5.4 cmtw_demo_rskrx65n_2m” |
| | | 24 | Added “5.6 Downloading Demo Projects” |
| | | 25 | Added “6. Appendices” |
| 1.32 | Nov 16, 2018 | -- | Added document number in XML |
| | | 6 | Added RXC v3.01.00 to “2.5 Supported Toolchains” |
| | | 13 | Updated “2.13 Adding the FIT Module to Your Project” |
| | | 14 | Added “2.14 “for”, “while” and “do while” statements” |
| | | 26 | Added table for Rev.1.32 |
| 1.40 | Feb 01, 2019 | 8 | Corrected interrupt priority level macro names |
| | | 16-22 | Removed ‘Reentrant’ description in each API function. |
| 2.00 | May.20.19 | — | Supported the following compilers:<br>- GCC for Renesas RX<br>- IAR C/C++ Compiler for Renesas RX |
| | | 1 | Added the section of Target compilers.<br>Deleted related documents. |
| | | 6 | 2.3 Software Requirements<br>Requires r_bsp v5.20 or higher |
| | | 9 | Updated the section of 2.10 Code Size |
| | | 26 | Table 6.1 Confirmed Operation Environment:<br>Added table for Rev.2.00 |
| | | 29 | Deleted the section of Website and Support. |
| | | Program | Changed below for support GCC and IAR compiler:<br>1. Deleted the inline expansion of the R_CMTW_GetVersion function.<br>2. Replaced evenaccess with the macro definition of BSP.<br>3. Replaced the declaration of interrupt functions with the macro definition of BSP. |

| | | | |
|---|---|---|---|
| 2.10 | Aug.15.19 | 1, 7 | Added support for RX72M |
| | | 10 | Added code size corresponding to RX72M |
| | | 26 | 6.1 Confirmed Operation Environment:<br>Added Table for Rev.2.10 |
| | | Program | Added support for RX72M. |
| 2.20 | Dec.30.19 | 1, 7 | Added support for RX66N, RX72N |
| | | 6 | 2.4 Limitations<br>Added limitations. |
| | | 10 | Added code size corresponding to RX66N, RX72N |
| | | 25 | 6.1 Confirmed Operation Environment:<br>Added Table for Rev.2.20 |
| | | Program | Added support for RX66N, RX72N. |

# 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

## 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/.

## Trademarks

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