

RX Family

CMTW FIT to FSP Migration Guide

Introduction

This application note provides a comprehensive guide on migrating an existing application that uses the CMTW Firmware Integration Technology (FIT) module to the CMTW FSP module (Renesas Flexible Software Package). It includes detailed instructions, migration steps, and demonstrations to assist users throughout the migration process.

Target Modules

- FIT_Module: r_cmtw_rx
- FSP Modules: r_cmtw

Contents

1. Overview	3
1.1 Stack Diagram	3
1.2 API Function	3
2. Migrating Project FIT to FSP	5
2.1 Creating a new FSP project	5
2.2 Migrating Configuration	7
2.3 Generate Code	10
2.4 Migrating Application Code	12
2.5 Build and run the project.	15
3. Reference Documents	16
4. Appendices	16
4.1 FIT & FSP Environment	16
4.2 Reference FSP Application Code	16
4.3 Comparison Reference	19
4.3.1 Compare Functions (Details)	19
4.3.2 Compare Configuration (Details)	23
Revision History	25

1. Overview

1.1 Stack Diagram

The figure below illustrates the stack diagram of the CMTW FIT module and the CMTW FSP Module

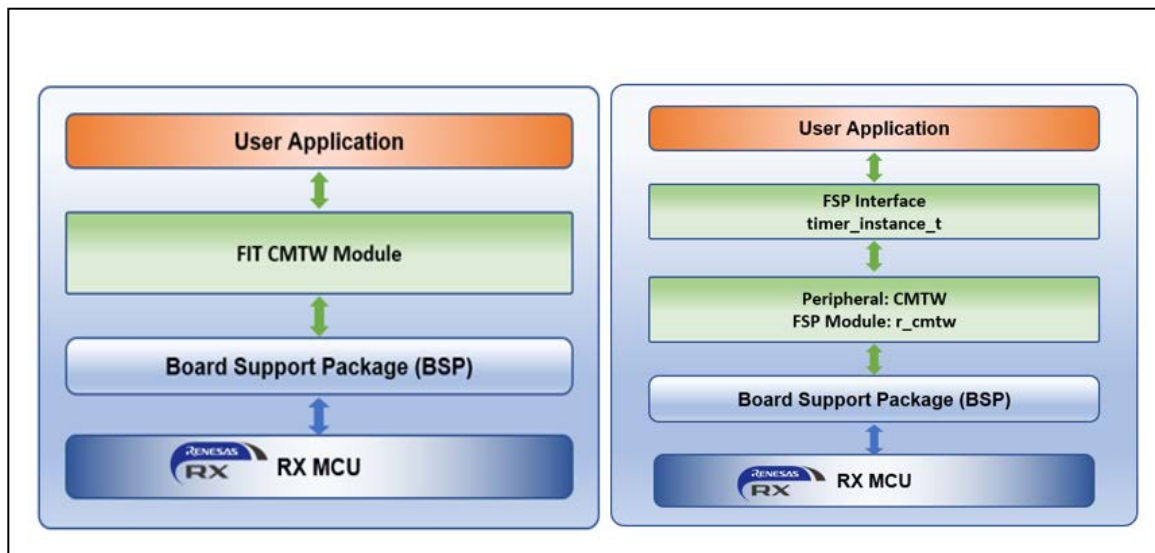


Figure 1-1 CMTW FIT and CMTW FSP stack diagram

CMTW FIT and CMTW FSP support standard timer functionalities, including one-shot and periodic modes.

1.2 API Function

This section summarizes the API Functions of the CMTW FIT and CMTW FSP module.

FIT	FSP
R_CMTW_Open	open
R_CMTW_Control	stop
Use <i>CMTW_CMD_STOP</i> to configure and control	
R_CMTW_Control	start
Use <i>CMTW_CMD_START</i> or <i>CMTW_CMD_RESUME</i> to configure and control	
R_CMTW_Control	reset
Use <i>CMTW_CMD_RESTART</i> to configure and control	
R_CMTW_Close	close
Callback function set in R_CMTW_Open	callbackSet

FIT	FSP
Not supported	periodSet
	infoGet
	statusGet
	enable
	disable
	dutyCycleSet
	compareMatchSet
R_CMTW_GetVersion	Not supported in specific modules FSP has common functions to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)

For more details, please refer to the sections:

[Compare Functions \(Details\)](#) : Comparison FIT and FSP.

[Migrate Application Code](#) : Example of using FSP CMTW API in an application.

2. Migrating Project FIT to FSP

2.1 Creating a new FSP project

Refer to the reference document “RX FSP Migration Guide” for creating RX FSP project.

To create a new project in e² Studio, navigate to the top bar menu and select:

File → New → Renesas C/C++ project → Renesas RX.

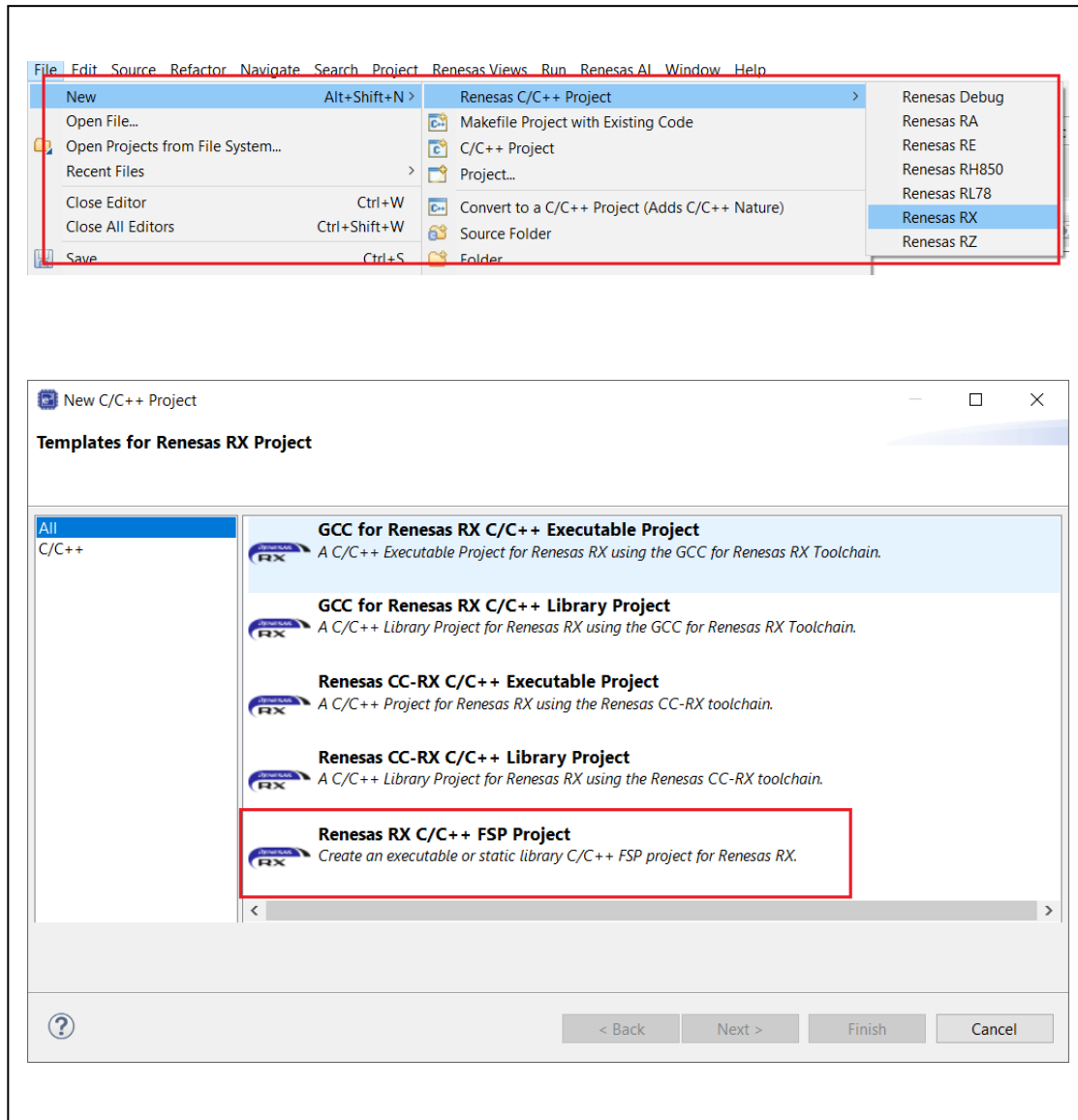


Figure 2-1 Create a new Renesas RX C/C++ FSP Project

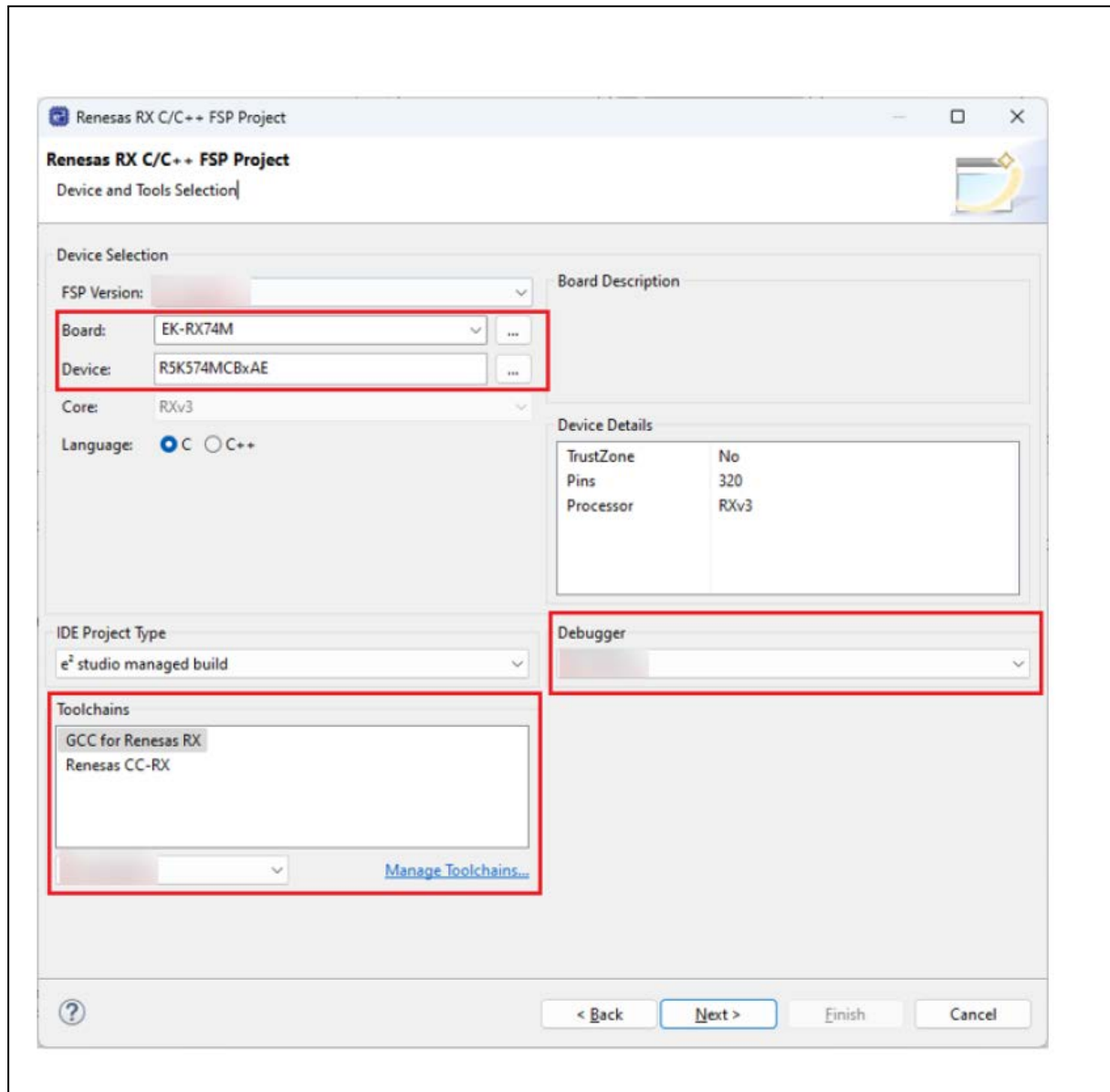


Figure 2-2 Select Board, Debugger and toolchains

2.2 Migrating Configuration

This section describes migration configuration

Configuration of Demo as shown in table below:

Configuration	Value
Channel	0 and 1
mode	periodic
period	500
Period Unit	ms
Compare Match interrupt Priority	Priority 5
Callback	CMTW_cb0 (channel 0) CMTW_cb1 (channel 1) (user's function name)

Steps of updating configuration of FSP project

Users need to add the “Timer, Compare Match Timer (r_cmtw)” in stack as below:

On stack tab, click on New stack→Timer→Timer, Compare Match Timer (r_cmtw)

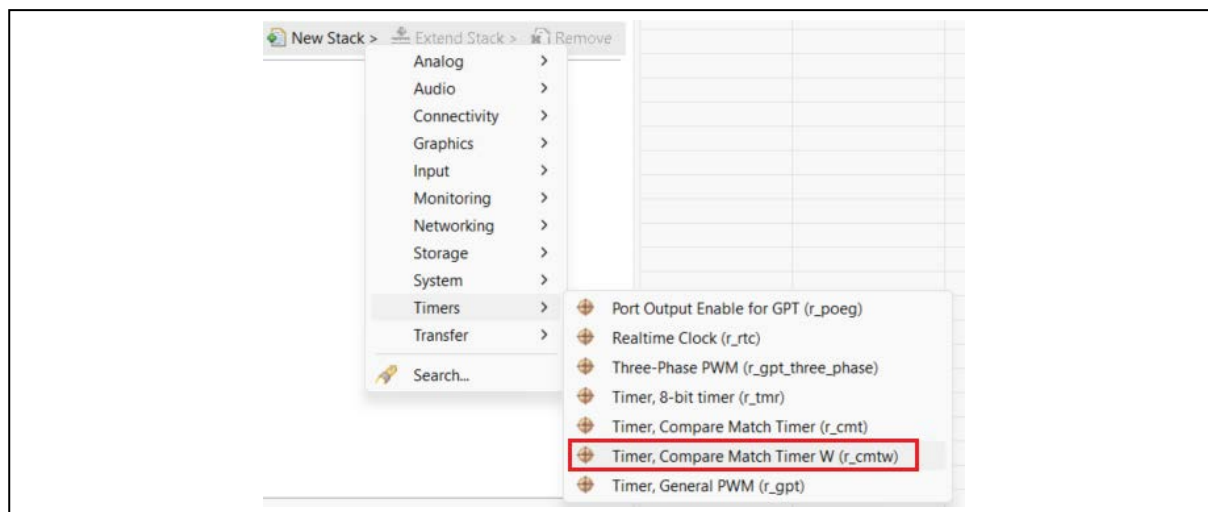


Figure 2-3 Add CMTW module in stack.

Note: The CMTW FIT function automatically finds an unused CMTW channel and configures it for the requested frequency. In this demonstration, two channels (CMTW0 and CMTW1) are used. When migrating to the CMTW FSP module, it is necessary to manually add two CMTW HAL modules—one for each channel (CMTW0 and CMTW1) and configure the settings for both HAL instances accordingly.

Click on the “**g_timer0 Timer, Compare Match Timer (r_cmtw)**” or “**g_timer1 Timer, Compare Match Timer (r_cmtw)**” module once it appears.

The “Properties” window will display its properties.

Configure CMTW FSP module according to the configuration of demo

The screenshot shows the Renesas IDE interface for configuring the CMTW FSP module. The 'Stacks Configuration' window is open, showing the 'HAL/Common Stacks' list. The 'g_timer0 Timer, Compare Match Timer W (r_cmtw)' module is selected and highlighted with a red box. Below this, the 'Properties' window for the selected module is open, showing a table of properties and their values. The 'Properties' window is also highlighted with a red box.

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer0 Timer, Compare Match Timer W (r_cmtw)	
General	
Name	g_timer0
Channel	0
Mode	Periodic
Period	500
Period Unit	Milliseconds
Counter Size	32-bit
Counter Clear Source	Compare Match
Output	
TOC0 Output	Disabled
TOC1 Output	Disabled
TOC0 Output Enable	Output Disable
TOC1 Output Enable	Output Disable
Input	
TIC0 Capture Trigger	Rising Edge
TIC1 Capture Trigger	Rising Edge
TIC0 Input Enable	Input Disable
TIC1 Input Enable	Input Disable
Interrupts	
Callback	cmtw_cb0
Compare Match Interrupt Priority	Priority 5
Input Capture 0 Interrupt Priority	Priority 5
Input Capture 1 Interrupt Priority	Priority 5
Output Compare 0 Interrupt Priority	Priority 5
Output Compare 1 Interrupt Priority	Priority 5
Pins	
CMTW0_TIC0	<unavailable>
CMTW0_TIC1	<unavailable>
CMTW0_TOC0	<unavailable>
CMTW0_TOC1	<unavailable>

Figure 2-4 Configure for CMTW FSP Channel 0

[CMTW_FSP_EK_RX74M] FSP Configuration X

Stacks Configuration

Generate Project

Threads New Thread Remove

- HAL/Common
 - g_ioport I/O Port (r_ioport)
 - g_timer0 Timer, Compare Match Timer W (r_cmtw)
 - g_timer1 Timer, Compare Match Timer W (r_cmtw)

HAL/Common Stacks New Stack > Extend Stack >

- g_ioport I/O Port (r_ioport)
- g_timer0 Timer, Compare Match Timer W (r_cmtw)
- g_timer1 Timer, Compare Match Timer W (r_cmtw)**

Objects New Object > Remove

Summary BSP Clocks Pins Interrupts Event Links Linker Sections **Stacks** Components

Problems Console Properties X Smart Browser Smart Manual Search Debug Renesas Debug Virtual Console

g_timer1 Timer, Compare Match Timer W (r_cmtw)

Settings

Property	Value
Common	
Parameter Checking	Default (BSP)
Module g_timer1 Timer, Compare Match Timer W (r_cmtw)	
General	
Name	g_timer1
Channel	1
Mode	Periodic
Period	500
Period Unit	Milliseconds
Counter Size	32-bit
Counter Clear Source	Compare Match
Output	
TOC0 Output	Disabled
TOC1 Output	Disabled
TOC0 Output Enable	Output Disable
TOC1 Output Enable	Output Disable
Input	
TIC0 Capture Trigger	Rising Edge
TIC1 Capture Trigger	Rising Edge
TIC0 Input Enable	Input Disable
TIC1 Input Enable	Input Disable
Interrupts	
Callback	cmtw_cb1
Compare Match Interrupt Priority	Priority 5
Input Capture 0 Interrupt Priority	Priority 5
Input Capture 1 Interrupt Priority	Priority 5
Output Compare 0 Interrupt Priority	Priority 5
Output Compare 1 Interrupt Priority	Priority 5
Pins	
CMTW1_TIC2	<unavailable>
CMTW1_TIC3	<unavailable>
CMTW1_TOC2	<unavailable>
CMTW1_TOC3	<unavailable>

Figure 2-5 Configure for CMTW FSP Channel 1

2.3 Generate Code

After completing stacks and HAL configurations and making user selections, click Generate Project Content as shown in Figure 2-5. This generates the configuration code for selected modules. You can find generated code using the Project Explorer window in e² studio. The configuration.xml file, containing all settings, can be edited in the GUI-based configuration editor for any necessary reconfigurations. The CMTW configuration can be found in **rx_gen/hal_data.c** as shown in Figure 2-6 This file will be created if it doesn't exist.

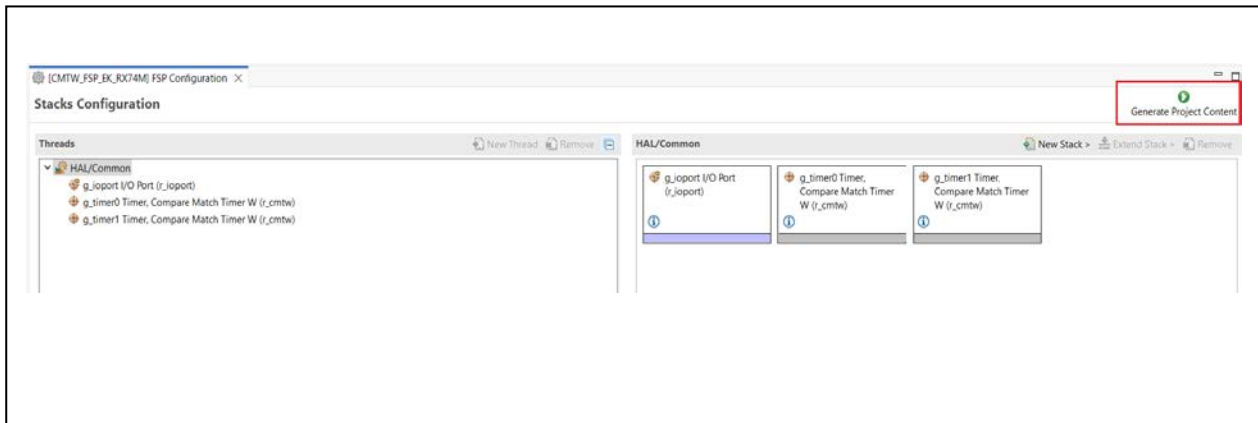


Figure 2-6 Generate Project Content

g_timer0 Timer, Compare Match Timer W (r_cmtw)

Settings	Property	Value
Common	Parameter Checking	Default (BSP)
	Module g_timer0 Timer, Compare Match Timer W (r_cmtw)	
General	Name	g_timer0
	Channel	0
	Mode	Periodic
	Period	500
	Period Unit	Milliseconds
Counter	Counter Size	32-bit
	Counter Clear Source	Compare Match
	Output	
TOC0 Output	TOC0 Output	Disabled
	TOC1 Output	Disabled
	TOC0 Output Enable	Output Disable
	TOC1 Output Enable	Output Disable
Input	TIC0 Capture Trigger	Rising Edge
	TIC1 Capture Trigger	Rising Edge
	TIC0 Input Enable	Input Disable
	TIC1 Input Enable	Input Disable
Interrupts	Callback	cmtw_cb0
	Compare Match Interrupt Priority	Priority 5
	Input Capture 0 Interrupt Priority	Priority 5
	Input Capture 1 Interrupt Priority	Priority 5
	Output Compare 0 Interrupt Priority	Priority 5

```

const cmtw_extended_cfg_t g_timer0_extnd =
{
    .toc0 = CMTW_OUTPUT_PIN_RETAIN,
    .toc1 = CMTW_OUTPUT_PIN_RETAIN,
    .toc0_control = CMTW_OUTPUT_CONTROL_DISABLED,
    .toc1_control = CMTW_OUTPUT_CONTROL_DISABLED,
    .capture_ic0_source = CMTW_SOURCE_EDGE_RISING,
    .capture_ic1_source = CMTW_SOURCE_EDGE_RISING,
    .ic0_control = CMTW_INPUT_CONTROL_DISABLED,
    .ic1_control = CMTW_INPUT_CONTROL_DISABLED,
    .clear_source = CMTW_CLEAR_SOURCE_COMPARE_MATCH_CMCOR,
    .counter_size = TIMER_VARIANT_32_BIT,

    .capture_ic0_ipl = (5),
    .capture_ic1_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR0)
    .capture_ic0_irq = VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR0,
    #else
    .capture_ic0_irq = FSP_INVALID_VECTOR,
    #endif
    #if defined(VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR1)
    .capture_ic1_irq = VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR1,
    #else
    .capture_ic1_irq = FSP_INVALID_VECTOR,
    #endif
    .compare_oc0_ipl = (5),
    .compare_oc1_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR0)
    .compare_oc0_irq = VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR0,
    #else
    .compare_oc0_irq = FSP_INVALID_VECTOR,
    #endif
    #if defined(VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR1)
    .compare_oc1_irq = VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR1,
    #else
    .compare_oc1_irq = FSP_INVALID_VECTOR,
    #endif
};

const timer_cfg_t g_timer0_cfg =
{
    .mode = TIMER_MODE_PERIODIC,
    /* Actual period: 0.5 seconds. */ .period_counts = (uint32_t) 0x47868c,
    .source_div = (timer_source_div_t) 3, .channel = 0, .p_callback = cmtw_cb0,
    #if defined(NULL)
    .p_context = NULL,
    #else
    .p_context = &NULL,
    #endif
    .p_extnd = &g_timer0_extnd,
    .cycle_end_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_CMTW0_CMTI)
    .cycle_end_irq = VECTOR_NUMBER_CMTW0_CMTW0_CMTI,
    #else
    .cycle_end_irq = FSP_INVALID_VECTOR,
    #endif
};

/* Instance structure to use this module. */
const timer_instance_t g_timer0 =
{
    .p_ctrl = &g_timer0_ctrl, .p_cfg = &g_timer0_cfg, .p_api = &g_timer_on_cmtw };

```

g_timer1 Timer, Compare Match Timer W (r_cmtw)

Settings	Property	Value
Common	Parameter Checking	Default (BSP)
	Module g_timer1 Timer, Compare Match Timer W (r_cmtw)	
General	Name	g_timer1
	Channel	1
	Mode	Periodic
	Period	500
	Period Unit	Milliseconds
Counter	Counter Size	32-bit
	Counter Clear Source	Compare Match
	Output	
TOC0 Output	TOC0 Output	Disabled
	TOC1 Output	Disabled
	TOC0 Output Enable	Output Disable
	TOC1 Output Enable	Output Disable
Input	TIC0 Capture Trigger	Rising Edge
	TIC1 Capture Trigger	Rising Edge
	TIC0 Input Enable	Input Disable
	TIC1 Input Enable	Input Disable
Interrupts	Callback	cmtw_cb1
	Compare Match Interrupt Priority	Priority 5
	Input Capture 0 Interrupt Priority	Priority 5
	Input Capture 1 Interrupt Priority	Priority 5
	Output Compare 0 Interrupt Priority	Priority 5

```

cmtw_instance_ctrl_t g_timer1_ctrl;
const cmtw_extended_cfg_t g_timer1_extnd =
{
    .toc0 = CMTW_OUTPUT_PIN_RETAIN,
    .toc1 = CMTW_OUTPUT_PIN_RETAIN,
    .toc0_control = CMTW_OUTPUT_CONTROL_DISABLED,
    .toc1_control = CMTW_OUTPUT_CONTROL_DISABLED,
    .capture_ic0_source = CMTW_SOURCE_EDGE_RISING,
    .capture_ic1_source = CMTW_SOURCE_EDGE_RISING,
    .ic0_control = CMTW_INPUT_CONTROL_DISABLED,
    .ic1_control = CMTW_INPUT_CONTROL_DISABLED,
    .clear_source = CMTW_CLEAR_SOURCE_COMPARE_MATCH_CMCOR,
    .counter_size = TIMER_VARIANT_32_BIT,

    .capture_ic0_ipl = (5),
    .capture_ic1_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR0)
    .capture_ic0_irq = VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR0,
    #else
    .capture_ic0_irq = FSP_INVALID_VECTOR,
    #endif
    #if defined(VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR1)
    .capture_ic1_irq = VECTOR_NUMBER_CMTW0_CAPTURE_CMCICR1,
    #else
    .capture_ic1_irq = FSP_INVALID_VECTOR,
    #endif
    .compare_oc0_ipl = (5),
    .compare_oc1_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR0)
    .compare_oc0_irq = VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR0,
    #else
    .compare_oc0_irq = FSP_INVALID_VECTOR,
    #endif
    #if defined(VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR1)
    .compare_oc1_irq = VECTOR_NUMBER_CMTW0_COMPARE_CMCOCR1,
    #else
    .compare_oc1_irq = FSP_INVALID_VECTOR,
    #endif
};

const timer_cfg_t g_timer1_cfg =
{
    .mode = TIMER_MODE_PERIODIC,
    /* Actual period: 0.5 seconds. */ .period_counts = (uint32_t) 0x47868c,
    .source_div = (timer_source_div_t) 3, .channel = 0, .p_callback = cmtw_cb1,
    #if defined(NULL)
    .p_context = NULL,
    #else
    .p_context = &NULL,
    #endif
    .p_extnd = &g_timer1_extnd,
    .cycle_end_ipl = (5),
    #if defined(VECTOR_NUMBER_CMTW0_CMTW0_CMTI)
    .cycle_end_irq = VECTOR_NUMBER_CMTW0_CMTW0_CMTI,
    #else
    .cycle_end_irq = FSP_INVALID_VECTOR,
    #endif
};

/* Instance structure to use this module. */
const timer_instance_t g_timer1 =
{
    .p_ctrl = &g_timer1_ctrl, .p_cfg = &g_timer1_cfg, .p_api = &g_timer_on_cmtw };

```

Figure 2-7 Generated files for CMTW

2.4 Migrating Application Code

The typical steps in using the CMTW HAL module in an application are:

1. Configure module by using FSP Configuration as described in section 2.2.
2. Initialize the module using the .open API.
3. Start the CMTW HAL module by calling the start API.
4. Respond to the timer callback as needed (user code)

```

void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;

    /* Initialize Timers Channel 0 */
    err = g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }

    /* Start Timers Channel 0 */
    err = g_timer0.p_api->start(g_timer0.p_ctrl);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }

    /* Initialize Timers Channel 1 */
    err = g_timer1.p_api->open(g_timer1.p_ctrl, g_timer1.p_cfg);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }

    /* Start Timers Channel 1 */
    err = g_timer1.p_api->start(g_timer1.p_ctrl);
    /* If there were an error this would demonstrate error detection of API calls. */
    if (FSP_SUCCESS != err)
    {
        /* This nop is for optimization suppression. */
        __asm volatile ("nop");
    }

    /* End of demo, spin here */
    while(1);
}

```

Figure 2-8 Overall CMTW FSP Application Code

Initialization

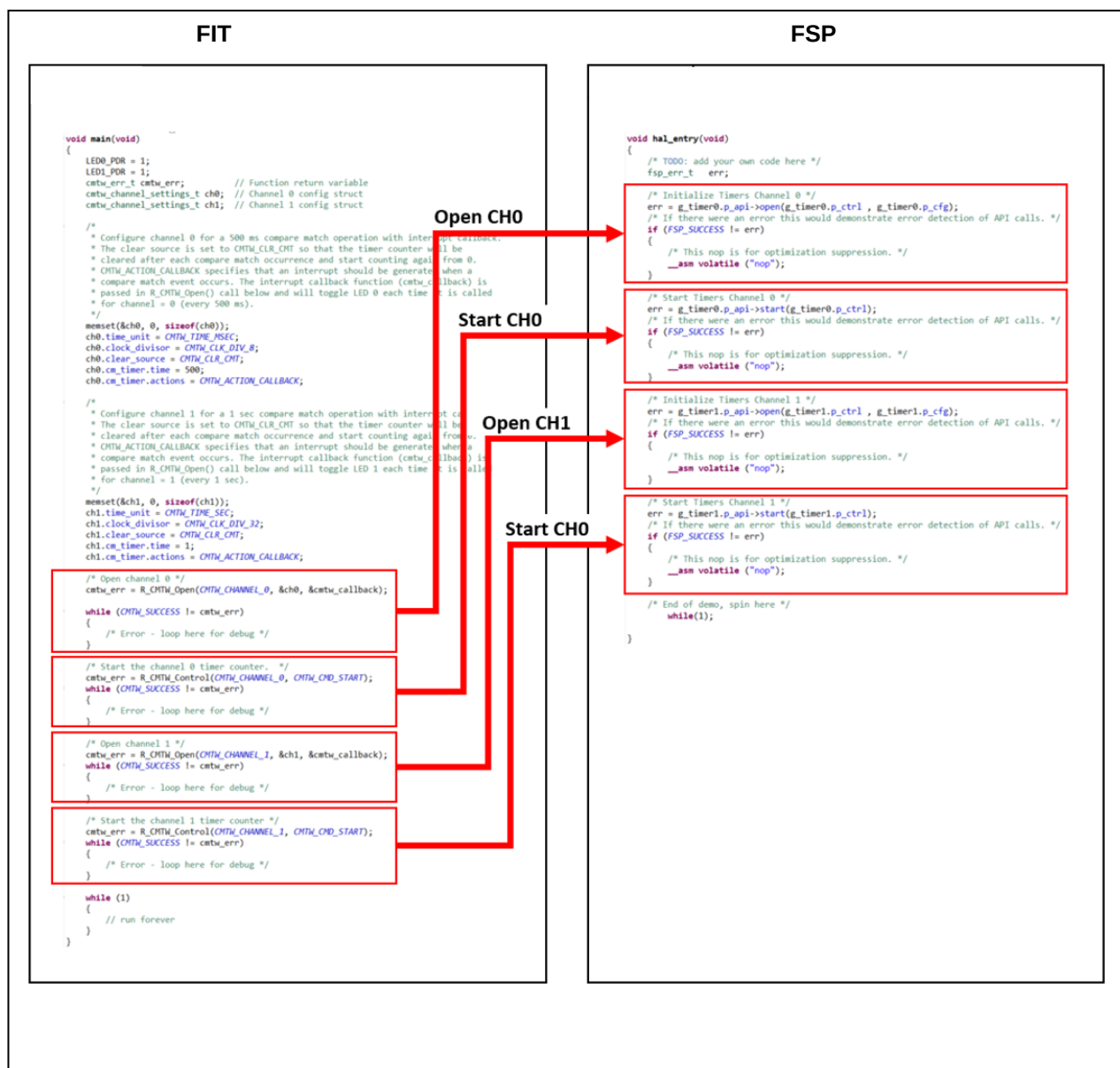


Figure 2-9 CMTW Initialization Code

Note: In CMTW FIT, the function `R_CMTW_CreatePeriodic()` finds an unused CMTW channel, configures it for the requested frequency, associates a user callback function with the timer's interrupt, and powers up and starts the timer immediately. In CMTW FSP, it is necessary to initialize the channel used during the configuration steps, as described in Section 2.2. After that, call `.open` to initialize the timer and apply configurations. Subsequently, call `.start` to start the timer.

Callback function

FIT	FSP
<pre> /* Function Name: cmtw_callback[] void cmtw_callback(void *pdata) { cmtw_callback_data_t cb_data; cmtw_callback_data_t *p_cb_data = (cmtw_callback_data_t *)pdata; /* Copy the callback data. The event/count fields * are not used here but are shown for reference. */ cb_data.channel = p_cb_data->channel; cb_data.event = p_cb_data->event; cb_data.count = p_cb_data->count; if (cb_data.channel == 0) { LED0 ^= LED_OFF; } if (cb_data.channel == 1) { LED1 ^= LED_OFF; } } </pre>	<pre> void cmtw_cb0(timer_callback_args_t *p_args) { bsp_leds_t leds = g_bsp_leds; // Toggle LED0 with each callback static volatile bsp_io_level_t led_level = BSP_IO_LEVEL_HIGH; led_level ^= BSP_IO_LEVEL_HIGH; R_BSP_PinAccessEnable(); R_BSP_PinWrite((bsp_io_port_pin_t)leds.p_leds[0], led_level); R_BSP_PinAccessDisable(); } /* Function Name : cmt_cb1[] void cmtw_cb1(timer_callback_args_t *p_args) { bsp_leds_t leds = g_bsp_leds; // Toggle LED1 with each callback static volatile bsp_io_level_t led_level = BSP_IO_LEVEL_HIGH; led_level ^= BSP_IO_LEVEL_HIGH; R_BSP_PinAccessEnable(); R_BSP_PinWrite((bsp_io_port_pin_t)leds.p_leds[1], led_level); R_BSP_PinAccessDisable(); } </pre>

Figure 2-10 CMTW Callback Code

Note: In CMTW FIT, when the callback function is called, the channel of the CMTW that triggered the callback is passed as an argument. Arguments are passed as void. Therefore, the callback function's argument is converted to a void pointer. See the examples below for reference.

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

```

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

    /* Copy the callback data. The event/count fields are not used here but are shown for reference. */
    cb_data.channel = p_cb_data->channel;
    cb_data.event = p_cb_data->event;
    cb_data.count = p_cb_data->count;

    if (cb_data.channel == 0)
    {
        LED0 ^= LED_OFF;
    }

    if (cb_data.channel == 1)
    {
        LED1 ^= LED_OFF;
    }
}

```

In FSP, when the callback function is called, the timer_callback_args_t is passed as the argument. The following shows Callback function parameter data

```

/** Callback function parameter data */
typedef struct st_timer_callback_args
{
    /** Placeholder for user data. Set in @ref timer_api_t::open function in @ref timer_cfg_t. */
    void const * p_context;
    timer_event_t event;          ///< The event can be used to identify what caused the callback.

    /** Most recent capture, only valid if event is TIMER_EVENT_CAPTURE_A or TIMER_EVENT_CAPTURE_B. */
    uint32_t capture;
} timer_callback_args_t;

```

For more details about FSP sample code, please refer to this section [Reference FSP Application Code](#)

2.5 Build and run the project.

1. Build the sample application, download it to the RSK board, and execute the application using a debugger.
2. The demo program will toggle the LED with a periodicity of 2 Hertz.

3. Reference Documents

RX Family CMTW Module Using Firmware Integration Technology(R01AN2199)
 Renesas Flexible Software Package (FSP)
 RX FSP Migration Guide

Technical Update/Technical News

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

4. Appendices

4.1 FIT & FSP Environment

Item	Contents
FIT Module	r_cmtw_rx v2.90
FSP Module	RX FSP v1.0.0 Module: r_cmtw
Board used	Renesas EK-RX74M (product No.: RTKEK74Mxxxxxxx)
Integrated development environment	Renesas Electronics e ² studio
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family GCC for Renesas RX

4.2 Reference FSP Application Code

Add this FSP sample code to file "hal_entry.c"

```
#include "hal_data.h"

FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

extern bsp_leds_t g_bsp_leds;

/*****
*****//**
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function
 * is called by main() when no RTOS is used.
*****
*****/

void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err;

    /* Initialize Timers Channel 0 */
    err = g_timer0.p_api->open(g_timer0.p_ctrl, g_timer0.p_cfg);
    /* If there were an error this would demonstrate error detection of API
    calls. */
    if (FSP_SUCCESS != err)
    {
        /* This nop is for optimization suppression. */
    }
}
```

```

    __asm volatile ("nop");
}

/* Start Timers Channel 0 */
err = g_timer0.p_api->start(g_timer0.p_ctrl);
/* If there were an error this would demonstrate error detection of API
calls. */
if (FSP_SUCCESS != err)
{
    /* This nop is for optimization suppression. */
    __asm volatile ("nop");
}

/* Initialize Timers Channel 1 */
err = g_timer1.p_api->open(g_timer1.p_ctrl , g_timer1.p_cfg);
/* If there were an error this would demonstrate error detection of API
calls. */
if (FSP_SUCCESS != err)
{
    /* This nop is for optimization suppression. */
    __asm volatile ("nop");
}

/* Start Timers Channel 1 */
err = g_timer1.p_api->start(g_timer1.p_ctrl);
/* If there were an error this would demonstrate error detection of API
calls. */
if (FSP_SUCCESS != err)
{
    /* This nop is for optimization suppression. */
    __asm volatile ("nop");
}

/* End of demo, spin here */
while(1);
}

void cmtw_cb0(timer_callback_args_t *p_args)
{
    bsp_leds_t leds = g_bsp_leds;
    // Toggle LED0 with each callback
    static volatile bsp_io_level_t led_level = BSP_IO_LEVEL_HIGH;
    led_level ^= BSP_IO_LEVEL_HIGH;

    R_BSP_PinAccessEnable();
    R_BSP_PinWrite((bsp_io_port_pin_t)leds.p_leds[0], led_level);
    R_BSP_PinAccessDisable();
}

/*****
* Function Name : cmt_cb1
* Description   : This is the callback function that was assigned to the CMT
*                  driver in main. It is called by the CMT driver at a 2 Hz rate
*                  and toggles LED0 with each call generating 1Hz blink rate.
* Arguments      : pdata - pointer to void
* Return value   : none
*****/
void cmtw_cb1(timer_callback_args_t *p_args)
{
    bsp_leds_t leds = g_bsp_leds;

```

```

// Toggle LED1 with each callback
static volatile bsp_io_level_t led_level = BSP_IO_LEVEL_HIGH;
led_level ^= BSP_IO_LEVEL_HIGH;

R_BSP_PinAccessEnable();
R_BSP_PinWrite((bsp_io_port_pin_t)leds.p_leds[1], led_level);
R_BSP_PinAccessDisable();
}

/*****
*****//**
* This function is called at various points during the startup process. This
implementation uses the event that is
* called right before main() to set up the pins.
*
* @param[in] event Where at in the start up process the code is currently
at
*****
*****/
void R_BSP_WarmStart(bsp_warm_start_event_t event)
{
    if (BSP_WARM_START_RESET == event)
    {
#ifdef BSP_FEATURE_FLASH_LP_VERSION != 0

        /* Enable reading from data flash. */
        R_FACI_LP->DFLCTL = 1U;

        /* Would normally have to wait tDSTOP(6us) for data flash recovery.
Placing the enable here, before clock and
        * C runtime initialization, should negate the need for a delay since
the initialization will typically take more than 6us. */
#endif
    }

    if (BSP_WARM_START_POST_C == event)
    {
        /* C runtime environment and system clocks are setup. */

        /* Configure pins. */
        R_IOPORT_Open (&IOPORT_CFG_CTRL, &IOPORT_CFG_NAME);
    }
}

```

4.3 Comparison Reference

4.3.1 Compare Functions (Details)

FIT API		FSP API	
Public API		FSP Interface API Public API	FSP Peripheral API Private API
R_CMTW_Open		open	R_CMTW_Open
R_CMTW_Control		stop	R_CMTW_Stop
Use <i>CMTW_CMD_STOP</i> to configure and control			
R_CMTW_Control		start	R_CMTW_Start
Use <i>CMTW_CMD_START</i> or <i>CMTW_CMD_RESUME</i> to configure and control			
R_CMTW_Control		reset	R_CMTW_Reset
Use <i>CMTW_CMD_RESTART</i> to configure and control			
R_CMTW_Close		close	R_CMTW_Close
R_CMTW_Open		callbackSet	R_CMTW_CallbackSet
Not supported		periodSet	R_CMTW_PeriodSet
		infoGet	R_CMTW_InfoGet
		statusGet	R_CMTW_StatusGet
		enable	R_CMTW_Enable
		disable	R_CMTW_Disable
		dutyCycleSet	Not supported
		compareMatchSet	
Not supported		Not supported Users can call private API directly, as no public API is provided for this function.	R_CMTW_OutputEnable
			R_CMTW_OutputDisable

FIT API		FSP API	
Public API		FSP Interface API Public API	FSP Peripheral API Private API
R_CMTW_GetVersion		Not supported in specific modules FSP has common functions to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)	Not supported in specific modules

Compare API functions

FSP Timer interfaces are an instance of FSP general Timer functions.

API Functions

Open Function	
Description	Initializes the timer module and applies configurations.
FIT	R_CMTW_Open (cmtw_channel_t channel, cmtw_channel_settings_t *pconfig, void (* const pcallback)(void *pdata))
FSP	.open (timer_ctrl_t * const p_ctrl, timer_cfg_t const * const p_cfg);

Stop Function	
Description	Stop the timers of the CMTW channel.
FIT	R_CMTW_Control (cmtw_channel_t channel, cmtw_cmd_t cmd) Use the command of CMTW_RX_CMD_STOP is equivalent to FSP's .start.
FSP	.close (timer_ctrl_t * const p_ctrl);

Start Function	
Description	Start the timer without resetting the counter to zero
FIT	R_CMTW_Control (cmtw_channel_t channel, cmtw_cmd_t cmd) Use the command of CMTW_RX_CMD_RESUME or CMTW_CMD_START is equivalent to FSP's .start.
FSP	.start (timer_ctrl_t * const p_ctrl);

Close Function	
Description	Stop and disable a CMTW channel that is in the open or running state.
FIT	R_CMTW_Close (cmtw_channel_t channel)
FSP	.close (timer_ctrl_t * const p_ctrl);

Restart Function	
Description	Reset the timer to zero
FIT	R_CMTW_Control (uint32_t channel, CMTW_commands_t command, void * pdata) Use the command of CMTW_RX_CMD_RESTART is equivalent to FSP's .reset. Note : the timer starts running automatically after the reset
FSP	.reset (timer_ctrl_t * const p_ctrl); Note : the timer doesn't starts running automatically after the reset. To initiate the timer in FSP, you need to call the .start function.

Callback Set	
Description	FIT The callback function in is specified during the creating operation FSP User can specify the callback function in the FSP Configuration. The callback can be updated, and there is an option to provide memory for the callback structure using the .callbackSet function.
FIT	R_CMTW_Open (cmtw_channel_t channel, cmtw_channel_settings_t *pconfig, void (* const pcallback)(void *pdata))
FSP	.callbackSet (timer_ctrl_t * const p_api_ctrl, void (* p_callback)(timer_callback_args_t *), void const * const p_context, timer_callback_args_t * const p_callback_memory);

Additional function

Get Version Function	
Description	Return the driver version number at runtime.
FIT	R_CMTW_GetVersion (void);
FSP	-

Enable Function	
Description	Enables external event triggers that start, stop, clear, or capture the counter.
FIT	-
FSP	.enable (timer_ctrl_t * const p_ctrl);

Disable Function	
Description	Disables external events triggers that start, stop, clear, or capture the counter.
FIT	-
FSP	.disable (timer_ctrl_t * const p_ctrl);

Period Set Function	
Description	Updates new period immediately and the counter is reset to 0.
FIT	-
FSP	.periodSet (timer_ctrl_t * const p_ctrl, uint32_t const period);

Enable output pin function	
Description	Enable output for TOC0 and/or TOC1 Note: Users are required to call the private API directly because RX FSP does not offer a corresponding public API.
FIT	-
FSP	R_CMTW_OutputEnable (timer_ctrl_t * const p_ctrl, cmtw_io_pin_t pin)

Disable output pin function	
Description	Disable output for TOC0 and/or TOC1 Note: Users are required to call the private API directly because RX FSP does not offer a corresponding public API.
FIT	-
FSP	R_CMTW_OutputDisable (timer_ctrl_t * const p_ctrl, cmtw_io_pin_t pin)

Get timer information Function	
Description	Gets timer information and stores it in provided pointer p_info.
FIT	-
FSP	.infoGet (timer_ctrl_t * const p_ctrl, timer_info_t * const p_info);

Get timer status Function	
Description	Retrieves the current state and counter value stores them in p_status
FIT	-
FSP	.statusGet (timer_ctrl_t * const p_ctrl, timer_status_t * const p_status);

4.3.2 Compare Configuration (Details)

FIT Configuration	FSP Configuration
Parameter checking	Common> Parameter Checking
Locking function for CMTW APIs	Not supported
Use channel 0	
Use channel 1	
>interrupt priority level for Compare Match CMTWx >interrupt priority level for Input Capture 0 CMTWx >interrupt priority level for Input Capture 1 CMTWx >interrupt priority level for Output Compare 0 CMTWx >interrupt priority level for Output Compare 1 CMTWx (x : 0 to 1)	Module g_timerx Timer, Compare Match Timer (r_cmtw) > Interrupts > Compare Match Interrupt Priority > Interrupts > Input Capture 0 Interrupt Priority > Interrupts > Input Capture 1 Interrupt Priority > Interrupts > Output Compare 0 Interrupt Priority > Interrupts > Output Compare 1 Interrupt Priority (x : 0 to 1)
Using CMTW nested interrupt	Not supported
Not supported in GUI User configured this variable in application code	Module g_timerx Timer, Compare Match Timer (r_cmtw) > Interrupts > Callback
Not supported in GUI User configured this variable in application code • Mode • Period • Period unit • Counter Size • Counter Clear Source	Module g_timerx Timer, Compare Match Timer (r_cmtw) >General > Channel >General > Mode >General > period >General > period Unit >General > Counter Size >General > Counter Clear Source (x : 0 to 1)
Not supported in GUI User configured this variable in application code • TOC0 Output • TOC1 Output • TIC0 Capture Trigger • TIC1 Capture Trigger	Module g_timerx Timer, Compare Match Timer (r_cmtw) >Output > TOC0 Output >Output > TOC1 Output > Input > TIC0 Capture Trigger > input > TIC1 Capture Trigger

Not supported	Module g_timerx Timer, Compare Match Timer (r_cmtw) >Output > TOC0 Output Enable >Output > TOC1 Output Enable > Input > TIC0 Capture Enable > input > TIC1 Capture Enable
---------------	---

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Sep 30, 2026	-	First edition issued

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.