

RX Family

WDT FIT to FSP Migration Guide

Introduction

This application note provides a comprehensive guide on migrating an existing application that uses the WDT Firmware Integration Technology (FIT) module to the WDT 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_wdt_rx
- FSP Module: r_wdt

Contents

1. Overview	3
1.1 Stack Diagram	3
1.2 API Functions	4
2. Migrating Project FIT to FSP.....	5
2.1 Creating a new FSP Project.....	5
2.2 Migrating Configuration	6
2.3 Generate Code	9
2.4 Migrating Application Code	10
2.5 Build and Run the project.	13
3. Reference Documents	14
4. Appendices.....	15
4.1 FIT & FSP Environment	15
4.2 Reference FSP Application Code.....	16
4.3 Comparison Reference	22
4.3.1 Compare Functions (Details).....	22
4.3.2 Compare Configuration (Details).....	24
Revision History	25

1. Overview

1.1 Stack Diagram

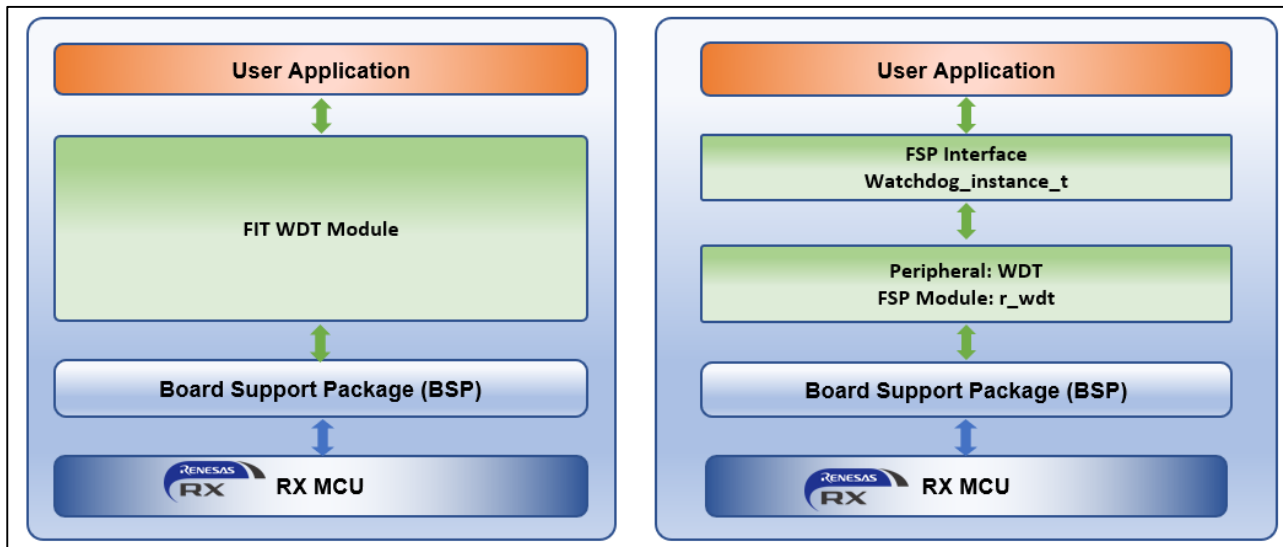


Figure 1-1 WDT FIT and WDT FSP Stack Diagram

1.2 API Functions

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

FIT	FSP
R_WDT_Open	open
R_WDT_Control Use Commands to configure and control.	refresh statusGet counterGet
R_WDT_GetVersion	Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)
Not supported	statusClear
Not supported	timeoutGet
Not supported	callbackSet

For more details, please refer to the sections:

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

[Migrate Application Code](#): example of using FSP WDT 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.

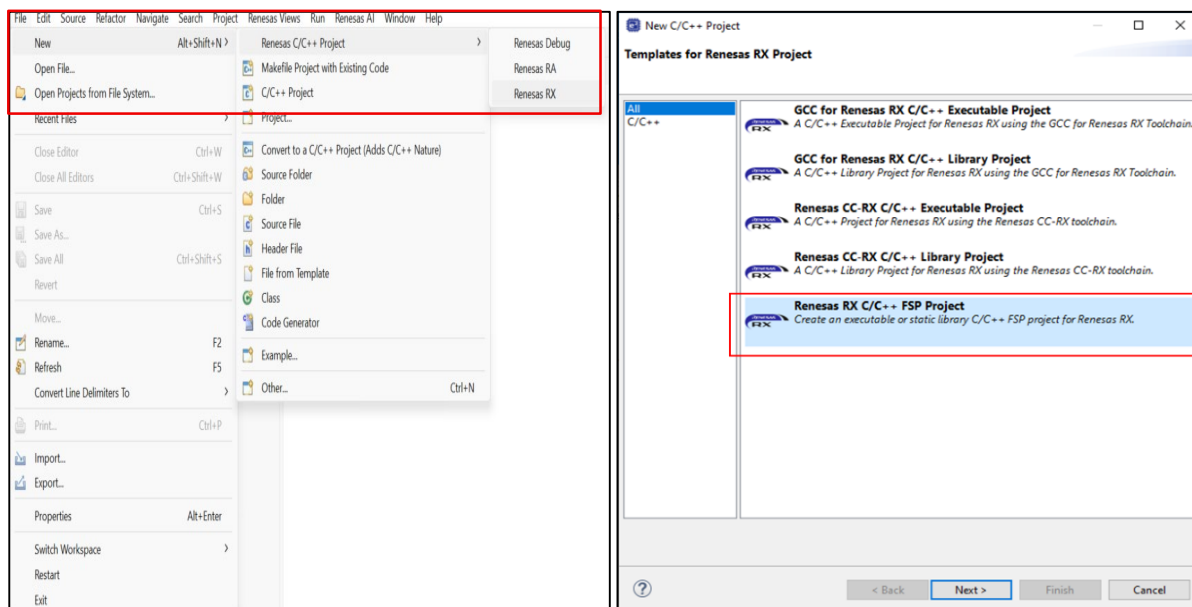


Figure 2-1 Create a new Renesas RX C/C++ FSP Project.

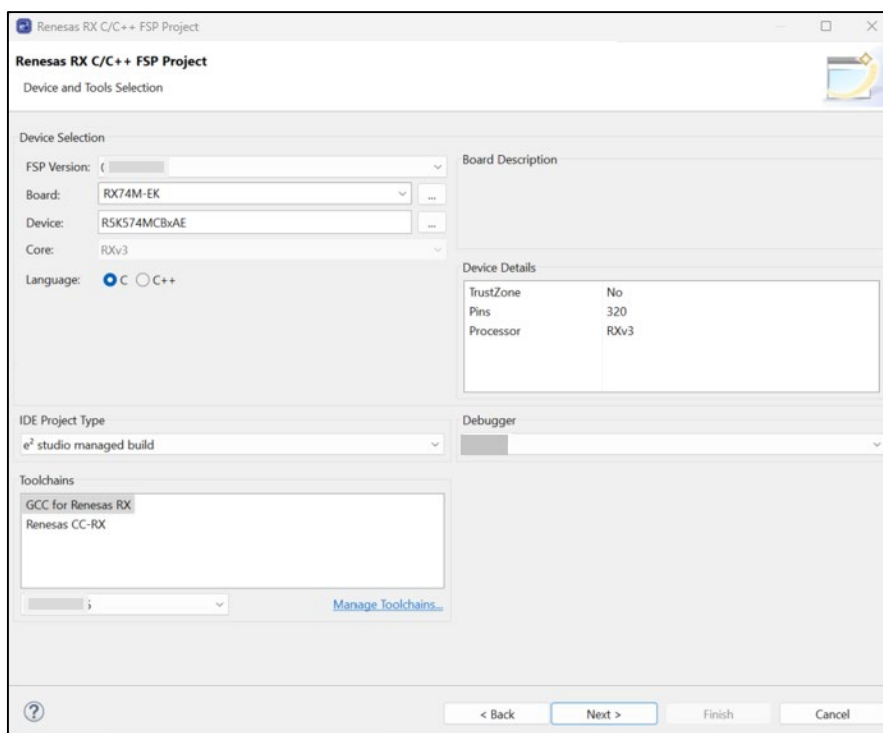


Figure 2-2 Select Board.

2.2 Migrating Configuration

This section describes migration configuration.

Configuration for WDT in this application notes shown as table below:

Configuration	Value
Timeout	16,384 Cycles
Clock Division Ratio	PCLK/8192
Window Start Position	100 (Window Position Not Specified)
Window End Position	25
Reset Control	Reset Output
Stop Control	WDT Count Disable in Low Power Mode
NMI Callback	wdt_callback

Steps of updating configuration on FSP Project.

On Configuration Stacks Tab

Add HAL WDT module: go to the New Stack→Monitoring→Watchdog(r_wdt)

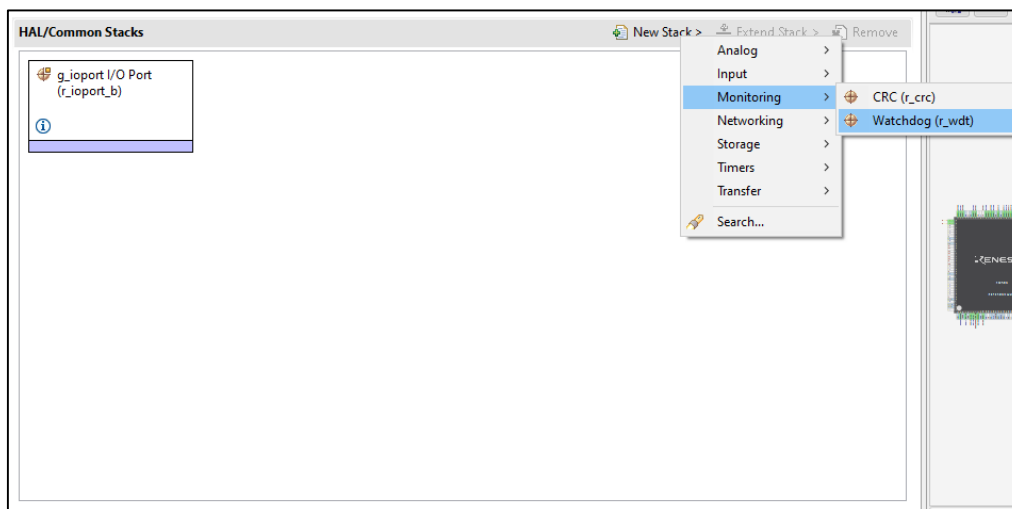


Figure 2-3 Add module (r_wdt) on stack.

The new stack(g_wdt0) has appeared after adding Watchdog(r_wdt).

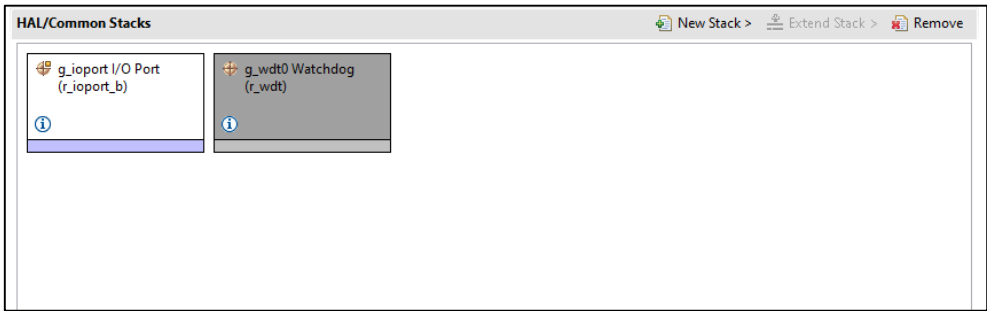


Figure 2-4 Module (g_wdt0) is added on stack.

After that, go to the Properties to set up for g_wdt0 stack as below:

g_wdt0 Watchdog (r_wdt)		
Settings	Property	Value
	▼ Common	
	Parameter Checking	Default (BSP)
	Register Start NMI Support	Disabled
	▼ Module g_wdt0 Watchdog (r_wdt)	
	Name	g_wdt0
	Timeout	16,384 Cycles
	Clock Division Ratio	PCLK/8192
	Window Start Position	100 (Window Position Not Specified)
	Window End Position	25
	Reset Control	Reset Output
	Stop Control	WDT Count Disabled in Low Power Mode
	NMI Callback	wdt_callback

Figure 2-5 Setting Properties for g_wdt0.

In this WDT demo using other modules. User need to add module IRQ and CMT on the stack.

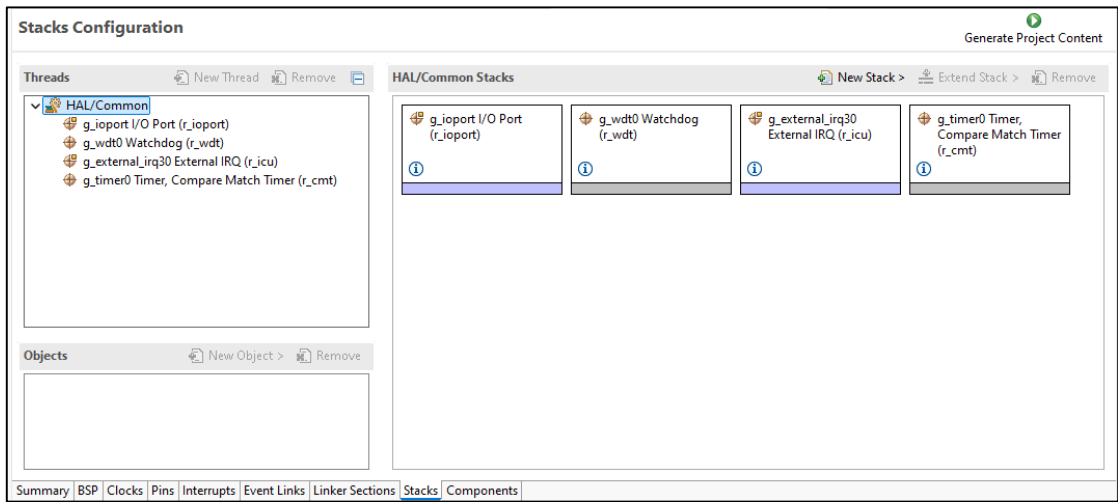


Figure 2-6 Adding IRQ and CMT.

Configuration for module CMT

g_timer0 Timer, Compare Match Timer (r_cmt)		
Settings	Property	Value
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_timer0 Timer, Compare Match Timer (r_cmt)	
	▼ General	
	Name	g_timer0
	Channel	0
	Mode	One-Shot
	Period	0x1000
	Period Unit	Raw Counts
	▼ Interrupts	
	Callback	switch_debounce_isr_callback
	Compare Match Interrupt Priority	Priority 11

Figure 2-7 Setting Properties for CMT.

Configuration for module IRQ

g_external_irq30 External IRQ (r_icu)		
Settings	Property	Value
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_external_irq30 External IRQ (r_icu)	
	Name	g_external_irq30
	Channel	30
	Trigger	Rising
	Digital Filtering	Disabled
	Filter Source	PCLK filter
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	switch_isr_callback1
	Pin Interrupt Priority	Priority 31 (highest)
	▼ Pins	
	IRQ30	P414
	IRQ30-DS	None

Figure 2-8 Setting Properties for IRQ.

Note: Channel, Pins setting depends on User's board.

2.3 Generate Code

After all the stacks/HAL configuration is completed and all the user selections are made, user must click on the button of Generate Project Content.

The Generate Project Content function takes the selected and configured modules and automatically generates the complete and correct configuration code. The code is added to the folders visible in the Project Explorer window in e2 studio. The configuration.xml file in the project folder holds all the generated configuration settings. This file can be opened in the GUI-based configuration editor to make further edits and changes. Once a project has been generated, you can go back and reconfigure any of the modules and settings if required using this editor.

When you click the Generate Project Content button, the stacks configuration contents are written to files into path: **rx_gen\ hal_data.c**.

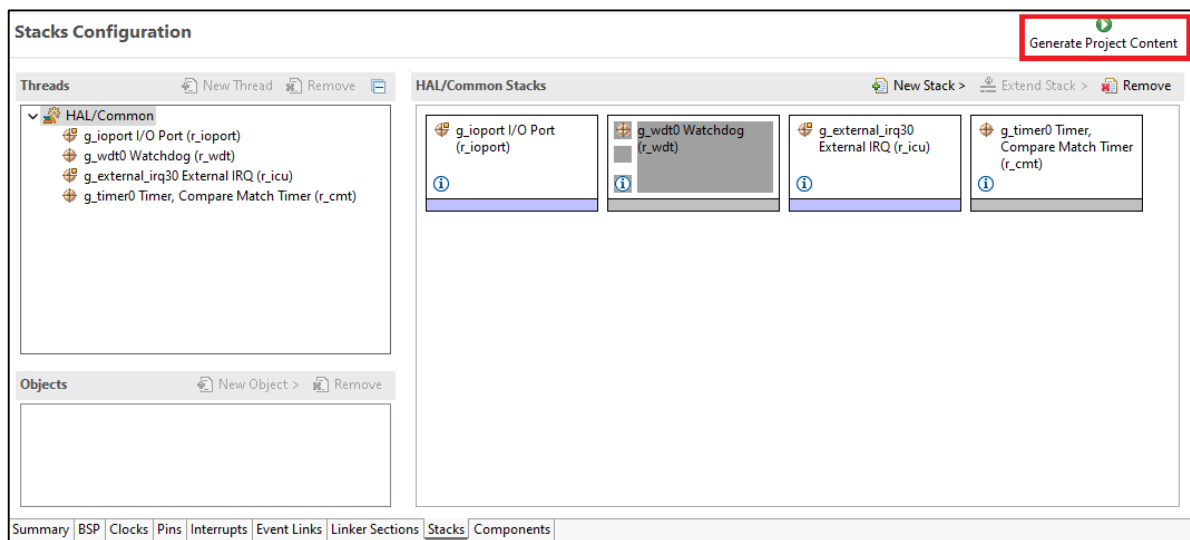


Figure 2-9 Generate Project Content.

The stacks configuration content for **g_wdt0(rx_gen\ hal_data.c)**:

```
const wdt_cfg_t g_wdt0_cfg =
{
    .timeout = WDT_TIMEOUT_16384,
    .clock_division = WDT_CLOCK_DIVISION_8192,
    .window_start = WDT_WINDOW_START_100,
    .window_end = WDT_WINDOW_END_25,
    .reset_control = WDT_RESET_CONTROL_RESET,
    .stop_control = WDT_STOP_CONTROL_ENABLE,
    .p_callback = wdt_callback, };

/* Instance structure to use this module. */
const wdt_instance_t g_wdt0 =
{
    .p_ctrl = &g_wdt0_ctrl, .p_cfg = &g_wdt0_cfg, .p_api = &g_wdt_on_wdt };

```

2.4 Migrating Application Code

The typical steps in using the WDT FSP module in an application:

1. Configure module by using FSP Configuration as described in section 2.2.
2. Initialize the WDT by using .open API.
3. Refresh WDT by using .refresh API.
4. Get the WDT status by using .statusGet and .counterGet APIs.
5. Handle interrupts by using callback.

WDT Initialization Code

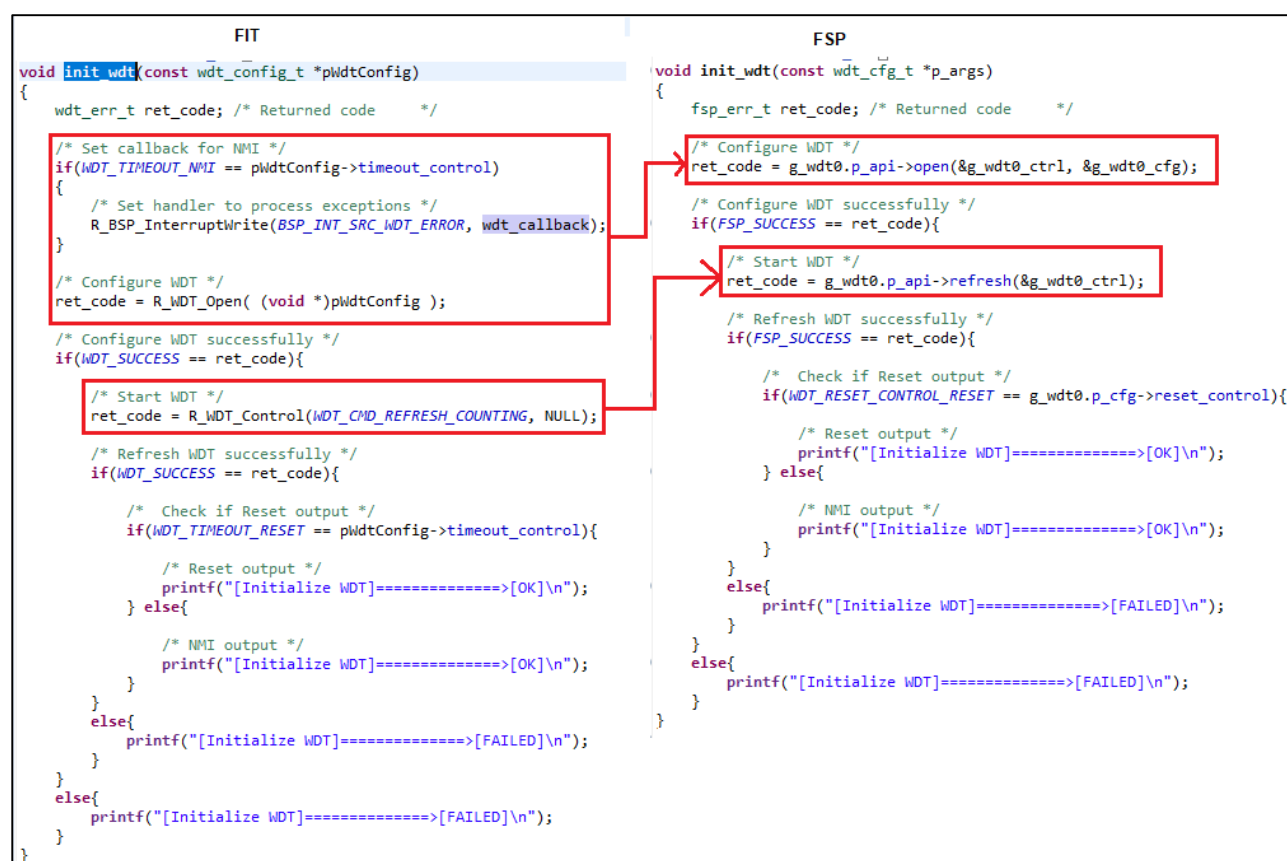


Figure 2-10 Compare initialization code of WDT between FIT and FSP.

WDT status checking Code

FIT	FSP
<pre> void check_reset_source() { uint16_t wdt_status = 0; /* Get WDT Status */ R_WDT_Control(WDT_CMD_GET_STATUS, &wdt_status); /* Check if it was underflow or refresh error */ if((wdt_status & REFEF_MASK) (wdt_status & UNOFF_MASK)){ /* Check if it was underflow error */ if(wdt_status & UNOFF_MASK) { printf("UNDERFLOW ERROR - Reset\nThe program was started, press RESET to input new WDT configuration\n"); } /* Check if it was underflow error */ else { printf("REFRESH ERROR - Reset\nThe program was started, press RESET to input new WDT configuration\n"); } while(1){ R_WDT_Nop(); } } else { printf("\n=====The Program was started=====n"); } } </pre>	<pre> void check_reset_source() { wdt_status_t wdt_status; /* Get WDT Status */ g_wdt0_p_api->statusGet(&g_wdt0_ctrl, &wdt_status); /* Check if it was underflow or refresh error */ if((wdt_status == WDT_STATUS_UNDERFLOW) (wdt_status == WDT_STATUS_REFRESH_ERROR)){ /* Check if it was underflow error */ if((wdt_status == WDT_STATUS_UNDERFLOW)) { printf("UNDERFLOW ERROR - Reset\nThe program was started, press RESET to input new WDT configuration\n"); } /* Check if it was underflow error */ else { printf("REFRESH ERROR - Reset\nThe program was started, press RESET to input new WDT configuration\n"); } while(1){ } } else { printf("\n=====The Program was started=====n"); } } </pre>

Figure 2-11 Compare Get status of WDT between FIT and FSP.

FIT	FSP
<pre> bool is_refresh_permitted(const wdt_config_t *pWdtConfig) { bool result = false; /* Result */ uint16_t timeout_cycles = 0; /* Timeout cycles */ uint16_t window_start_val = 0; /* Window start value */ uint16_t window_end_val = 0; /* Window end value */ uint16_t wdt_status = 0; /* WDT Status */ /* Get setting from WDT configuration */ timeout_cycles = gs_timeout_cycles_opt[pWdtConfig->timeout]; window_start_val = gs_window_start_opt[pWdtConfig->window_start >> 12]; window_end_val = gs_window_end_opt[pWdtConfig->window_end >> 8]; /* Check if window start was less than window end value */ if(window_start_val <= window_end_val){ window_end_val = 0; } /* Get counter value */ R_WDT_Control(WDT_CMD_GET_STATUS, &wdt_status); /* Check if it is refresh-permitted period */ switch(pWdtConfig->window_start){ case WDT_WINDOW_START_100: { if((((wdt_status & CNTVAL_MASK) * 100) / timeout_cycles) < window_end_val) { result = false; } else{ result = true; } break; } case WDT_WINDOW_START_75: case WDT_WINDOW_START_50: case WDT_WINDOW_START_25: { if((((wdt_status & CNTVAL_MASK) * 100) / timeout_cycles) > window_start_val) (((wdt_status & CNTVAL_MASK) * 100) / timeout_cycles) < window_end_val){ result = false; } else{ result = true; } break; } default: { break; } } return result; } </pre>	<pre> bool is_refresh_permitted(const wdt_cfg_t *pWdtConfig) { bool result = false; /* Result */ uint16_t timeout_cycles = 0; /* Timeout cycles */ uint16_t window_start_val = 0; /* Window start value */ uint16_t window_end_val = 0; /* Window end value */ uint32_t counter_value = 0; /* counter value */ /* Get setting from WDT configuration */ timeout_cycles = gs_timeout_cycles_opt[pWdtConfig->timeout]; window_start_val = gs_window_start_opt[pWdtConfig->window_start]; window_end_val = gs_window_end_opt[pWdtConfig->window_end]; /* Check if window start was less than window end value */ if(window_start_val <= window_end_val){ window_end_val = 0; } /* Get counter value */ g_wdt0_p_api->counterGet(&g_wdt0_ctrl, &counter_value); /* Check if it is refresh-permitted period */ switch(pWdtConfig->window_start){ case WDT_WINDOW_START_100: { if(((counter_value * 100) / timeout_cycles) < window_end_val) { result = false; } else{ result = true; } break; } case WDT_WINDOW_START_75: case WDT_WINDOW_START_50: case WDT_WINDOW_START_25: { if((((counter_value * 100) / timeout_cycles) > window_start_val) (((counter_value * 100) / timeout_cycles) < window_end_val)) { result = false; } else{ result = true; } break; } default: { break; } } return result; } </pre>

Figure 2-12 Compare Get counter status of WDT between FIT and FSP.

Switch Callback & Refresh Code

FIT	FSP
<pre> void switch_press_callback(void) { /* Check if switch SW1 is held down */ if (check_sw_is_pressed(SWITCHPRESS_1)) { /* Turn on LED3 */ LED3_PDR = 1; LED3_ = LED_ON; printf("[SW1 was pressed]=====>[OK]\n"); /* Refresh WDT */ R_WDT_Control(WDT_CMD_REFRESH_COUNTING, NULL); } } </pre>	<pre> void switch_press_callback(void) { /* Check if switch SW1 is held down */ if (check_sw_is_pressed(SWITCHPRESS_1)) { /* Turn on LED3 */ LED(3, LED_ON); printf("[SW1 was pressed]=====>[OK]\n"); /* Refresh WDT */ g_wdt0.p_api->refresh(&g_wdt0_ctrl); } } </pre>

Figure 2-13 Compare refresh code of WDT between FIT and FSP.

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

2.5 Build and Run the project.

In the demonstration, the program configure WDT by using the FSP Configuration. After configuring WDT, it begins counting down and LED0 is blinked. When counter value is in refresh-permitted period or in refresh-prohibited period, Renesas Debug Virtual Console will print out to notify user. If it is in refresh-permitted period and SW1 is pressed, WDT is refreshed. Otherwise, if it is in refresh- prohibited period and SW1 is pressed, refresh error occurs.

In addition, after WDT underflow error or refresh error occurs, the program will automatically reset and restart.

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. The program automatically configures the WDT using the default settings and starts counting down.

Input Example

```
Renesas Debug Virtual Console X
-----The Program was started-----
[Initialize WDT]=====>[OK]
Press SW1 to refresh Watchdog
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-permitted
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
WDT was within refresh-prohibited
-----The Program was started-----
[Initialize WDT]=====>[OK]
Press SW1 to refresh Watchdog
WDT was within refresh-permitted
WDT was within refresh-permitted
```

Figure 2-14 Input and Result on Console.

3. Reference Documents

RX Family WDT Module Using Firmware Integration Technology(R01AN3200)

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

This section describes confirmed operation environment for the WDT FIT/FSP module.

Table 4.1 Confirmed Operation Environment (Rev.1.00)

Item	Contents
FIT Module	r_wdt_rx v3.10
FSP module	RX FSP v1.0.0 Module: r_wdt
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"
#include "wdt_func.h"
FSP_CPP_HEADER
void R_BSP_WarmStart(bsp_warm_start_event_t event);
FSP_CPP_FOOTER

/*****
*****//**
 * 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)
{
    /* Check if reset was WDT RESET source or not */
    check_reset_source();

    /* Delay 1s before starting the program */
    R_BSP_SoftwareDelay(1000000, BSP_DELAY_UNITS_MICROSECONDS);

    /* Initialize Switches */
    init_switches();

    /* Initialize WDT */
    init_wdt(&g_wdt0_cfg);

    /* Request user press SW1 refresh WDT */
    printf("Press SW1 to refresh Watchdog\n");

    /* Main processing */
    while(1)
    {
        /* Blink LED */
        blink_led();

        /* Check if WDT was within refresh-permitted */
        if( is_refresh_permitted(&g_wdt0_cfg) == true ){
            printf("WDT was within refresh-permitted\n");
        }
        else{
            printf("WDT was within refresh-prohibited\n");
        }
    }
}

/*****
*****//**
 * 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)
    {
#if 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. */
    IOPORT_CFG_OPEN (&IOPORT_CFG_CTRL, &IOPORT_CFG_NAME);
}
}
```

WDT sample project include some files for setting switches, output to console as sample folder structure below:

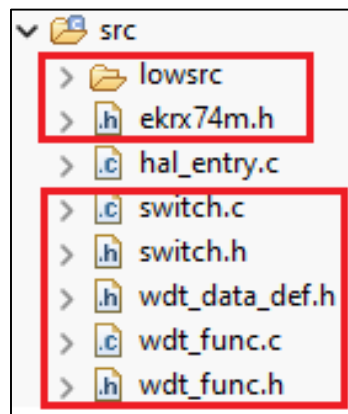


Figure 4-1 Sample folder structure.

In this application note include reference code in file **wdt_func.c** to introduce using WDT API functions.

FSP sample code in file “wdt_func.c”

Initialization Code:

```

/*****
*****
* Function Name: init_wdt
* Description  : Initialize WDT
* Arguments   : pWdtConfig -
*               WDT configuration
* Return Value : None
*****
*****/
void init_wdt(const wdt_cfg_t *p_args)
{
    fsp_err_t ret_code; /* Returned code */

    /* Configure WDT */
    ret_code = g_wdt0.p_api->open(&g_wdt0_ctrl, p_args);

    /* Configure WDT successfully */
    if(FSP_SUCCESS == ret_code){

        /* Start WDT */
        ret_code = g_wdt0.p_api->refresh(&g_wdt0_ctrl);

        /* Refresh WDT successfully */
        if(FSP_SUCCESS == ret_code){

            /* Check if Reset output */
            if(WDT_RESET_CONTROL_RESET == g_wdt0.p_cfg->reset_control){

                /* Reset output */
                printf("[Initialize WDT]=====>[OK]\n");
            } else{

                /* NMI output */
                printf("[Initialize WDT]=====>[OK]\n");
            }
        }
        else{
            printf("[Initialize WDT]=====>[FAILED]\n");
        }
    }
    else{
        printf("[Initialize WDT]=====>[FAILED]\n");
    }
}
/*****
*****
* End of function init_wdt()
*****
*****/

```

WDT status checking Code

```

/*****
*****
* Function Name: check reset source
* Description : Check if reset was WDT RESET source or not, and print out messages
accordingly
* Arguments : None
* Return Value : None
*****
*****/
void check_reset_source()
{
    wdt_status_t wdt_status;

    /* Get WDT Status */
    g_wdt0.p_api->statusGet(&g_wdt0_ctrl, &wdt_status);

    /* Check if it was underflow or refresh error */
    if((wdt_status == WDT_STATUS_UNDERFLOW) || (wdt_status == WDT_STATUS_REFRESH_ERROR) ){

        /* Check if it was underflow error */
        if( (wdt_status == WDT_STATUS_UNDERFLOW) )
        {
            printf("[UNDERFLOW ERROR - Reset]\nThe program was started, press RESET to input
new WDT configuration\n");
        }
        /* Check if it was underflow error */
        else
        {
            printf("[REFRESH ERROR - Reset]\nThe program was started, press RESET to input
new WDT configuration\n");
        }

        while(1){
        }

    }
    else
    {
        {
            printf("\n=====The Program was started=====\\n");
        }
    }
}
/*****
*****
* End of function check reset source()
*****
*****/

```

Check WDT status by .counterGet API

```

/*****
*****
* Function Name: is refresh permitted
* Description   : Check whether WDT was in refresh-permitted period or not
* Arguments    : pWdtConfig -
*                WDT configuration
* Return Value : true -
*                in refresh-permitted period
*                false -
*                in refresh-prohibited period
*****
*****/
bool is_refresh_permitted(const wdt_cfg_t *pWdtConfig)
{
    bool result = false; /* Result */
    uint16_t timeout_cycles = 0; /* Timeout cycles */
    uint16_t window_start_val = 0; /* Window start value */
    uint16_t window_end_val = 0; /* Window end value */
    uint32_t counter_value = 0; /* counter value */

    /* Get setting from WDT configuration */
    timeout_cycles = gs_timeout_cycles_opt[pWdtConfig->timeout];
    window_start_val = gs_window_start_opt[pWdtConfig->window_start];
    window_end_val = gs_window_end_opt[pWdtConfig->window_end];

    /* Check if window start was less than window end value */
    if(window_start_val <= window_end_val){
        window_end_val = 0;
    }

    /* Get counter value */
    g_wdt0.p_api->counterGet(&g_wdt0_ctrl, counter_value);

    /* Check if it is refresh-permitted period */
    switch(pWdtConfig->window_start){
        case WDT_WINDOW_START_100:
        {
            if( ((*counter_value * 100) / timeout_cycles) < window_end_val )
            {
                result = false;
            }
            else{
                result = true;
            }
            break;
        }
        case WDT_WINDOW_START_75:
        case WDT_WINDOW_START_50:
        case WDT_WINDOW_START_25:
        {
            if( (((*counter_value * 100) / timeout_cycles) > window_start_val ) ||
                (((*counter_value * 100) / timeout_cycles) < window_end_val) )
            {
                result = false;
            }
            else{
                result = true;
            }
            break;
        }
        default:
        {
            break;
        }
    }

    return result;
}
/*****
*****
* End of function is_refresh_permitted()
*****
*****/

```

Refresh WDT

```

/*****
*****
* Function Name: switch press callback
* Description   : Switch Press callback.
* Arguments    : None
* Return Value  : None
*****
*****/
void switch_press_callback(void)
{
    /* Check if switch SW1 is held down */
    if (check_sw_is_pressed(SWITCHPRESS_1))
    {
        /* Turn on LED3 */
        LED(3, LED_ON);

        printf("[SW1 was pressed]=====>[OK]\n");

        /* Refresh WDT */
        g_wdt0.p_api->refresh(&g_wdt0_ctrl);
    }
}
/*****
*****
* End of function switch press callback()
*****
*****/

```

4.3 Comparison Reference

4.3.1 Compare Functions (Details)

Compare API functions (details)

FIT	FSP	
	FSP Interface API (Public API)	FSP Instance API (Private API)
R_WDT_Open	open	R_WDT_Open
R_WDT_Control Use Commands to configure and control.	refresh statusGet counterGet	R_WDT_Refresh R_WDT_StatusGet R_WDT_CounterGet
R_WDT_GetVersion	Not supported in specific modules FSP has common function to get FSP pack version. R_FSP_VersionGet (in bsp_api.h)	Not supported
Not supported	statusClear	R_WDT_StatusClear
Not supported	timeoutGet	R_WDT_TimeoutGet
Not supported	callbackSet	R_WDT_CallbackSet

Compare functions in detail

Open Function	
Description	Initializes the WDT.
FIT	R_WDT_Open (void * const p_cfg)
FSP	open (wdt_ctrl_t * const p_ctrl, wdt_cfg_t const * const p_cfg);

Control Function	
Description	Control and get status of module
FIT	R_WDT_Control (wdt_cmd_t const cmd, uint16_t * p_status) Command: WDT_CMD_GET_STATUS WDT_CMD_REFRESH_COUNTING
FSP	refresh (wdt_ctrl_t * const p_ctrl) Refresh the counter statusGet (wdt_ctrl_t * const p_ctrl, wdt_status_t * const p_status); Read the WDT status flags counterGet (wdt_ctrl_t * const p_ctrl, uint32_t * const p_count); Read the current count value of the WDT

Additional Functions	
Description	List additional functions or specific function which only have in FIT but don't have in FSP and vice versa.
FIT	Get Version Function: R_WDT_GetVersion (void) This function is used to get the driver version information.
FSP	Information Retrieval: statusClear (wdt_ctrl_t * const p_ctrl, const wdt_status_t status) Clear the WDT status and error flags. timeoutGet (wdt_ctrl_t * const p_ctrl, wdt_timeout_values_t * const p_timeout) Read timeout information for the watchdog timer. callbackSet (wdt_ctrl_t * const p_ctrl, void (* p_callback)(wdt_callback_args_t *), void const * const p_context, wdt_callback_args_t * const p_callback_memory) Updates the user callback and has option of providing memory for callback structure.

4.3.2 Compare Configuration (Details)

Compare FIT and FSP Configuration:

FIT Configuration	FSP Configuration
Parameter Checking	Common> Parameter Checking
Not supported	Common> Register Start NMI Support
Configure module in user application code: • Timeout • Clock Division Ratio • Window Start Position • Window End Position • Reset Control • Stop Control	Set in FSP Configuration>Module WDT >Name >Timeout >Clock Division Ratio >Window Start Position > Window End Position >Reset Control >Stop Control >NMI Callback

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. RENASAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENASAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENASAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENASAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENASAS 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/.