Renesas Synergy™ Platform

Thread Monitor Framework Module Guide

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application and write code, using the included application project code as a reference and efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are available in the Renesas Synergy Knowledge™ Base (as described in the References section at the end of this document), and should be valuable resources for creating more complex designs.

The Thread Monitor Framework provides a high-level API for system monitoring applications using the watchdog timer (WDT) or independent watchdog timer (IWDT) to monitor program execution. The Thread Monitor Framework uses the WDT or IWDT peripherals on the Synergy MCU device.

Contents

1. Thread Monitor Framework Module Features

The Thread Monitor Framework supports the following features:

- The Thread Monitor Framework interface monitors RTOS threads using a watchdog timer. The Thread Monitor forces a watchdog reset of the microcontroller when any of the monitored threads do not behave as expected.
- The Thread Monitor is designed to support any Synergy device with either a WDT or IWDT peripheral, and a HAL module with no changes to the API.
- In profiling mode, the minimum and maximum counter values for registered threads can be determined. When in profiling mode, the watchdog timer is always refreshed and does not reset the device.
- Both the WDT and IWDT HAL modules are supported by this framework module.

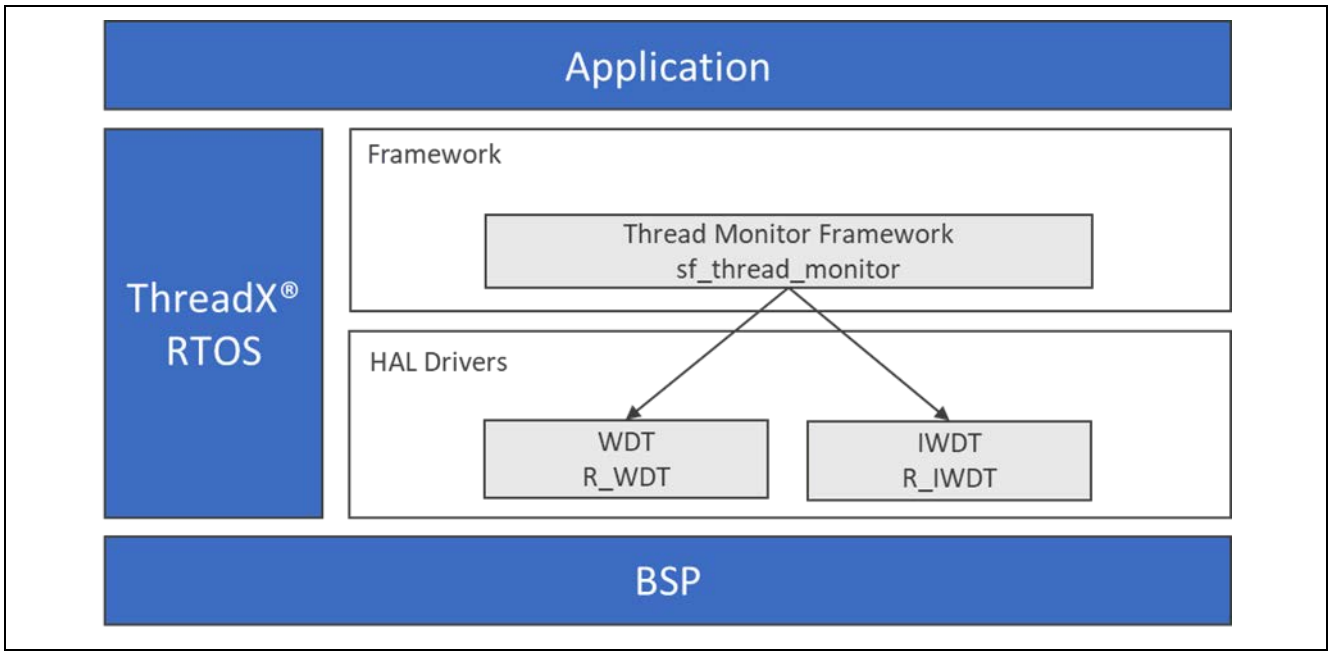

Figure 1. Thread Monitor Framework Module Block Diagram

2. Thread Monitor Framework Module APIs Overview

The Thread Monitor Framework defines APIs for opening and closing the framework and registering and unregistering threads for monitoring. The following table has a complete list of the available APIs, an example API call, and a short description of each API. A table of return status values follows the API summary table.

Table 1. Thread Monitor Framework Module APIs Summary

Function Name	Example API Call and Definition
open	`g_sf_thread_monitor.p_api->open (g_sf_thread_monitor.p_ctrl,g_sf_thread_monitor.p_cfg);` Configures the WDT or IWDT module. From the configuration data, the timeout period of the WDT/IWDT is determined. A thread created to monitor registered threads.
close	`g_sf_thread_monitor.p_api->close (g_sf_thread_monitor.p_ctrl);` Suspends the thread monitoring thread. The watchdog peripheral no longer refreshes.
threadRegister	`g_sf_thread_monitor.p_api-> threadRegister (g_sf_thread_monitor.p_ctrl, &p_min_max_struct);` Registers a thread for monitoring.

Function Name	Example API Call and Definition
threadUnregister	`g_sf_thread_monitor.p_api-> threadUnregister (g_sf_thread_monitor.p_ctrl);` Removes a thread from monitoring.
countIncrement	`g_sf_thread_monitor.p_api-> countIncrement (g_sf_thread_monitor.p_ctrl);` Safely increments a monitored thread's count value.
versionGet	`g_sf_thread_monitor.p_api-> versionGet(&version);` Retrieves the API version and stores it in the version pointer.

Note: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the *SSP User's Manual* **API References** for the associated module.

Table 2. Status Return Values

Name	Description
SSP_SUCCESS	API Call Successful
SSP_ERR_ASSERTION	Pointer is null.
SSP_ERR_IN_USE	Thread monitor has already been opened.
SSP_ERR_INVALID_MODE	Low-level watchdog peripheral returns an error when opened.
SSP_ERR_UNSUPPORTED	Data structure could not be allocated.
SSP_ERR_INVALID_ARGUMENT	One or more configuration options is invalid for the low-level driver.
SSP_ERR_NOT_OPEN	`SF_THREAD_MONITOR_Open` has either not been called or it was not called successfully.
SSP_ERR_INSUFFICIENT_SPACE	Not enough entries in the threads-to-be-monitored array to add this thread. Increases the value of THREAD_MONITOR_CFG_MAX_NUMBER_OF_THREADS in `sf_thread_moinitor_cfg.h`

Note: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* **API References** for the associated module for a definition of all relevant status return values.

3. Thread Monitor Framework Module Operational Overview

A thread registers a counter variable with the Thread Monitor along with minimum and maximum expected values for this counter variable. The thread, which is monitored, increments the counter variable while it runs. At a period of half the watchdog timeout period, the Thread Monitor checks the counter variables of registered threads. If any fall outside of the minimum and maximum values, the watchdog timer can reset the microcontroller. If all fall within their expected range, the watchdog timer is refreshed, and the counter variables are cleared to zero.

3.1 Thread Monitor Framework Module Important Operational Notes and Limitations

3.1.1 Thread Monitor Framework Module Operational Notes

The following figure shows a flowchart for the operation of the Thread Monitor Framework module:

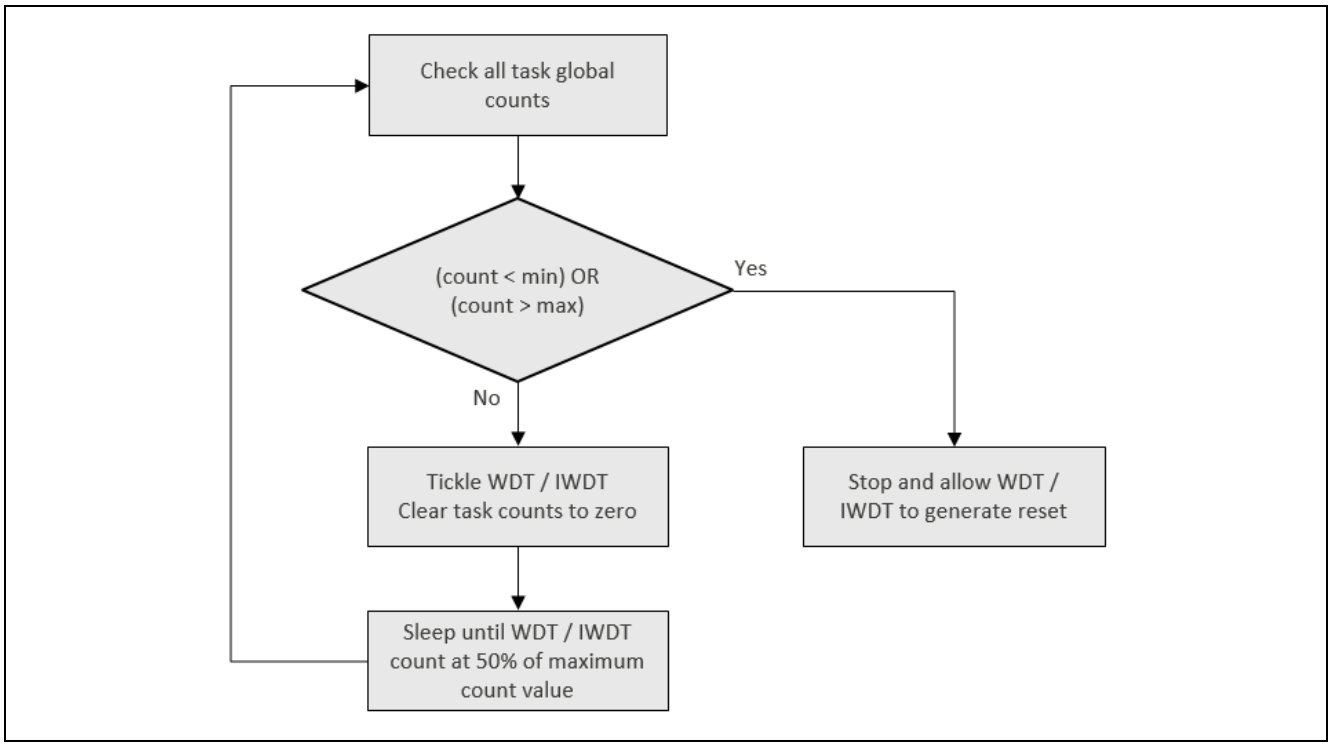

Figure 2. Thread Monitor Framework Operation

The following figure shows when the WDT/IWDT refreshes (Note that the valid refresh period is the central 50% of the count period, 25% on either side of the 50% count value):

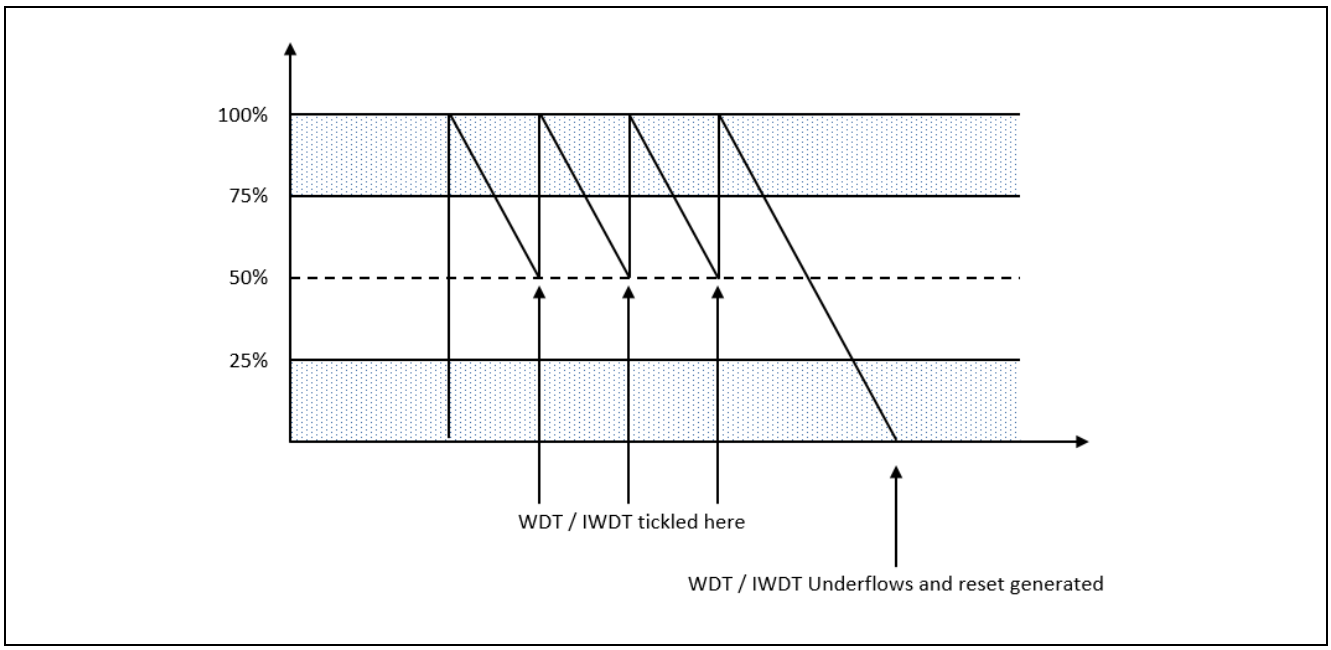

Figure 3. WDT/IWDT Refresh Operation

- The IWDT has its own clock source to improve safety.
- The WDT can be started from the application.
- When the thread is not executing sleep mode, set the stop-control property of the WDT to WDT Count Enabled in Low Power Mode. When the thread is not executing (asleep), the ThreadX® executes a WFI instruction effectively putting the device into soft sleep, which causes the WDT to stop counting.

Note: Do not open or refresh the WDT in a monitored thread file, it is opened or refreshed automatically by the Thread Monitor Framework.

- Internally, the Thread Monitor Framework runs at the rate of half of the watchdog timer reset period. This rate ensures the Thread Monitor Framework runs at 50% of the watchdog count value—well within the valid refresh window. The Thread Monitor calculates the reset period internally by querying the lower-level watchdog driver.

3.1.2 Thread Monitor Framework Module Limitations

- The Thread Monitor Framework has a close API call. When WDT and IWDT are being used, it is not possible to stop them. If the Thread Monitor Framework is closed, some other provision for refreshing the watchdog must be made or the device resets.
- Debugging mode-support is required when running with a SEGGER J-Link® on some devices; WDT/IWDT does not count when using J-Link debugging hardware. The Thread Monitor Framework thread typically synchronizes to the WDT/IWDT counter but skips this synchronization step when it is running with J-Link.
- Assign a high priority (low number in ThreadX) to the Thread Monitor Framework thread; any delays in running the Thread Monitor Framework could refresh the watchdog outside of the valid refresh window, causing the microcontroller to reset. The Thread Monitor Framework thread does not run for long and does not impact the performance of the system.

Refer to the latest *SSP Release Notes* for any additional operational limitations for this module.

4. Including the Thread Monitor Framework Module in an Application

This section describes how to include the Thread Monitor Framework in an application using the SSP configurator.

Note: It is assumed you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these tasks, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the Thread Monitor Framework to an application, simply add it to a thread using the stacks selection sequence given in the following table. (The default name for the Thread Monitor Framework is g_sf_thread_monitor0. This name can be changed in the associated Properties window.)

Table 3. Thread Monitor Framework Module Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
g_sf_thread_monitor0 Thread Monitor Framework	Threads	New Stack> Framework> Services> Thread Monitor Framework on sf_thread_monitor

When the Thread Monitor Framework on `sf_thread_monitor` is added to the thread stack as shown in the following figure, the configurator reports (via the Add WDT driver) that at least one watchdog timer is required to complete the stack. Any lower-level modules that need additional configuration information have box text highlighted in Red. Modules with a Gray band are individual modules that stand alone. Modules with a Blue band are shared or common and need only be added once and can be used by multiple stacks. Modules with a Pink band can require the selection of lower-level modules; these are either optional or recommended (this is indicated in the block with the inclusion of this text.) If the addition of lower-level modules is required, the module description includes Add in the text. Clicking on any Pink banded modules brings up the New icon and displays possible choices.

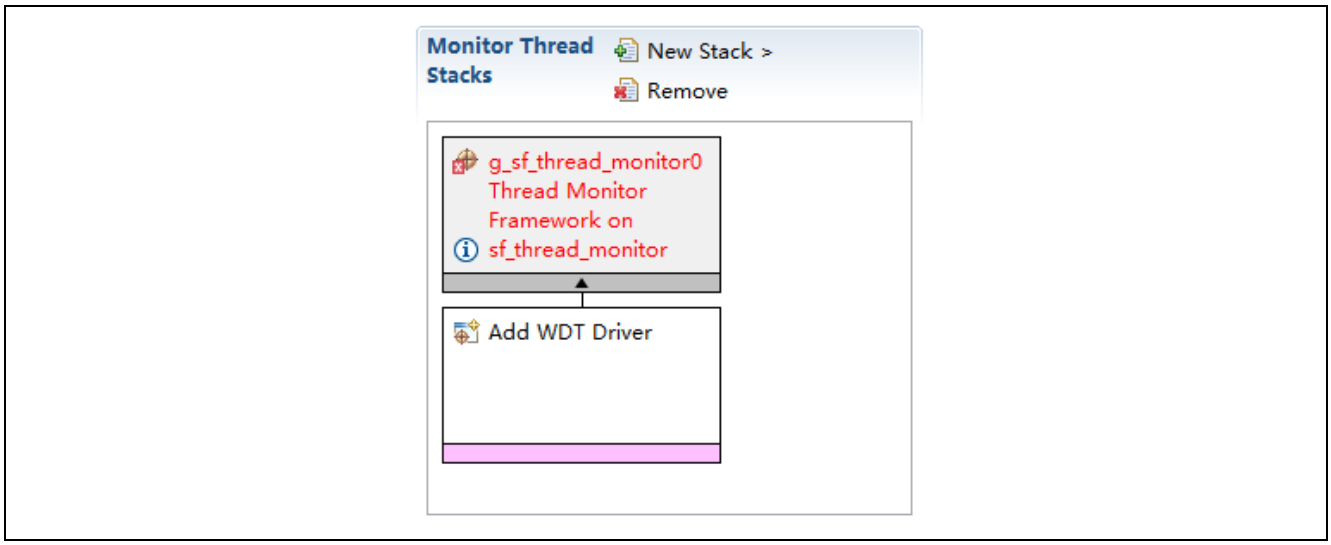

Figure 4. Thread Monitor Framework Module Stack

5. Configuring the Thread Monitor Framework Module

The Thread Monitor Framework module must be configured by the user for the desired operation. The SSP configuration window automatically identifies (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Only those properties that can be changed without causing conflicts are available for modification. Other properties are 'locked' and are unavailable for changes, and they are identified with a lock icon for the 'locked' property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error-prone than previous 'manual' approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP Configurator, and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available.

Note: You may want to open your ISDE, create the module and explore the property settings in parallel while looking over the following configuration table settings. This helps to orient you and can be a useful 'hands-on' approach to learning the ins and outs of developing with SSP.

Table 4. Configuration Settings for the Thread Monitor Framework Module on sf_thread_monitor

Parameter	Value	Description
Parameter Checking	Enabled, Disabled, BSP (Default: BSP)	Controls whether to include code for API parameter checking.
Maximum Number of Monitored Threads	5	Maximum number of threads that can be monitored.
Stack size of the Thread Monitor (bytes)	512	Stack size of the thread monitor
Name	g_sf_thread_monitor0	Module name.
Profiling Mode	Enabled, Disabled (Default: Disabled)	Whether profiling mode should be enabled.
Thread Monitor Thread Priority	1	Priority of thread monitor internal thread.
Name of generated initialization function	sf_thread_monitor_init0	Name of generated initialization function
Auto Initialization	Enable, Disable (Default: Enable)	Auto initialization selection

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Settings other than the defaults for lower-level modules can be desirable in some cases. For example, it might be useful to set the maximum number of monitored threads to a desired value. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

Note: Most of the property settings for modules are intuitive and usually can be determined by inspection of the associated Properties window from the SSP configurator.

5.1 Configuration Settings for the Thread Monitor Framework Low Level Modules

Typically, only a small number of settings must be modified from the default for lower-level modules as indicated via the red text in the thread stack block. Notice that some of the configuration properties must be set to a certain value for proper framework operation and are locked to prevent user modification. The following tables identify all the settings within the properties section for the module:

Table 5. Configuration Settings for the Independent Watchdog Timer on r_iwdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enables or disables the parameter checking
Name	g_wdt0	Module Name
NMI Callback	NULL	Callback Name. A user callback function can be registered in external_irq_api_t::open. If this callback function is provided, it will be called from the interrupt service routine (ISR) each time the IRQn triggers. Warning: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

Table 6. Configuration Settings for the Watchdog Timer on r_wdt

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled (Default: BSP)	Enable or disable the parameter checking
Name	g_wdt0	Module Name

ISDE Property	Value	Description
Start Mode	Register, Auto (Default: Register)	Configures the start mode as register start or auto-start.
Start Watchdog After Configuration	True, False (Default: True)	Controls whether WDT is started during initialization
Timeout	1024 cycles, 4096 cycles, 8192 cycles, 16384 cycles (Default: 16384 cycles)	WDT timeout period.
Clock Division Ratio	PCLK/4, PCLK/64, PCLK/128, PCLK/512, PCLK/2048, PCLK/8192 (Default: PCLK/8192)	WDT clock divider
Window Start Position	100% (Window Position Not Specified); 75%, 50%, 25% (Default: 100% (Window Position Not Specified))	Permitted refresh period start position.
Window End Position	0% (Window Position Not Specified), 25%, 50%, 75% (Default: 0% (Window Position Not Specified))	Permitted refresh period end position.
Reset Control	Reset Output, NMI Generated (Default: Reset Output)	Select whether WDT should reset the MCU or generate an NMI.
Stop Control	WDT Count Enabled in Low Power Mode, WDT Count Disabled in Low Power Mode (Default: WDT Count Disabled in Low Power Mode)	Select whether the WDT should stop counting in low power modes.
NMI Callback	NULL	Callback. A user callback function can be registered in open. If this callback function is provided, it is called from the interrupt service routine (ISR) each time the IRQn triggers. ATTENTION: Since the callback is called from an ISR, care should be taken not to use blocking calls or lengthy processing. Spending excessive time in an ISR can affect the responsiveness of the system.

Note: The example values and defaults are for a project using the Synergy S7G2 MCU Group. Other MCUs may have different default values and available configuration settings.

5.2 Thread Monitor Framework Module Clock Configuration

Use the ISDE to configure the WDT clock using the Clocks tab.

The WDT is initially based on the PCLKB frequency. Set the PCLKB frequency in the e^2 studio Integrated Solution Developer Environment (ISDE) using the clock configurator or the CGC Interface at run time.

With PCLKB running at 60 MHz, the WDT maximum timeout period in default value is approximately 2.24 seconds.

The formula is: Clock Division Ratio / PCLKB * Timeout Period (PCLKB Clock Cycles) = 8192 / 60MHz * 16384 ≈ 2.24s

The IWDT clock runs at 15 kHz resulting in a maximum possible timeout period just under 35 seconds.

The formula is: Clock Division Ratio / IWDTCLK * Timeout Period (IWDTCLK Clock Cycles) = 256 / 15kHz * 2048 ≈ 35s

Note: The settings of IWDT Timeout Period and IWDT Dedicated Clock Frequency Divisor can be found in Properties of BSP tab in configurator.

5.3 Thread Monitor Framework Module Pin Configuration

The Thread Monitor Framework does not require any pins for its operation.

5.4 Threads for the Thread Monitor Framework Module Application

Any thread monitored by the Thread Monitor Framework must be instrumented to work with the monitoring module. When a thread to be monitored is registered with the Thread Monitor, the expected minimum and maximum count values are passed via a pointer to a structure of type `sf_thread_monitor_counter_min_max_t` containing the values.

The thread's counter and minimum and maximum values must be registered with the Thread Monitor Framework by calling `g_sf_thread_monitor.p_api->threadRegister()`.

Each time round the monitored thread's loop, the counter value should be updated by calling `g_sf_thread_monitor.p_api->countIncrement()`.

5.5 Other Thread Monitor Framework Module Settings

The Thread Monitor Framework does not use any interrupts; the WDT/IWDT must be configured to generate a reset.

6. Using the Thread Monitor Framework Module in an Application

The typical steps in using the Thread Monitor Framework in an application are:

1. Initialize the Thread Monitor using the `open` API
2. Register thread that needs to be monitored with the `threadRegister` API
3. Use the `countIncrement` API to increment the count every time the registered thread is executed.
4. Unregister the thread with the `threadUnregister` API when there is no longer a need to monitor the registered thread.
5. Close the Thread Monitor with the `close` API (only if the Thread Monitor is no longer required in an application).

Often, the increment count step is repeated periodically. The following diagram shows these common steps in a typical operational flow:

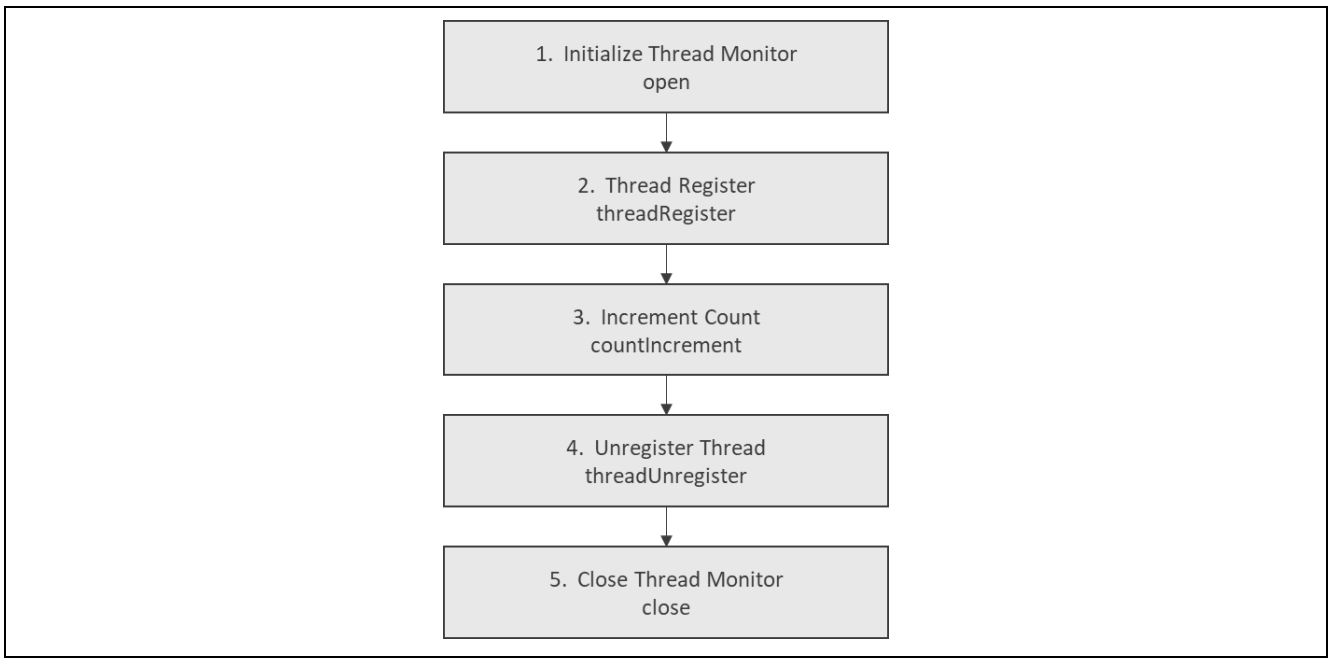

Figure 5. Typical Thread Monitor Framework Module Application

7. The Thread Monitor Framework Module Application Project

The application project associated with this module guide demonstrates the steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the Thread Monitor Framework module. You can also read over the code (in `monitor_thread_entry.c`, `monitored_thread1_entry.c`, and `monitored_thread2_entry.c`) used to illustrate the Thread Monitor Framework API's in a complete design.

The application project demonstrates the typical use of the Thread Monitor Framework APIs. The monitor thread initializes the Thread Monitor Framework. Monitor threads 1 and 2 register themselves in the thread monitor and enter a loop. The loop works until you press SW 4 (for monitored thread 2) or SW 5 (for monitored thread 1). In this case, the user button was pressed, the corresponding thread stops incrementing its counter, which consequently leads to a device reset. A LED is lit shortly before device reset — if thread 1 stops incrementing LED1 is lit, and if thread 2 stops incrementing LED2 is lit.

Table 7. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e² studio	v6.2.1	Integrated Solution Development Environment
SSP	v1.5.0	Synergy Software Platform
IAR EW for Renesas Synergy	v8.23.1	IAR Embedded Workbench® for Renesas Synergy™
SSC	v6.2.1	Synergy Standalone Configurator
SK-S7G2	v3.0, v3.1, v3.3	Starter Kit

The following figure illustrates a simple flow diagram of the application project:

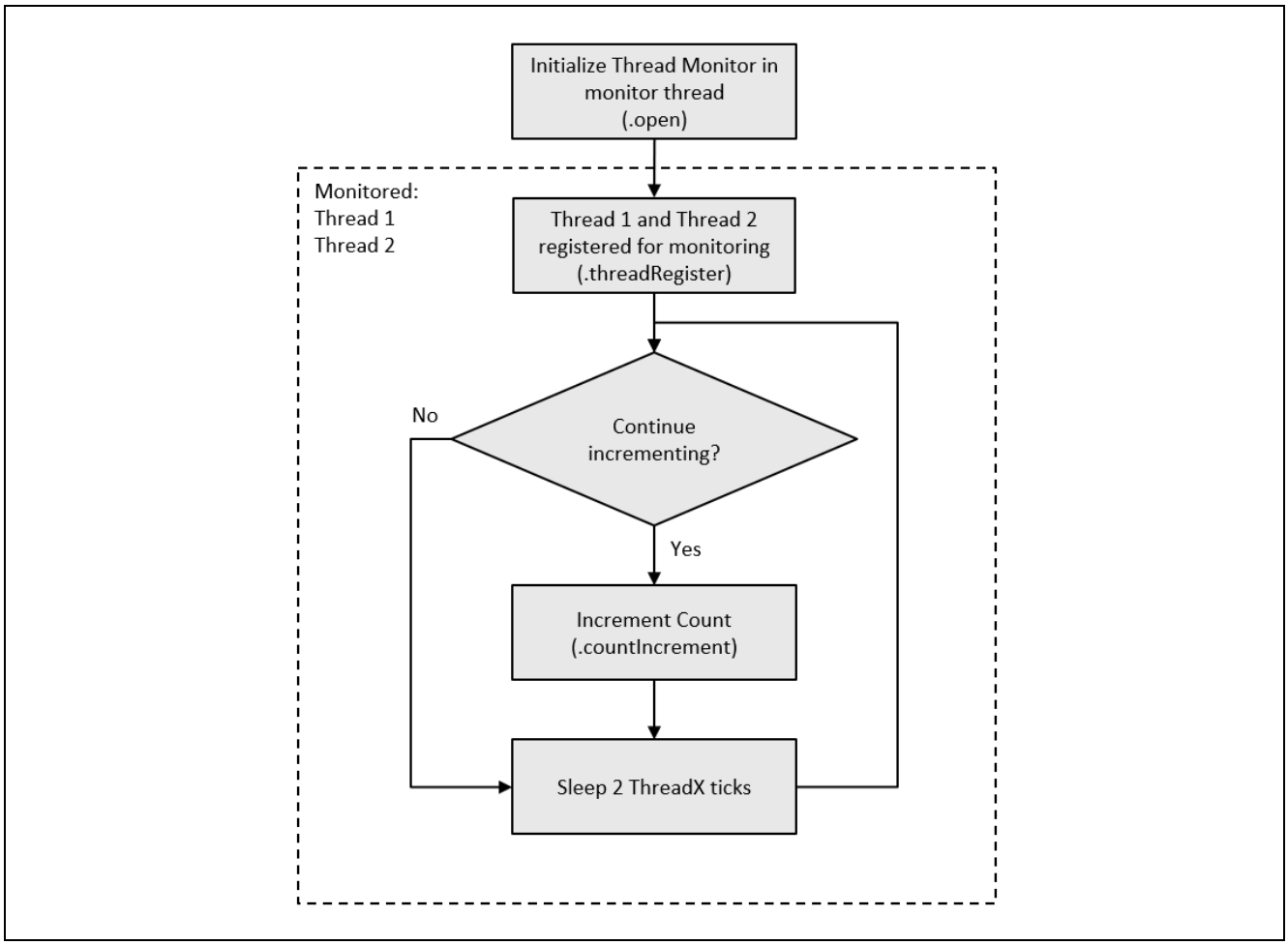

Figure 6. Thread Monitor Framework Implementation Block Diagram

The `monitor_thread_entry.c`, `monitored_thread1_entry.c`, and `monitored_thread2_entry.c` files located in the project, once it has been imported into the ISDE. You can open these files within the ISDE and follow along with the description provided to help identify key uses of APIs.

The first section of `monitor_thread_entry.c` has the header files referencing the Thread Monitor instance structure and has an LED structure, which facilitates onboard LED management. The next section is the entry function for the monitor thread. The Thread Monitor Framework is automatically initialized using the open API and is ready for usage. The function contains a very important line:

`R_DBG->DBGSTOPCR_b.DSWDT = 0` enables counting of the WDT in debug mode. After that, LEDs are initialized and turned off, and the thread enters an infinite loop.

Monitored threads 1 and 2 are very similar, their user-editable source files, `monitored_thread1_entry.c` and `monitored_thread2_entry.c`, are organized into sections: The first section is the include section to access the Thread Monitor, the `monitor_thread.h` file. Next is the variable that enables IRQ callbacks to

disable the incrementing of the monitored thread counter and external variable (LEDs structure) used to manipulate LED1 (monitored thread 1) or LED2 (monitored thread 2). After the variable comes the thread entry function. At the beginning, the thread count range structure is defined. The maximum count is set to 60 and the minimum is set to 50. The two values were determined using the profiling modes in the Thread Monitor Framework; they need to be updated in case the WDT configuration changes or the thread sleep-value is changed (currently set to 2) using the profiling mode in the Thread Monitor Framework.

For example, if the thread sleep-ticks number is doubled, the minimum and maximum values are twice as small. Monitored threads 1 and 2 were reporting minimum 55 and maximum 56 counts, so range 50 to 60 should be sufficient. After the thread count range is determined and configured, the flag used to stop incrementing the thread counter is set to false; this allows the thread to continue incrementing the counter and prevents the Thread Monitor/WDT from resetting the device. Subsequently, the External IRQ driver is opened to handle user buttons. As monitored, Thread 1 uses SW5; the driver needs to react to IRQ11. Similarly, as monitored thread 2 uses SW4, the driver needs to react to IRQ10. Finally, the thread enters a loop, sleeping for two ThreadX ticks, and checking the “disable increment” flag. If the flag is set to false, the thread increments its counter. If otherwise set, the counter does not increment and a device reset occurs.

After the entry function, an IRQ callback is defined (in both monitored thread entry files.) Basically, the IRQ callback functions (for monitored threads 1 and 2) set the “disable increment” flag to true and light the corresponding LEDs (LED1 for thread 1, LED2 for thread 2.)

A few key properties are configured in this application project. These properties are used to support the required operations and the physical properties of the target board and MCU. The following tables list the properties with the values set for this specific project. You can also open the application project and view these settings in the Properties window as a hands-on exercise.

Table 8. Thread Monitor Framework Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking	Default (BSP)
Maximum Number of Monitored Threads	2
Stack size of the Thread Monitor Thread (bytes)	512
Name	g_sf_thread_monitor
Profiling Mode	Disabled
Thread Monitor Thread Priority	1
Name of generated initialization function	sf_thread_monitor_init0
Auto Initialization	Enable

Table 9. Thread Monitor Framework — WDT Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking	Default (BSP)
Name	g_wdt
Start Mode	Register
Start Watchdog After Configuration	True
Timeout	16,384 Cycles
Clock Division Ratio	PCLK/8192
Window Start Position	75%
Window End Position	25%
Reset Control	Reset Output
Stop Control	WDT Count Enabled in Low Power Mode
NMI Callback	NULL

Table 10. Monitored Thread 1 — External IRQ Driver Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking	Default (BSP)
Name	g_external_irq10_sw5
Channel	10
Trigger	Rising
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
Interrupt enabled after initialization	True
Callback	irq10_sw5_callback
Interrupt Priority	Priority 1

Table 11. Monitored Thread 2 — External IRQ Driver Configuration Settings for the Application Project

ISDE Property	Value Set
Parameter Checking	Default (BSP)
Name	g_external_irq11_sw4
Channel	11
Trigger	Rising
Digital Filtering	Disabled
Digital Filtering Sample Clock (Only valid when Digital Filtering is enabled)	PCLK / 64
Interrupt enabled after initialization	True
Callback	irq11_sw4_callback
Interrupt Priority	Priority 2

8. Customizing the Thread Monitor Framework Module for a Target Application

Some configuration settings are normally changed by the developer from those shown in the application project. For example, the user can easily change the sleep intervals for monitored threads, which affect the number of the thread's counter increments. Alternatively, some changes to the WDT can be made. For example, the WDT timeout can be decreased to less than 16,384 cycles or the clock division ratio can be set to PCLKB/2048. Also, changing the Window Start and End positions affects the WDT characteristics by narrowing (or widening) the WDT counter ranges that allow for WDT tickling. After modifying parameters, the profiling procedure should be executed to determine each thread's minimum and maximum count values.

In the Thread Monitor Framework, enable profiling in the thread monitor's properties. Set the minimum and maximum structures for each monitored thread to have their maximum value set to 0 and minimum value set to any big number, for example 0xffffffff. When the application is running in debug mode, the following expressions can be watched to see the measured minimum, maximum, and current thread counters:

```
((sf_thread_monitor_instance_ctrl_t*)g_sf_thread_monitor.p_ctrl)        -
>thread_counters[X].current_count
((sf_thread_monitor_instance_ctrl_t*)g_sf_thread_monitor.p_ctrl)        -
>thread_counters[X].minimum_count
((sf_thread_monitor_instance_ctrl_t*)g_sf_thread_monitor.p_ctrl)        -
>thread_counters[X].maximum_count
```

Where "X" stands for the monitored thread index on Thread Monitor's thread list.

The current count is the count value measured until the application was stopped by a breakpoint. The minimum count is the minimum number of times the thread increments its counter during the current debugging session. The maximum count is the maximum number of times thread incrementing its counter during current debugging session. After the minimum and maximum counter values are measured, profiling mode should be disabled and the thread's minimum and maximum count structured should have their fields set to measured minimum and maximum counts. The user can consider the actual minimum value to be

slightly less than the measured minimum value and the actual maximum value to be slightly greater than the measured maximum value to provide some extra tolerance.

Finally, the WDT (Watchdog Timer) can be substituted with IWDT (Independent Watchdog Timer) for a more reliable thread-monitoring implementation.

9. Running the Thread Monitor Framework Module Application Project

To run the Thread Monitor Framework application project and to see it executed on a target kit, you can simply import it into your ISDE, compile and run debug.

Note: The following steps are described in sufficient detail for someone experienced with the basic flow through the Synergy development process. If these steps are unfamiliar, refer to the first few chapters of the *SSP User's Manual* for a description of how to accomplish these steps.

To create and run the Thread Monitor Framework application project, simply follow these steps:

1. Import and build the example project included with this module guide according to the *Renesas Synergy Project Import Guide* (r11an0023eu0121-synergy-ssp-import-guide.pdf).
2. Connect to the host PC via a micro USB cable to J19 on SK-S7G2.
3. Start to debug the application
4. Use the SW4 and SW5 buttons to invoke a device reset. After pressing a button, LED1 or LED2 lights until the device is reset.

10. Thread Monitor Framework Module Conclusion

This Module Guide has provided all the background information needed to select, add, configure and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy Platform makes these steps less time consuming and removes common errors, such as conflicting configuration settings or the incorrect selection of lower-level drivers. The use of high-level APIs (as demonstrated in the application project) illustrates additional development-time savings in allowing work to begin at a high level and avoiding the time required in older development environments to use, or, in some cases, create, lower-level drivers.

11. Thread Monitor Framework Module Next Steps

After you have mastered a simple Thread Monitor Framework example, you can proceed with more complex scenarios, composed of a larger number of threads with different minimum and maximum counter values specific to your actual business needs.

12. Thread Monitor Framework Module Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Synergy Gallery.

Links to all the most up-to-date sf_thread_monitor module reference materials and resources are available on the Synergy Knowledge Base: https://en-support.renesas.com/search/sf_thread_monitor%20Module%20Guide%20Resources.

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.31.17	–	Initial version
1.01	Jan.07.19	–	Updated for v1.5.0

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.4.0-1 November 2017)

Corporate Headquarters

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

Trademarks

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

Contact information

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