Renesas Synergy™ Platform

ThreadX® Source Module Guide

Introduction

This document provides an easy reference to users of the ThreadX® source component in e² studio. Properties are explained in greater detail than the footer comment supplied with each property. The included context-specific usage explains if and when to change a default value. This document should make it easier to use the ThreadX source component, without having to cross reference with the Express Logic ThreadX User Guide, and also help developers quickly get familiar with ThreadX features.

Contents

1. NetX and NetX Duo Source Module Features

The NetX and NetX Duo Source Module provides configurable options for a wide range of ThreadX properties. These include properties for:

- Timer settings
- Stack settings
- Priority settings

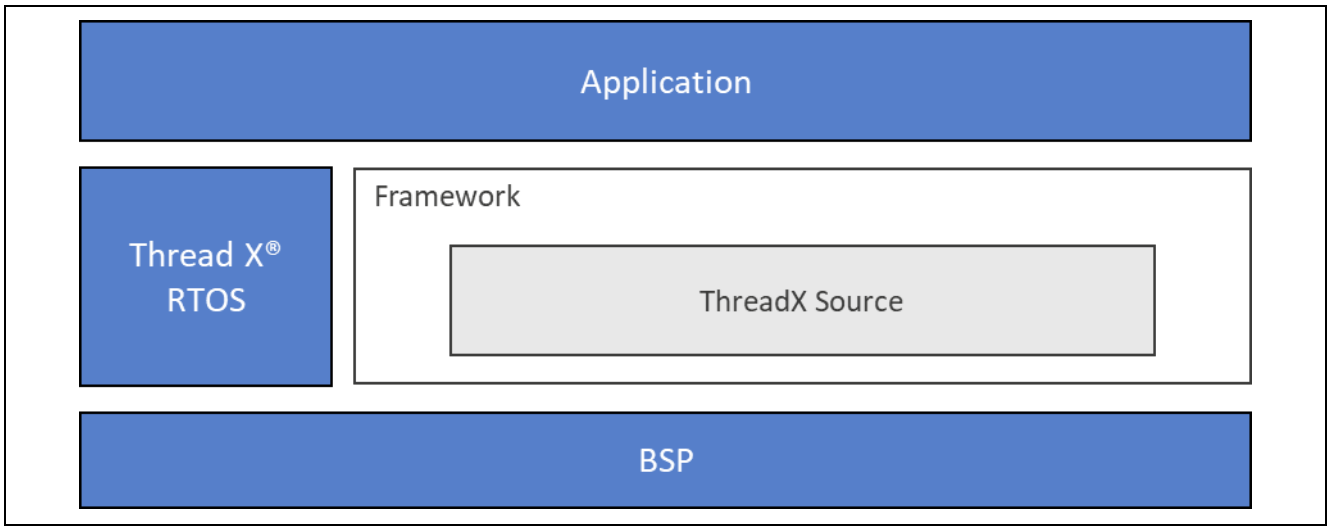

Figure 1. NetX and NetX Duo Source Module

2. ThreadX Module APIs Overview

There are no APIs associated with the ThreadX Source module. This module is used to configure various ThreadX properties.

3. ThreadX Module Operational Overview

The operational overview provides guidance on when to include the ThreadX source component and how changes are rebuilt for use by a project. The actual properties and their descriptions are in section 5 of this document.

3.1 When to Include the ThreadX Duo Source Component

Adding the ThreadX source component enables the developer in the Synergy configurator environment to customize the ThreadX library, change values from default settings, and enable or disable certain features. Otherwise, a developer must use the prebuilt ThreadX library. In most projects beyond the simplest, the developer typically wants to customize their ThreadX environment. Note that the ThreadX source component is automatically added when adding a higher-level source component (like NetX, NetX Duo, GUIX, or USBX).

Without adding the ThreadX source component, the e^2 studio configurator uses a prebuilt library with the ThreadX default settings.

3.2 Changing ThreadX Source Properties

After changing ThreadX property settings, the developer must click the **Generate Project Content** button to update the project configurator in e^2 studio. The ThreadX library **must** be rebuilt to rebuild the project. Simply changing a property (or applying a #define in the preprocessor list) without rebuilding the project does not have any affect. In that case, e^2 studio simply uses the previously built library.

4. Including the ThreadX Module in an Application

In the e^2 studio configurator you can add the ThreadX Source component by selecting the HAL/Common item from the Threads list and pressing the “New Stack” button and navigating the menu to X-Ware -> ThreadX -> ThreadX Source. The ThreadX Source component can be added to any of the application threads instead of adding it to the HAL/Common item, but if the thread is removed the configuration may be lost, so it is recommended to add it to the HAL/Common item.

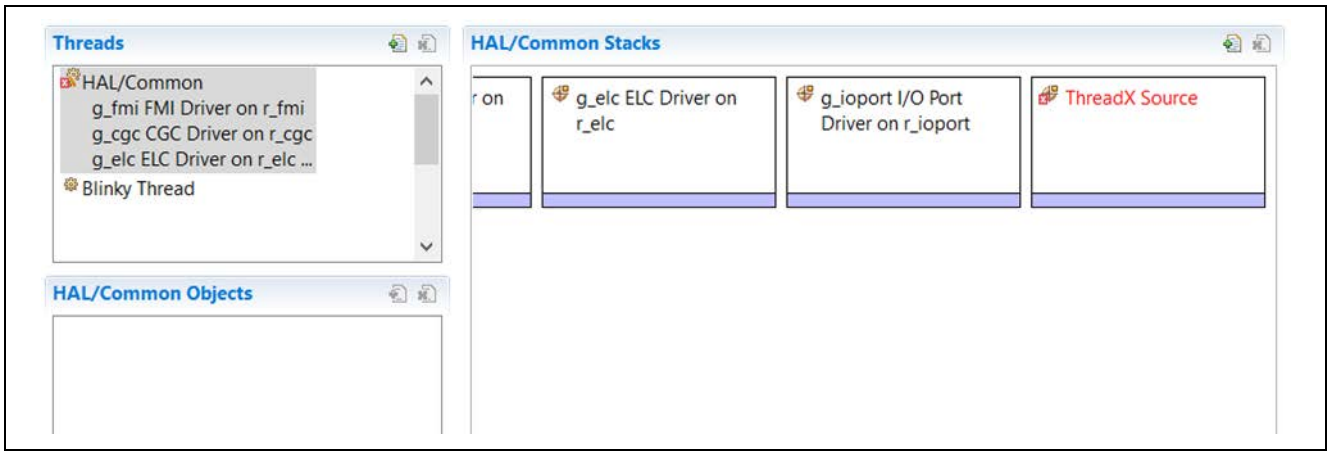

Figure 2. ThreadX Source Module Stack

5. Configuring the ThreadX Source Module

The list of properties below may not match the order of the ThreadX property table. They are grouped according to general category for easier reference.

Default settings are based on user experience and are often the choice applicable to common use cases.

Error Checking – default value enabled – Generally enabled during development and debugging phase and disabled when building a release version. When enabled, ThreadX includes error-checking services that check inputs and other parameters before calling the actual API. Some of the things checked include:

- NULL pointer input
- Invalid non-pointer parameters, such as an invalid IP address type or IP address equal to zero.
- Required configurable option must be enabled e.g. performance information must be enabled to call the get services.
- The data structure IDs must match what is expected, for example:
  thread_ptr -> tx_thread_id == TX_THREAD_ID /* check the IP instance structure */
- Size of the data structure e.g. the ThreadX thread, matches the size of the data structure in the ThreadX library.

These last two checks guard against an application using a different version of the ThreadX library than the application is using.

Timer ticks per second – default value not displayed, 100 ticks per second is used – Define the common timer tick reference for use by other middleware components. The default value is 10ms, but may be replaced by a port specific version in tx_port.h, or by the user as a compilation option. This affects all other components that may rely on timer timeouts for operation (specially communication protocols).

Max Priorities – default value not displayed, 32 priorities are used – Define the priority levels for ThreadX. Legal values range from 32 to 1024 and MUST be evenly divisible by 32.

Minimum Stack – default value not displayed, 200 is used – Define the minimum stack for a ThreadX thread on this processor. If the size supplied during thread creation is less than this value, the thread created call returns an error.

Timer Thread Stack Size – default value not displayed, 1024 is used – Define the system timer thread's default stack size and priority. These are only applicable if Timer Process in ISR is disabled.

Timer Thread Priority – default value not displayed, priority 0 is used – The timer thread priority should be the highest on the system. Using a lower value would cause timers to be preempted by other threads with hard to predict consequences.

Trace Time Mask – default value not displayed – TBD.

Timer Process in ISR – default value enabled Determine if timer expirations (application timers, timeouts, and tx_thread_sleep) calls should be processed within the system timer thread or directly in the timer ISR. By default, the timer expiration processing is done directly from the timer ISR, thereby eliminating the timer thread control block, stack, and context switching to activate it. The default setting saves memory and improves timer processing speed, but any timer code runs at interrupt level, blocking all threads on the

system. The default for this option is enabled for historical and compatibility reasons, but it is recommended to change it to be disabled.

Reactivate Inline - **default value disabled –** Determine if in-line timer reactivation should be used within the timer expiration processing. By default, this line timer is disabled and a function call is used. When enabled, reactivating is performed in-line resulting in faster timer processing but slightly larger code size.

Stack Filling – default value disabled – Determine is stack filling is enabled. By default, ThreadX stack filling is enabled, which places an 0xEF pattern in each byte of each thread's stack. This is used by e² studio ThreadX OS plugin by the ThreadX run-time stack checking feature.

Stack Checking – default value disabled – When enabled, stack filling is automatically enabled which is necessary for the stack checking logic.

Preemption Threshold – default value disabled - If the application does not use the preemption-threshold feature, it may be disabled to reduce code size and improve performance. For more information on Preemption Threshold see the ThreadX User Guide.

Redundant Clearing – default value enabled – Determine if global ThreadX variables should be cleared. If the compiler startup code clears the .bss section prior to ThreadX running, this option can be used to eliminate unnecessary clearing of ThreadX global variables.

No Timer – default value disabled – Determine if no timer processing is required. This option helps eliminate the timer processing when not needed. The user also has to comment out the call to tx_timer_interrupt, which is typically made from assembly language in tx_initialize_low_level.

Notify Callbacks – default value disabled – Determine if the notify callback option should be disabled. By default, notify callbacks are disabled. If the application does not use notify callbacks, they may be disabled to reduce code size and improve performance.

Inline Thread Resume Suspend – default value disabled – Determine if the tx_thread_resume and tx_thread_suspend services should have their internal code in-line. This results in a larger image, but improves the performance of the thread resume and suspend services.

Not Interruptible – default value disabled – Determine if the internal ThreadX code is non-interruptible. This results in smaller code size and less processing overhead, but increases the interrupt lockout time.

5.1 Using TraceX with ThreadX

If TraceX is enabled in the ThreadX source component, the project containing the ThreadX library must be rebuilt, or TraceX macros that log events is not executed.

Event Trace – default value disabled – Determine if the trace event logging code should be enabled. This causes slight increases in code size and overhead, but provides the ability to generate system trace information that is available for viewing in TraceX.

Trace Buffer Name– default value g_tx_trace_buffer – When event trace is enabled, this is the name of the variable that holds the trace events. This is a circular buffer that is statically allocated for this use (a global variable).

Trace Buffer Size – default value 65536 (64 kBytes) – When event trace is enabled, this is the siye of the buffer that holds the trace records. A bigger buffer holds more information, allowing for a longer recording time, but it uses RAM that is not be available to other parts of the application.

Trace Buffer Number of Registries – default value 30 – This is the total number of objects for which trace information is recorded. An object in this sense is a thread, a mutex, a semaphore, and so on; or other type of objects that generate trace information from higher level software stacks, like NetX (IP instance, …), FileX (media, file, …) or others.

5.1.1 ThreadX Statistics Properties

If enabled, ThreadX keeps statistics on its internal operations. These statistics are at the component level, e.g. thread, mutex, semaphore, and others. Disabling these statistics reduces processing time slightly.

The value of these statistics is to be able to diagnose problems or optimize application performance without having to stop or interrupt program flow or write tedious debug code.

Examples:

tx_byte_pool_info_get.c

If an application is experiencing unexpected memory fragmentation in a byte pool, check the fragments created statistic. This is incremented every time an allocation creates a new memory fragment.

tx_mutex_info_get.c

If an application is suspected of experiencing priority inversion situations, these can be verified by checking the mutex priority inversion statistic.

The following is a partial list of APIs for ThreadX statistics.

Thread info – Statistics at the level of each thread:

```
UINT tx_thread_info_get(TX_THREAD *thread_ptr,
                        CHAR **name,
                        UINT *state,
                        ULONG *run_count,
                        UINT *priority,
                        UINT *preemption_threshold,
                        ULONG *time_slice,
                        TX_THREAD **next_thread,
                        TX_THREAD **next_suspended_thread);
```

Thread Performance Info – Performance Statistics on each thread:

```
UINT tx_thread_performance_info_get(TX_THREAD *thread_ptr,
                                    ULONG *resumptions,
                                    ULONG *suspensions,
                                    ULONG *solicited_preemptions,
                                    ULONG *interrupt_preemptions,
                                    ULONG *priority_inversions,
                                    ULONG *time_slices,
                                    ULONG *relinquishes,
                                    ULONG *timeouts,
                                    ULONG *wait_aborts,
                                    TX_THREAD **last_preempted_by);
```

Thread System Performance Info – Performance Statistics for all threads on the system:

```
UINT tx_thread_performance_system_info_get(ULONG *resumptions,
                                           ULONG *suspensions,
                                           ULONG *solicited_preemptions,
                                           ULONG *interrupt_preemptions,
                                           ULONG *priority_inversions,
                                           ULONG *time_slices,
                                           ULONG *relinquishes,
                                           ULONG *timeouts,
                                           ULONG *wait_aborts,
                                           ULONG *non_idle_returns,
                                           ULONG *idle_returns);
```

Block Pool Performance Info – default value disabled – Determine if block pool performance gathering is required by the application. When enabled, ThreadX gathers various block pool performance information. This increases the size of each block pool structure and has a small impact on performance.

Byte Pool Performance Info – default value disabled – Determine if byte pool performance gathering is required by the application. When enabled, ThreadX gathers various byte pool performance information. This increases the size of each byte pool structure and has a small impact on performance.

Event Flags Performance Info – default value disabled – Determine if event flags performance gathering is required by the application. When enabled, ThreadX gathers various event flags performance information. This increases the size of each byte pool structure and has a small impact on performance.

Mutex Performance Info – default value disabled – Determine if mutex performance gathering is required by the application. When enabled, ThreadX gathers various mutex performance information. This increases the size of each byte pool structure and has a small impact on performance.

Queue Performance Info – default value disabled - Determine if queue performance gathering is required by the application. When enabled, ThreadX gathers various queue performance information. This increases the size of each byte pool structure and has a small impact on performance.

Semaphore Performance Info – default value disabled – Determine if semaphore performance gathering is required by the application. When enabled, ThreadX gathers various semaphore performance information. This increases the size of each byte pool structure and has a small impact on performance.

Thread Performance Info – default value disabled – Determine if thread performance gathering is required by the application. When enabled, ThreadX gathers various thread performance information. This increases the size of each byte pool structure and has a small impact on performance.

Timer Performance Info– default value disabled – Determine if timer performance gathering is required by the application. When enabled, ThreadX gathers various timer performance information. This increases the size of each byte pool structure and has a small impact on performance.

Hardware Thread Stack Monitoring – default value enabled – When enabled, the Synergy Hardware Stack Monitors are used to monitor threads for stack overflow conditions.

Systick Interrupt Priority – default value priority value is 0 (highest) -- The systick timer thread priority should be the highest on the system. Allows ThreadX to carry out context switching to support multiple tasking. Using a lower value would cause systick handler to be preempted by other threads with hard to predict consequences.

IAR Library Support – default value dependent on compiler – When enabled, ThreadX does the necessary adjustments to work correctly with the IAR C run-time library.

EPK Support – default is disabled – When enabled, the Execution Profile Kit code is used. Refer to the Execution Profile Kit documentation for further information. Note that enabling this option does not generate the EPK code, this code needs to be added to the project by the user.

5.1.2 ThreadX Extension Points

The following is a list of extension points that can be used to add custom data to different structures defined by ThreadX.

TX_THREAD_EXTENSION_0 – default value empty
TX_THREAD_EXTENSION_1 – default value empty
TX_THREAD_EXTENSION_2 – default value empty
TX_THREAD_EXTENSION_3 – default value empty – Define the TX_THREAD control block extensions for this port. The main reason for the multiple macros is so that backward compatibility can be maintained with existing ThreadX kernel awareness modules. These macros are used inside the TX_THREAD structure definition and give an opportunity to incorporate its own data into this structure. Note that other software (including other Express Logic software components) may already use these extension points and can conflict with user definitions made here. If the user would like to add its own data to the structure, a safer place would be the TX_THREAD_USER_EXTENSION macro.

TX_BLOCK_POOL_EXTENSION – default value empty – Extension point used to add custom data to the block pool structure.
TX_BYTE_POOL_EXTENSION – default value empty – Extension point used to add custom data to the byte pool structure.

Note: For information on other extension macros, check the ThreadX User Guide.

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

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Oct.23.17	—	Initial Release
1.01	Apr.30.19	—	Updates for SSP v1.6.0

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/.