ThreadX RTOS porting with EFR32

Redpine Apps Engineering

Exported on 12/14/2022

Table of Contents

1 Introduction

A real-time operating system (RTOS) is a software that supplements computer hardware complexities. An RTOS rapidly switches between the tasks, giving the impression that multiple programs are being executed at the same time on a single processing core.

Azure RTOS ThreadX is a high-performance real-time kernel designed specifically for embedded applications. Benefits of using ThreadX RTOS include improved responsiveness, ease of use, reduced software maintenance. ThreadX also has a number of add-on modules (TCP/IP network stack, FAT filesystem, GUI framework, USB stack) that can be used according to requirements.

By default, ThreadX has 32 priority levels, ranging from priority 0 through priority 31. Numerically smaller values imply higher priority. Hence, priority 0 represents the highest priority, while priority (**TX_MAX_PRIORITIES**-1) represents the lowest priority.

2 Objective

This document explains how to port ThreadX RTOS to a featured example in silabs SDK to work with EFR32 Host MCU and RS9116.

3 Prerequisite

3.1 Hardware

- RS9116 EVK Board
- USB Cables
- EFR32 Host MCU
- Interconnect board and SPI ribbon cable
- Windows PC to run Simplicity Studio

3.2 Software

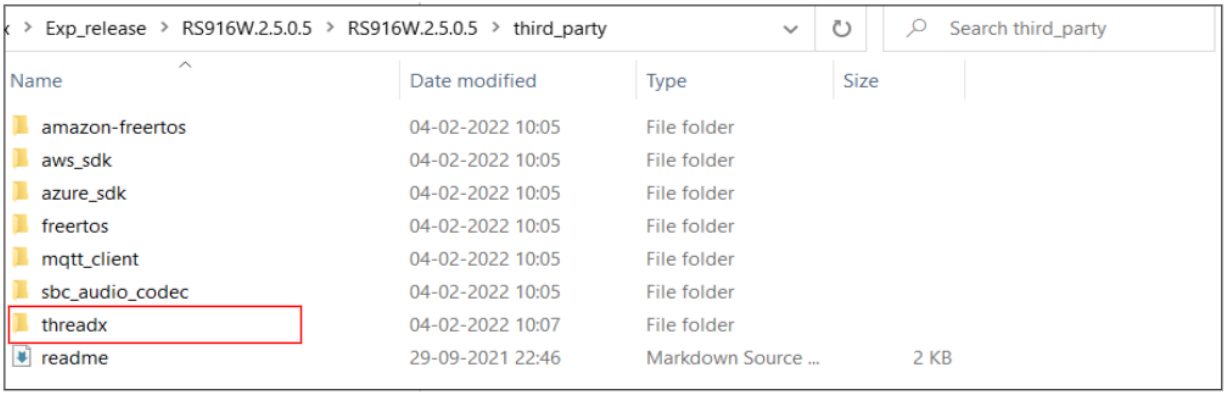
- Simplicity Studio v5 or above
- RS9116W release SDK
- Azure RTOS ThreadX library

4 Porting steps (Azure RTOS ThreadX)

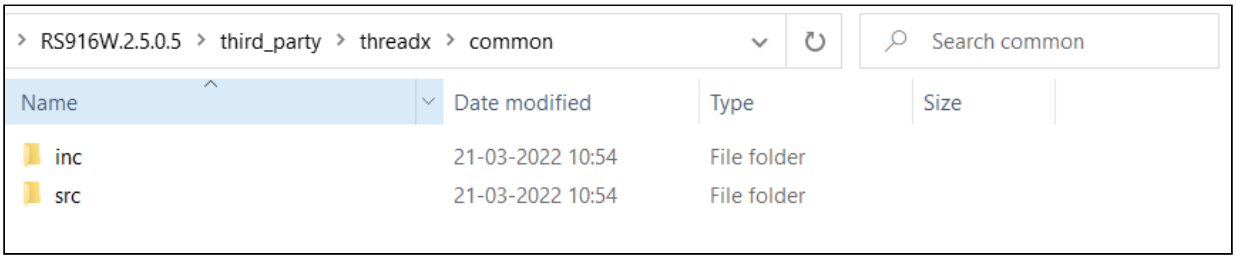
Note:
In this guide, wlan_throughput_brd_4180b_gg11 example from RS9116W 2.5.0.5 release SDK is used as reference for explanation.

4.1 Step-1: Get ThreadX library

- Downalod ThreadX RTOS library named 'threadx-master' form link[1]
- Download 2.5.0.5 release SDK from link[2]
- In the Silicon Labs release SDK, create a folder named ‘threadx’ in <release>/third_party

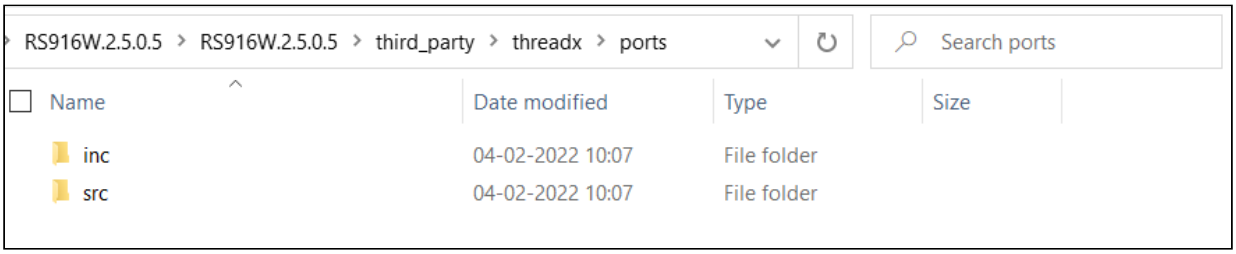


- Create a folder named ‘common’ in <release>/third_party/threadx
- Copy threadx-master/common/inc and threadx-master/common/src folders to <release>/third_party/threadx/common/



- Create a folder named ‘ports’ in <release>/third_party/threadx
- Copy ‘inc’ and ‘src’ folders present in threadx-master/ports/cortex_m4/gnu/ to <release>/third_party/threadx/ports/

1 https://github.com/azure-rtos/threadx/
2 https://www.dropbox.com/s/uiahvc95i0b55q9/RS916W.2.5.0.5.zip?dl=0

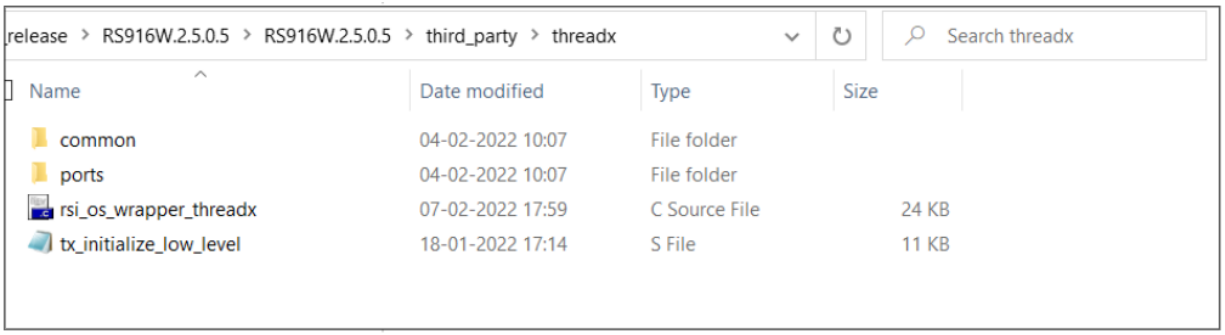


- Copy threadx-master/ports/cortex_m4/gnu/example_build/tx_initialize_low_level.s file to <release>/third_party/threadx/
- An example os porting file named 'rsi_os_wrapper_threadx.c' is available at , download the file and copy it to <release>/third_party/threadx

Note:

1. The sample OS porting file is made for ThreadX RTOS. The detailed explanation of rsi_os_wrapper_threadx.c file is present in Appendix.
2. This is an example OS porting file used for reference, you can create your own porting file as per requirements using the information given in Appendix.

- The threadx folder will contain following contents



4.2 Step-2: Project setup

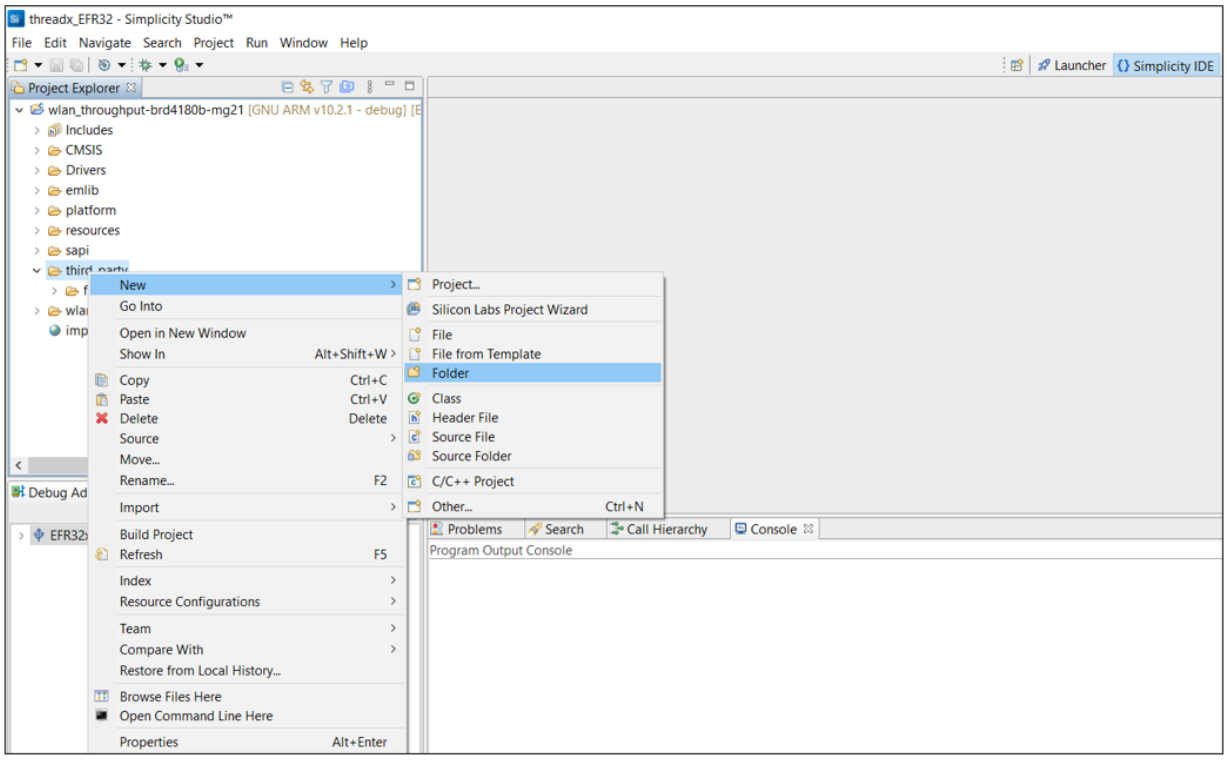
In this step, we will import the project to Simplicity Studio and include all files and folders related to ThreadX library and rsi_os_threadx_wrapper

4.3 **Import the project to Simplicity Studio**

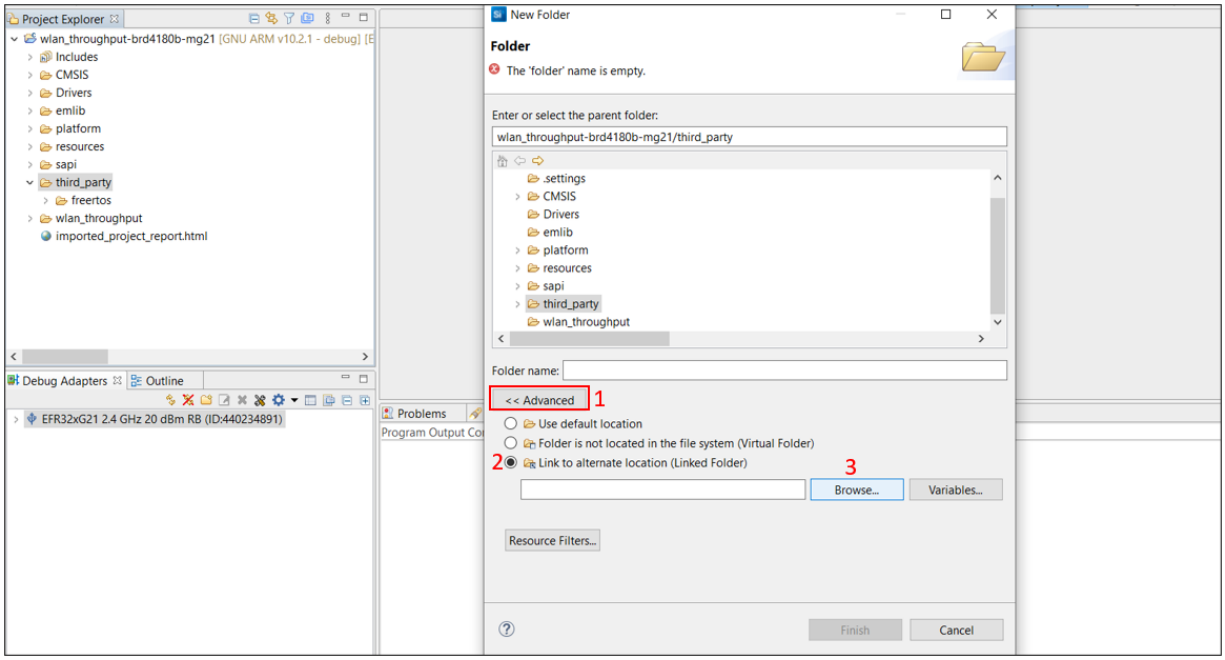
- In simplicity studio, click on ***File***, select ***import***, and click on ***Browse***
- Navigate to the *release_SDK/examples/featured/wlan_throughput/projects*, click on ***Select Folder***
- Choose the project file according to you board, click on ***Next*** → ***Next*** → ***Finish***
- After successfully importing the project, the project name appears in the 'Project Explorer' tab

4.4 Update file configurations

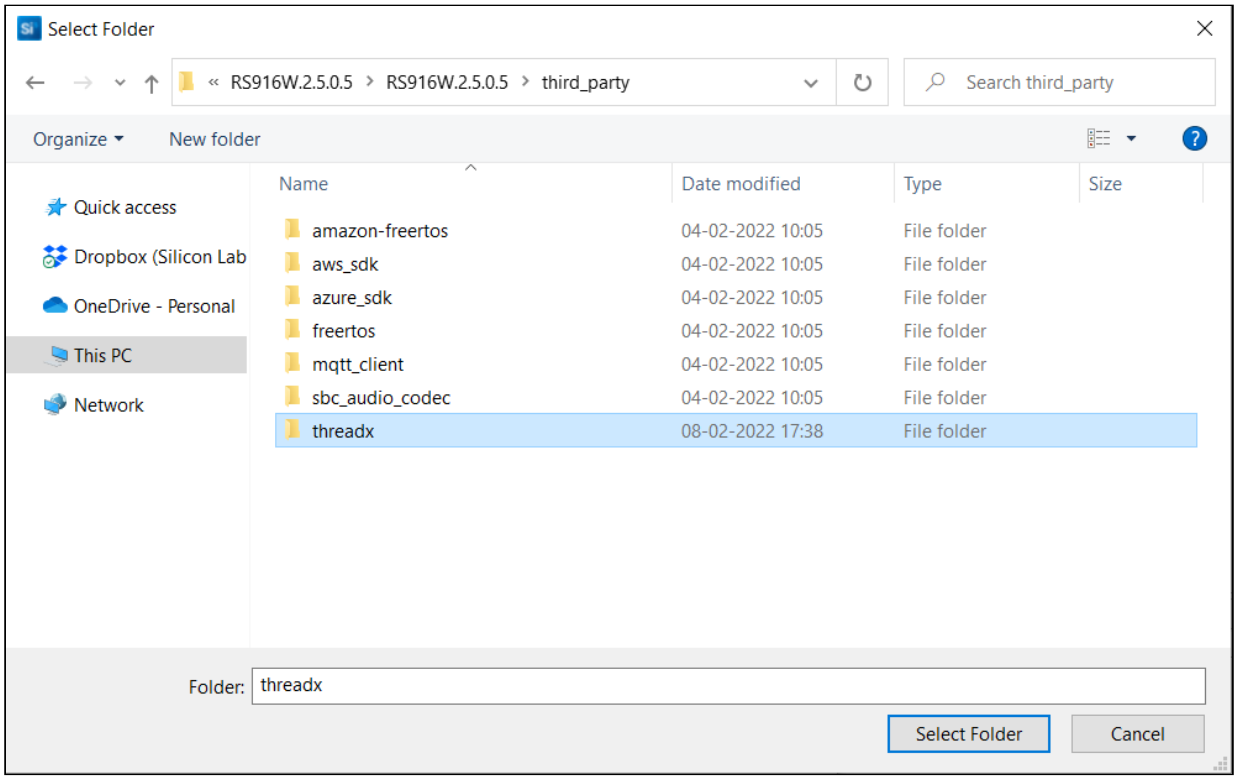
- Expand the project, right-click on *third_party* and select ***New*** → ***Folder***



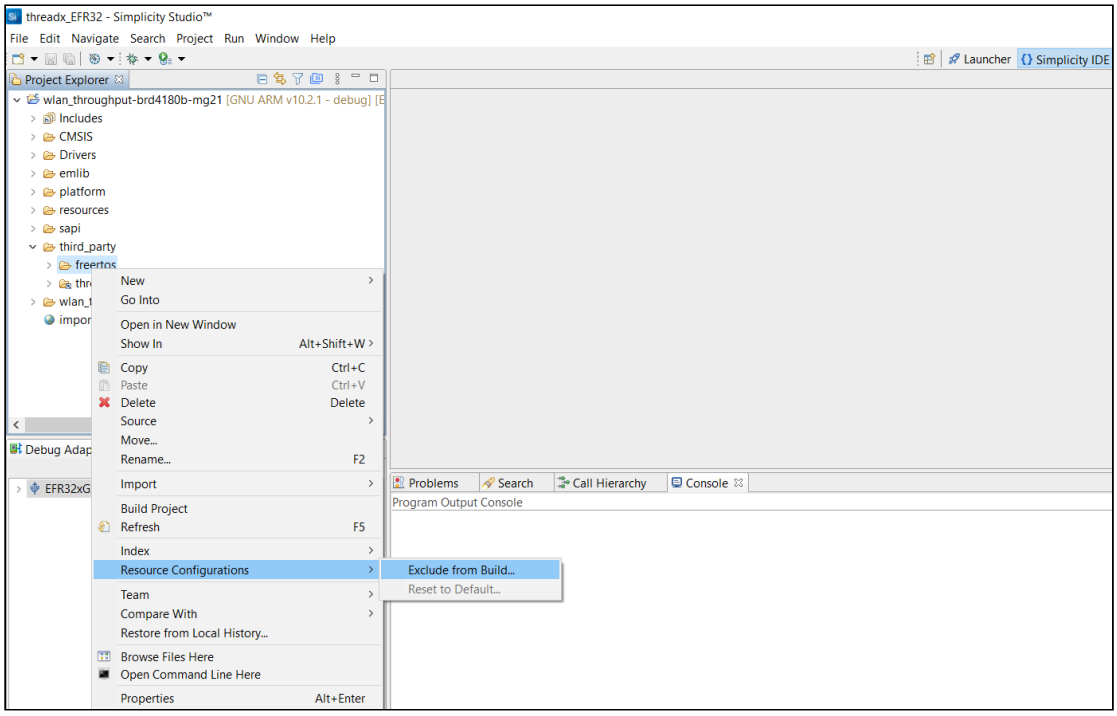
- Select ***Advanced → Link to alternate location → Browse***



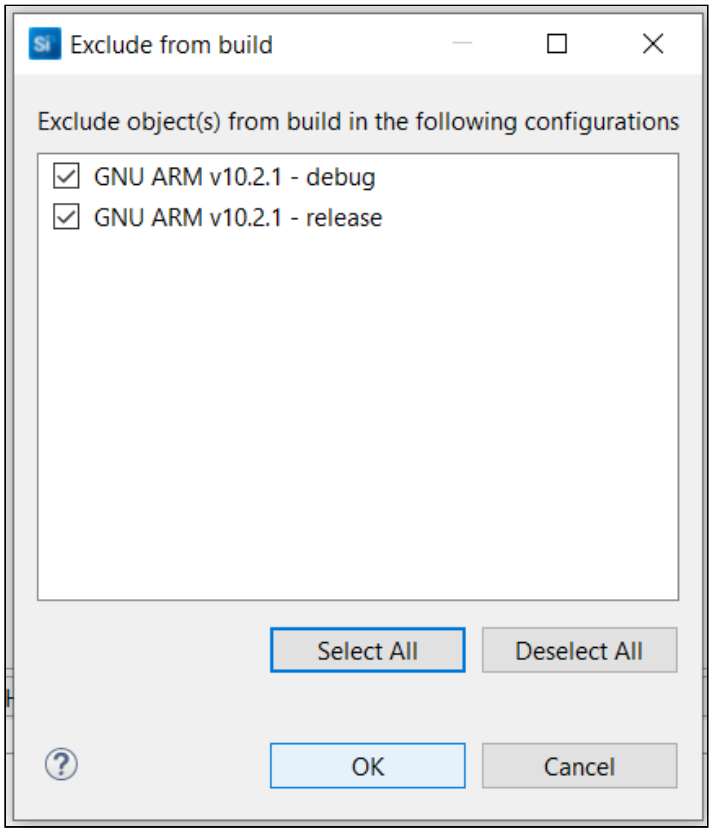
- Navigate to *release_SDK/third_party*, select threadx folder, click ***Select Folder***



- Click on ***Finish*** to complete the process
- Exclude FreeRTOS related files from build
  - Right click on **<project_name>/third_party/freertos** folder, select ***Resource configurations*** → ***Exclude from build***

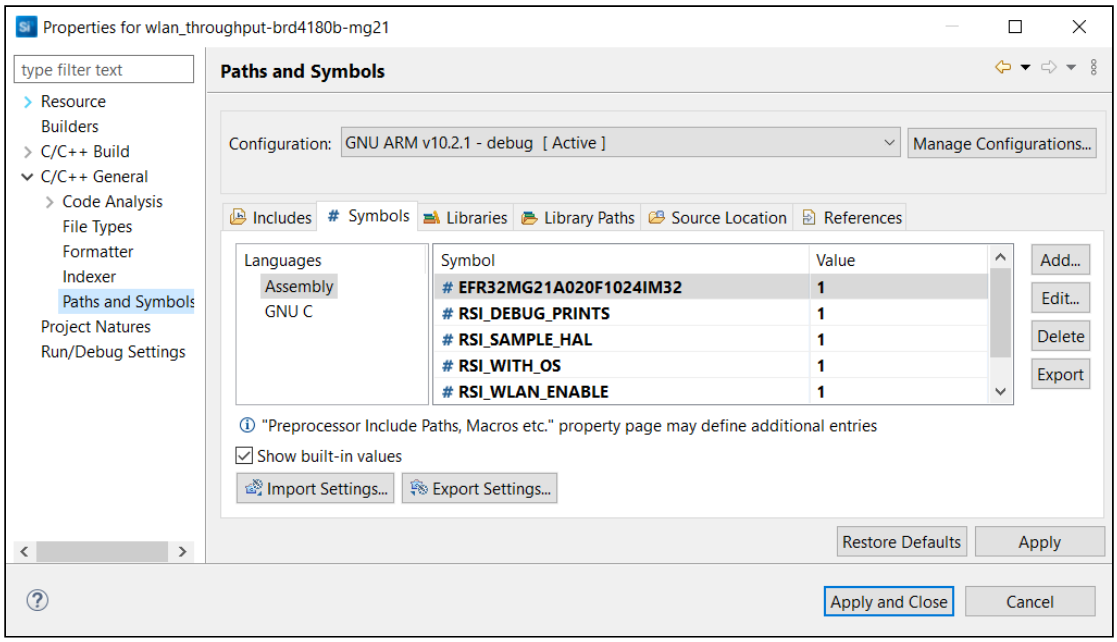


  - Select All. Click ***OK*** to save changes.

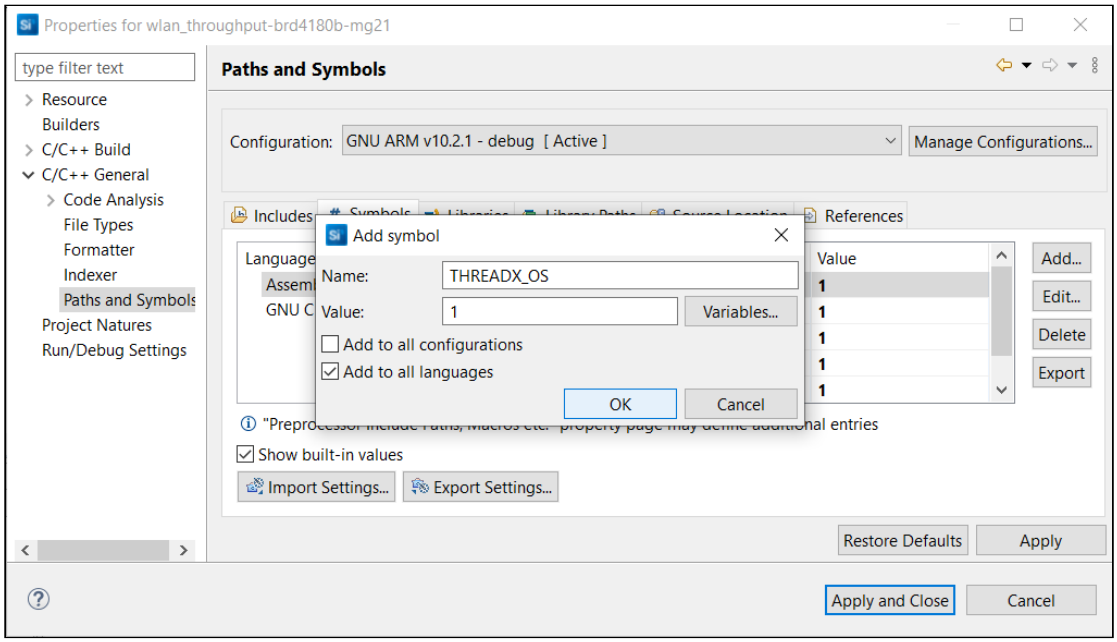


- Repeat the process to exclude ***<project_name>/sapi/rtos/port.c*** and ***<project_name>/sapi/rtos/rsi_os_wrapper.c*** files

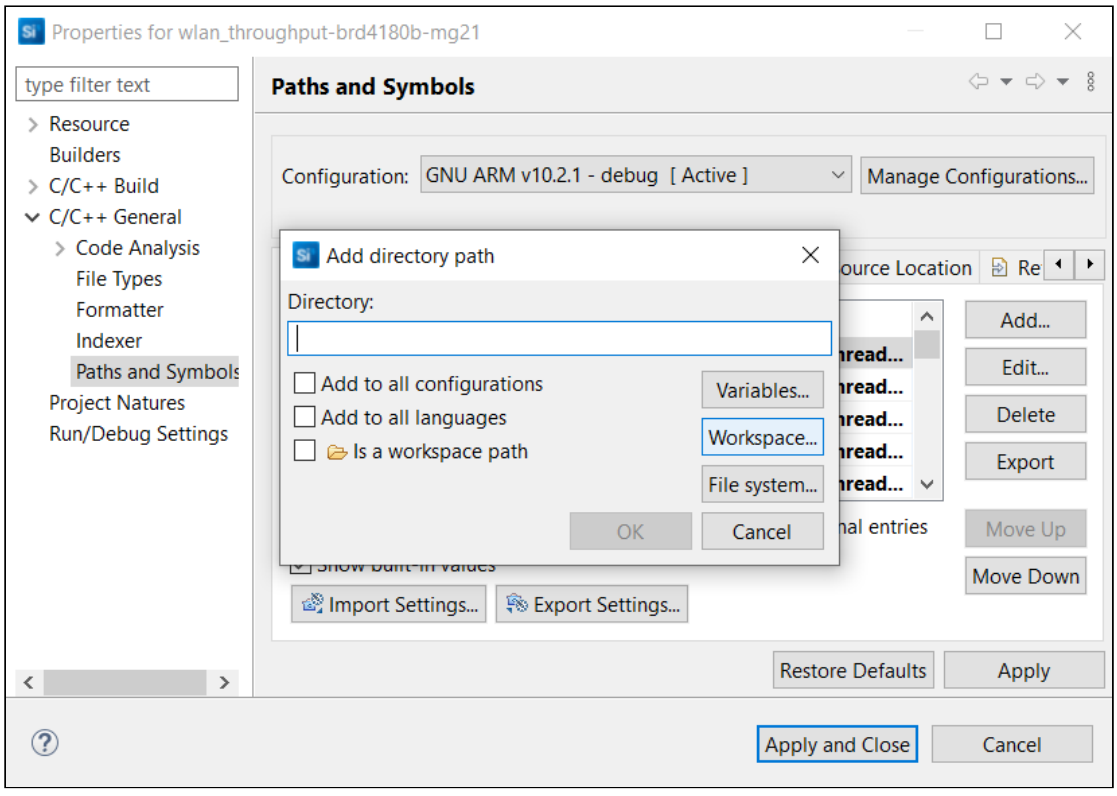
- Add THREADX_OS macro to the file system
  - Right-click on project name and select ***properties*** option
  - Go to ***C/C++ General → Paths and Symbols → # Symbols → Assembly***



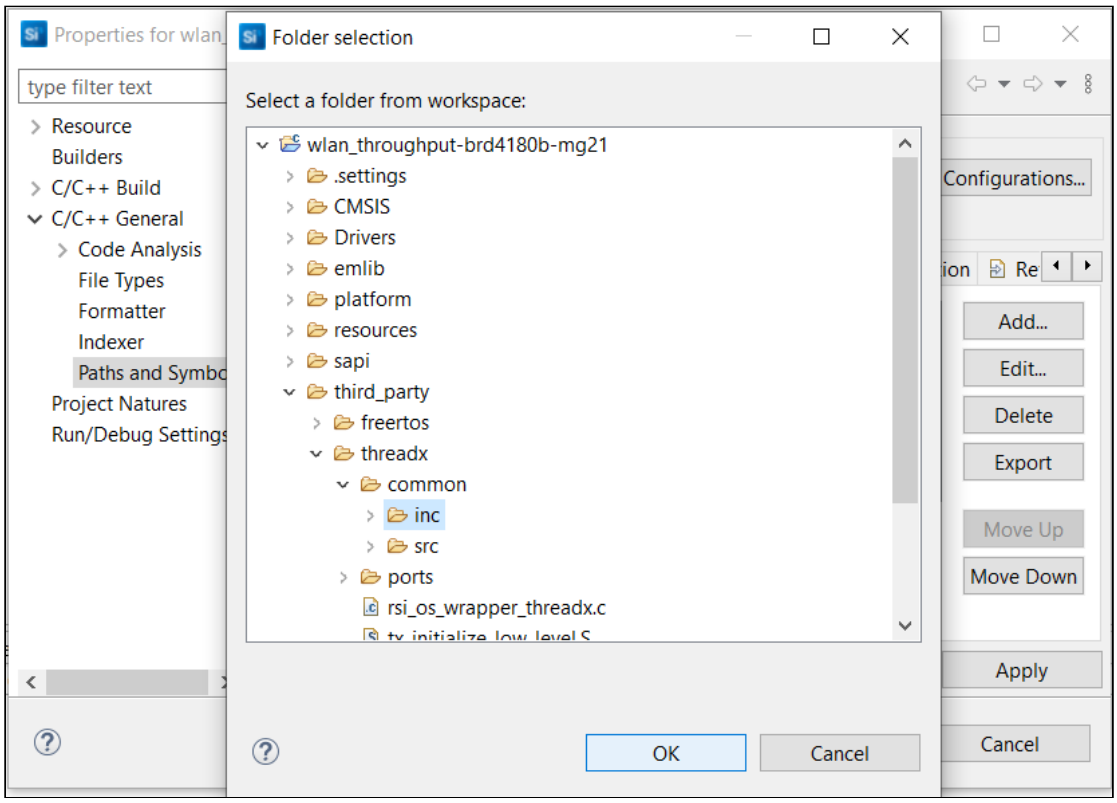
- Click on **Add**. Enter Name: THREADX_OS, Value: 1
- Select **Add to all languages**, click on **OK** → **Apply and Close**



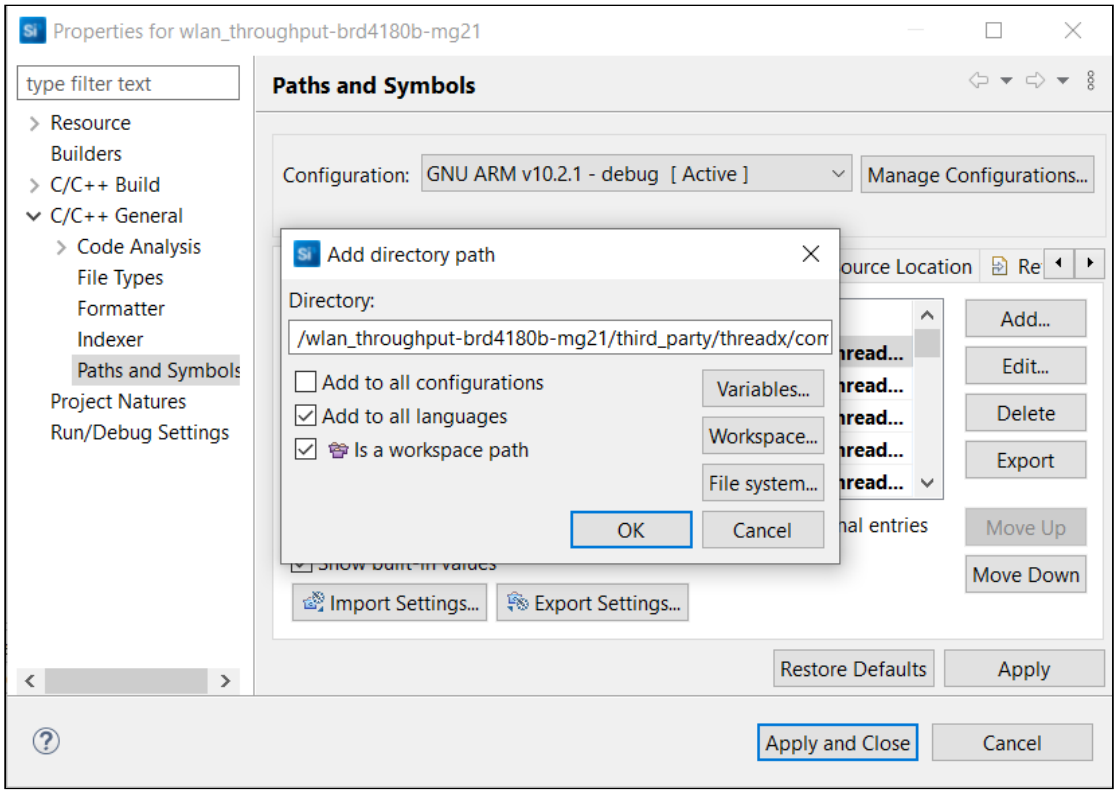
- Add includes for ThreadX RTOS files
  - Right-click on the project name and select **properties** option
  - Go to ***C/C++ General*** → ***Paths and Symbols*** → ***includes*** → ***Assembly***. Click on ***Add***.
  - Select ***Workspace***



- Navigate to the ***<project_name>/third_party/threadx/common***. Select **inc** folder, click **OK**



- Now, select **Add to all Languages**, click **OK**



- Repeat the process and add ***<project_name>/third_party/threadx/ports/inc*** folder to includes.
- Click **Apply and Close** to save the changes

4.5 Code Changes

sapi/include/rsi_os.h file

- Add an include to 'tx_api.h' file

```
#ifdef THREADX_OS
#include "tx_api.h"
#endif
```

```
/*****************************************************************************

#ifndef RSI_OS_H
#define RSI_OS_H
#include "rsi_error.h"
#ifdef THREADX_OS
#include "tx_api.h"
#endif
/*****************************************************
 * *                        Macros
 * ****************************************************/
```

- Modify defines related to semaphore, mutex and task handlers as below

```
typedef uint32_t rsi_reg_flags_t;
#ifndef THREADX_OS
  // Handle to manage Semaphores.
  typedef uint32_t rsi_semaphore_handle_t;
  // Handle to manage Mutex.
  typedef uint32_t rsi_mutex_handle_t;
  // Task handler
  typedef void *rsi_task_handle_t;
#else
  //Handle to manage Semaphores.
  typedef TX_SEMAPHORE rsi_semaphore_handle_t;
  // Handle to manage Mutex.
  typedef TX_MUTEX rsi_mutex_handle_t;
  // Task handler
  typedef TX_THREAD *rsi_task_handle_t;
#endif
```

```
/*************************************************************
 * *                    Type Definitions
 * ************************************************************/
typedef uint32_t rsi_reg_flags_t;
#ifndef THREADX_OS
  // Handle to manage Semaphores.
  typedef uint32_t rsi_semaphore_handle_t;
  // Handle to manage Mutex.
  typedef uint32_t rsi_mutex_handle_t;

  // Task handler
  typedef void *rsi_task_handle_t;
#else
  // Handle to manage Semaphores.
  typedef TX_SEMAPHORE rsi_semaphore_handle_t;
  // Handle to manage Mutex.
  typedef TX_MUTEX rsi_mutex_handle_t;

  // Task handler
  typedef TX_THREAD *rsi_task_handle_t;
#endif

```

platform/hal/rsi_hal_mcu_platform_init.c file

- Change SYSTICK interrupt priority as

```
#ifndef THREADX_OS
#define SYSTICK_INTR_PRI    ((1<<__NVIC_PRIO_BITS)-1)
#else
#define SYSTICK_INTR_PRI  0
#endif
```

```
//! systick interrupt priority
#ifndef THREADX_OS
#define SYSTICK_INTR_PRI  ((1<<__NVIC_PRIO_BITS)-1)
#else
#define SYSTICK_INTR_PRI 0
#endif
```

platform/hal/rsi_hal_mcu_timer.c file

- Modify includes for RTOS files

```
#ifdef RSI_WITH_OS
  #ifndef THREADX_OS
    /* FreeRTOS includes. */
    #include "FreeRTOS.h"
    #include "task.h"
    #include "timers.h"

    #if defined(SysTick)
      #undef SysTick_Handler
      /* FreeRTOS SysTick interrupt handler prototype */
      extern void SysTick_Handler     (void);
      /* FreeRTOS tick timer interrupt handler prototype */
      extern void xPortSysTickHandler (void);
      /* SysTick */
    #endif
  #else
    /*THREADX includes*/
    #include "tx_api.h"
    #include "tx_trace.h"
    #include "tx_thread.h"
    #include "tx_timer.h"
    #include "time.h"
  #endif
#endif
```

```
#ifdef RSI_WITH_OS
  #ifndef THREADX_OS
    /* FreeRTOS includes. */
    #include "FreeRTOS.h"
    #include "task.h"
    #include "timers.h"

    #if defined(SysTick)
      #undef SysTick_Handler
      /* FreeRTOS SysTick interrupt handler prototype */
      extern void SysTick_Handler     (void);
      /* FreeRTOS tick timer interrupt handler prototype */
      extern void xPortSysTickHandler (void);
    #endif  /* SysTick */
  #else
    #include "tx_api.h"
    #include "tx_trace.h"
    #include "tx_thread.h"
    #include "tx_timer.h"
    #include "time.h"
  #endif
#endif
/* Counts 1ms timeTicks */
```

- Modify rsi_delay_ms() function, add a call to threadx API

```
#ifdef RSI_WITH_OS
  #ifndef THREADX_OS
    vTaskDelay(delay_ms);
  #else
    tx_thread_sleep(delay_ms);
  #endif
#else
  start = rsi_hal_gettickcount();
  do {
  } while (rsi_hal_gettickcount() - start < delay_ms);
#endif
```

```
void rsi_delay_ms(uint32_t delay_ms)
{
#ifndef RSI_WITH_OS
   uint32_t start;
#endif
  if (delay_ms == 0)
    return;

#ifdef RSI_WITH_OS
#ifndef THREADX_OS
  vTaskDelay(delay_ms);
#else
  tx_thread_sleep(delay_ms);
#endif
#else
  start = rsi_hal_gettickcount();
  do {
  } while (rsi_hal_gettickcount() - start < delay_ms);
#endif
}
```

- Add condition check for SysTickHandler() function

```
#ifndef THREADX_OS
void SysTick_Handler(void)
{
  /* Increment counter necessary in Delay()*/
  msTicks++;
  #ifdef RSI_WITH_OS
    if (xTaskGetSchedulerState() != taskSCHEDULER_NOT_STARTED) {
        xPortSysTickHandler();
      }
  #endif
}
#endif
```

```
197 #ifndef RSI_HAL_USE_RTOS_SYSTICK
198 /*
199   SysTick handler implementation that also clears overflow flag.
200 */
201 #ifndef THREADX_OS
202 void SysTick_Handler(void)
203 {
204   /* Increment counter necessary in Delay()*/
205   msTicks++;
206 #ifdef RSI_WITH_OS
207   if (xTaskGetSchedulerState() != taskSCHEDULER_NOT_STARTED) {
208     xPortSysTickHandler();
209   }
210 #endif
211 }
212 #endif
```

third_party/threadx/tx_low_level_initialize.s file

- Add condition check for dynamic memory allocation

```
#ifdef USE_DYNAMIC_MEMORY_ALLOCATION
    LDR     r0, =_tx_initialize_unused_memory       @ Build address of
unused memory pointer
    LDR     r1, =__RAM_segment_used_end__           @ Build first free
address
    ADD     r1, r1, #4                              @
    STR     r1, [r0]                                @ Setup first unused
memory pointer
#endif
```

```
102@    /* Set base of available memory to end of non-initialised RAM area.  */
103@
104#ifdef USE_DYNAMIC_MEMORY_ALLOCATION
105    LDR     r0, =_tx_initialize_unused_memory       @ Build address of unused memory pointer
106    LDR     r1, =__RAM_segment_used_end__           @ Build first free address
107    ADD     r1, r1, #4                              @
108    STR     r1, [r0]                                @ Setup first unused memory pointer
109#endif
110@
```

- Change '_vectors' to '__Vectors' in the next code snippet (double underscore)

```
@
@    /* Setup Vector Table Offset Register.  */
@
    MOV     r0, #0xE000E000                         @ Build address of NVIC registers
    LDR     r1, =__Vectors                           @ Pickup address of vector table
    STR     r1, [r0, #0xD08]                        @ Set vector table address
@
@    /* Set system stack pointer from vector value.  */
@
    LDR     r0, =_tx_thread_system_stack_ptr        @ Build address of system stack pointer
    LDR     r1, =__Vectors                           @ Pickup address of vector table
    LDR     r1, [r1]                                @ Pickup reset stack pointer
    STR     r1, [r0]                                @ Save system stack pointer
```

- Add condition check for SysTick_Handler function, to avoid conflicts while working with Baremetal.

```
#ifdef THREADX_OS
@ /* System Tick timer interrupt handler */
    .global  __tx_SysTickHandler
    .global  SysTick_Handler
    .thumb_func
__tx_SysTickHandler:
    .thumb_func
SysTick_Handler:
@ VOID TimerInterruptHandler (VOID)
@ {
@
    PUSH    {r0, lr}
#ifdef TX_ENABLE_EXECUTION_CHANGE_NOTIFY
    BL      _tx_execution_isr_enter             @ Call the ISR enter function
#endif
    BL      _tx_timer_interrupt
#ifdef TX_ENABLE_EXECUTION_CHANGE_NOTIFY
    BL      _tx_execution_isr_exit              @ Call the ISR exit function
#endif
    POP     {r0, lr}
    BX      LR
@ }
#endif
```

third_party/threadx/common/src/tx_initialize_kernel_enter.c file

- Comment thread scheduling code line in *tx*_initialize_kernel_enter() function.

```
    _tx_thread_system_state =  TX_INITIALIZE_IS_FINISHED;

    /* Call any port specific pre-scheduler processing.  */
    TX_PORT_SPECIFIC_PRE_SCHEDULER_INITIALIZATION

    /* Enter the scheduling loop to start executing threads!  */
//    _tx_thread_schedule();

#ifdef TX_SAFETY_CRITICAL

    /* If we ever get here, raise safety critical exception.  */
    TX_SAFETY_CRITICAL_EXCEPTION(__FILE__, __LINE__, 0);
#endif
```

wlan_throughput/rsi_throughput_app.c file

- Invert the priorities for application task and wireless driver task. Always make sure driver task is of higher priority (Lowest priority number).

```
106 // Memory length for driver
107 #define GLOBAL_BUFF_LEN 15000
108
109 // Wlan task priority
110 #define RSI_APPLICATION_TASK_PRIORITY 2
111
112 // Wireless driver task priority
113 #define RSI_DRIVER_TASK_PRIORITY 1
114
```

  - Follow the guide[3] and make necessary changes to run the application such as SSID, PSK etc.
  - Follow trouble shoot(see page 22), for necessary changes to avoid errors.
- Clean and build the project, if there are no errors, flash and run the project on EFR32.

3 https://docs.silabs.com/rs9116-wiseconnect/latest/wifibt-wc-featured-examples/wlan-throughput-readme#wi-fi-configuration

5 Appendix

- **rsi_os_wrapper_threadx.c file**

This is a sample OS wrapper file, which contains function handlers for the Azure RTOS ThreadX platform.

API	Corresponding ThreadX API	Description
rsi_task_create()	tx_thread_create()	This API is used to create different tasks in OS supported platforms.
rsi_mutex_create()	_tx_mutex_create()	This function is OS Abstraction layer API which creates the mutex.
rsi_mutex_lock()	_tx_mutex_get()	This function is OS Abstraction layer API which takes the mutex.
rsi_mutex_unlock()	_tx_mutex_put()	This function is OS Abstraction layer API which gives the mutex.
rsi_mutex_destroy()	_tx_mutex_delete()	This function is OS Abstraction layer API which destroy/ delete the mutex.
rsi_semaphore_create()	_tx_semaphore_create	This function is OS Abstraction layer API which creates the semaphore.
rsi_semaphore_destroy()	_tx_semaphore_delete()	This function is OS Abstraction layer API which destroys the semaphore.
rsi_selamphore_wait()	Tx_semaphore_wait()	This API is used by Wireless Library to acquire or wait for semaphore.
rsi_semaphore_post()	_tx_semaphore_put()	This API is used by wireless library to release semaphore, which was acquired.
rsi_semaphore_reset()	_tx_semaphore_delete()	This API is used by wireless library Wireless Library to the semaphore to initial state.
rsi_task_destroy()	_tx_thread_delete()	This API is used to delete/destroy the task created.
rsi_start_os_scheduler()	_tx_thread_schedule()	This API schedules the tasks created.
rsi_free()	_tx_byte_release()	This API is used to free the memory bytes allocated.

API	Corresponding ThreadX API	Description
rsi_task_suspend()	tx_thread_suspend()	This API suspends the task.
rsi_malloc()	tx_byte_allocate()	This API is used to assign memory required by a task while its creation.
	tx_application_define()	This API is used by the wireless library to define the initial system, memory initialization.

6 Trouble shoot

- If the application seems to hang while execution, increase the RSI_APPLICATION_STACK_TASK_SIZE

7 References

- To learn more about Azure RTOS ThreadX and its implementation, refer link[4].

4 https://docs.microsoft.com/en-us/azure/rtos/threadx/about-this-guide

8 Scope for improvement

- Memory optimization, to use less stack for application tasks.