



STM32 AZURE RTOS workshop

ThreadX

- The purpose of this session is to give you a first overview of how we support AZURE-RTOS ThreadX in the STM32Cube ecosystem

Agenda

- Prepare project for CubeMX and CubelIDE
- Add ThreadX in CubeMX
- Add ThreadX into CubelIDE project
- Use TraceX
- Use multiple Threads with preemption priorities
- CMSIS_OS2 optional layer*
- ThreadX features overview

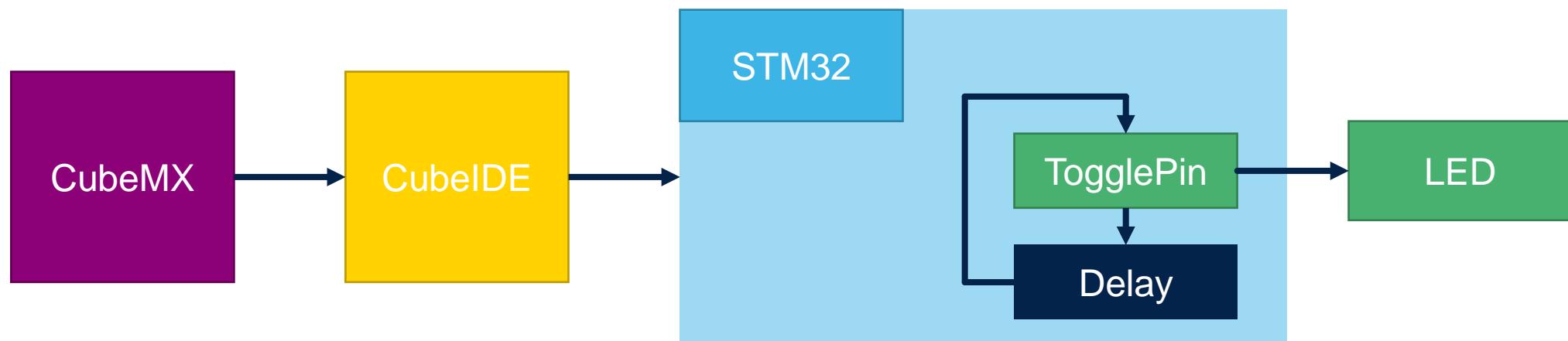
Links

Useful links

- ThreadX user guide <https://docs.microsoft.com/en-us/azure/rtos/threadx/about-this-guide>
- ThreadX api description <https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter4>
- ThreadX cheatsheet https://rristm.github.io/stm32_threadx
- Github: <https://github.com/STMicroelectronics/x-cube-azrtos-h7>

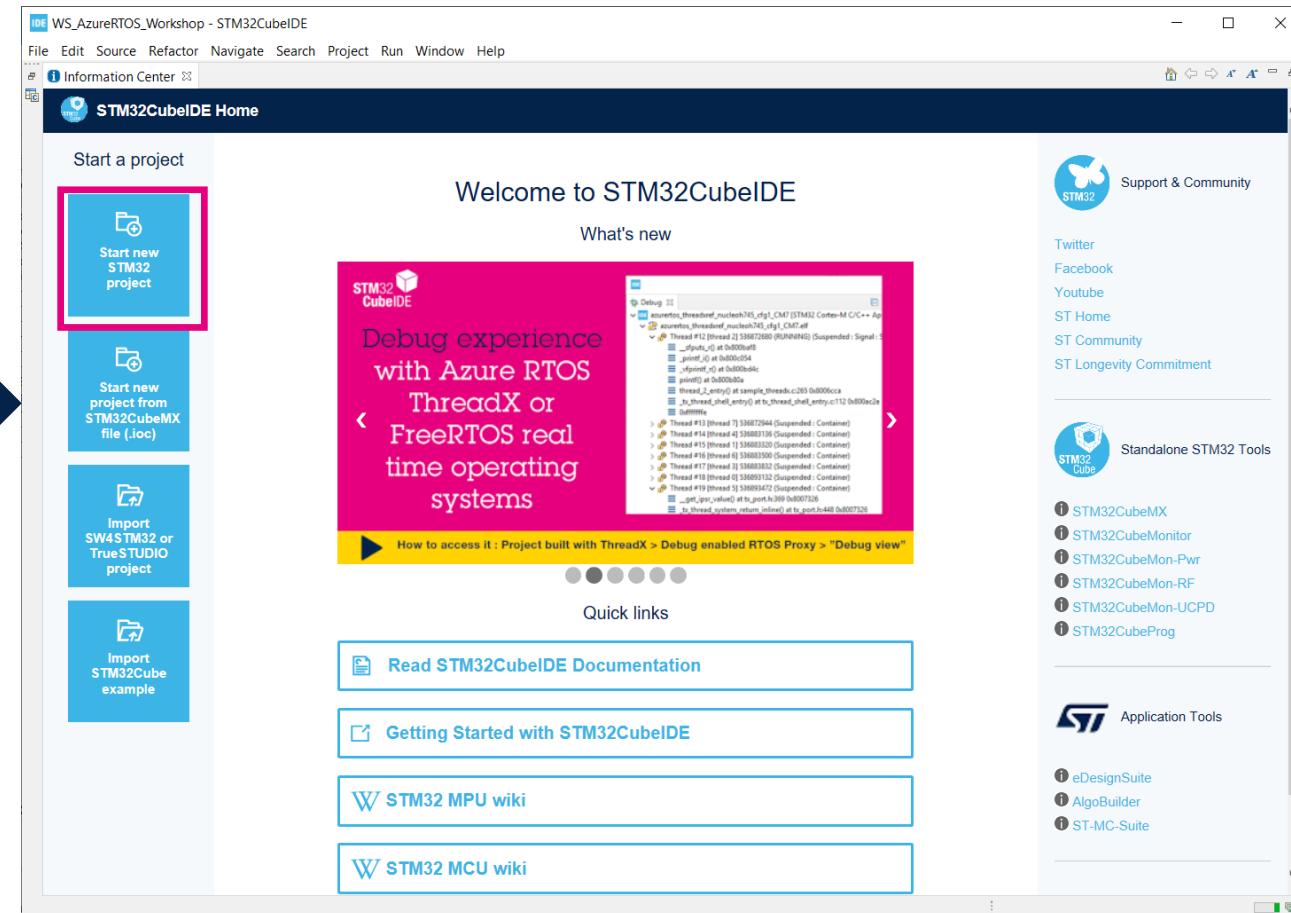
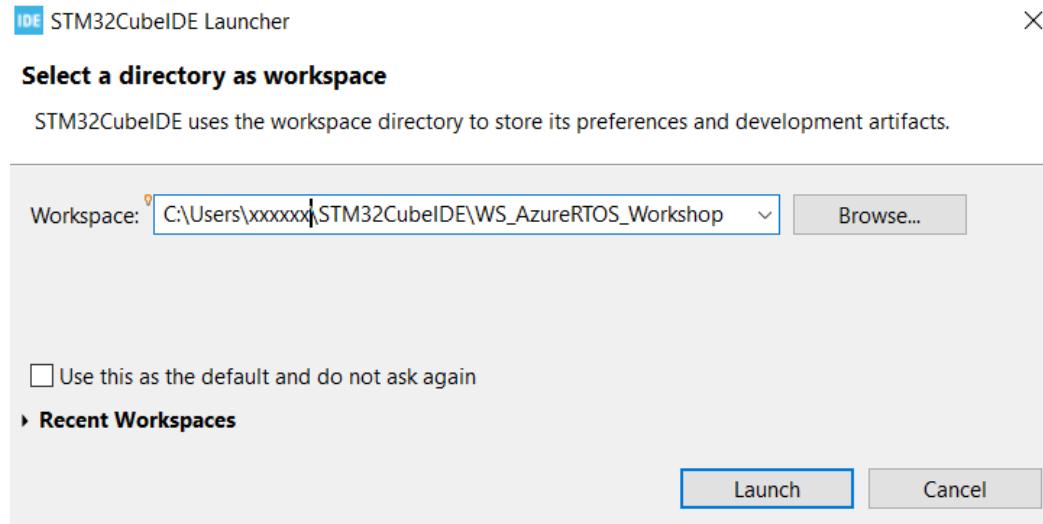
Step 1 : Start led blink from scratch

- This short hands-on will show you how to start a project from scratch
- The point is to get first running led blink in few clicks
- We will then use this base to go further with ThreadX (no RTOS yet)



Launch STM32CubeIDE

- Launch CubeIDE
- Select a workspace, click on Launch



Start a new project

- Click on board selector

IDE STM32 Project

Target Selection

⚠ STM32 target or STM32Cube example selection is required

MCU/MPU Selector **Board Selector** Example Selector Cross Selector

MCU/MPU Filters

Part Number

Core >

Series >

Line >

Package >

Other >

Peripheral >

Features Block Diagram Docs & Resources Datasheet Buy

STM32U5 psacertified™ SESIP level three First STM32 MCU to receive SESIP and PSA level 3 certification

MCUs/MPUs List: 1903 items

+ Display similar items Export

*	Part ...	Reference	Market...	Unit Pr...	Board	Package	Flash	RAM	IO	Freq.
★	STM32F...	STM32F...	Active	0.597		LQFP48	32 kBytes	4 kBytes	39	48 MHz
★	STM32F...	STM32F...	Active	0.722		LQFP48	64 kBytes	8 kBytes	39	48 MHz
★	STM32F...	STM32F...	Active	1.1		LQFP48	256 kByt...	32 kBytes	37	48 MHz
★	STM32F...	STM32F...	Active	0.424		TSSOP20	16 kBytes	4 kBytes	15	48 MHz
★	STM32F...	STM32F...	Active	0.518		LQFP32	32 kBytes	4 kBytes	25	48 MHz
★	STM32F...	STM32F...	Active	0.754	NU... ST...	LQFP64	64 kBytes	8 kBytes	55	48 MHz
★	STM32F...	STM32F...	Active	1.21		LQFP64	256 kByt...	32 kBytes	51	48 MHz
★	STM32F...	STM32F...	Active	0.97		LQFP48	16 kBytes	4 kBytes	39	48 MHz

?

< Back Next > Finish Cancel

Look for Nucleo H723

- Type H723 in the search box
- Then click on Nucleo-H723ZG

The screenshot shows the STM32 Project IDE interface. In the top-left, there is a search bar with the text "H723". Below it, the "Board Selector" tab is active. On the left, there are "Board Filters" with a dropdown menu set to "H723" and "NUCLEO-H723ZG" selected. The main area displays the "NUCLEO-H723ZG" board details, including a large image of the STM32U5 chip, PSA certification logos, and a text box stating "First STM32 MCU to receive SESIP and PSA level 3 certification". At the bottom, there is a "Boards List" table with one item, showing columns for Overview, Commercial..., Type, Marketing S..., Unit Price (...), and Mounted D... . The entire "Boards List" table is highlighted with a pink box.

*	Overview	Commercial...	Type	Marketing S...	Unit Price (...)	Mounted D...
		NUCLEO-H7...	Nucleo-144	Active	29.0	STM32H723ZG...

Check proposed board

- Click next

The screenshot shows the STM32 Project IDE interface. The title bar says "STM32 Project". The main window is titled "Target Selection" with the sub-instruction "Select STM32 target or STM32Cube example". Below this, there are four tabs: "MCU/MPU Selector" (selected), "Board Selector", "Example Selector", and "Cross Selector".

On the left, there is a "Board Filters" sidebar with the following options:

- Commercial Part Number: H723 (selected)
- Vendor
- Type
- MCU/MPU Series
- Other
- Peripheral

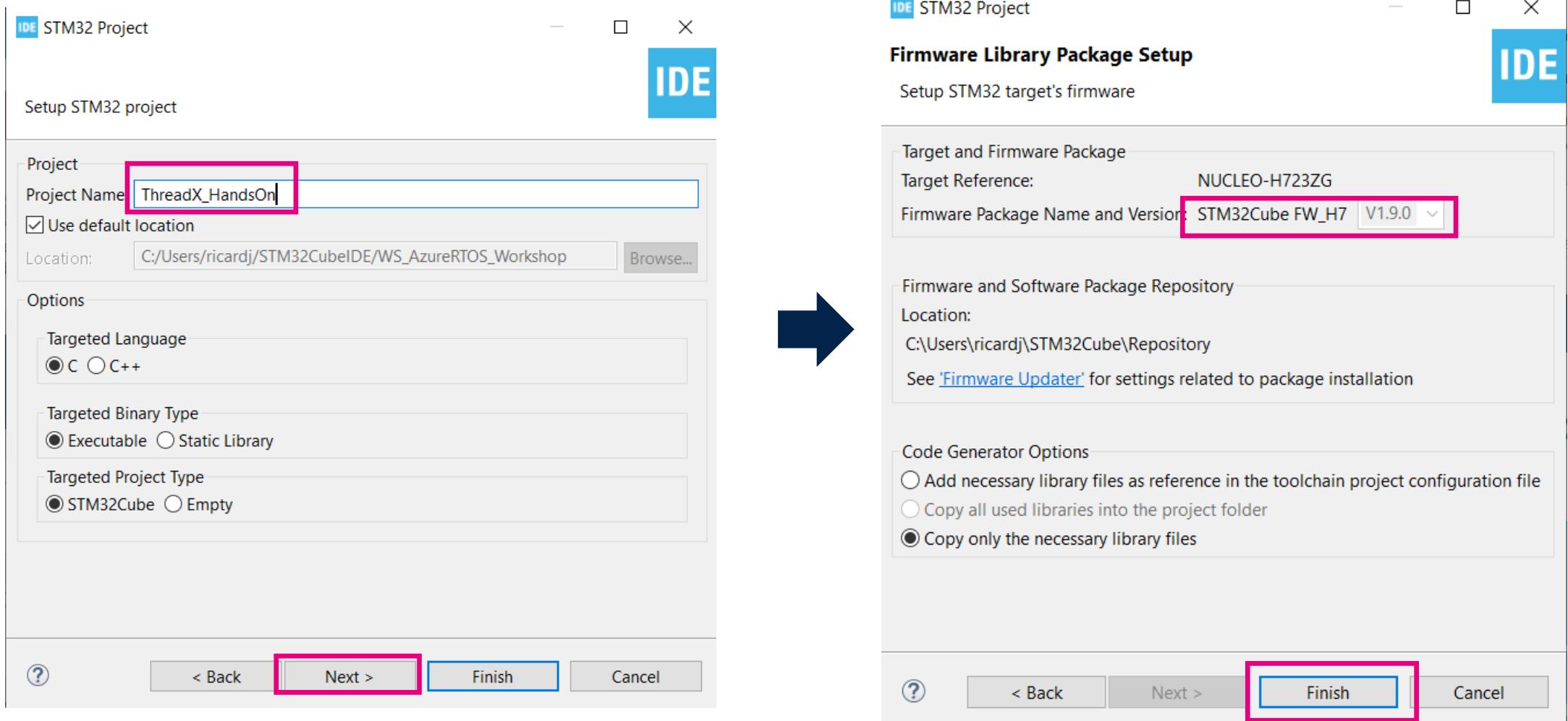
The main content area displays the "STM32H7 Series" and the "NUCLEO-H723ZG" board. It includes:

- Features:** ACTIVE Active, Product is in mass production.
- Datasheet:** Unit Price (US\$) : 29.0, Mounted Device : STM32H723ZGTx
- Description:** The STM32 Nucleo-144 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller. For the compatible boards, the internal or external SMPS significantly reduces power.
- Boards List:** 1 item (NUCLEO-H723ZG - Nucleo-144)

At the bottom, there are navigation buttons: "?", "< Back", "Next >" (highlighted with a red box), "Finish", and "Cancel".

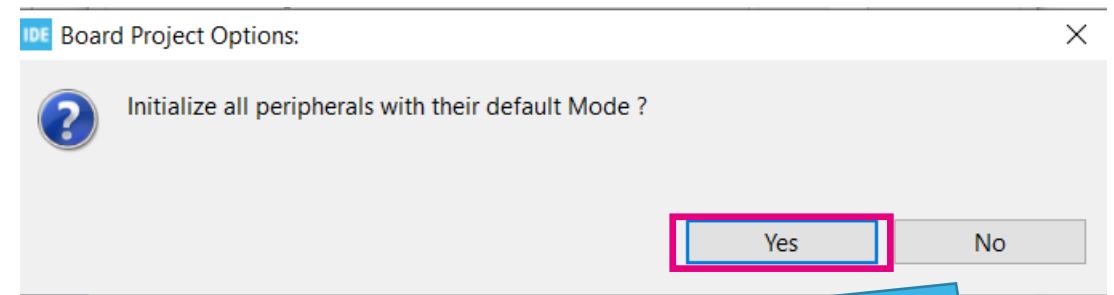
Setup the project name

- Type the project name then next and check FW version



Finish the project setup

- In pop-up asking of peripherals initialization
- Click **Yes**
- Then second popup to ask to open device configuration perspective.
- Click **Yes**



This will setup all GPIOs to correct mode



This will create your project structure with all needed files and open the STM32CubeMX embedded in STM32CubeIDE

What you get after few clicks

In project explorer you get all the project structure

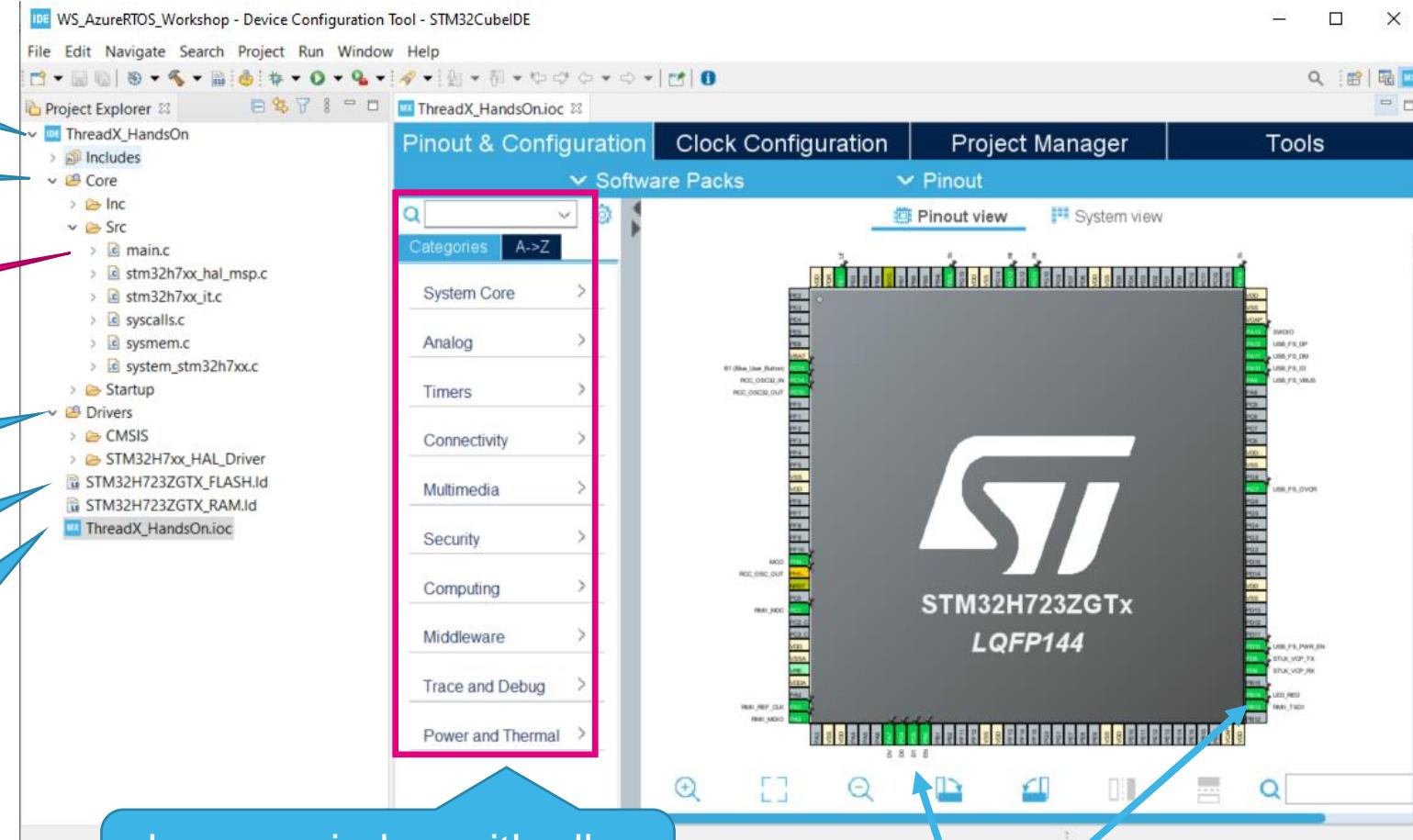
Core contains the specific project files

Our main.c

Drivers contains HAL and CMSIS

STM32H723GTX_FLASH.Id is the linker file. Handling placement in STM32 memory

.ioc file is the one opened here in the embedded STM32CubeMX



shows a window with all configurable peripherals

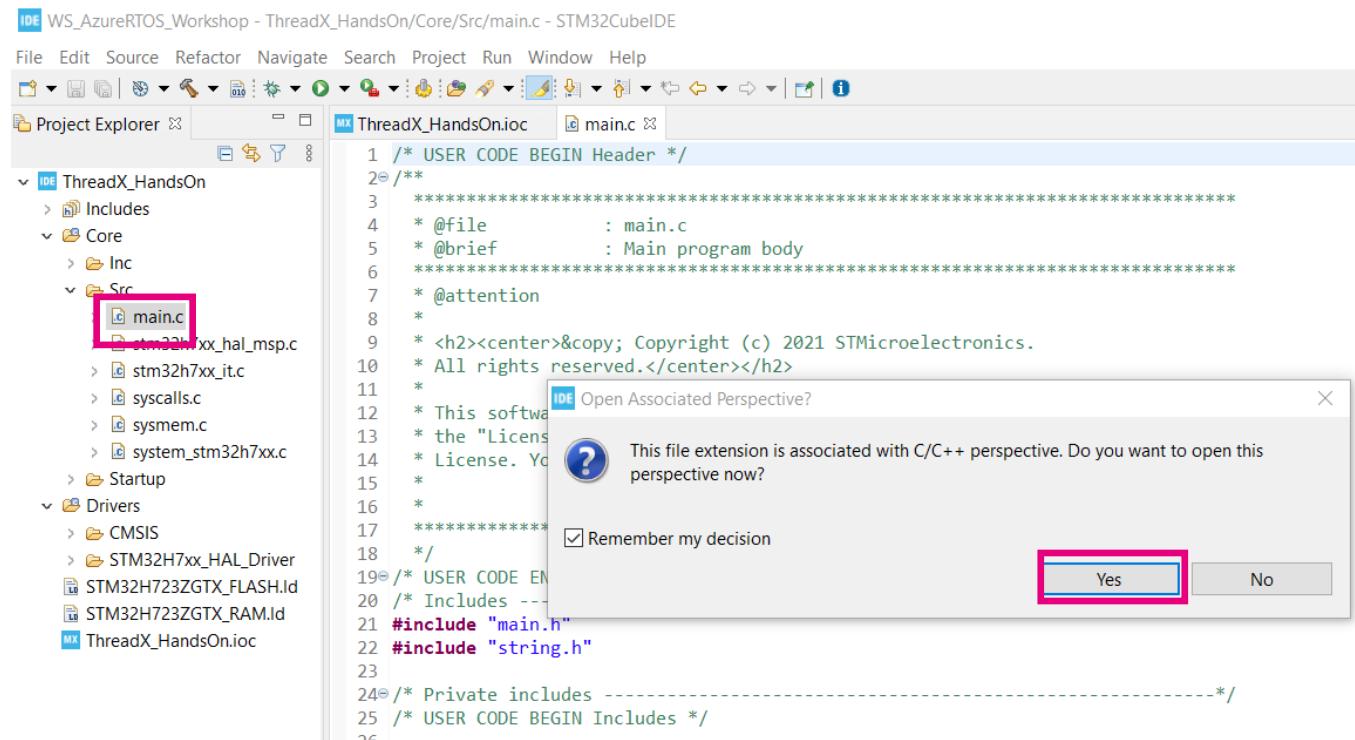
All GPIOs already setup according to this specific board

Edit main.c to add your code

- Look for main.c and double click
- Answer yes to the open perspective
- Then go down to the main() function and find the while(1) loop line 129

```
/* USER CODE BEGIN WHILE */
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
}
```



Adding LED toggling using code completion



Core\Src\main.c

- Just before comment `/* USER CODE END WHILE */` type:
- `HAL_GPIO_T` and the CTRL-Space bar you should get:

A screenshot of a code editor showing a portion of the `main.c` file. The code is as follows:

```
127  /* Infinite loop */
128  /* USER CODE BEGIN WHILE */
129  while (1)
130  {
131      HAL_GPIO_TogglePin(GPIOx, GPIO_Pin)
132  /* USER CODE END WHILE */
133
134  /* USER CODE BEGIN 3 */
```

The cursor is at line 131, and the code `HAL_GPIO_TogglePin(GPIOx, GPIO_Pin)` is highlighted. A tooltip or code completion dropdown is visible, showing the function signature `void HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)`.

- First argument `LED_GREEN_GPIO_Port` (type `LED` CTRL+Space)
- Second argument `LED_GREEN_Pin`
- Add the final `'.'`

A screenshot of a code editor showing the same portion of the `main.c` file. The code now includes the first argument:

```
127  /* Infinite loop */
128  /* USER CODE BEGIN WHILE */
129  while (1)
130  {
131      HAL_GPIO_TogglePin(LED_G, GPIO_Pin)
132  /* USER CODE END WHILE */
133
134  /* USER CODE BEGIN 3 */
```

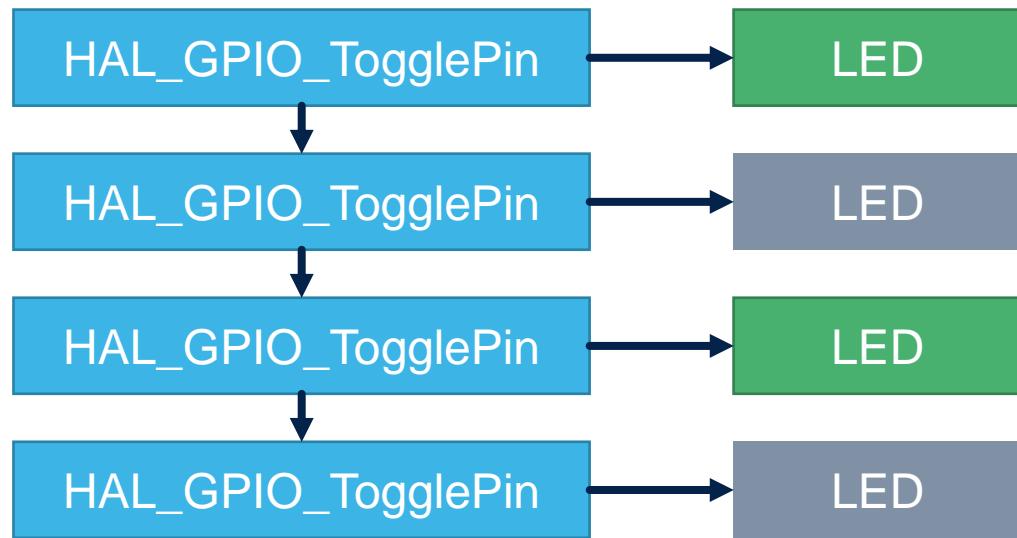
A tooltip or code completion dropdown is visible, showing the available options: `# LED_GREEN_GPIO_Port`, `# LED_GREEN_Pin`, `# LED_RED_GPIO_Port`, and `# LED_YELLOW_GPIO_Port`.

`HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);`

HAL GPIO Toggle

- Each time HAL_GPIO_TogglePin is called
 - LED will turn on/off
- Without delay impossible to see in run
 - Only in debug stepping

Toggling too fast, no delay



Adding the delay between each LED toggle

- Starting from

```

126  /* Infinite loop */
127  /* USER CODE BEGIN WHILE */
128  while (1)
129  {
130      HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
131
132      /* USER CODE END WHILE */
133
134

```

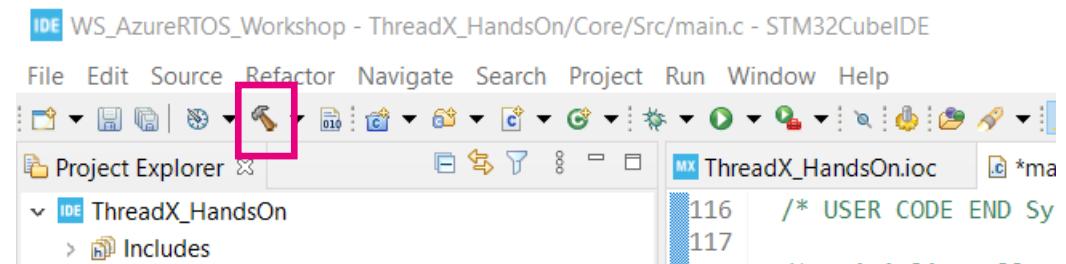
- Type HAL_Delay
- Choose a delay in ms : 1000
- Add final semicolon
- Now compile the project using the hammer

```

128  /* USER CODE BEGIN WHILE */
129  while (1)
130  {
131      HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
132      HAL_Delay(1000);
133      /* USER CODE END WHILE */
134
135      /* USER CODE BEGIN 3 */
136  }

```

HAL_Delay(1000);



Compilation result

No errors

IPE ThreadX_base_repo - ThreadX_01/Core/Src/main.c - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer ThreadX_01 Binaries Includes Core Src main.c stm32h7xx_hal_msp.c stm32h7xx_it.c syscalls.c sysmem.c system_stm32h7xx.c Startup Drivers Debug STM32H723ZGTX_FLASH.Id STM32H723ZGTX_RAM.Id ThreadX_01.ioc

```
112 SystemClock_Config();  
113 /* USER CODE BEGIN SysInit */  
114  
115 /* USER CODE END SysInit */  
116  
117 /* Initialize all configured peripherals */  
118 MX_GPIO_Init();  
119 MX_ETH_Init();  
120 MX_USART3_UART_Init();  
121 MX_USB_OTG_HS_USB_Init();  
122 /* USER CODE BEGIN 2 */  
123  
124 /* USER CODE END 2 */  
125  
126 /* Infinite loop */  
127 /* USER CODE BEGIN WHILE */  
128 while (1)  
129 {  
130     HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);  
131     HAL_Delay(1000);  
132     /* USER CODE END WHILE */  
133  
134     /* USER CODE BEGIN 3 */  
135 }  
136 /* USER CODE END 3 */  
137  
138 }  
139  
140 /** @brief System Clock Configuration  
141 * @retval None  
142 */
```

CDT Build Console [ThreadX_01]

```
arm-none-eabi-gcc -mcpu=cortex-m7 -g3 -c -x assembler-with-cpp -MMD -MP "Core/Startup/startup_stm32h723zgtx.d" -MT "Core/Startup/startup_stm3  
arm-none-eabi-gcc ".Core/Src/main.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..../Drivers/STM3  
arm-none-eabi-gcc ".Core/Src/stm32h7xx_hal_msp.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..  
arm-none-eabi-gcc ".Core/Src/stm32h7xx_it.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..../Driv  
arm-none-eabi-gcc ".Core/Src/syscalls.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..../Drivers/  
arm-none-eabi-gcc ".Core/Src/sysmem.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..../Drivers/STI  
arm-none-eabi-gcc ".Core/Src/system_stm32h7xx.c" -mcpu=cortex-m7 -std=gnu11 -g3 -DDEBUG -DUSE_HAL_DRIVER -DSTM32H723xx -c -I..../Core/Inc -I..  
arm-none-eabi-gcc -o "ThreadX_01.elf" "@objects.list" -mcpu=cortex-m7 -T"\\Radek\DD_Projects\2021_01_21_ThreadX\Materials\ThreadX_base_repo\T  
Finished building target: ThreadX_01.elf
```

arm-none-eabi-size ThreadX_01.elf
arm-none-eabi-objdump -h -S ThreadX_01.elf > "ThreadX_01.list"
arm-none-eabi-objcopy -O binary ThreadX_01.elf "ThreadX_01.bin"
text data bss dec hex filename
21936 216 1984 24056 5df8 ThreadX_01.elf

Finished building: default.size.stdout

Finished building: ThreadX_01.list

Finished building: ThreadX_01.bin

09:39:46 Build Finished. 0 errors, 0 warnings. (took 15s.745ms)

Writable Smart Insert 134:1:4313

Outline Build Targets

- main.h
- string.h
- DMARxDscrTab : ETH_DMADescTypeDef
- DMATxDscrTab : ETH_DMADescTypeDef
- Rx_Buff : uint8_t[1]
- DMARxDscrTab : ETH_DMADescTypeDef
- DMATxDscrTab : ETH_DMADescTypeDef
- Rx_Buff : uint8_t[1]
- DMARxDscrTab : ETH_DMADescTypeDef
- DMATxDscrTab : ETH_DMADescTypeDef
- Rx_Buff : uint8_t[1]
- TxConfig : ETH_TxPacketConfig
- heth : ETH_HandleTypeDef
- huart3 : UART_HandleTypeDef
- SystemClock_Config(void) : void
- MX_GPIO_Init(void) : void
- MX_ETH_Init(void) : void
- MX_USART3_UART_Init(void) : void
- MX_USB_OTG_HS_USB_Init(void) : void
- main(void) : int
- SystemClock_Config(void) : void
- MX_ETH_Init(void) : void
- MX_USART3_UART_Init(void) : void
- MX_USB_OTG_HS_USB_Init(void) : void
- MX_GPIO_Init(void) : void
- Error_Handler(void) : void

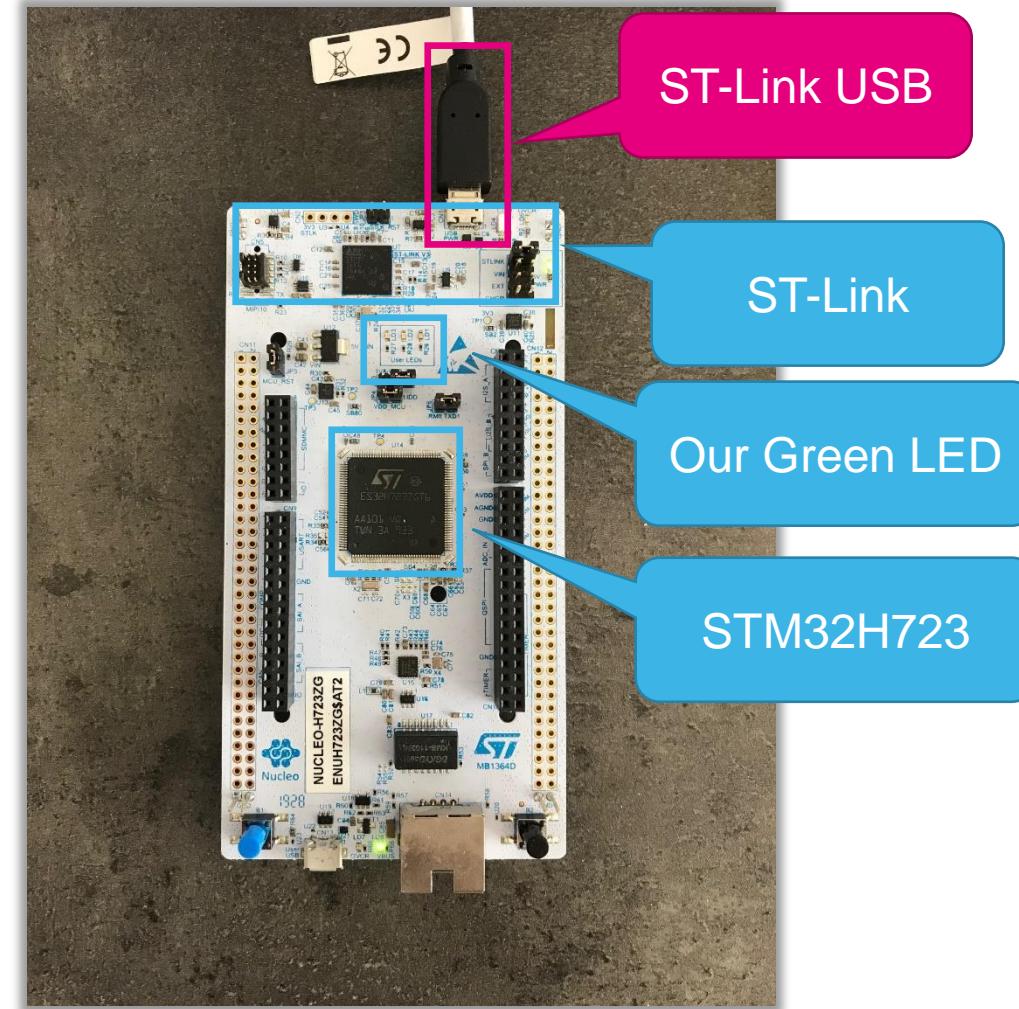
Build Analyzer Static Stack Analyzer

ThreadX_01.Id - /ThreadX_01/Debug - Jun 14, 2021, 9:39:45 AM

Memory Regions	Memory Details
ITCMRAM	Start address 0x00000000 End address 0x00010000 Size 64 KB
RAM	Start address 0x20000000 End address 0x20020000 Size 128 KB
FLASH	Start address 0x08000000 End address 0x08100000 Size 1024 KB
RAM_D1	Start address 0x24000000 End address 0x24050000 Size 320 KB
RAM_D2	Start address 0x30000000 End address 0x30008000 Size 32 KB
RAM_D3	Start address 0x38000000 End address 0x38004000 Size 16 KB

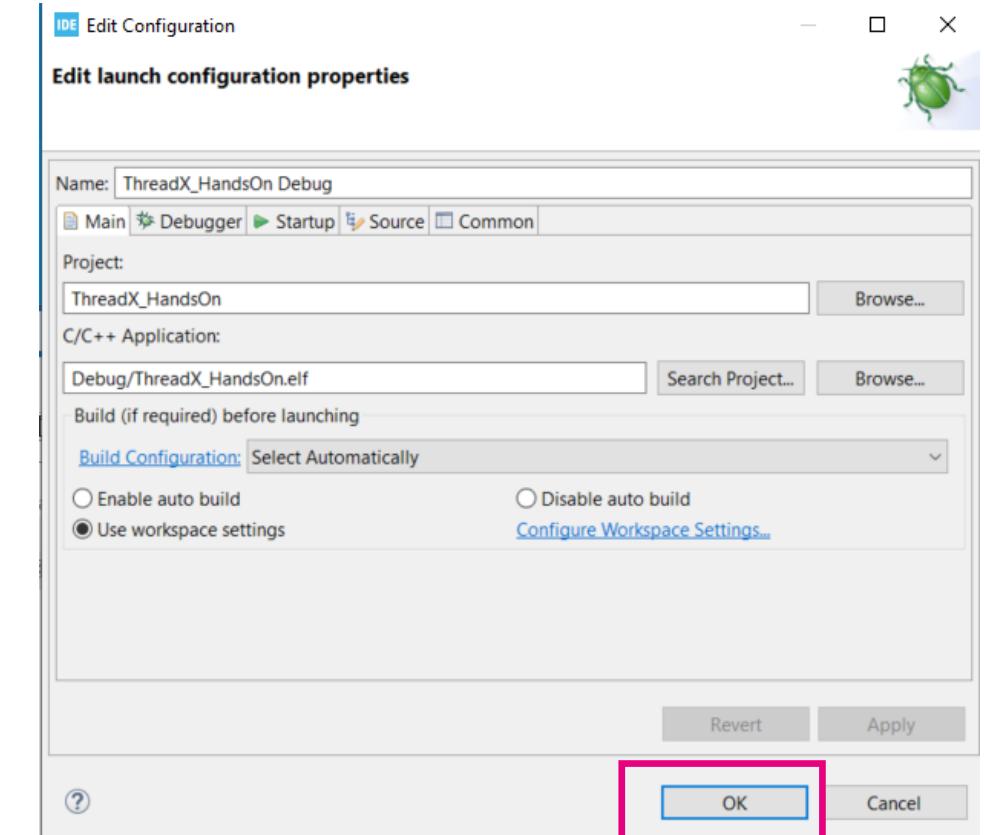
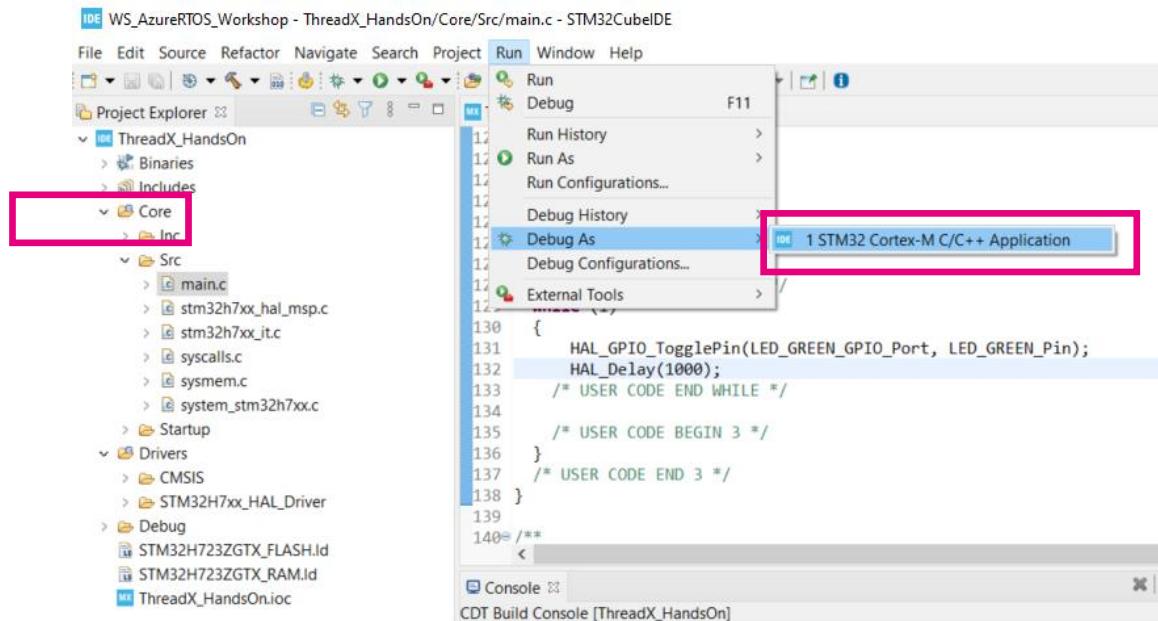
Connect the board to the computer

- Time to transfer firmware to the board
- First connect the Nucleo board to the computer



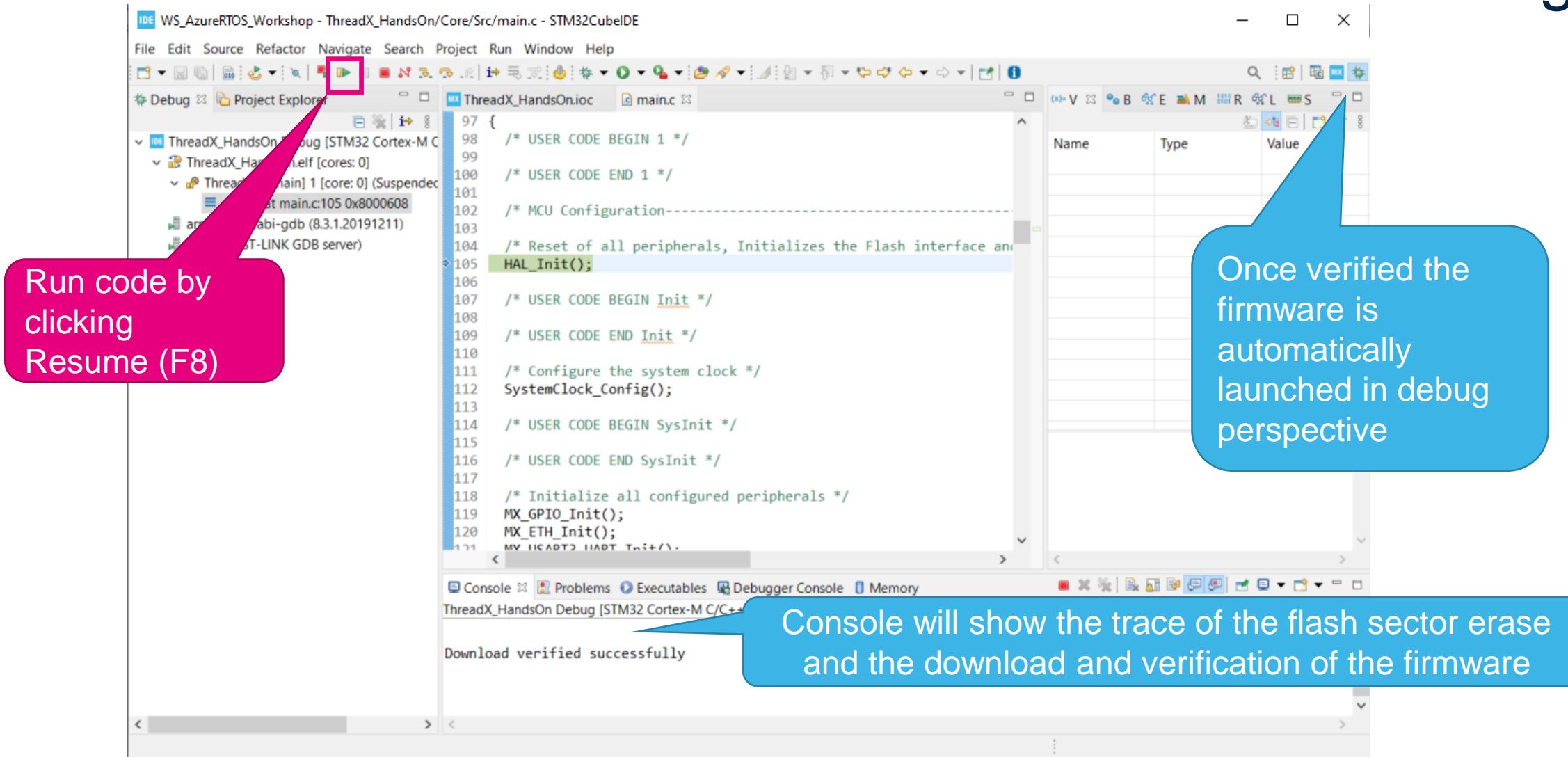
Download firmware and Debug on target

- First click on the project name
- Then menu Run/Debug As/STM32 ...



- In Edit Configuration window click OK

LED is Blinking

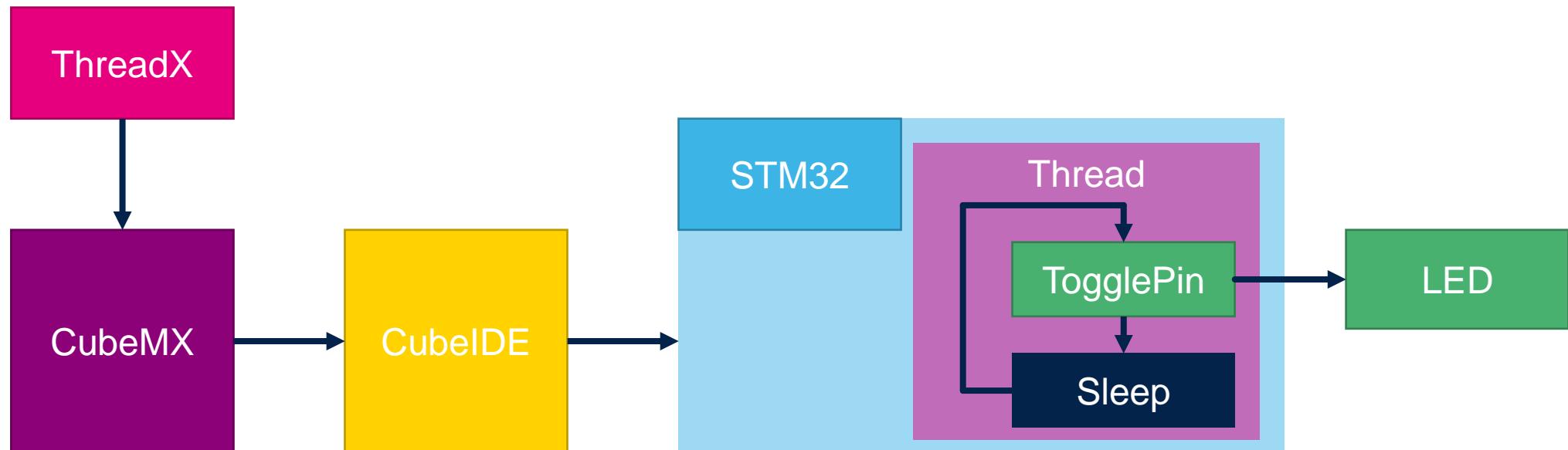


- To exit debugger click on or CTRL-F2

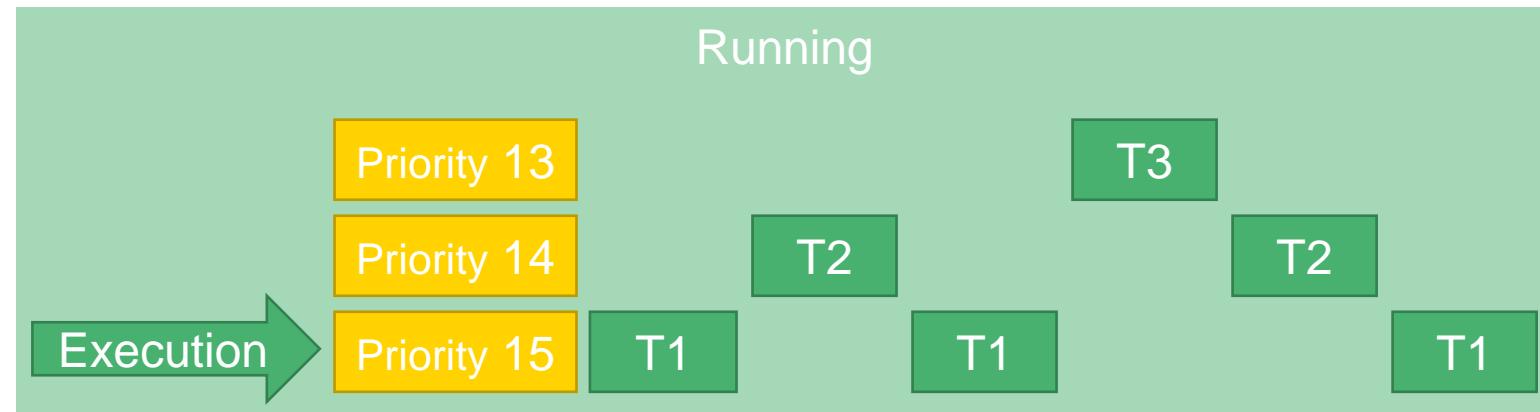
Step 2 : Create first thread

Step 2 : Create first thread

- First step is to enable ThreadX in STM32CubeMX
- We want now to blink the LED inside a ThreadX thread
- This will update the project structure to support ThreadX
- Then we will be able to simply create a thread



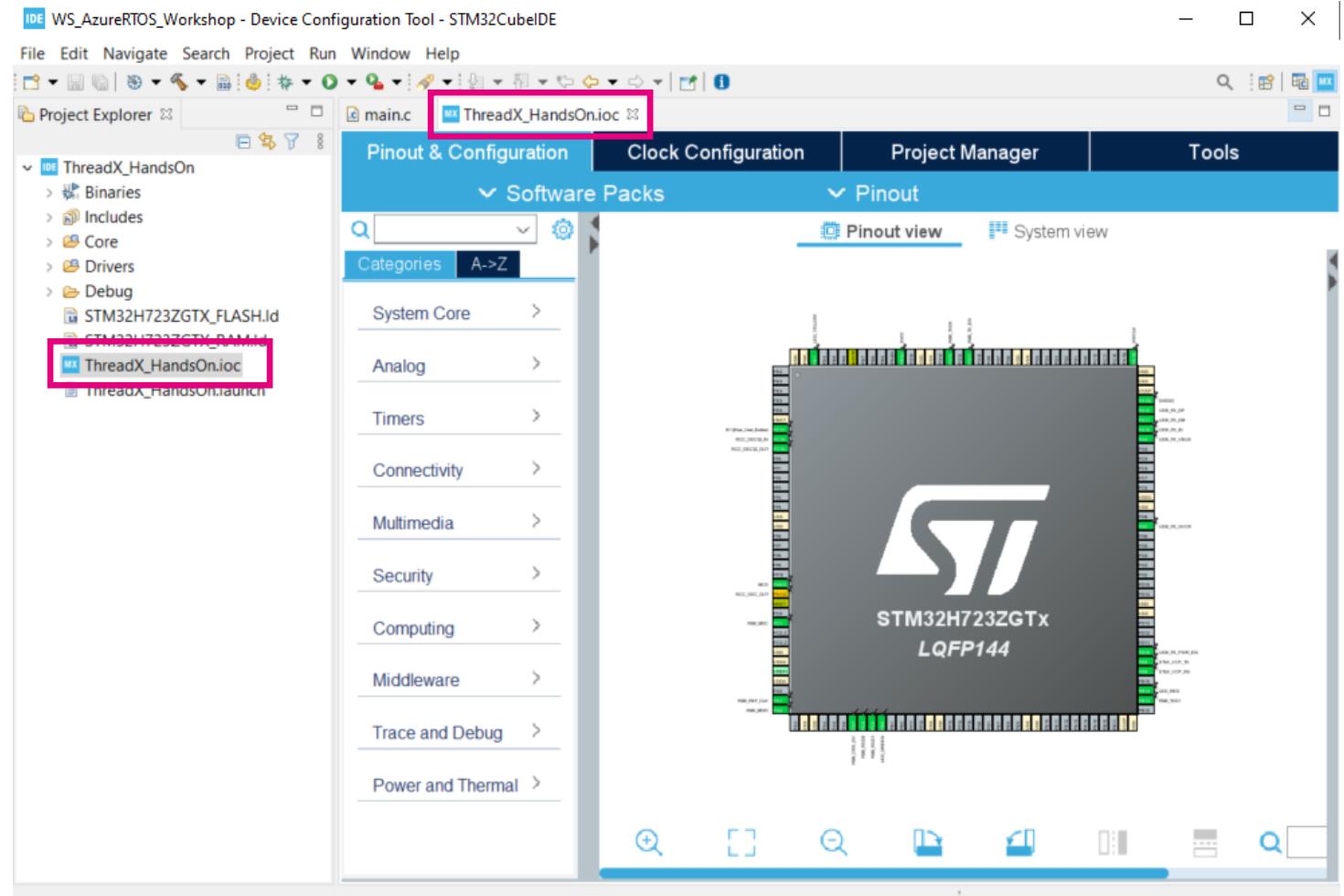
- RTOS runs in threads
 - At least one thread for user program needed (If no RTOS runs in idle)
 - Preemption priority (higher priority thread run, like IRQ)



- <https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#thread-execution-1>
- https://rristm.github.io/stm32_threadx/show/threadx_thread.md

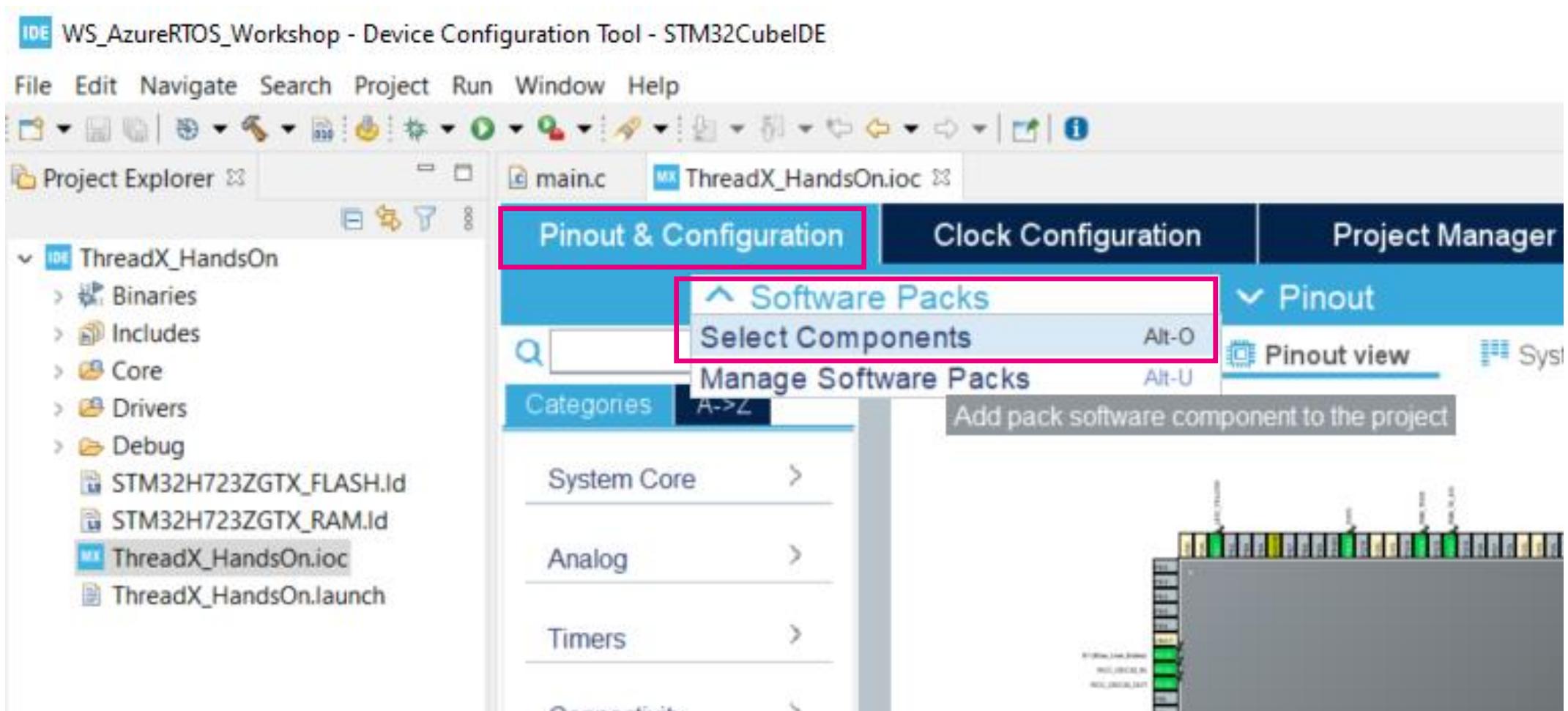
Switch to embedded CubeMX

- Terminate debug from first hands-on (CTRL-F2)
- Double click or right click/open on the file: ThreadX_HandsOn.ioc

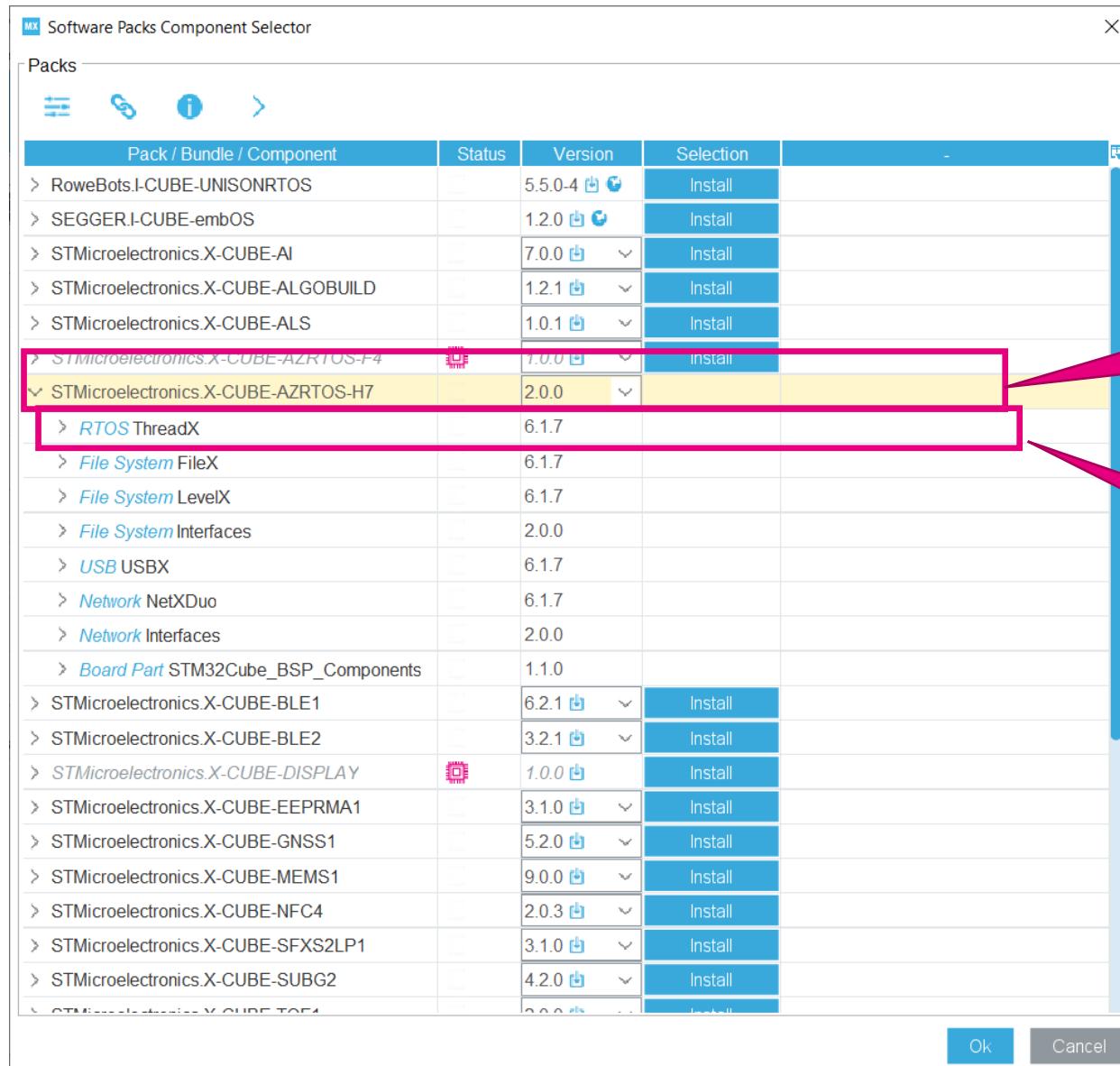


Select a software pack component

- Click on Software Pack and Select Components:



Component selection window



Open the X-CUBE-AZRTOS-H7

Open RTOS_ThreadX

Component selection

Software Packs Component Selector

Packs

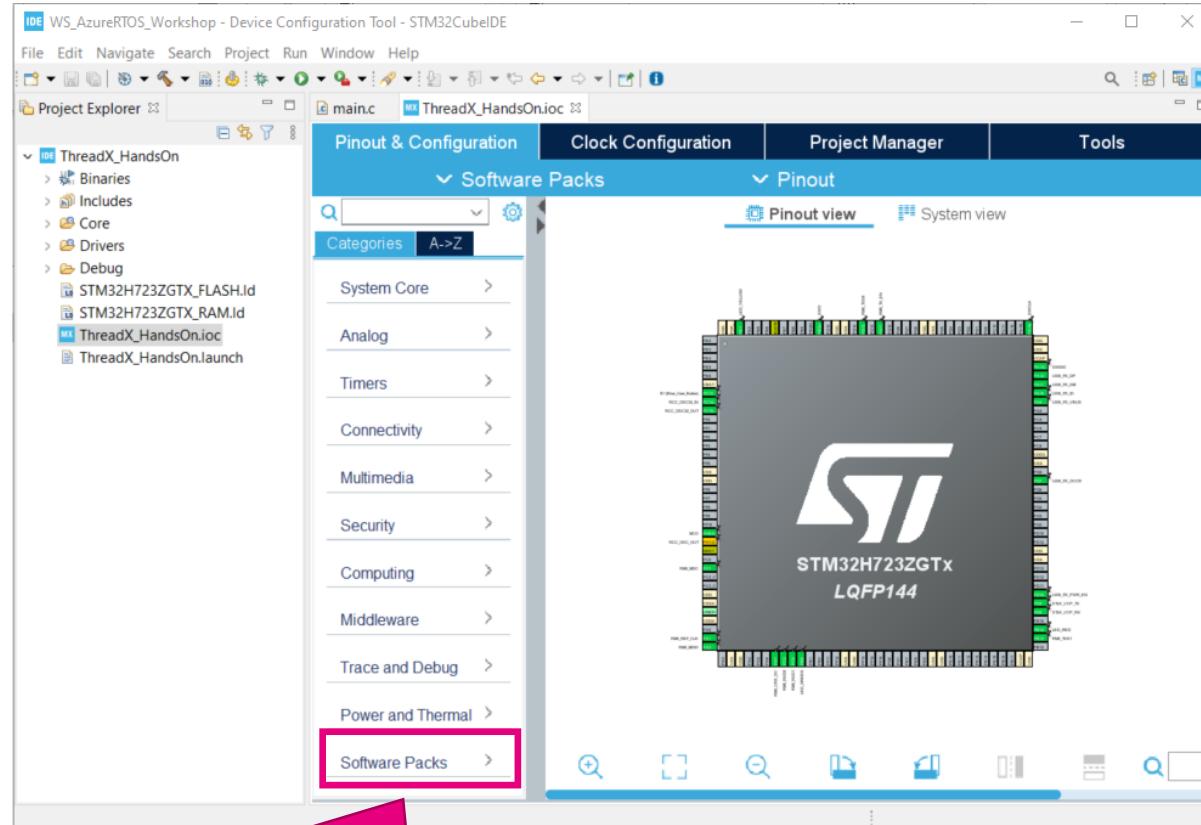
Pack / Bundle / Component	Status	Version	Selection
> RoweBots.I-CUBE-UNISONRTOS		5.5.0-4	Install
> SEGGERI-CUBE-embOS		1.2.0	Install
> STMicroelectronics.X-CUBE-AI		7.0.0	Install
> STMicroelectronics.X-CUBE-ALGOBUILD		1.2.1	Install
> STMicroelectronics.X-CUBE-ALS		1.0.1	Install
> STMicroelectronics.X-CUBE-AZRTOS-F4		1.0.0	Install
STMicroelectronics.X-CUBE-AZRTOS-H7	✓	2.0.0	
RTOS ThreadX	✓	6.1.7	
ThreadX / Core	✓		<input checked="" type="checkbox"/>
ThreadX / PerformanceInfo			<input type="checkbox"/>
ThreadX / TraceX support			<input type="checkbox"/>
ThreadX / Low Power support			<input type="checkbox"/>
> File System FileX		6.1.7	
> File System LevelX		6.1.7	
> File System Interfaces		2.0.0	
> USB USBX		6.1.7	
> Network NetXDuo		6.1.7	
> Network Interfaces		2.0.0	
> Board Part STM32Cube_BSP_Components		1.1.0	
> STMicroelectronics.X-CUBE-BLE1		6.2.1	Install
> STMicroelectronics.X-CUBE-BLE2		3.2.1	Install
> STMicroelectronics.X-CUBE-DISPLAY		1.0.0	Install
> STMicroelectronics.X-CUBE-EEPROMA1		3.1.0	Install
> STMicroelectronics.X-CUBE-GNSS1		5.2.0	Install
> STMicroelectronics.X-CUBE-MEMORY		0.0.0	Install

Ok Cancel

Select:
Application - azure_rtos_app

Confirm selection

Software pack included in the interface



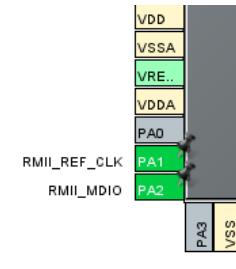
New category added, open it

Power and Thermal

Software Packs

STMicroelectronics.X-CUBE-AZRTOS-H7.2.0.0

Select X-CUBE-AZRTOS H7.2.0.0



Component selection in software pack

Select:
RTOS ThreadX

STMicroelectronics.X-CUBE-AZRTOS-H7.2.0.0 Mode

RTOS ThreadX

System Core

Analog

Timers

Connectivity

Multimedia

Security

Computing

Middleware

Trace and Debug

Power and Thermal

Software Packs

STMicroelectronics.X-CUBE-AZRTOS-H7.2.0.0

RTOS ThreadX

Configuration

Software Packs

Pinout

STMicroelectronics.X-CUBE-AZRTOS-H7.2.0.0 Mode and Configuration

RTOS ThreadX

Configuration

Reset Configuration

AzureRTOS Application ThreadX User Constants

Configure the below parameters :

Search (Ctrl+F)

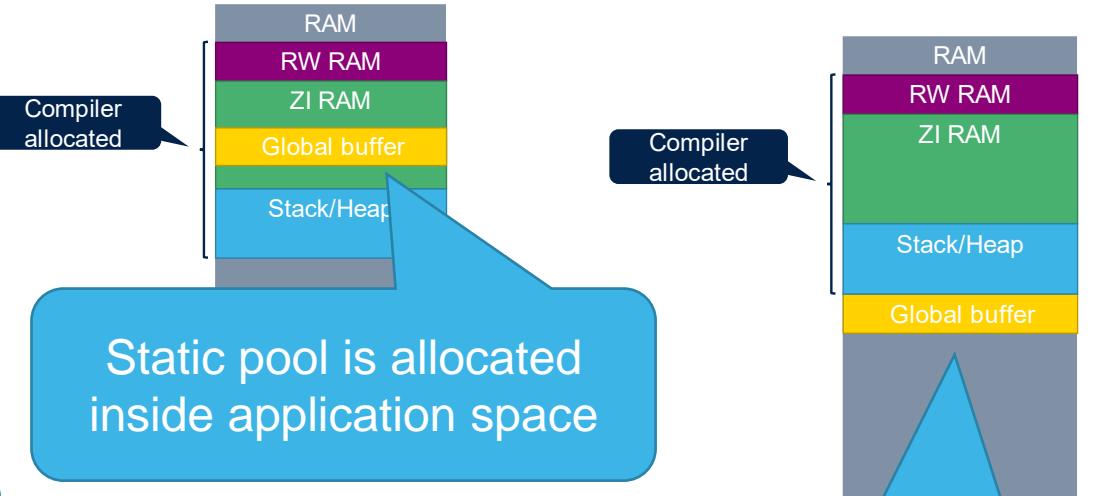
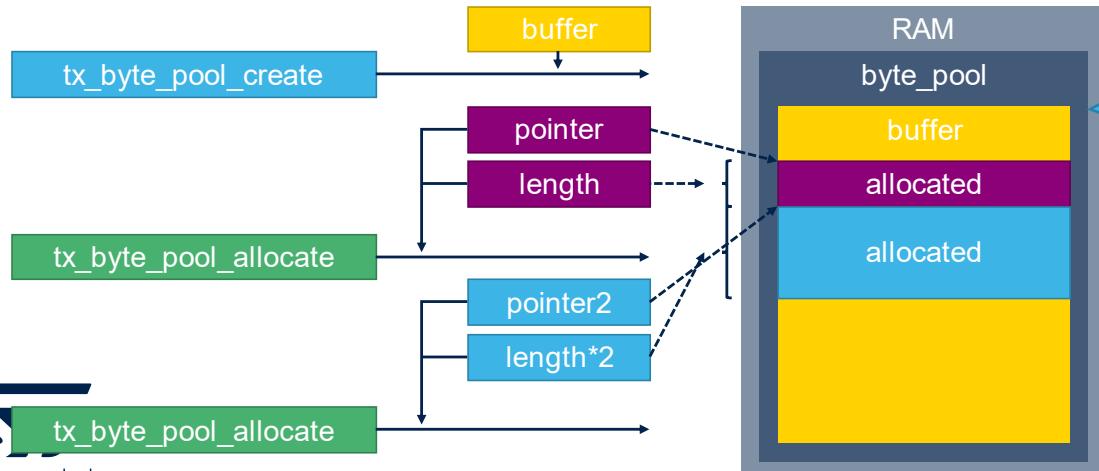
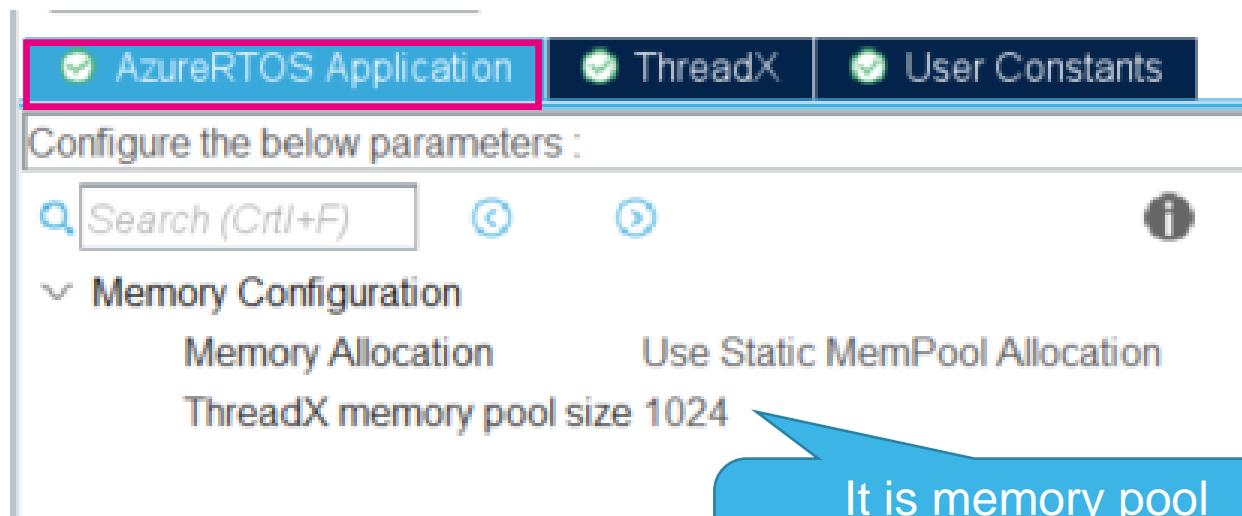
Version

ThreadX version 6.1.7

Core

Parameter	Value
TX_MINIMUM_STACK	200
TX_THREAD_USER_EXT...	
TX_DISABLE_STACK_FL...	Disabled
TX_ENABLE_STACK_C...	Disabled
TX_DISABLE_PREEMPTI...	Enabled
TX_DISABLE_REDUNDA...	Disabled
TX_DISABLE_NOTIFY_C...	Enabled

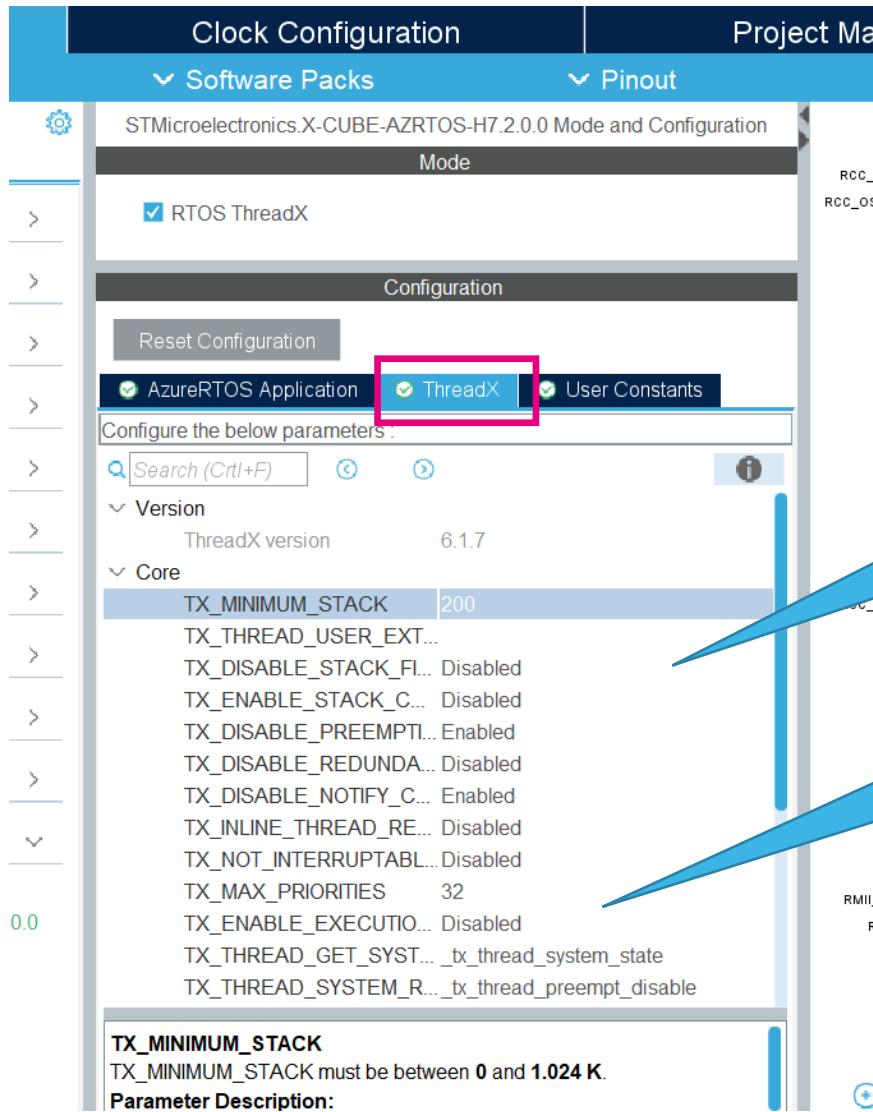
Azure RTOS application configuration TAB



Application can obtain
memory by calling
`tx_byte_pool_allocate`

Using this mempool is optional

ThreadX configuration

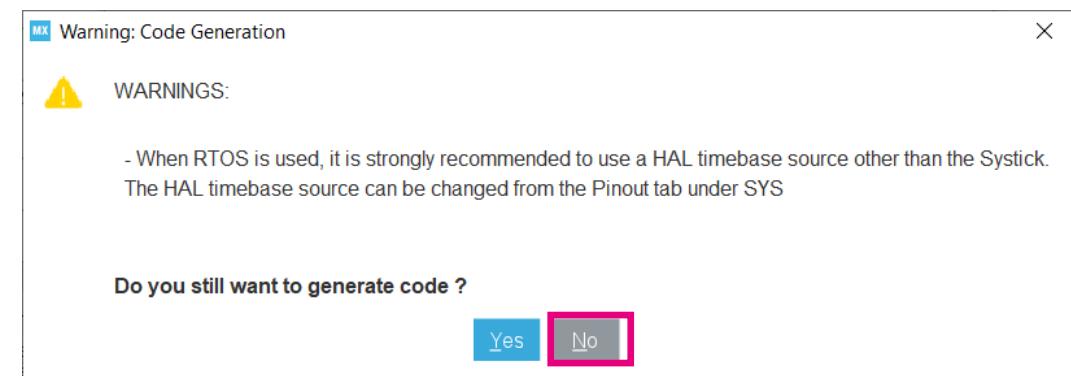
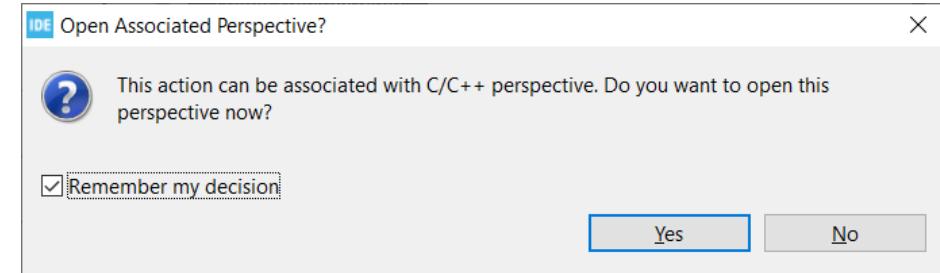


Can enable features of ThreadX

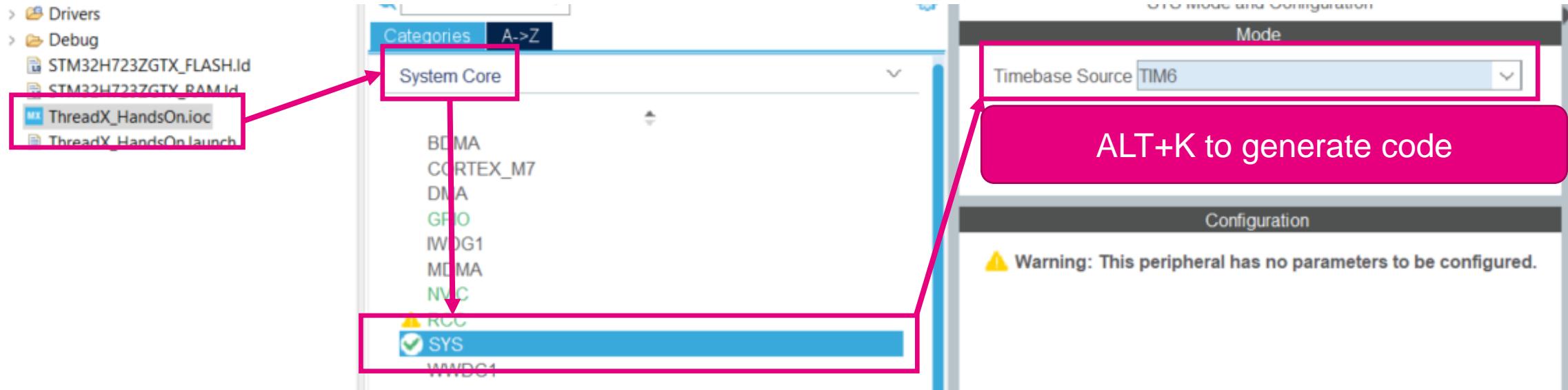
Fewer enabled feature will result
in smaller ThreadX footprint

Generate new code structure

- Type ALT-K or Project/Generate Code
- You can request automatic change of perspective to C/C++
- But you will get a warning regarding the time base
- Please answer “No” and let’s address this point

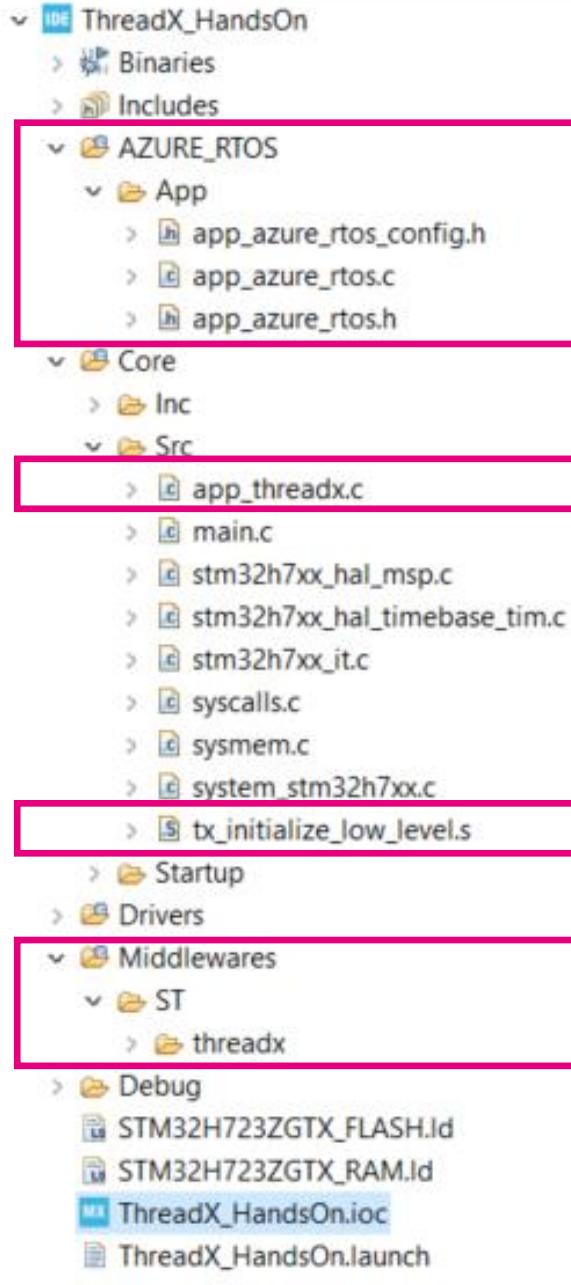


Selection of TimeBase



If HAL library don't have separated time source the compilation will fail because both libraries want to use SysTick_Handler interrupt

Code generation



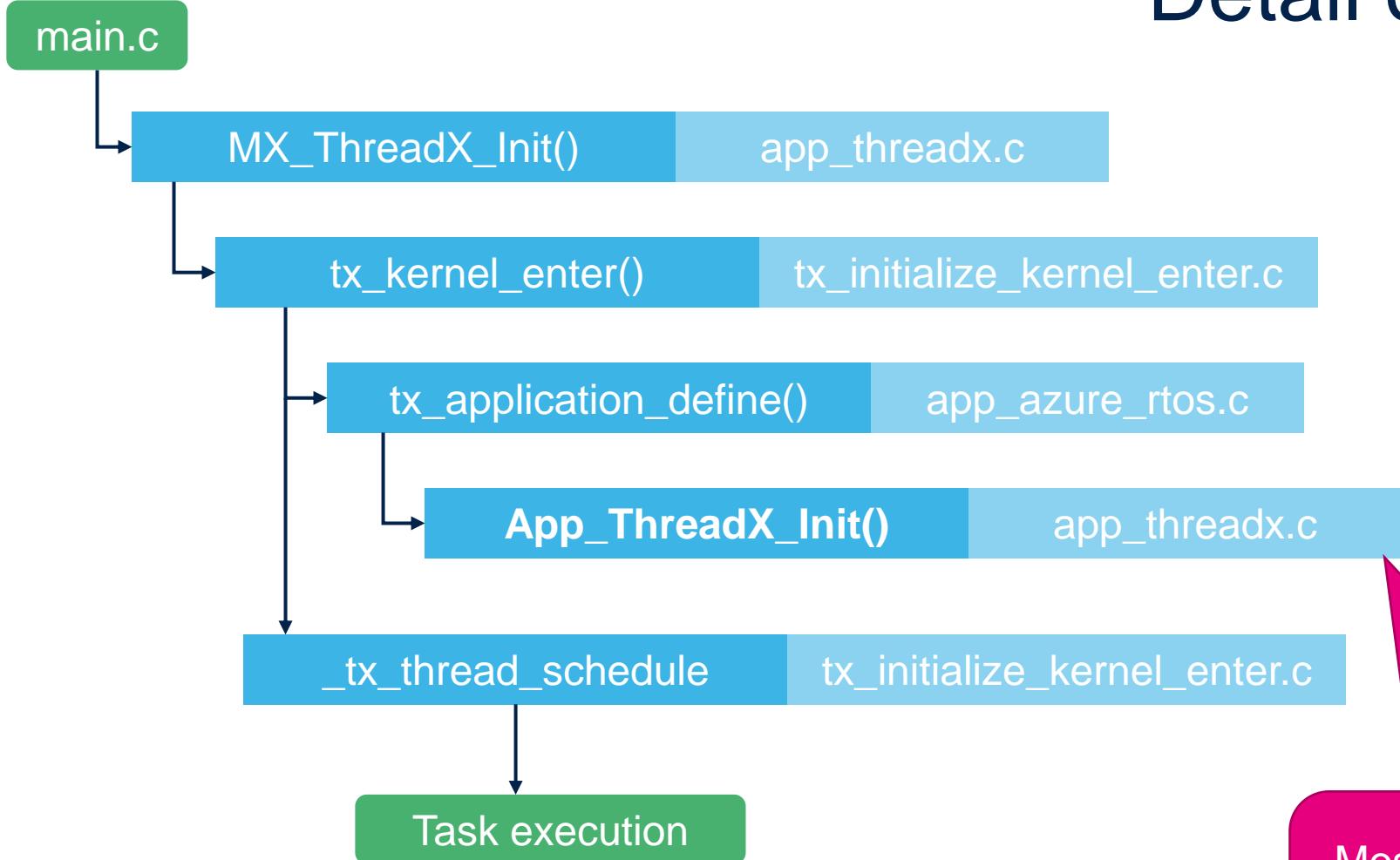
AZURERTOS\App contains the adaptation layer and configuration

app_threadx.c place for user to create threads

ThreadX low level interface to STM32

Middlewares\ST\threadx contains all threadx source code

Detail on code structure



Most important is `App_ThreadX_Init`
where user should create his first
threads

Creating first thread : variables declarations

- Open Core\Src\app_threadx.c and add

- To line 36 /* USER CODE BEGIN PD */
 - #define THREAD_STACK_SIZE 1024

```

35/* Private define -----
36/* USER CODE BEGIN PD */
37#define THREAD_STACK_SIZE 1024
38/* USER CODE END PD */
39

```

Our stack size

- To line 47 just after /* USER CODE BEGIN PV */

```

uint8_t thread_stack[THREAD_STACK_SIZE];
TX_THREAD thread_ptr;

```

```

45/* Private variables -----
46/* USER CODE BEGIN PV */
47uint8_t thread_stack[THREAD_STACK_SIZE];
48TX_THREAD thread_ptr;
49/* USER CODE END PV */
50

```

Our stack

- To line 52 after /* USER CODE BEGIN PFP */

```
VOID my_thread_entry(ULONG initial_input);
```

```

51/* Private function prototypes -----
52/* USER CODE BEGIN PFP */
53VOID my_thread_entry(ULONG initial_input);
54/* USER CODE END PFP */
55

```

Thread function prototype

STEP 2: Create the thread

- After the stack is allocated, we can create the thread: line 70
 - You can use completion with CTRL-Space at any time

```
tx_thread_create(  
    &thread_ptr,  
    "my_thread",  
    my_thread_entry,  
    0x1234,  
    thread_stack,  
    THREAD_STACK_SIZE,  
    15,  
    15,  
    1,  
    TX_AUTO_START  
);
```

Thread handle defined from line 48

Name of thread

Thread function from line 53

Thread function input argument (any value you want 0-4G)

Thread stack from line 47

Thread stack size from line 37

Thread priority

Thread preemption priority

Thread time slice

Start of thread

```
/* USER CODE BEGIN App_ThreadX_Init */  
tx_thread_create( &thread_ptr, "my_thread", my_thread_entry, 0x1234, thread_stack,  
THREAD_STACK_SIZE, 15, 15, 1, TX_AUTO_START);  
/* USER CODE END App_ThreadX_Init */
```

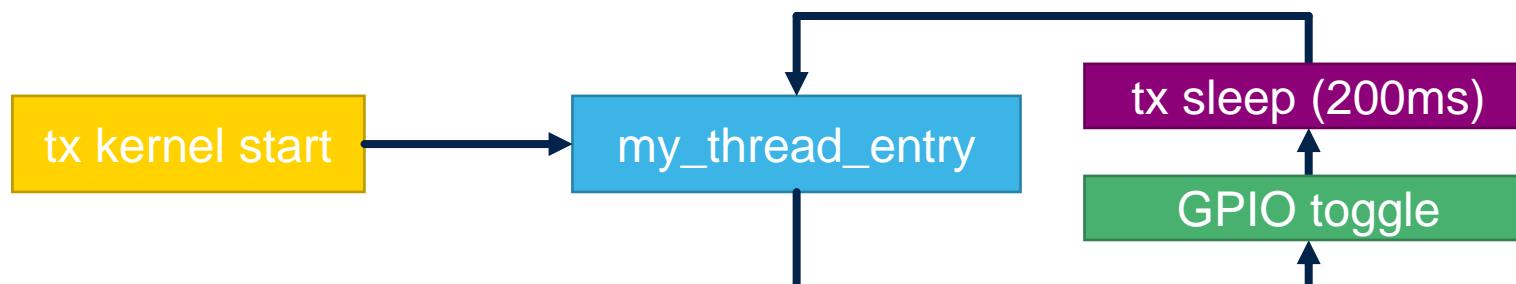
STEP 3: Create the thread's main function

- First, as we want to access to LED definitions, we need to include main.h
 - At line 26, just after /* USER CODE BEGIN Includes */
 - #include "main.h"
 - Then after line 95, /* USER CODE BEGIN 1 */

```
VOID my_thread_entry (ULONG initial_input)
{
    while(1){
        HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
        tx_thread_sleep(20);
    }
}
```

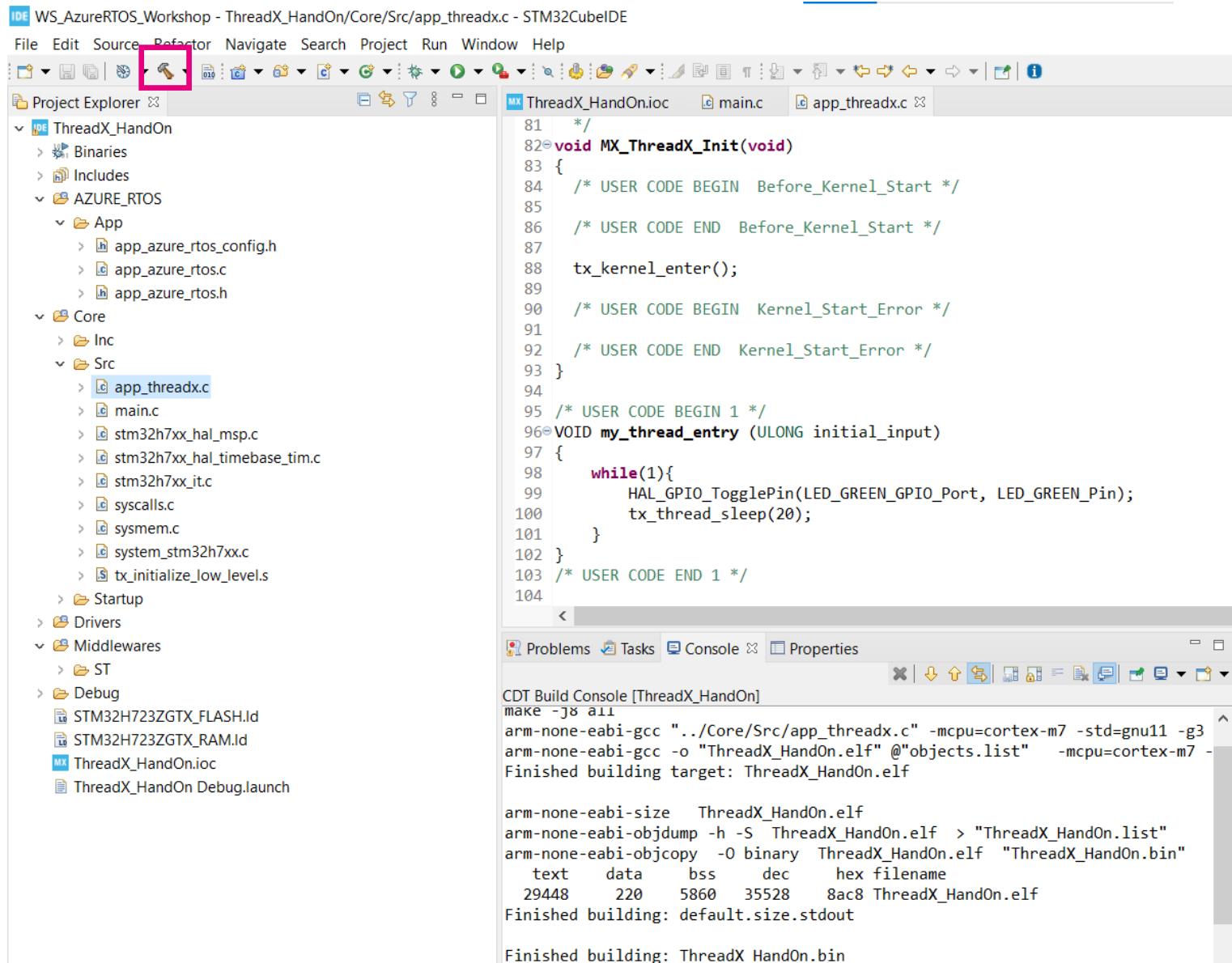
```
25/* Private includes -----
26/* USER CODE BEGIN Includes */
27#include "main.h"
28/* USER CODE END Includes */
29
```

```
95 /* USER CODE BEGIN 1 */
96 VOID my_thread_entry (ULONG initial_input)
97 {
98     while(1){
99         HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
100        tx_thread_sleep(20);
101    }
102 }
103 /* USER CODE END 1 */
```



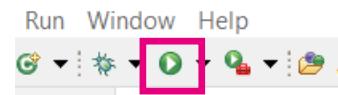
Build the project

- Use hammer icon or Project/Build Project



Flash the new thread project

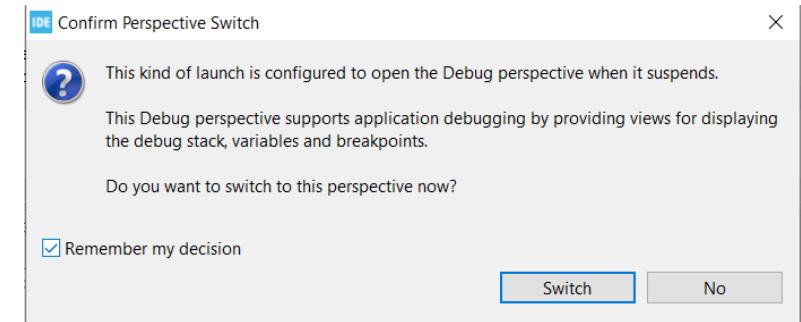
- The configuration from bare metal can be reused to flash the board.
- You can use either
 - the green arrow button
 - Or Run/Run
- You should get now a LED blinking with 400 ms period



Start the debugger

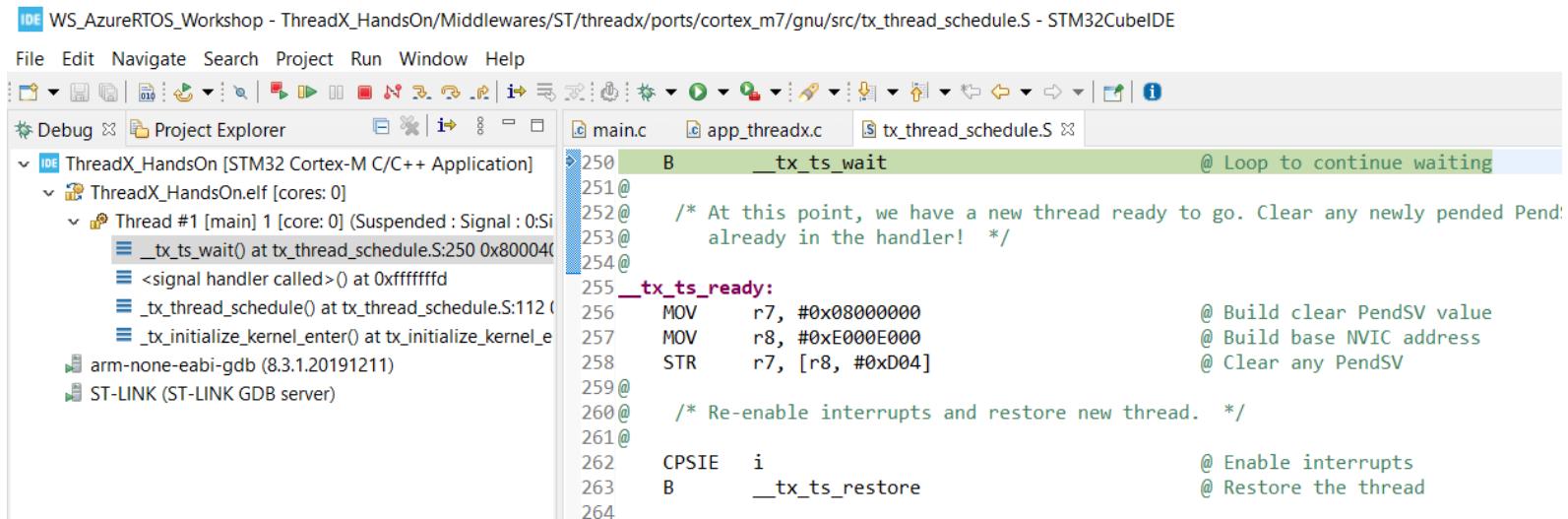
- Click on debugger icon :

- Pop-up window to switch perspective
 - Click remember my decision
 - Click on switch to activate the debug perspective
- Run the code by clicking either
 - Resume icon
 - F8
 - Run/Resume
- Green led should blink



Suspend the execution

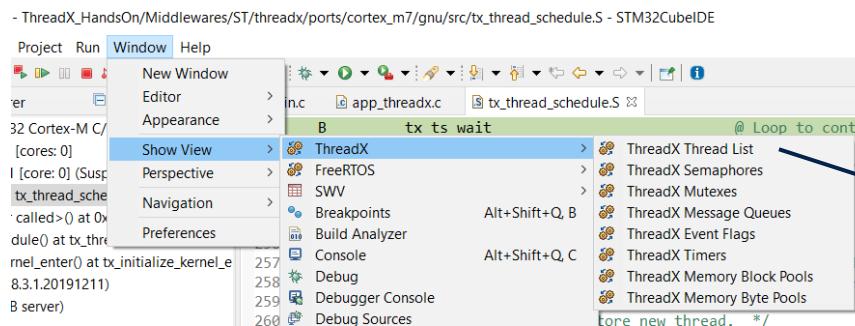
- Break the running firmware using either
 - Break icon 
 - Run/Suspend
- Suspended in the ThreadX scheduler



```
WS_AzureRTOS_Workshop - ThreadX_HandsOn/Middlewares/ST/threadx/ports/cortex_m7/gnu/src/tx_thread_schedule.S - STM32CubeIDE
File Edit Navigate Search Project Run Window Help
Debug Project Explorer main.c app_threadx.c tx_thread_schedule.S
250 B _tx_ts_wait @ Loop to continue waiting
251@
252@ /* At this point, we have a new thread ready to go. Clear any newly pended PendSV already in the handler! */
253@
254@
255 _tx_ts_ready:
256 MOV r7, #0x08000000 @ Build clear PendSV value
257 MOV r8, #0xE000E000 @ Build base NVIC address
258 STR r7, [r8, #0xD04] @ Clear any PendSV
259@
260/* Re-enable interrupts and restore new thread. */
261@
262 CPSIE i @ Enable interrupts
263 B _tx_ts_restore @ Restore the thread
264
```

ThreadX aware debugging

- Threads can be viewed using specific view
- Please open the ThreadX thread list view
 - Window>Show View/ThreadX/ThreadX Thread List



A screenshot of the STM32CubeIDE interface. The window title is 'WS_AzureRTOS_Workshop - ThreadX_HandsOn/Middlewares/ST/threadx/ports/cortex_m7/gnu/src/tx_thread_schedule.S - STM32CubeIDE'. The code editor shows a C file with assembly-like comments. A specific section of the code is highlighted:

```
250 B __tx_ts_wait @ Loop to continue waiting
251@ /* At this point, we have a new thread ready to go. Clear any newly pended PendSV - since we
252@ already in the handler! */
253@ ...
254@ ...
255 __tx_ts_ready:
256 MOV r7, #0x08000000 @ Build clear PendSV value
257 MOV r8, #0xE000E000 @ Build base NVIC address
258 STR r7, [r8, #0xD04] @ Clear any PendSV
259@ ...
260@ /* Re-enable interrupts and restore new thread. */
261@ ...
262 CPSIE i B __tx_ts_restore @ Enable interrupts
263@ ...
264 ...
265 #ifdef TX_ENABLE_FPU_SUPPORT
266 ...
267 .global tx_thread_fpu_enable
268 .thumb_func
269 tx_thread_fpu_enable:
270@ ...
```

The bottom part of the interface shows the 'ThreadX Thread List' view, which is a table with the following data:

Name	Prio...	State	Run Count	Stack Start	Stack End	Stack Size	Stack Ptr	Stack Usage
Main Thread	10	SLEEP (68)	127	0x200000f8	0x200002f7	512	0x20001cc	Disabled
System Timer Thre...	0	SUSPENDED	505	0x2000da0	0x2000119f	1024	0x2000106c	Disabled
Idle								

ThreadX debug view

The screenshot shows a debugger interface with the 'ThreadX Thread List' tab selected. The table displays the following data:

Name	Prio...	State	Run Count	Stack Start	Stack End	Stack Size	Stack Ptr	Stack Usage
my_thread	15	SLEEP (20)	39	0x240005dc	0x240009db	1024	0x2400089c	320
System Timer Thre...	0	SUSPENDED	38	0x24000da8	0x240011a7	1024	0x2400104c	348
Idle								

Annotations with arrows point from callout boxes to specific table columns:

- 'Thread name provided at creation time' points to the 'Name' column.
- 'Thread priority provided at creation time' points to the 'Prio...' column.
- 'Stack Ram location (in byte pool)' points to the 'Stack Start' column.
- 'Thread size provided at creation time' points to the 'Stack Size' column.
- 'Thread state' points to the 'State' column.
- 'How many times thread was scheduled' points to the 'Run Count' column.
- 'Current stack pointer' points to the 'Stack Ptr' column.
- 'Maximum stack used. To activate using the top left icon' points to the 'Stack Usage' column.

Add a breakpoint to see impact on thread list

- Resume execution
- Open app_threadx.c and go to end of file
- Click on the line containing the HAL_GPIO_TogglePin
- Set a breakpoint here with either methods
 - Double click on the column just on the left of the line numbers
 - CTRL-SHIFT-B
- As soon as the breakpoint is created, the execution will stop with maximum 200 ms delay
- The ThreadX thread view is now updated ...

Updated ThreadX thread list

The screenshot shows the STM32CubeIDE interface with the following details:

- Project Explorer:** Shows the project "ThreadX_HandOn" with the file "app_threadx.c" open.
- Code Editor:** Displays the C code for the ThreadX kernel initialization and a user-defined thread entry point. The line `HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);` is highlighted.
- Registers:** A table showing memory variables, with a row for `initial_input` set to `Value 4660`.
- Threads:** A table titled "ThreadX Thread List" showing two threads:

Name	Prio	State	Run Count	Stack Start	Stack End	Stack Size	Stack Ptr	Stack Usage
my_thread	15	RUNNING	52	0x240005dc	0x240009db	1024	0x2400089c	320
System Timer Thre...	0	SUSPENDED	51	0x24000da8	0x240011a7	1024	0x2400104c	348
Idle								
- Page Number:** 02-48

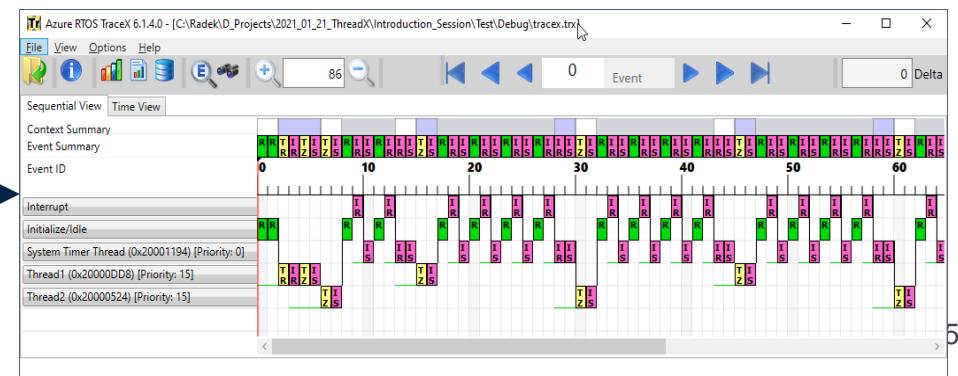
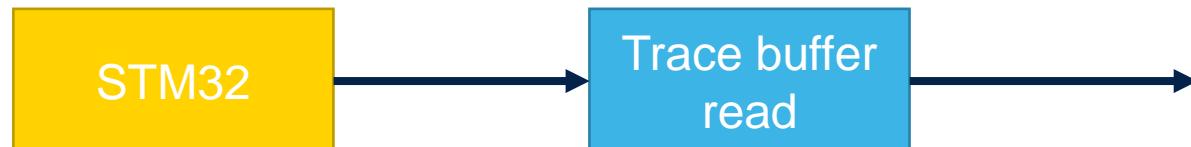
Conclusion

- Adding ThreadX application to a bare metal project consists in
 - Adding the ThreadX component in STM32CubeMX
 - Adapt ThreadX configuration in STM32CubeMX
 - Generate the code from STM32CubeMX
 - This will update the project structure with all needed files
 - Use ThreadX API to create ThreadX components thread, mutex, etc
- This is the integration of ThreadX in our Cube ecosystem

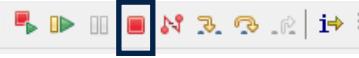
Step 3: Activate TraceX

Purpose

- We will see in the new hands-on how to activate TraceX feature of ThreadX
- TraceX allows a post mortem analysis of the thread sequence occurred during firmware execution
- To make this work we need to
 - activate the feature in the firmware
 - allocate memory to store the trace during firmware execution
 - execute the firmware to fill the trace in memory
 - write the content of the memory in a file on the PC
 - use Microsoft's TraceX tool to display the result



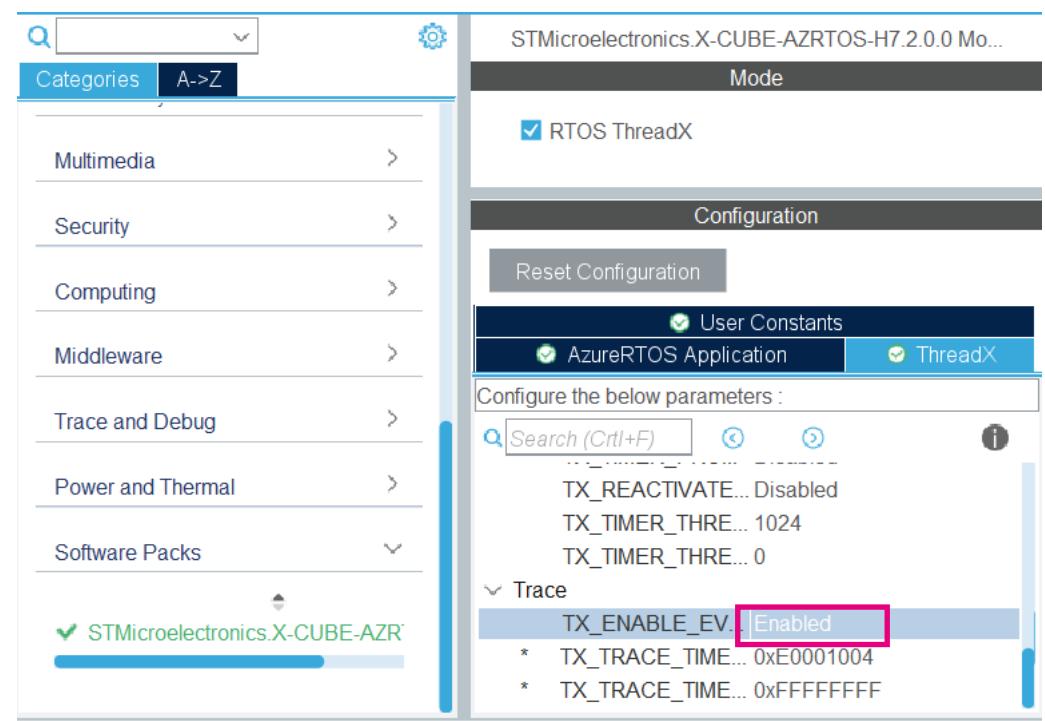
Activating traceX

- First close the debugger from previous hands-on
 - Click on the red square 
 - Run/Terminate
 - CTRL-F2
- Activate embedded STM32CubeMX: double click on ThreadX_HandsOn.ioc
- Go to the Software Packs/ Select component or ALT-O
- Open STMicroelectronics.X-CUBE-AZRTOS-H7/RTOS ThreadX
- Activate TraceX support
- OK to validate

STMicroelectronics.X-CUBE-AZRTOS-H7	<input checked="" type="checkbox"/>	2.0.0
RTOS ThreadX	<input checked="" type="checkbox"/>	6.1.7
ThreadX / Core	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ThreadX / PerformanceInfo	<input type="checkbox"/>	<input checked="" type="checkbox"/>
ThreadX / TraceX support	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ThreadX / Low Power support	<input type="checkbox"/>	<input checked="" type="checkbox"/>
File System FileX	<input type="checkbox"/>	6.1.7

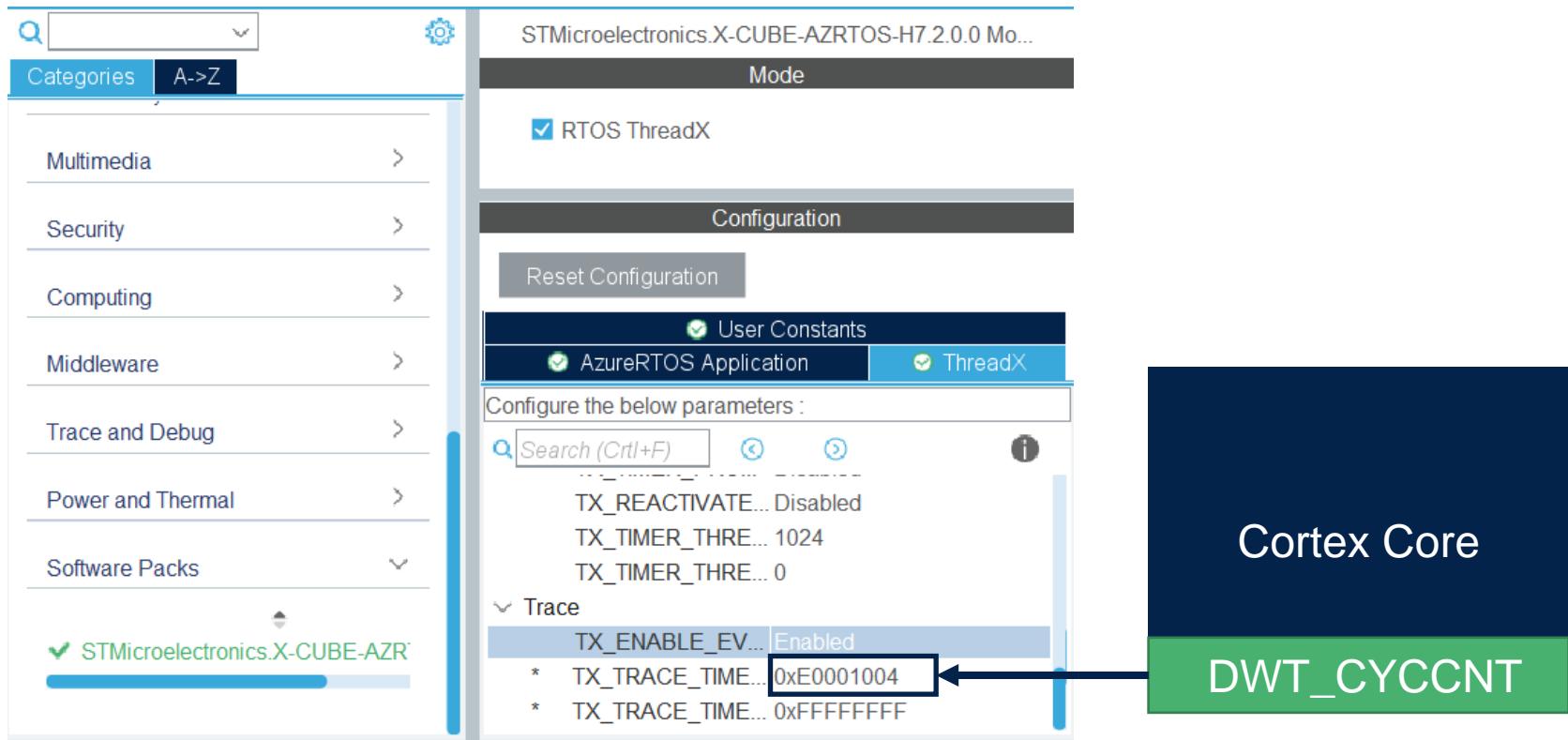
Enable the traces

- In left panel of CubeMX select Software pack
- In middle panel select ThreadX tab
- Change value of configuration TX_ENABLE_EVENT_TRACE to Enabled



Trace timestamp source

- The trace timestamp source is setup by default on the cycle counter of the Cortex at address 0xE0001004



Use ALT-K to update the generated code for TraceX support

Memory dedicated to trace

- In app_threadx.c into section /* USER CODE BEGIN PV */ Line 47 add

```
#define TRACEX_BUFFER_SIZE 64000
```

```
uint8_t tracex_buffer[64000] __attribute__ ((section (".trace")));
```

Buffer used for trace

put buffer to specific RAM section

```
49  
50 #define TRACEX_BUFFER_SIZE 64000  
51 uint8_t tracex_buffer[64000] __attribute__ ((section (".trace")));  
52 /* USER CODE END PV */
```

- To line 73 add:

```
tx_trace_enable(&tracex_buffer,TRACEX_BUFFER_SIZE,30);
```

Amount trace objects in buffer

Our trace buffer

buffer size

```
74  
75 tx_trace_enable(&tracex_buffer,TRACEX_BUFFER_SIZE,30);  
76 /* USER CODE END App_ThreadX_Init */  
77
```

Put trace buffer to AXI SRAM

- Add this to STM32H723ZGTX_FLASH.Id file around line 120

```
.trace (NOLOAD):
{
    . = ALIGN(4);
    *(.trace)
} > RAM_D1
```

Put .trace section to AXI SRAM

```
120 } >FLASH
121
122 .trace (NOLOAD):
123 {
124     . = ALIGN(4);
125     *(.trace)
126 } > RAM_D1
127
128 /* used by the startup to initialize data */
129 _sidata = LOADADDR(.data);
130
```

Now the trace buffer will be on known location 0x24000000

Execute firmware in debugger

- Fill the trace buffer by running project

- Compile project 

- Run debug 

- Run code 

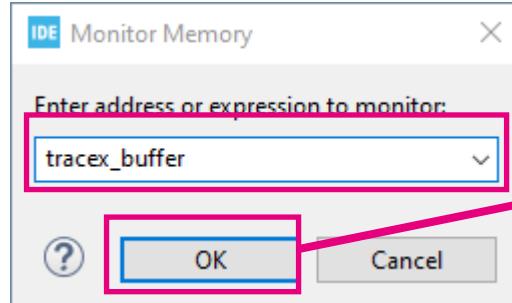
- Suspend debug 

- Read trace buffer

- Open memory window

- Add new memory area

- Type tracex_buffer



Address	0 - 3	4 - 7	8 - B	C - F
24000000	42545854	FFFFFFFFFF	00000024	30000024
24000010	00002000	D0050024	D0050024	F0F90024
24000020	D0120024	AAAAAA...	BBBBBBBB	CCCCCCCC
24000030	00018000	E8060124	A8070124	00040000
24000040	53797374	656D2054	696D6572	20546872
24000050	65616400	F14F0C00	9346009F	D8465FF0

If window is missing
Menu>Window>Show view>Memory

TraceBuffer export



Address	0 - 3	4 - 7	8 - B	C - F
24000000	42545854	FFFFFF	00000024	30000024
24000010	00002000	D0050024	D0050024	F0F90024

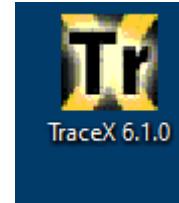
- Please click on export icon
- A pop-up should look like this
 - Select format **Raw binary**
 - Type length **64000**
 - Click on Browse and create a directory in tmp
 - for instance: c:\tmp\AZRTOSLogs
 - Then use filename **log1 trx**
 - Then press ok : the log is written



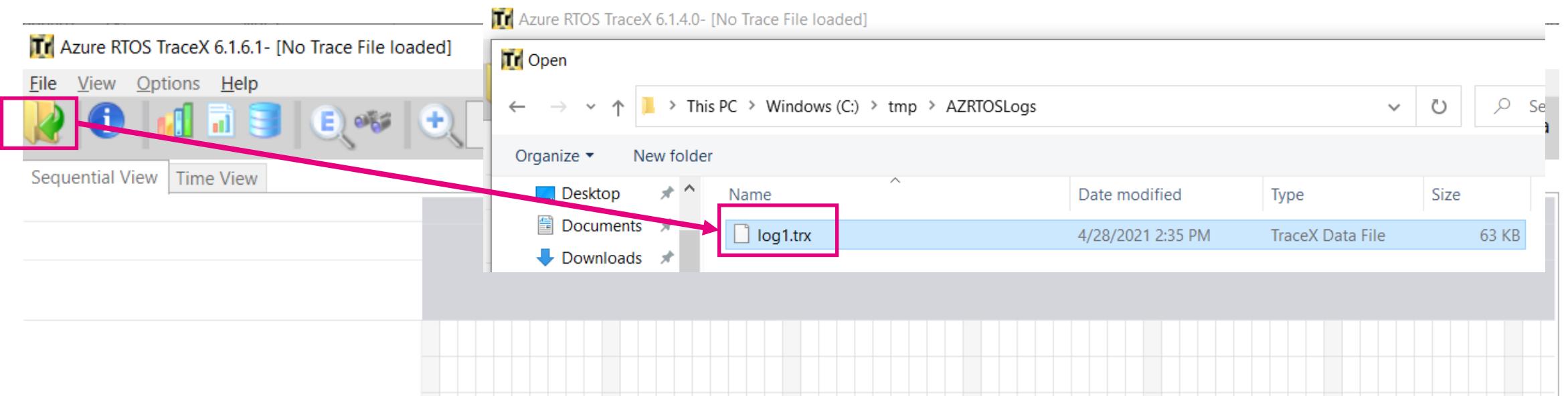
TRX extension is important for
TraceX App

Open TraceX

- You should have TraceX installed. Open the application

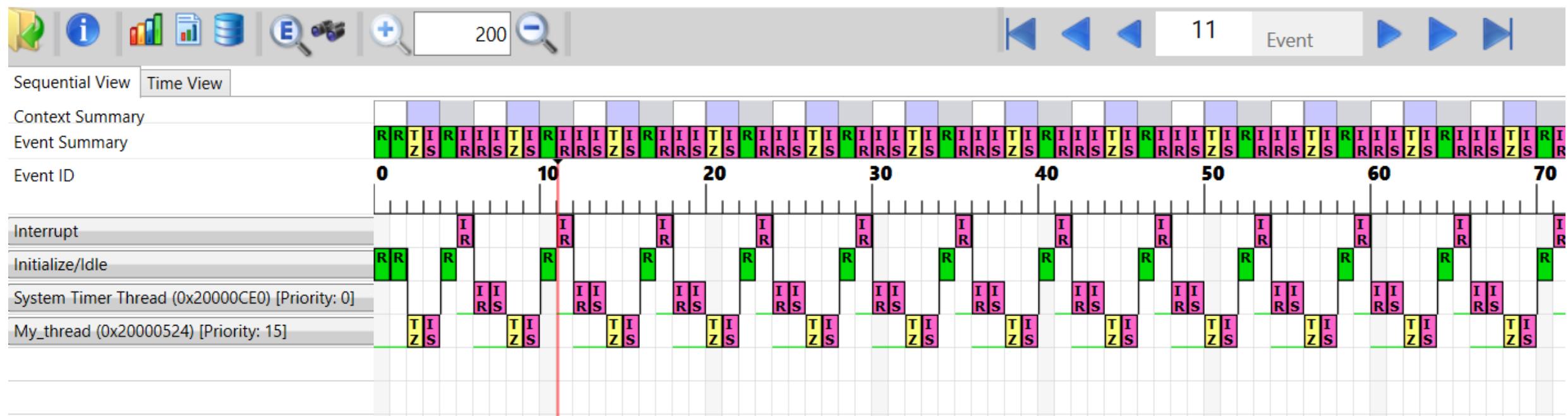
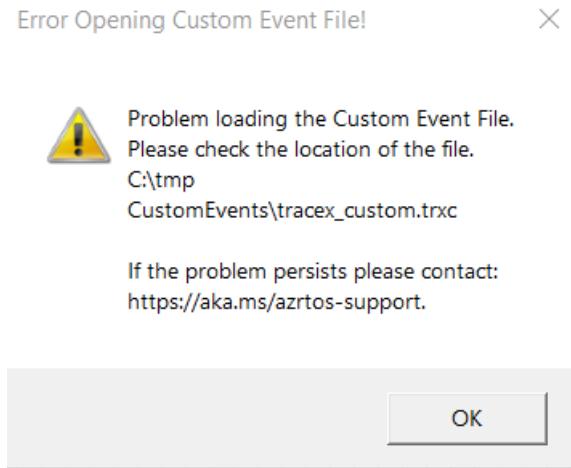


- Open file and select the log file just created



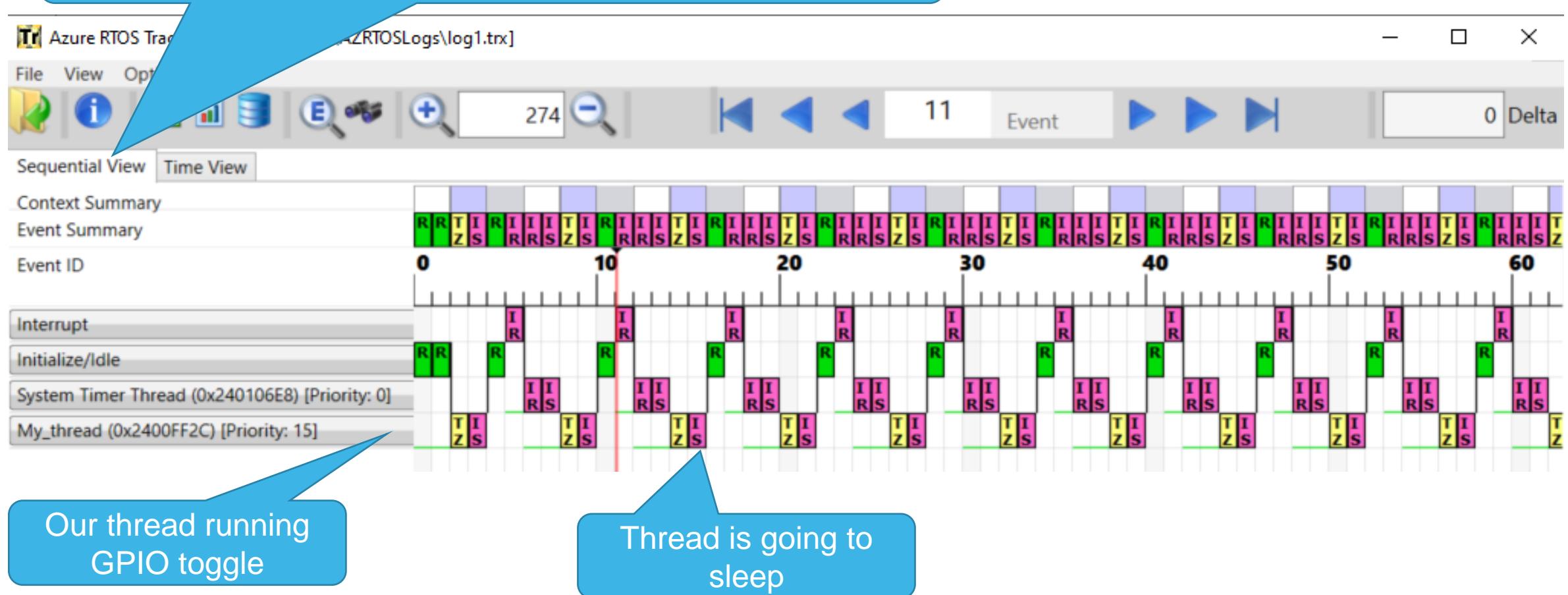
Show the trace

- An error pop-up should appear:
- Just press OK
- You get the trace



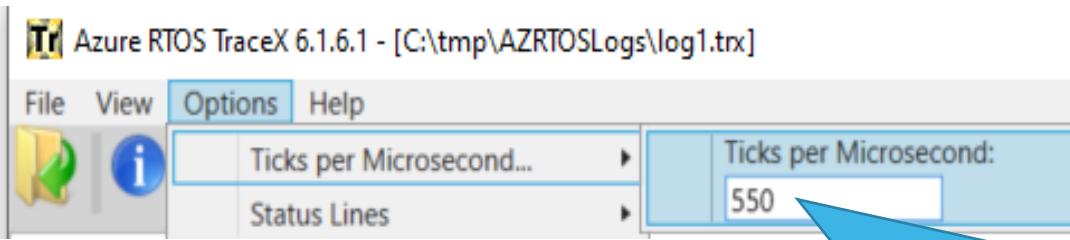
Analyzing the trace

The sequential view does not provide any timing information, just the scheduling sequence

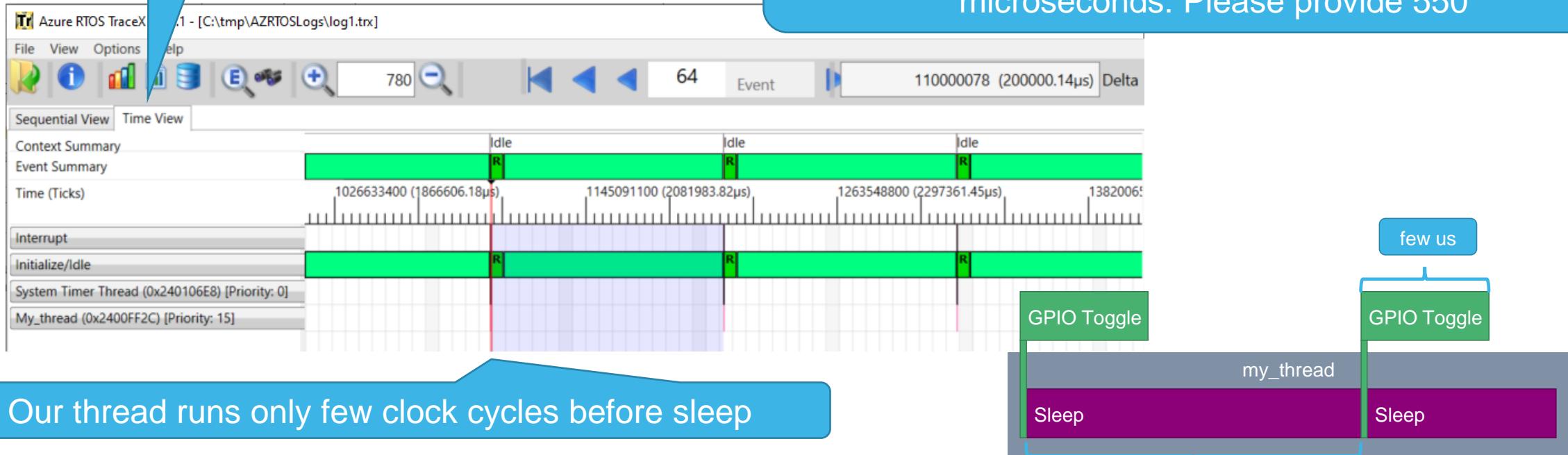


Using Time View

You can check the Time view



To be able to use it efficiently you need to provide the tick per microseconds in menu Options/Ticks per microseconds: Please provide 550



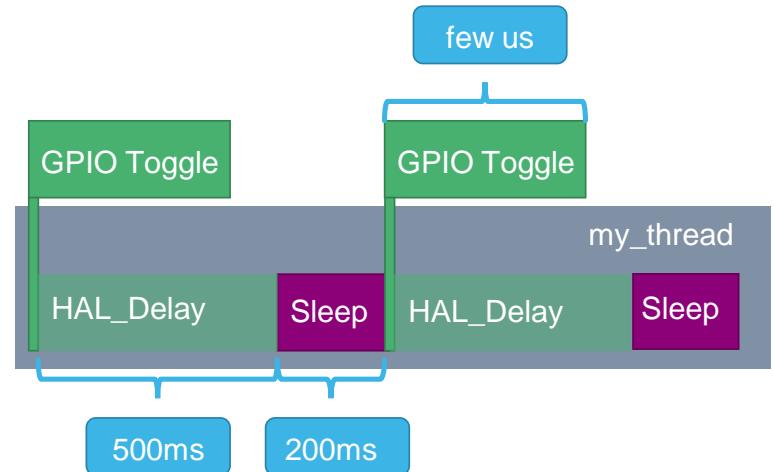
Our thread runs only few clock cycles before sleep

Change behaviour

- In order to show how TraceX can be used, we will add one change in the code
- This change consists in using HAL_Delay which is performing an active wait
- In MainThread add the following lines:

```
HAL_Delay(500);
```

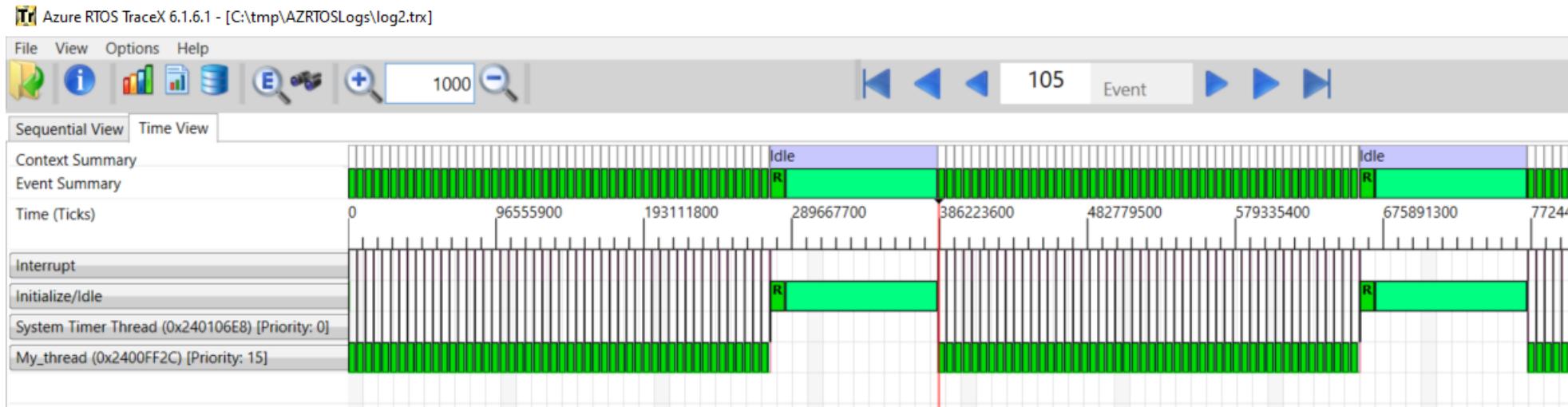
```
102 /* USER CODE BEGIN 1 */  
103 VOID my_thread_entry (ULONG initial_input)  
104 {  
105     while(1){  
106         HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);  
107         HAL_Delay(500);  
108         tx_thread_sleep(20);  
109     }  
110 }
```



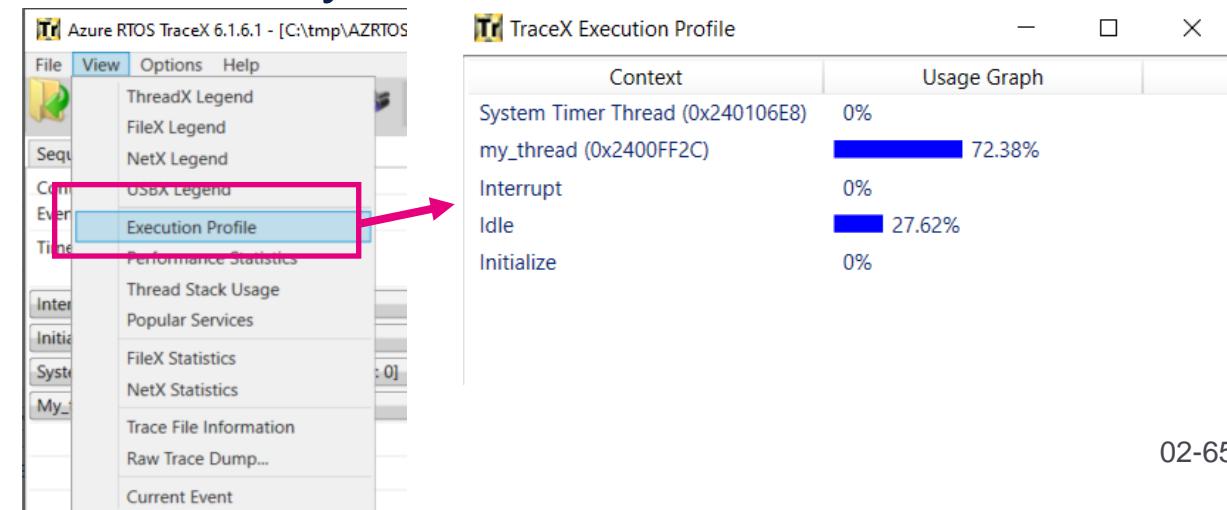
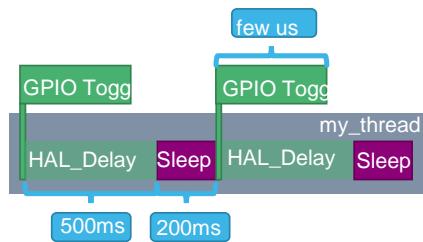
Obtain new trace

- You can launch directly the debugger. Recompilation will be done automatically.
- Launch the firmware : LED should blink with longer period
- Break after few seconds
- Export the trace buffer:
 - Click on memory. The TraceBuffer should be already displayed
 - Double click on Hex Integer to create a new rendering
 - Click on export button and replace log1.trx by log2.trx, then click OK
 - Go back to TraceX application, click on open file icon or File/Open
 - Default directory should be same as before, just select log2.trx
 - In the sequential view you shouldn't see much difference
 - Select Time view and unzoom

New trace



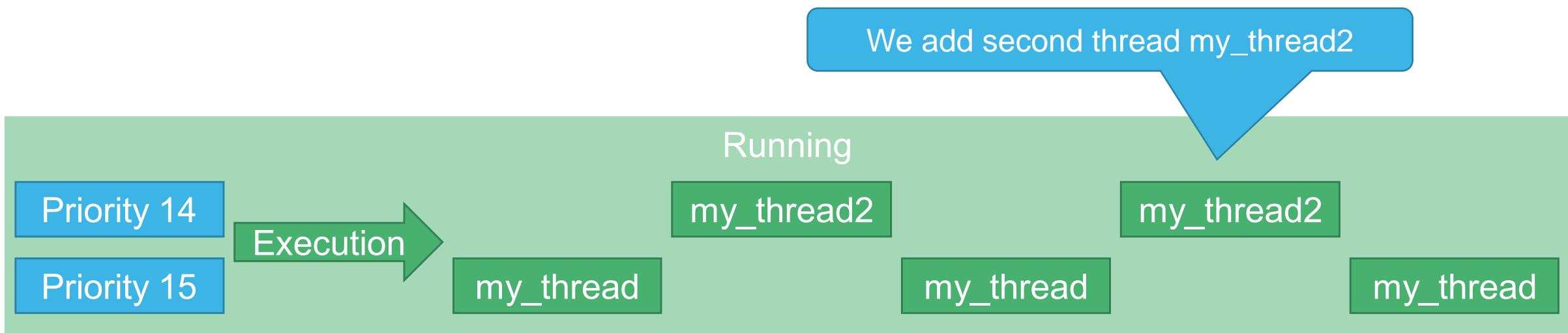
- In time view the my_thread active wait is immediately visible.
- Also, launching View/Execution profile
 - The my_thread time consumption is obvious!



Step 4 : Concurrent threads

Purpose

- Now we have seen the ThreadX environment, we will show one specific feature of this RTOS that is preemption threshold
- Let's have 2 concurrent threads and see how the scheduler manages the priorities



Add a second thread

Core\Src\app_threadx.c

- Edit App_threadx.c
 - Add new thread handle and stack

```
uint8_t thread_stack2[THREAD_STACK_SIZE];
TX_THREAD thread_ptr2;
```
 - Function prototype

```
VOID my_thread_entry2(ULONG initial_input);
```
 - Create thread priority 14

```
tx_thread_create(&thread_ptr2, "my_thread2", my_thread_entry2, 0x1234, thread_stack2, THREAD_STACK_SIZE, 14, 14, 1, TX_AUTO_START);
```

- Thread function

```
VOID my_thread_entry2 (ULONG initial_input)
{
    while(1){
        HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
        HAL_Delay(500);
        tx_thread_sleep(20);
    }
}
```

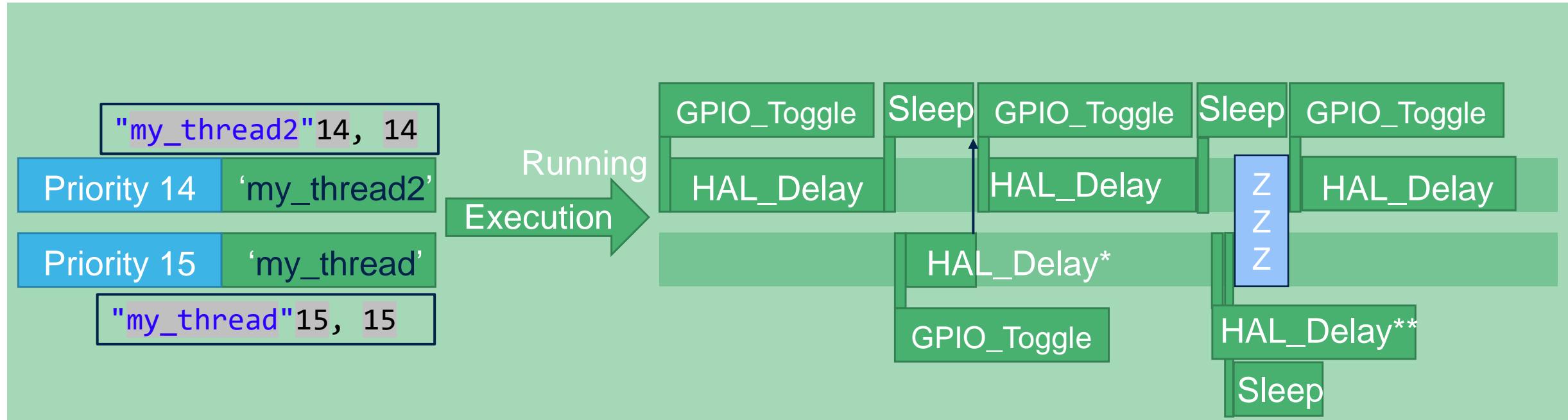
```
46 /* USER CODE BEGIN PV */
47 uint8_t thread_stack[THREAD_STACK_SIZE];
48 TX_THREAD thread_ptr;
49 uint8_t thread_stack2[THREAD_STACK_SIZE];
50 TX_THREAD thread_ptr2;
51 #define TRACEX_BUFFER_SIZE 64000
52 uint8_t tracex_buffer[64000] __attribute__ ((section ("".trace")));
53 /* USER CODE END PV */
```

```
55@/* Private function prototypes -----
56 /* USER CODE BEGIN PFP */
57 VOID my_thread_entry(ULONG initial_input);
58 VOID my_thread_entry2(ULONG initial_input);
59 /* USER CODE END PFP */
```

```
75 /* USER CODE BEGIN App_ThreadX_Init */
76 tx_thread_create( &thread_ptr, "my_thread", my_thread_entry, 0x1234, thread_stack, THREAD_STACK_SIZE, 15, 15, 1, TX_AUTO_START);
77 tx_thread_create(&thread_ptr2, "my_thread2", my_thread_entry2, 0x1234, thread_stack2, THREAD_STACK_SIZE, 14, 14, 1, TX_AUTO_START);
```

```
112@VOID my_thread_entry2 (ULONG initial_input)
113 {
114     while(1){
115         HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
116         HAL_Delay(500);
117         tx_thread_sleep(20);
118     }
119 }
120 /* USER CODE END 1 */
```

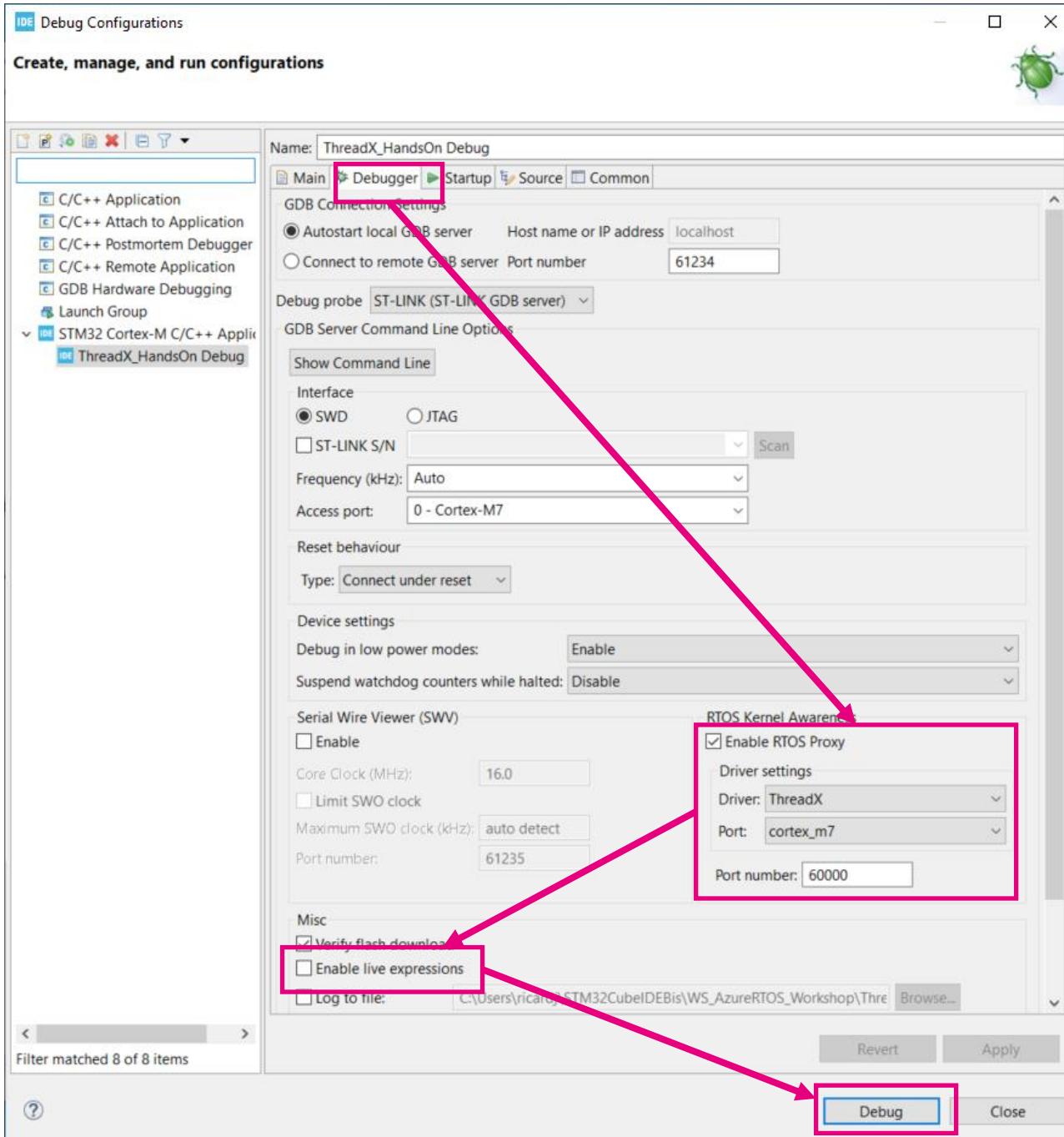
Expected behavior



*) HAL_Delay() is interrupted by higher prio thread which just became active

**) Continue with the interrupted HAL_Delay() which timer now has elapsed

Debugger ThreadX aware



- Open Run/Debug Configuration
- Select Debugger tab
- Enable RTOS Proxy
- Select Cortex M7
- Unselect ‘Enable Live Expression’
- Click on Debug

ThreadX Stack frames

- Execute the firmware for few seconds. 2 LEDS should blink with different periods

Get stack frame for each thread

The screenshot shows the WS_AzureRTOS_Workshop IDE interface. In the Project Explorer, three threads are listed under the ThreadX_HandsOnElf project:

- Thread #6 [my_thread2] (Running)
- Thread #7 [my_thread] (Suspended: Container)
- Thread #8 [System Timer Thread] (Suspended: Container)

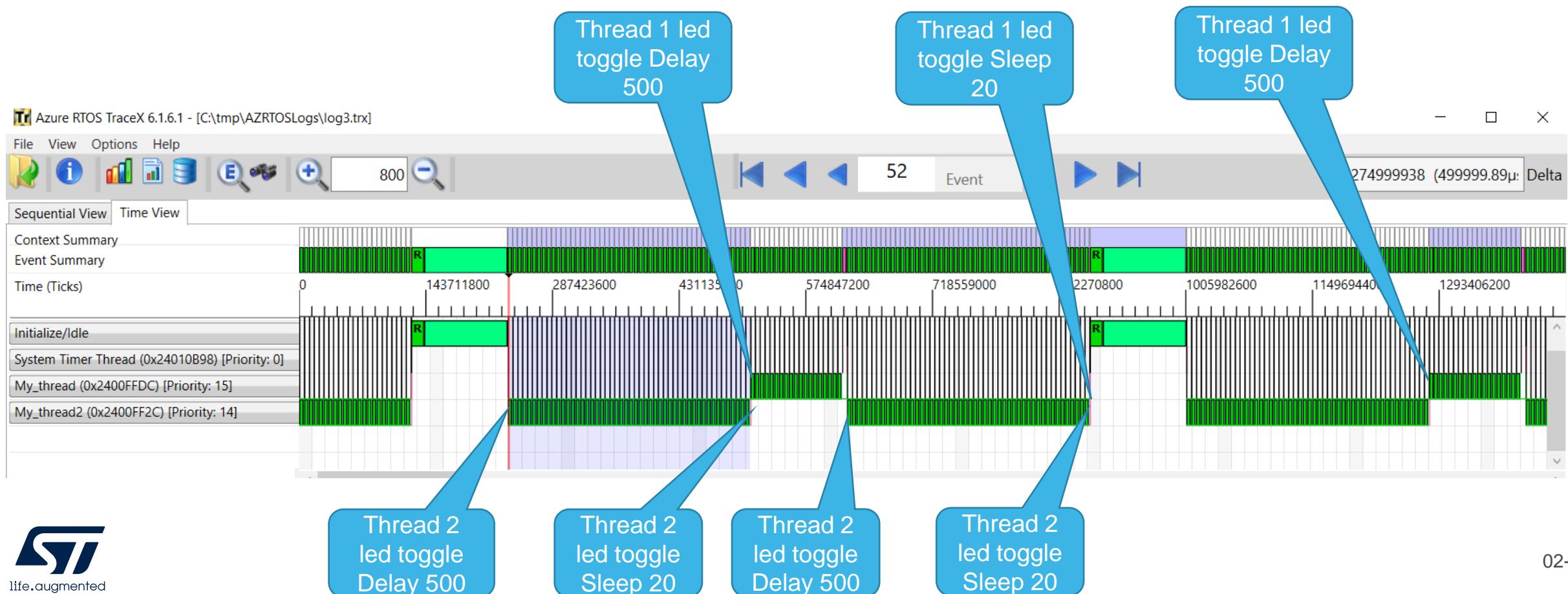
In the main.c code editor, the HAL_Delay function is shown, with the while loop highlighted. The ThreadX Thread List table at the bottom shows the following information:

Name	Prio...	State	Run Count	Stack Start	Stack End	Stack Size	Stack Ptr	Stack Usage
my_thread	15	READY	12	0x2401048c	0x2401088b	1024	0x2401071c	368
my_thread2	14	RUNNING	13	0x2401008c	0x2401048b	1024	0x2401031c	368
System Timer Thre...	0	SUSPENDED	12	0x24010c58	0x24011057	1024	0x24010ee4	372
Idle								

Annotations with red boxes highlight the stack frames for Thread #6, Thread #7, and Thread #8 in the Project Explorer. A blue callout points to the Project Explorer with the text "Get stack frame for each thread".

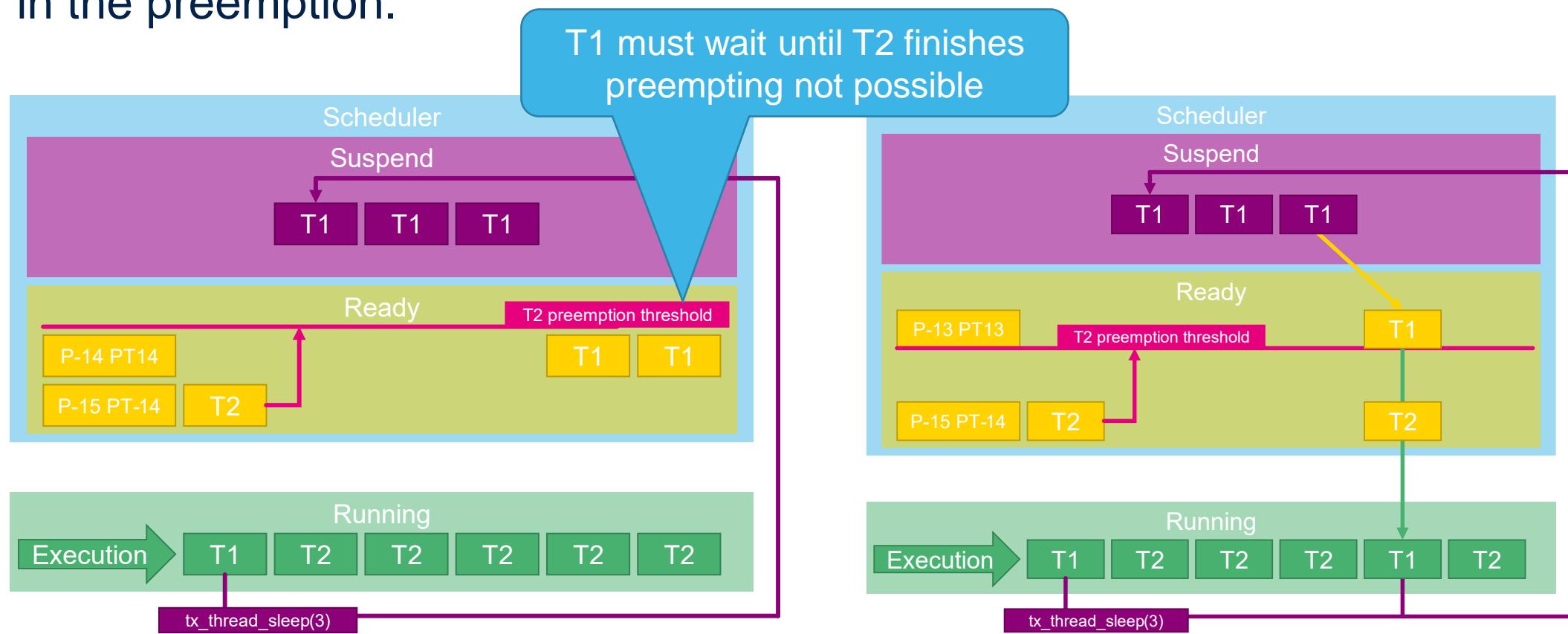
Get Trace

- Go to memory tab and export trace buffer to log3.trx
- Switch to TraceX, open log3.trx, time view, unzoom



Preemption threshold principle

- Preemption threshold is a ThreadX mechanism that introduces a kind of hysteresis in the preemption.

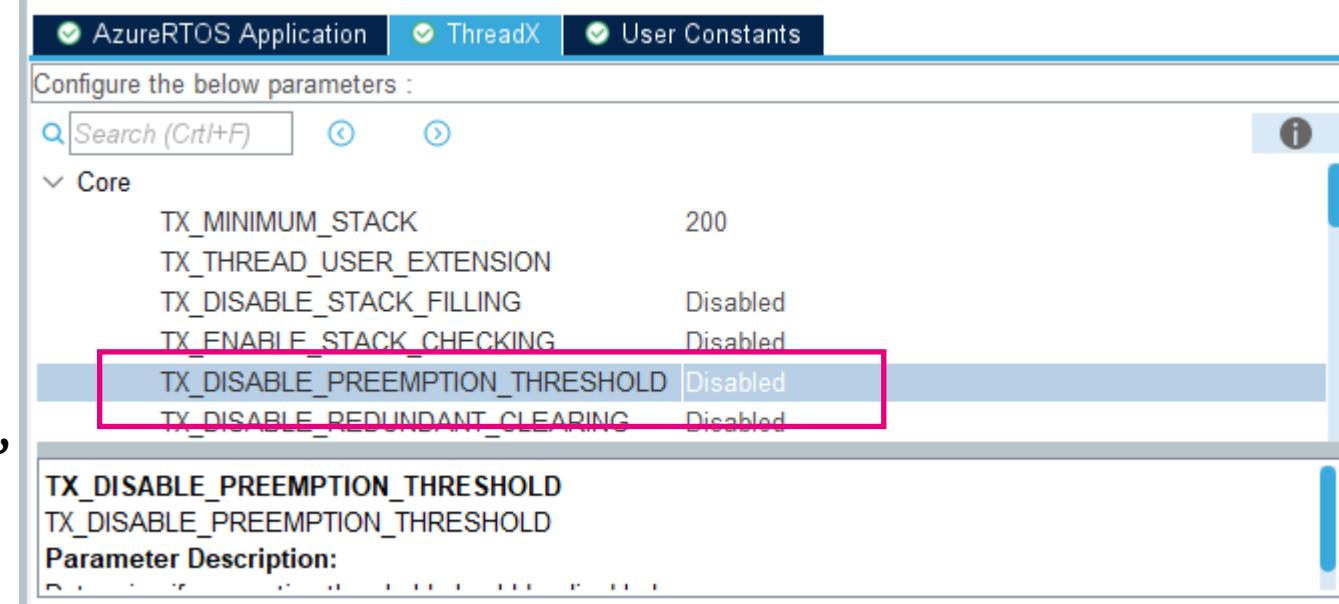


Activation of preemption threshold feature

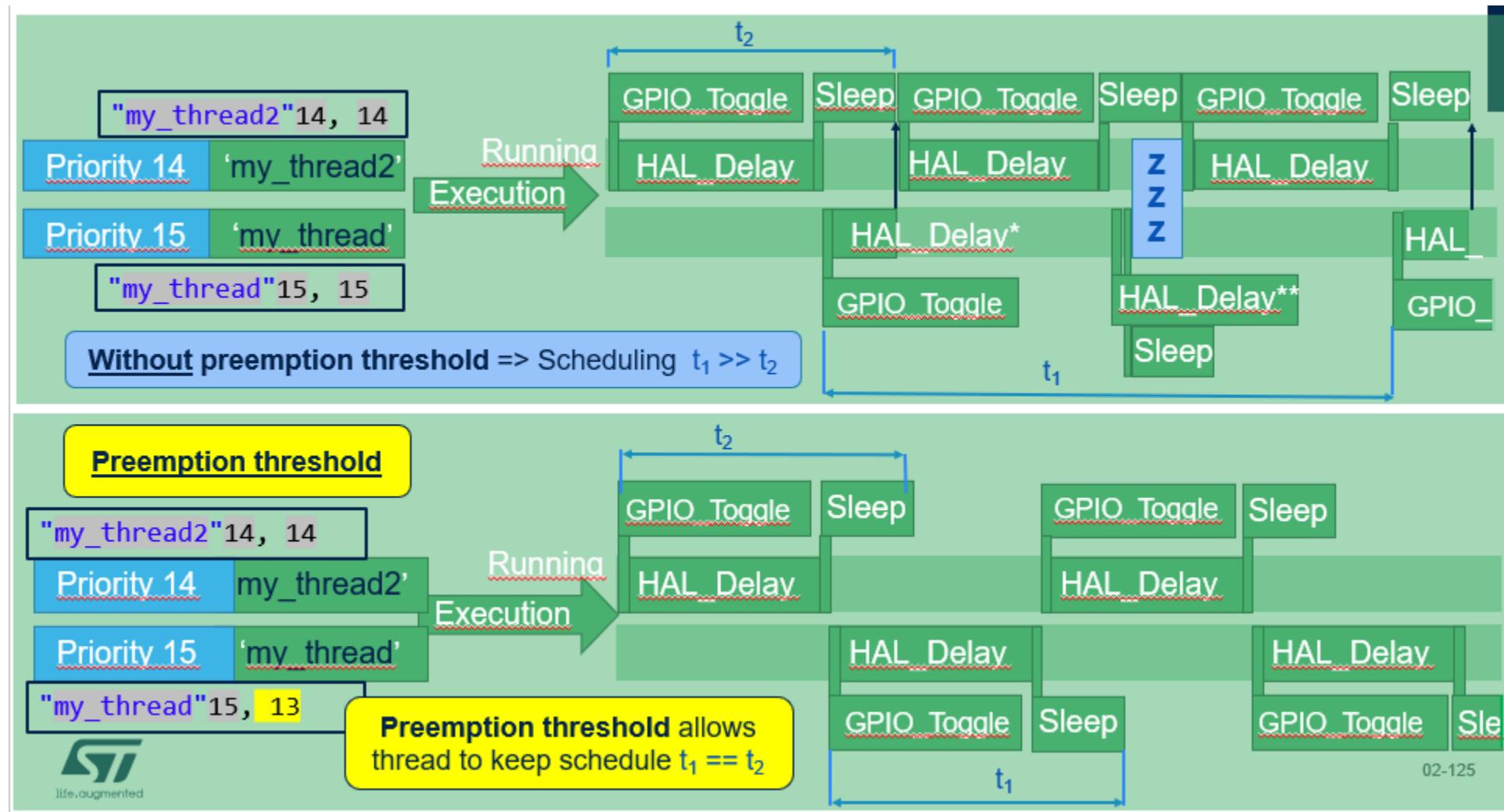
Core\Src\app_threadx.c

- Enable this feature in CubeMX
 - Set TX_DISABLE_PREEMPTION_THRESHOLD to DISABLE
 - ALT-K to update code
- Change my_thread preemption priority to 13

```
tx_thread_create(&thread_ptr, "my_thread",
    my_thread_entry, 0x1234,
    thread_stack, THREAD_STACK_SIZE,
    15, 13, 1,
    TX_AUTO_START);
```



Behavior of preemption threshold

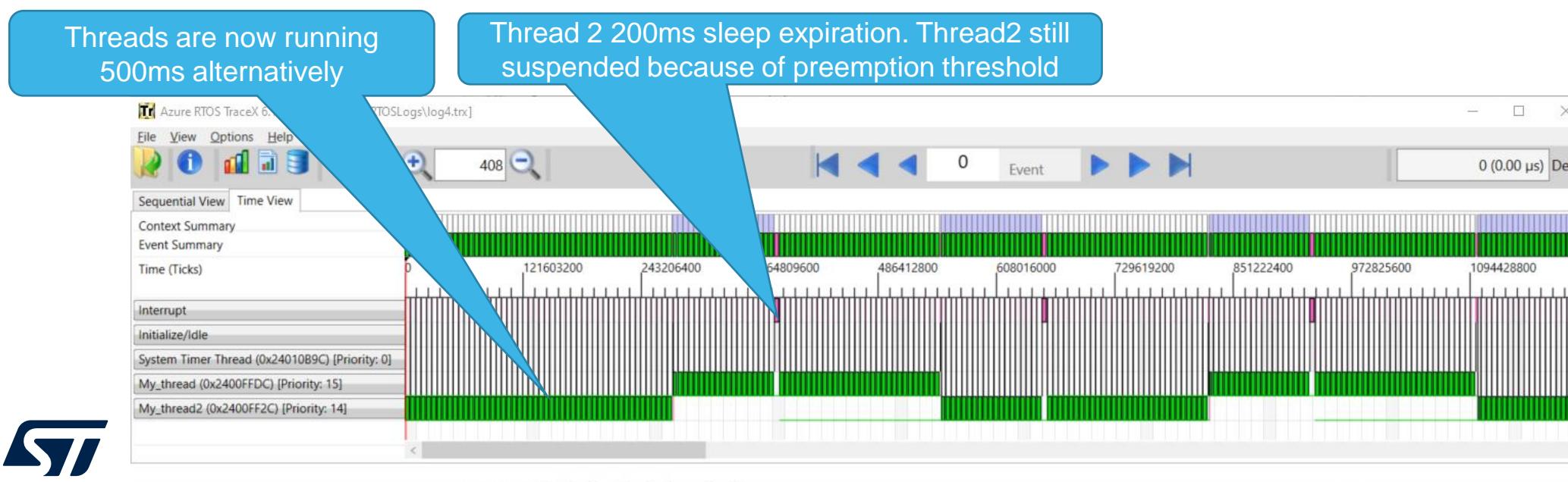


*) HAL_Delay() is interrupted by higher prio thread which just became active

**) Continue with the interrupted HAL_Delay() which timer now has elapsed

Check with TraceX

- Launch the debugger and execute the firmware for few seconds
- LEDs should blink alternatively with same period
- Go to memory tab and export trace buffer to log4.trx
- Switch to TraceX and open log4.trx, time view, unzoom



Conclusion

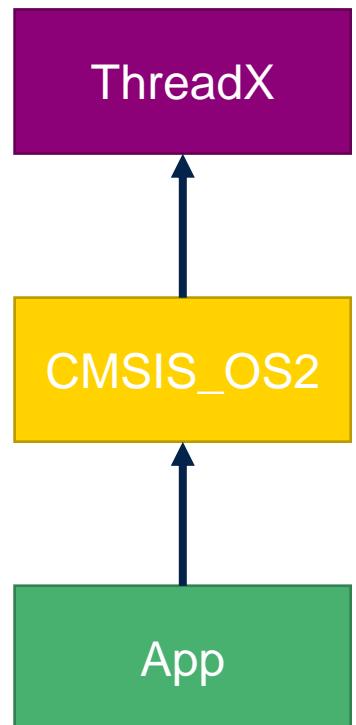
- We have seen how ThreadX is integrated in our environment
- It is easy to add ThreadX in a project and start using ThreadX API in an already prepared project structure
- STM32CubeIDE provides ThreadX specific views for debugging
- TraceX usage

Step 4 : CMSIS_OS2 wrapper

CMSIS_OS2 wrapper

- Can be added manually to project
- Copy cmsis_os2.h and cmsis_os2.c to your project
 - Copy cmsis_os2.c from Repository\Packs\STMicroelectronics\X-CUBE-AZRTOS-H7\2.0.0\Middlewares\ST\cmsis_rtos_threadx\
 - Copy to cmsis_os2.h from Repository\STM32Cube_FW_H7_V1.9.0\Drivers\CMSIS\RTOS2\Include\
- #Include “cmsis_os2.h” to your project
- Modify tx_user

```
#define TX_MAX_PRIORITIES 64
#define TX_THREAD_USER_EXTENSION
ULONG tx_thread_detached_joinable;
```
- You can now use CMSIS_OS2 API
- Check README.md for api supported by port

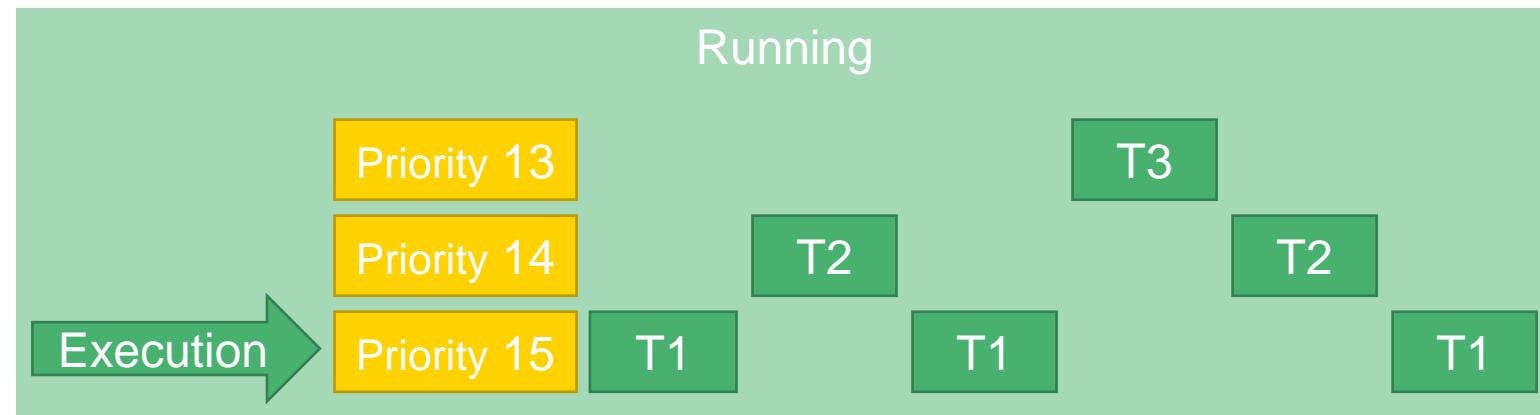


Step 2 : Short ThreadX features overview

List of components

- Threads
- Memory pools
- Semaphores
- Mutexes
- Queues
- Events
- Timers

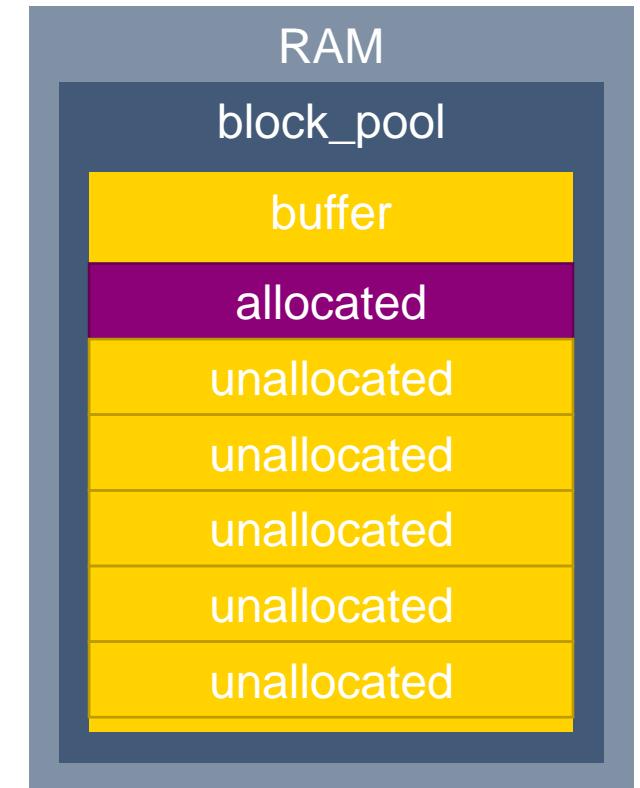
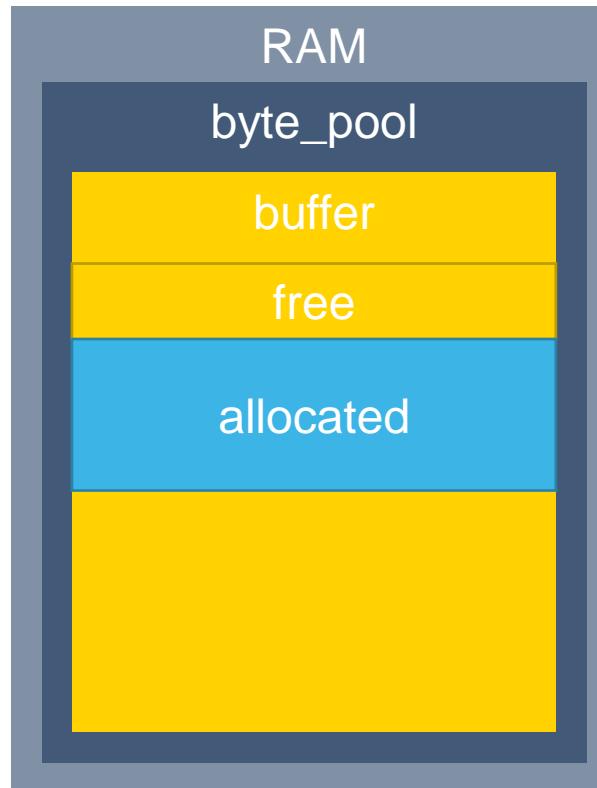
- Threads
 - Preemption priority
 - Priority threshold
 - Time slicing



<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#thread-execution-1>
https://rristm.github.io/stm32_threadx/show/threadx_thread.md

Memory pool

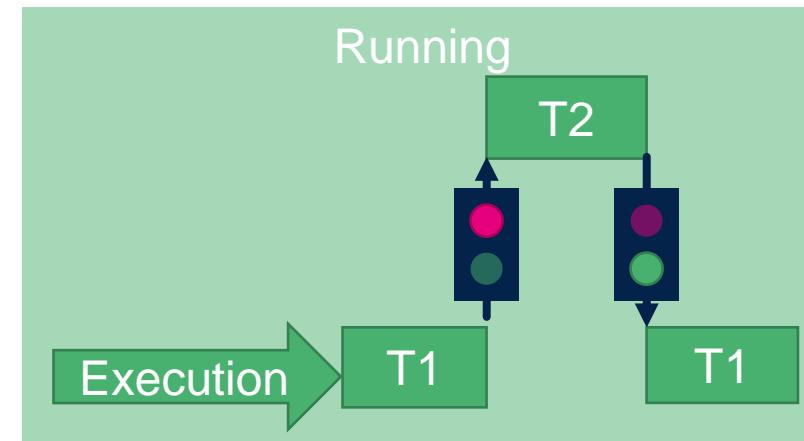
- Byte pool
 - Can allocate any value in bytes
 - Can be fragmented back
 - Fragmentation issues
- Block pool
 - Can give only one block fixed size
 - No fragmentation problems



<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#memory-byte-pools>
https://rristm.github.io/stm32_threadx/show/threadx_memory_pools.md

Semaphore

- Semaphore
 - Work as counting semaphore
 - Can work as binary semaphore

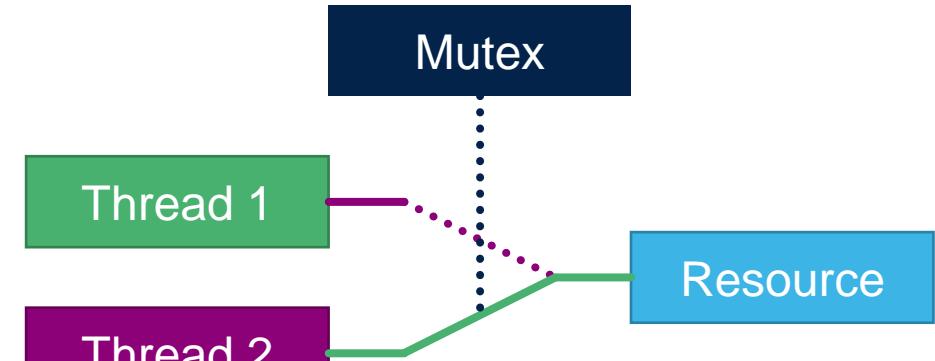
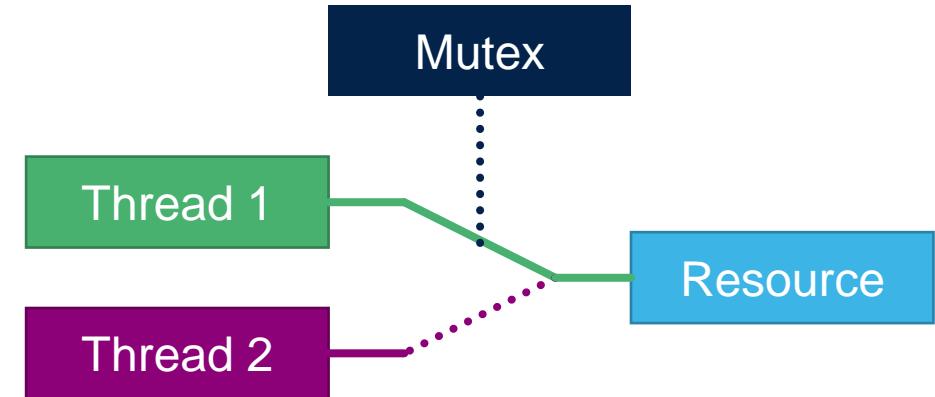


i

<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#counting-semaphores>

https://rristm.github.io/stm32_threadx/show/threadx_semaphore.md

- Mutex
 - Work as binary semaphore
 - Guards access to one resource
 - Thread can inherit priority



i

<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#mutexes>
https://rristm.github.io/stm32_threadx/show/threadx_mutex.md

- Queue
 - Used to send messages between threads
 - Up to 16 words (16*32bit)



i

<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#message-queues>

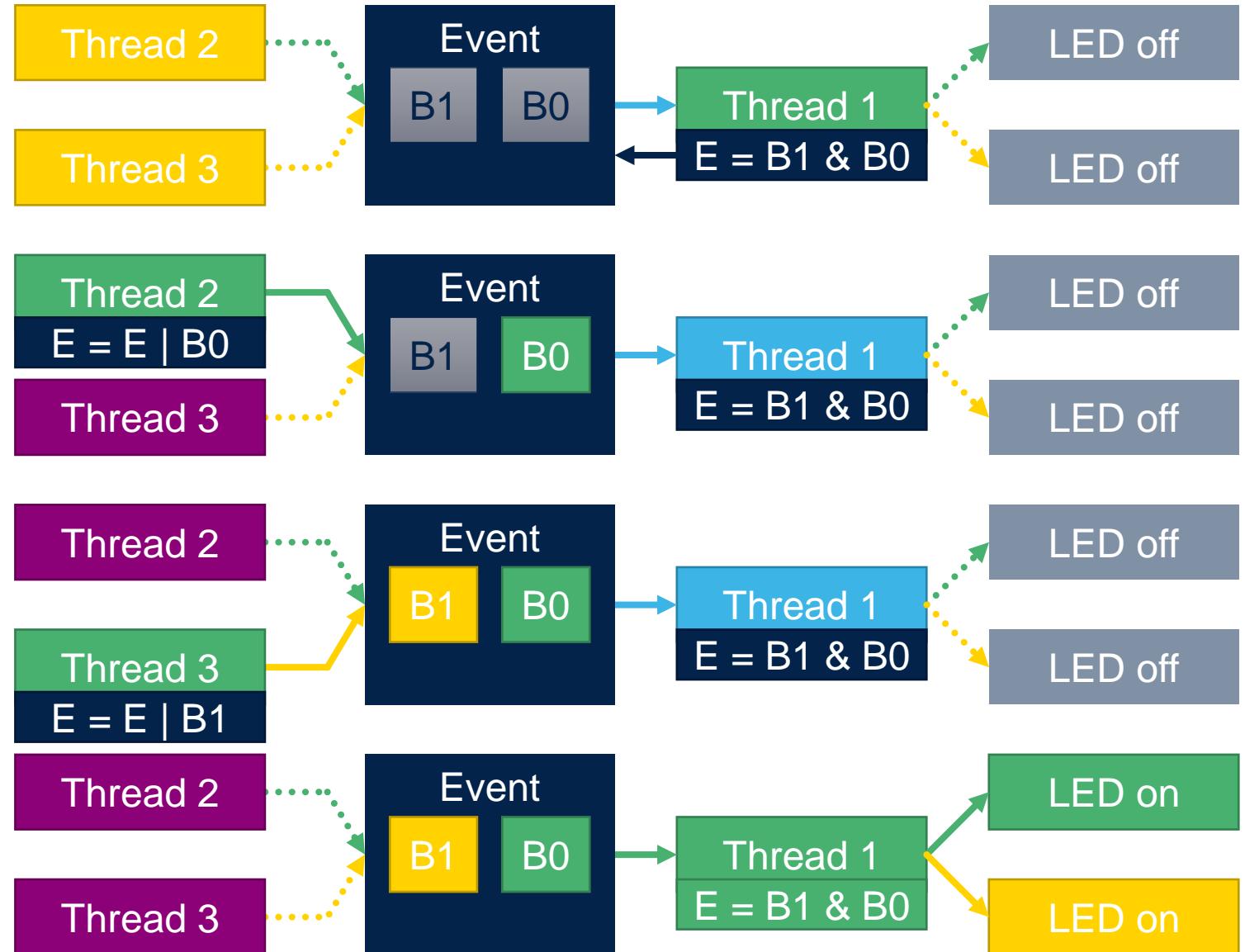
https://rristm.github.io/stm32_threadx/show/threadx_queue.md

Event

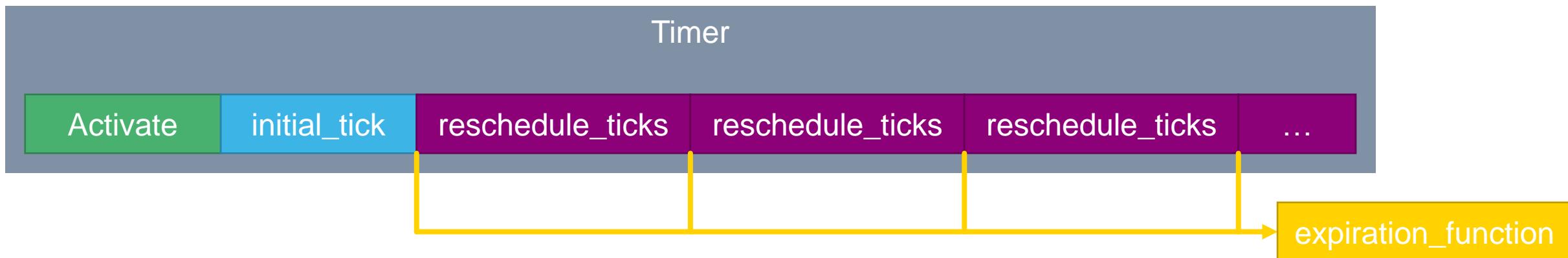
- Events
 - Used to synchronize thread
 - Base on OR / AND bit logic
 - One event have 32bits.

i

<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#event-flags>
https://rristm.github.io/stm32_threadx/show/threadx_events.md



- SW Timers
 - SW timer based on ThreadX ticks
 - Can be used to create delays or timeouts (only RTOS ticks)
 - Repetitive and one-shot mode



<https://docs.microsoft.com/en-us/azure/rtos/threadx/chapter3#application-timers-1>

https://rristm.github.io/stm32_threadx/show/threadx_timer.md

Low power

- Introduced in ThreadX 6.1.5

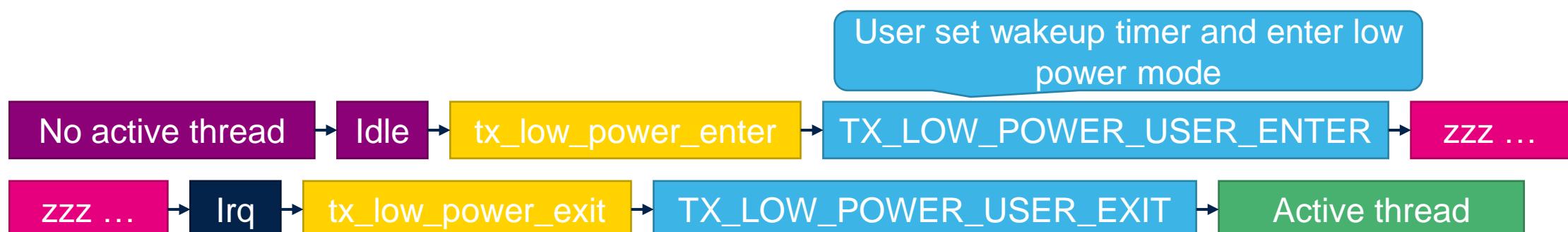
- tx_low_power functions
- Allow to calculate time needed for low power before enter
- Allow to set new time after low power exit
- Resume ThreadX operation after low power
- Activated by definition TX_LOW_POWER
- By defining `TX_LOW_POWER_USER_ENTER` function to user defined low power enter.



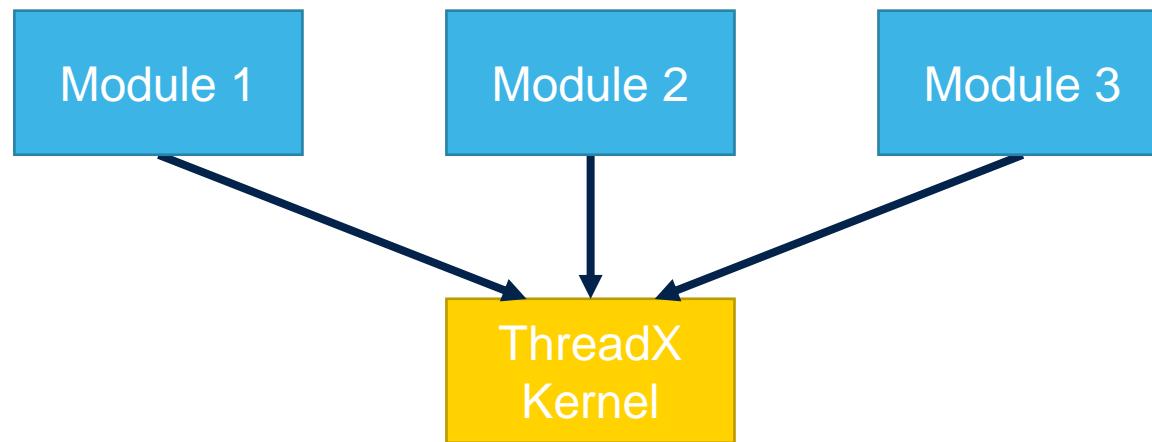
<https://github.com/azure-rtos/threadx>



Example:
X-CUBE-AZRTOS-H7\2.0.0\Projects\NUCLEO-H723ZG\Applications\ThreadX\Tx_LowPower



- Application can be separated to modules
- Modules possible to load on the fly
- Modules can be protected by MPU



i

<https://docs.microsoft.com/en-us/azure/rtos/threadx-modules/chapter1>

ThreadX footprint

- <https://docs.microsoft.com/en-us/azure/rtos/threadx/overview-threadx>

Azure RTOS ThreadX Service	Typical Size in Bytes
Core Services (Require)	2,000
Queue Services	900
Event Flag Services	900
Semaphore Services	450
Mutex Services	1,200
Block Memory Services	550
Byte Memory Services	900

Performance info

Performance info

- Allow to get performance statistics inside app
- Functions have *_performance_* in their name
- Here link how to enable it

As and example. *performance_system_info will give information's about system states

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

how many times threads were resumed for example

Adding this feature will increase ThreadX size

Thank you

© STMicroelectronics - All rights reserved.

The STMicroelectronics corporate logo is a registered trademark of the STMicroelectronics group of companies. All other names are the property of their respective owners.

