NobleProg

FreeRTOS Workshop
STM32U575

Grzegorz Mazur

Updated 12.02.2025

Contents

1. Development environment

x86-64-based PC running 64-bit Windows 11/10

Software installed (free):

- STM32CubeIDE from https://www.st.com/en/development-tools/stm32cubeide.html. (You need to register as ST user to get the software; you account date will also be needed for downloading the CubeIDE packages.)
- TeraTerm v5.3 or later from https://github.com/TeraTermProject
- STM32CubeIDE packages
- Pliki nagłówkowe z https://github.com/gbm-ii/ucintro, potrzebne do wybranych ćwiczeń.

Hardware:

- Nucleo-U575ZIQ
- USB cable with micro-B plug for connecting PC to Nucleo board

2. Project creation

CubeMX application generator, being a part of STM32CubeIDE, is used for project creation and setup.

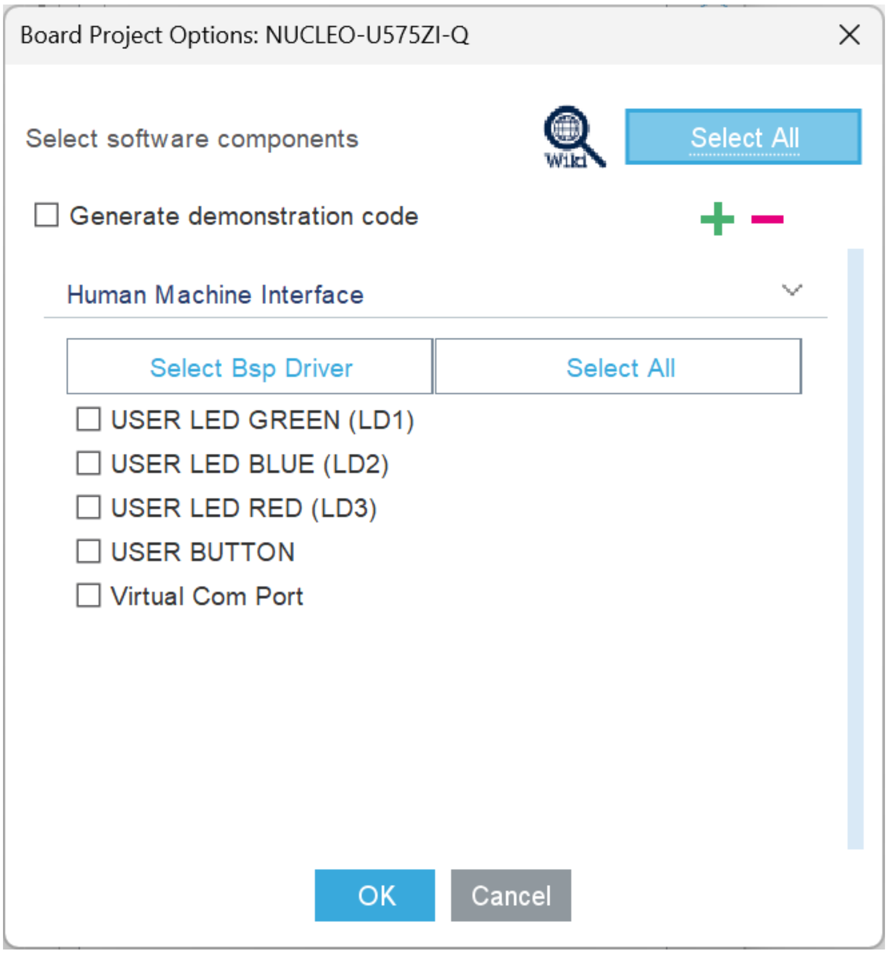
2.1. Starting the new project

> To avoid confusions, if STM32CubeIDE workspace was used before and another project was opened, before creating the new project close all the editor tabs (so that we don’t modify another project code by mistake).

To create a new project using CubeMX, start STM32CubeIDE, then:

- Select New STM32 Project from main window or File menu.
- After an automatic update of MCU database, hardware platform selection window opens. In this window, select the 2nd tab – Board Selector, in the left pane set filter MCU/MPU Series – STM32U5, then in the right bottom pane board list find NUCLEO-U575ZI-Q and select it. Press Next to confirm. If this is the first project using the selected MCU series, the support package will be downloaded and installed, which will take some time.
- In the next dialog, we enter the project name and select Finish.
- In the next presented dialog Board Project Options, we should deselect all the proposed items by pressing Unselect All button. The final state of this dialog is shown below.

After clicking OK, the project skeleton will be created and the MCU package view will be presented in the main CubeMX pane.



Board Project Options: NUCLEO-U575ZI-Q

Select software components

□ Generate demonstration code

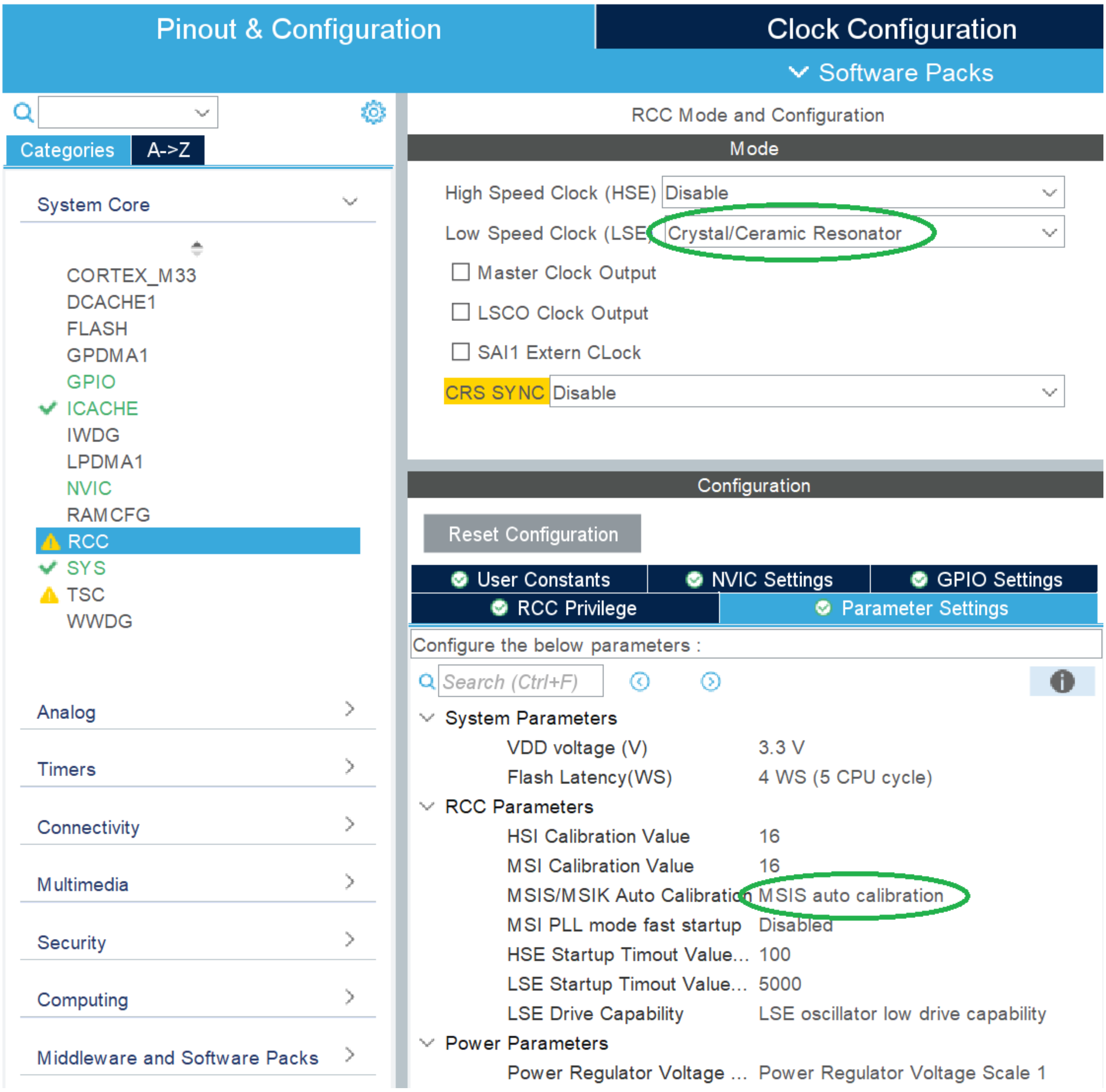
Human Machine Interface

□ USER LED GREEN (LD1)
□ USER LED BLUE (LD2)
□ USER LED RED (LD3)
□ USER BUTTON
□ Virtual Com Port

2.2. MCU clock setup

To obtain the accurate clock MCU frequency on Nulcleo-U575 board in the absence of high-frequency crystal, it is necessary to synchronize the internal low-precision MSIS clock generator to low-frequency generator using crystal. To achieve this:

- In Pinout & Configuration tab left pane, click System – RCC then in the middle-upper pane set Low Speed Clock to Crystal/Ceramic Resonator.
- In middle-lower pane, Parameter Settings tab, set MSIS/MSIK Auto Calibration to MSIS auto calibration.
- Switch to main Clock Configuration tab and make the changes highlighted in the picture below: select MSIS as PLL1 source and enter 160 MHz as HCLK value. After pressing Enter, confirm automatic clock tree setup. All the primary peripheral clocks should be automatically set to 160 MHz.



RCC Mode and Configuration

Mode

High Speed Clock (HSE) Disable

Low Speed Clock (LSE) Crystal/Ceramic Resonator

□ Master Clock Output

□ LSCO Clock Output

□ SAI1 Extern CLock

CRS SYNC Disable

Configuration

Configure the below parameters :

System Parameters

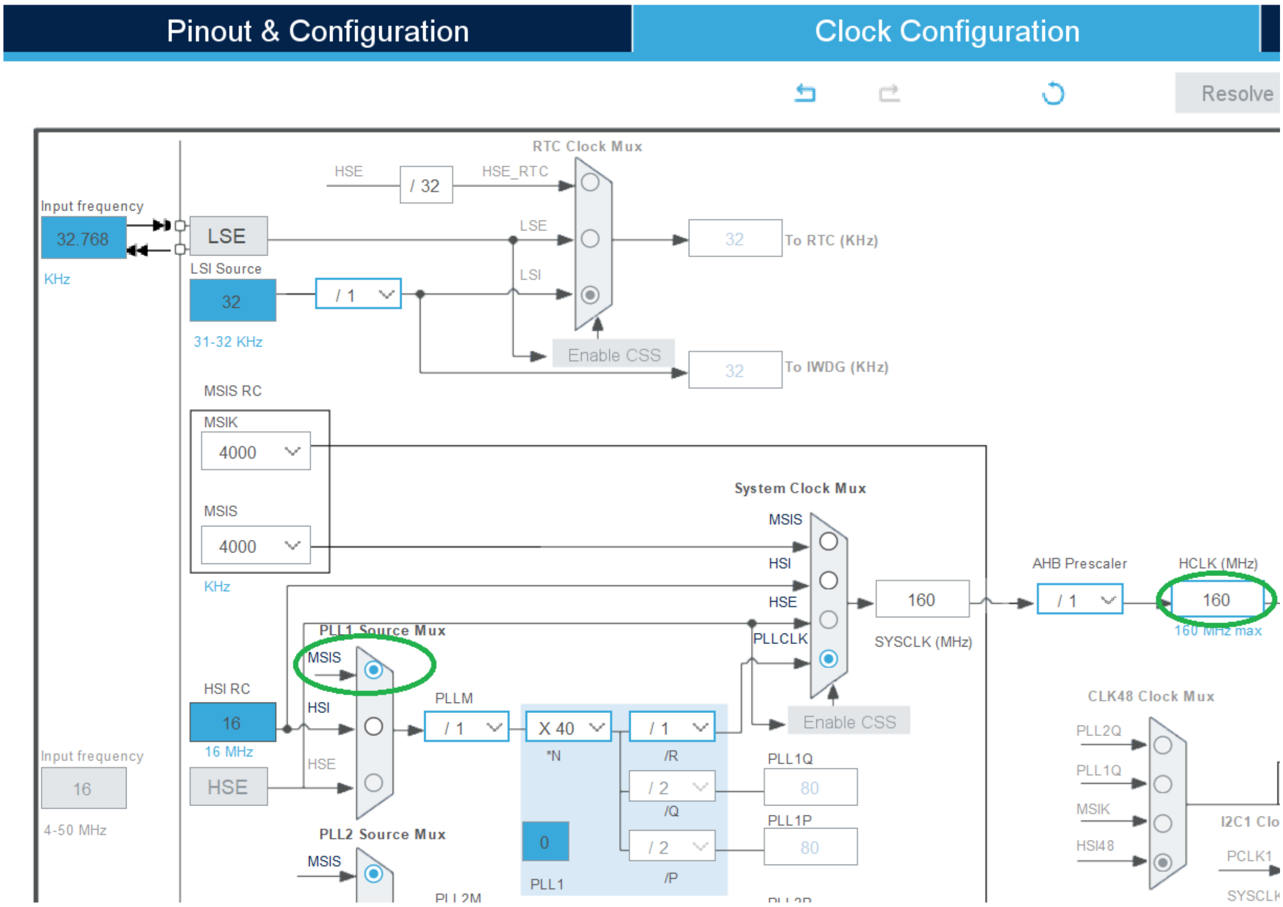
VDD voltage (V)	3.3 V
Flash Latency(WS)	4 WS (5 CPU cycle)

RCC Parameters

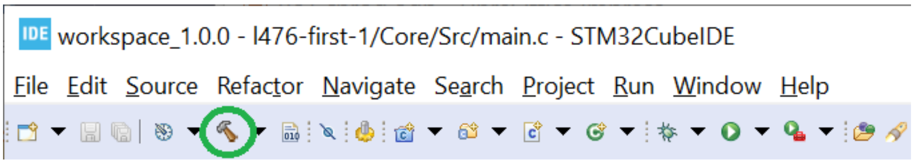
HSI Calibration Value	16
MSI Calibration Value	16
MSIS/MSIK Auto Calibration	MSIS auto calibration
MSI PLL mode fast startup	Disabled
HSE Startup Timout Value...	100
LSE Startup Timout Value...	5000
LSE Drive Capability	LSE oscillator low drive capability

Power Parameters

Power Regulator Voltage ...	Power Regulator Voltage Scale 1



- Save the configuration by clicking the Save icon in the main toolbar, confirm the code generation and perspective change.
- Verify that the project compiles successfully by clicking the Build icon (hammer symbol).



workspace_1.0.0 - l476-first-1/Core/Src/main.c - STM32CubeIDE

File Edit Source Refactor Navigate Search Project Run Window Help

- Analyze the content of files in Core folder: *Src/main.c*, *Src/stm32l4xx_it.c*, *Inc/main.h*, *Startup/startup_stm32l4???.s*
- View the linker script file in the main folder of the project.
- Check the memory usage in the lower right pane of CubeIDE window.

2.3. Adding custom code to CubeMX-generated source files

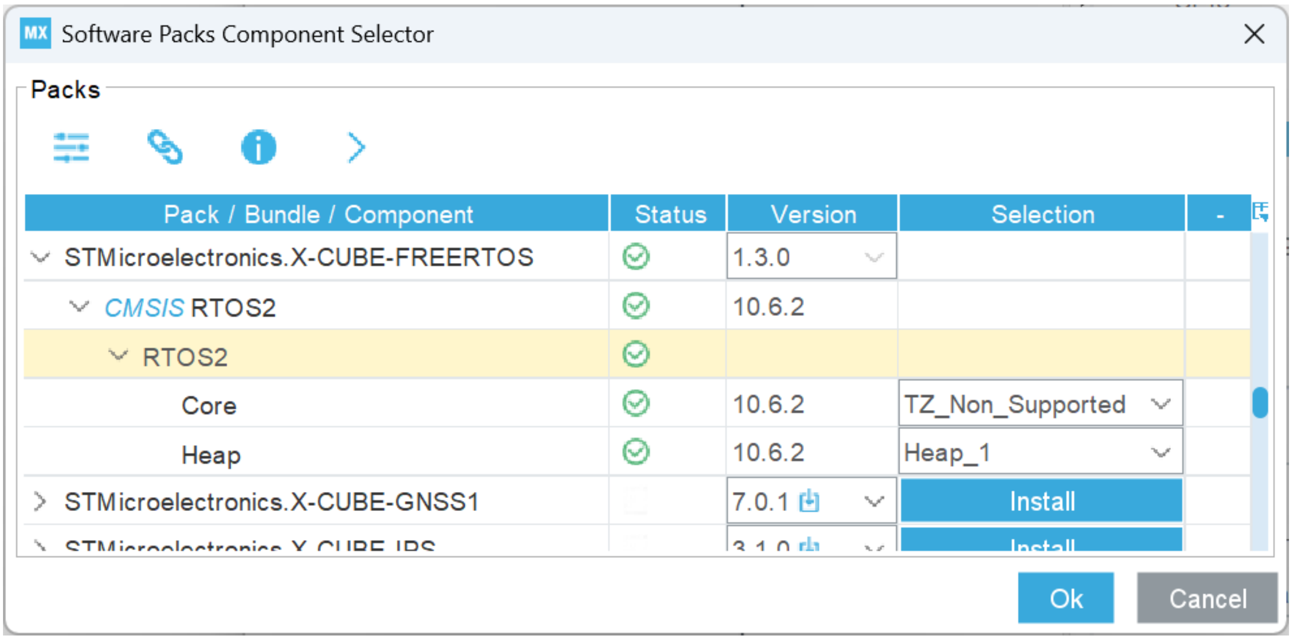
User may edit the files generated by CubeMX in Core folder by adding custom code. The user code may be added anywhere between `/* USER CODE BEGIN … */` and `/* USER CODE END … */` comments created by CubeMX. The code will be preserved when the application skeleton code is re-generated after configuration changes.

The CubeMX-generated code relies on ST HAL library, added as source code to the generated project. HAL uses hardware timer, by default configured to trigger interrupts at 1 kHz frequency. By default, if no RTOS is used in the project, ARM core timer SysTick is used for this purpose.

3. FreeRTOS-based project

Steps to create the FreeRTOS-based application:

- Open the project's .ioc file created in the previous step. In Pinout & Configuration tab left pane, click Middleware and Software Packs, then select X-CUBE-FREERTOS. This will open the pack installer.
- In the Software Packs Component Selector window, select STMicroelectronics – CMSIS RTOS2. Set Core option to TZ_NotSupported, Heap to Heap_1, then close the window.



Pack / Bundle / Component	Status	Version	Selection	-
STMicroelectronics.X-CUBE-FREERTOS	✓	1.3.0		
CMSIS RTOS2	✓	10.6.2		
RTOS2	✓			
Core	✓	10.6.2	TZ_Non_Supported	
Heap	✓	10.6.2	Heap_1	
STMicroelectronics.X-CUBE-GNSS1		7.0.1	Install	
STMicroelectronics.X-CUBE-IPS		3.1.0	Install	

- In CubeMX Pinout & Configuration tab left panel, select X-CUBE-FREERTOS. Tick the CMSIS RTOS2 option.
- Generate the application code by clicking either Save or Generate code icon.
- Open the generated *Core/Src/app_freertos.c* file and navigate to StartDefaultTask() routine.
- In the routine's infinite loop, add the following code (names of LED constants may be copied from main.h file):

```
osDelay(1000);
HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
```

- Compile the program by clicking the Build icon.

3.1. Flashing and running the program

To run the program:

- Connect the Nucleo board to PC using a USB cable with micro-B plug (top connector);
- Click the Run icon in the toolbar. When done for the first time, a debug settings dialog will open. Confirm the default settings by clicking OK button.

- When connected for the first time or after CubeIDE update, ST-Link update message may appear. In such a case, perform firmware update, then close the updater window and click Run again.
- While downloading the firmware to Nucleo board, watch the messages displayed in the bottom pane. After successful flashing you should see the following message:

```
Verifying ...

Download verified successfully

Shutting down...
Exit.
```

- If the MCU cannot be programmed, try to adjust debug settings. From Run menu select Run Configurations..., then select the project name from the pulldown list; in Debugger tab change Reset behaviour to None or Software System Reset. Additionally, you may try to set the communication frequency to no more than 4 MHz.
- After flashing, the program starts automatically. Check the operation – the red user LED should toggle every second.

3.2. Avoiding dynamic memory allocation

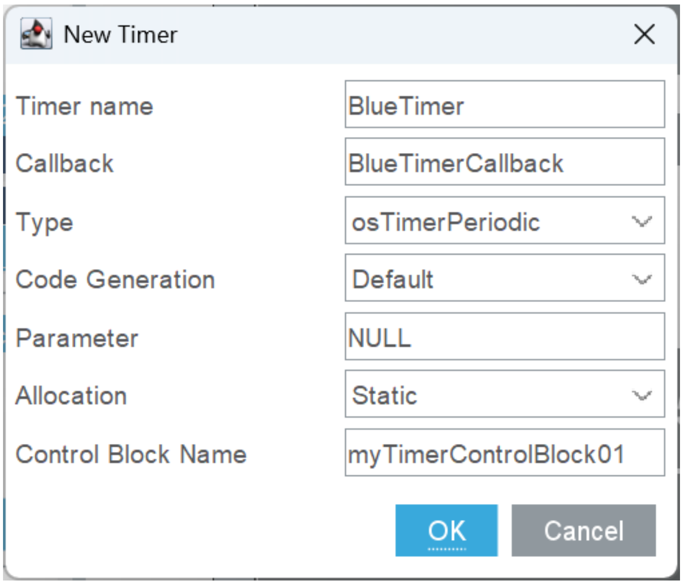
By default, all the FreeRTOS objects created by CubeMX use dynamic memorz allocation – the memory used for thread stacks, timer data, queues and semaphores is allocated dynamically during the object creation. Dynamic allocation has two disadvantages – it increases the memory footprint and makes it harder to monitor the memory usage and detect memory overflows. It is however possible to allocate this data statically. To do this, in project configuration view, select X-CUBE_FREERTOS, then in the middde bottom pane select CMSIS RTOS2 tab. To change the memory allocation of the default task, select Tasks and Queues tab, click on Default task entry and change the Allocation option to Static. The same may (and should) be done during the creation of every new object.

3.3. Using RTOS timers

RTOS timer is a mechanism for calling application-defined routine at a predefined time, once or periodically.

To create a timer:

- Open the project's .ioc file in CubeMX.
- In the right pane, select X-NUCLEO-FREERTOS.
- In the middle lower pane (), select CMSIS RTOS2 tab, then select Timers and Semaphores subtab.
- In Timer panel, click Add.
- Enter a meaningful name for the timer and its callback routine, set Type to osTimerPeriodic and chage Allocation to Static.



New Timer

Timer name	BlueTimer
Callback	BlueTimerCallback
Type	osTimerPeriodic
Code Generation	Default
Parameter	NULL
Allocation	Static
Control Block Name	myTimerControlBlock01

- Generate the updated code.
- In main.h add the definition for blue LED toggle period.

```
#define BLUE_PERIOD   400u   // ms
```

- Open the app_freertos.c file.
- In MX_FREERTOS_Init() function, ddd timer startup call to RTOS_TIMERS section.

```
osTimerStart(BlueTimerHandle, BLUE_PERIOD);
```

- Add LED toggling code to the callback routine.

```
/* BlueTimerCallback function */
void BlueTimerCallback(void *argument)
{
  /* USER CODE BEGIN BlueTimerCallback */
  HAL_GPIO_TogglePin(LED_BLUE_GPIO_Port, LED_BLUE_Pin);
  /* USER CODE END BlueTimerCallback */
}
```

- Compile and flash the code.

3.4. Detecting the button press using RTOS timer

Because of mechanical bouncing, reliable detection of key press requires sampling the button state with period exceeding the maximum bounce time. For small pushbuttons, the maximum bounce time is specified in range of 10..15 ms. The button state may be sampled with 20 ms period. In RTOS this may be achieved by using `osDelayUntil()` or, easier, by using RTOS timer. Our program will change the state of an LED on every button press.

- Create another timer as described in the previous section. Set the memory allocation option to static and give it a meaningful name (like BtnTimer).
- In *main.h*, define the constant representing the button state test period in milliseconds.

```
#define BTN_TEST_PERIOD      20u    // ms
```

- Configure the button timer in FreeRTOS similarly to the LED blink timer from the previous sections.
- Generate the code.
- In *app_freertos.c*, add the timer startup code as previously.
- Edit the timer callback routine, adding the code below.

```
/* BtnTimerCallback function */
void BtnTimerCallback(void *argument)
{
  /* USER CODE BEGIN BtnTimerCallback */
  static bool btn_was_pressed;
  bool btn_is_pressed = USER_BUTTON_GPIO_Port->IDR & USER_BUTTON_Pin;
  if (!btn_was_pressed && btn_is_pressed)
  {
        HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
  }
  btn_was_pressed = btn_is_pressed;
  /* USER CODE END BtnTimerCallback */
}
```

- (To use `bool` type name in C dialect older than C23, it is necessary to include the standard header file *stdbool.h* – it is convenient to include it via *Inc/main.h*.)
- Compile and test the code.

4. Using Queues and semaphores

In the next phase, we will implement a simple model of an RTOS-based data processing application. The application will communicate with a PC via the microcontrolers serial port connected to ST-Link VCP (Virtual Communication Port) UART and visible as VCP on the PC side. There is no need to create the new project – we may add the new functionality to the existing LED & button demo.

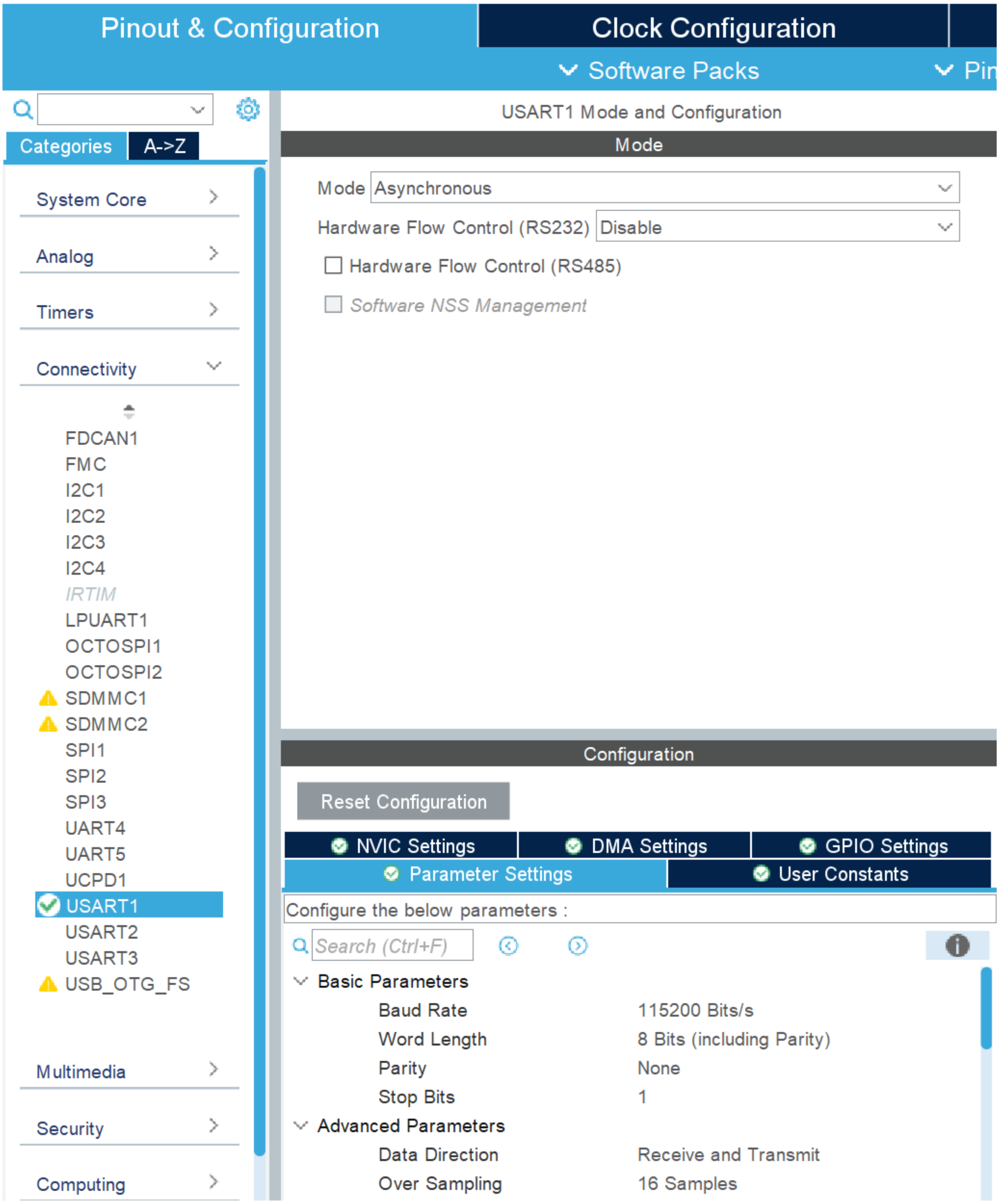
We will start by verifying the VCP connection:

- Open project configuration file in CubeMX, select Pinout & Configuration tab.
- Select Connectivity – USART1. Set Mode to Asynchronous.
- Check the configuration of USART1 – Parameter Settings tab in middle-lower pane. It should be configured to 115200 b/s baud rate, 8 date bits, no parity. Fix the default baud rate if needed.
- Select the NVIC Settings tab and enable global UART interrupt; this will be needed later.
- Generate the updated project code.
- Open the *Core/Src/app_freertos.c* and edit the `DefaultTask` routine. Below the line toggling the LED state, temporarily add the line:

```
USART1->TDR = '*';
```

- Compile and run the updated code.
- Test the operation using *TeraTerm*. Open *TeraTerm*; if initial dialog is displayed while opening, select Serial connection and close the dialog. Use menu option Setup-Serial port); from COM port list select the port described as ST Microelectronics ST-Link. Set connection parameters as in CubeMX USART1 configuration – 115200, 8N1. Leave the default values of other parameters of *TeraTerm* connection.
- Close the dialog. The transmitted characters should be in terminal window.
- After verifying the connection, remove the line of code sending the test characters.
- Put the extern declaration of huart1 structure in *main.h* (copy the definition from *main.c*, adding the `extern` keyword). Since there is no matching section in *main.h*. put it at the end of user private defines section.

```
extern UART_HandleTypeDef huart1;
```



USART1 Mode and Configuration

Mode

Mode Asynchronous

Hardware Flow Control (RS232) Disable

□ Hardware Flow Control (RS485)

□ *Software NSS Management*

Configuration

Configure the below parameters :

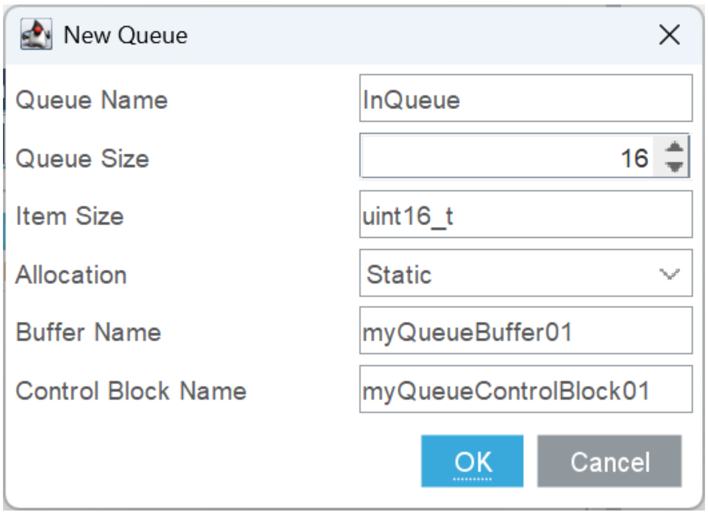
Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples

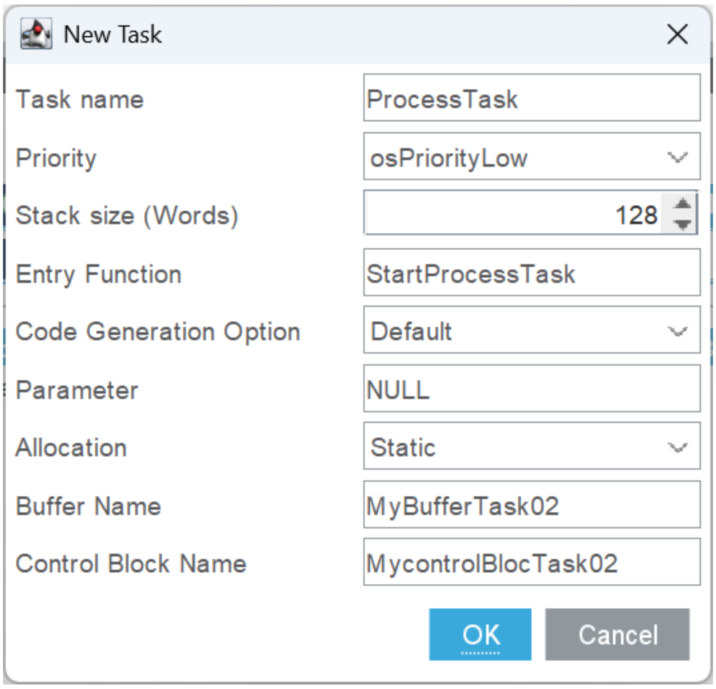
Create two queues with (mostly) default parameters – InQueue and OutQueue.



New Queue

Queue Name	InQueue
Queue Size	16
Item Size	uint16_t
Allocation	Static
Buffer Name	myQueueBuffer01
Control Block Name	myQueueControlBlock01

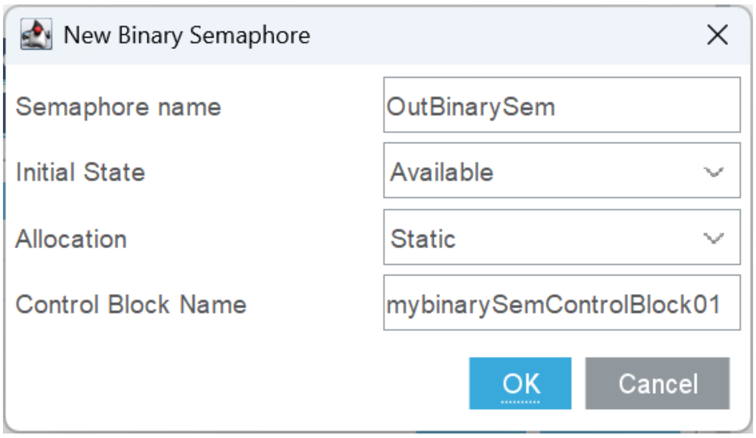
Create two tasks, named ProcessTask and OutputTask.



New Task

Task name	ProcessTask
Priority	osPriorityLow
Stack size (Words)	128
Entry Function	StartProcessTask
Code Generation Option	Default
Parameter	NULL
Allocation	Static
Buffer Name	MyBufferTask02
Control Block Name	MycontrolBlocTask02

Create a binary semaphore named OutBinarySem.



Semaphore name: OutBinarySem
Initial State: Available
Allocation: Static
Control Block Name: mybinarySemControlBlock01

Generate the code, then edit the *app_freertos.c*, adding the following fragments:

- define a byte variable `inchar`:

```
/* USER CODE BEGIN Variables */
static uint8_t inchar;
```

- edit the StartProcessTask function:

```
void StartProcessTask(void *argument)
{
  /* USER CODE BEGIN ProcessTask */
  HAL_UART_Receive_IT(&huart1, &inchar, 1);
  /* Infinite loop */
  for(;;)
  {
    uint16_t c;
    osMessageQueueGet(InQueueHandle, &c, 0, osWaitForever);
    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))
          c ^= 'a' ^ 'A';
    osMessageQueuePut(OutQueueHandle, &c, 0, osWaitForever);
  }
  /* USER CODE END ProcessTask */
}
```

- edit the StartOutputTask function:

```
void StartOutputTask(void *argument)
{
  /* USER CODE BEGIN OutputTask */
  /* Infinite loop */
  for(;;)
  {
    uint16_t c;
    osMessageQueueGet(OutQueueHandle, &c, 0, osWaitForever);
    osSemaphoreAcquire(OutBinarySemHandle, osWaitForever);
    HAL_UART_Transmit_IT(&huart1, (uint8_t *)&c, 1);
  }
  /* USER CODE END OutputTask */
}
```

- at the end of file, add the code for UART transfer interaction with RTOS objects:

```
/* USER CODE BEGIN Application */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
  osMessageQueuePut(InQueueHandle, &inchar, 0, 0);
  HAL_UART_Receive_IT(&huart1, &inchar, 1);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
{
  osSemaphoreRelease(OutBinarySemHandle);
}

/* USER CODE END Application */
```