Morse Micro
reaching farther™

MM-IoT CMSIS-Pack

USER GUIDE

Table of Contents

1 Introduction

This user guide demonstrates how to get started with embedded development using the MM-IoT CMSIS-pack.

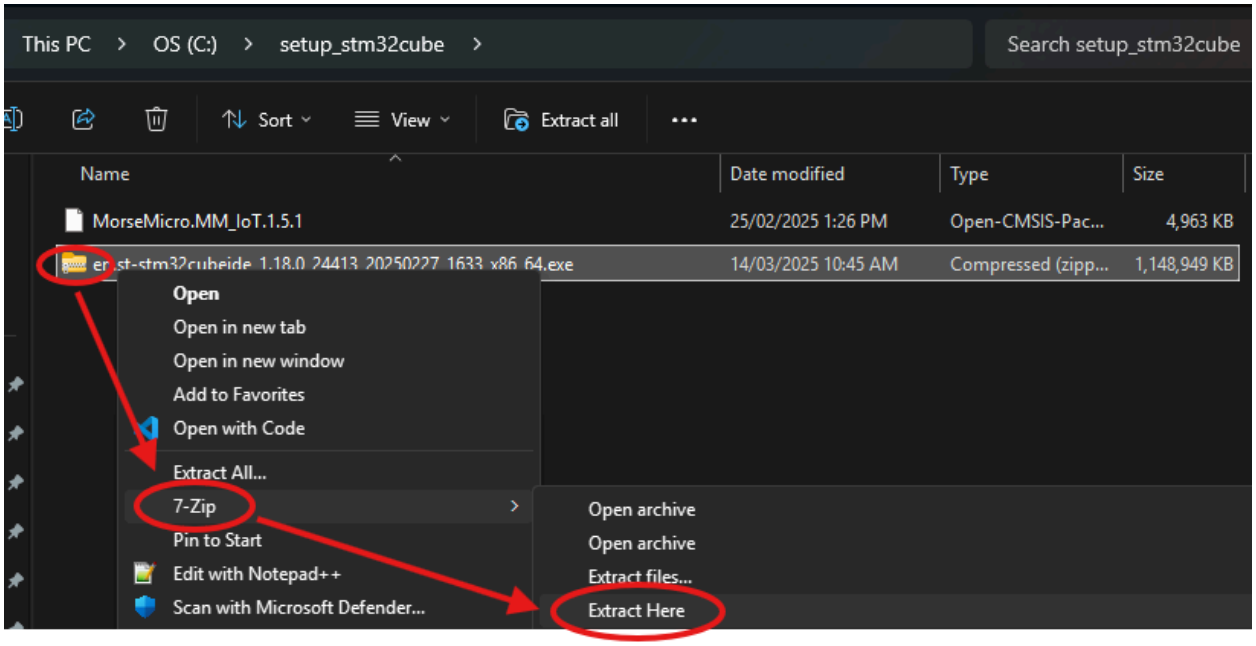
A CMSIS-Pack is a way to add support for software components, device parameters, and EVB support that can be used in ARM projects across a range of IDEs. In this document STM32CubeIDE is the focus, and CMSIS pack is used to add support for HaLow in Cube IDE based projects.

MM-IoT CMSIS-Pack a rebundled version of the Morse Micro IoT-SDK for embedded devices. Further details of the APIs and examples contained in the IoT-SDK can be found at (https://github.com/MorseMicro/mm-iot-sdk).

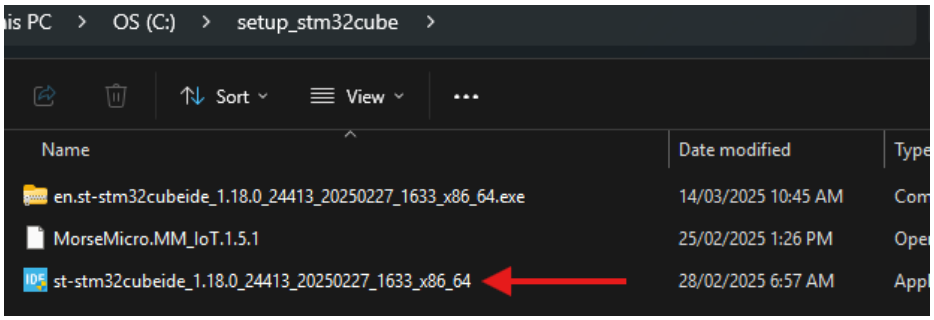
Note well: This guide has been written for STM32CubeIDE v 1.18.0. From STM32Cube v2.0.0 onwards, ST have changed their tooling to separate the IDE from the STM32CubeMX code generation tool. The steps in this guide still apply, but you should perform the CMSIS-pack installation, and code generation / IOC file editing with the STM32CubeMX tool directly (instead of having it bundled into the IDE).

2 Installing STM32CubeIDE

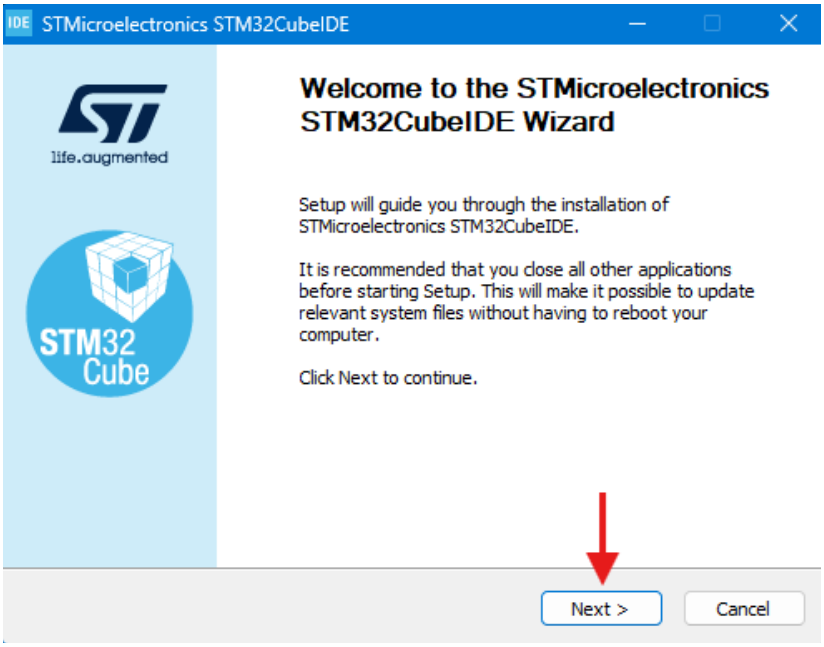
First, install STM32Cube IDE on your machine. To do so, you need to download the latest version from https://www.st.com/en/development-tools/stm32cubeide.html which results in a zip file. At the time of writing this document, v1.18.0 was the latest version. First unzip the downloaded file using any compression tool:

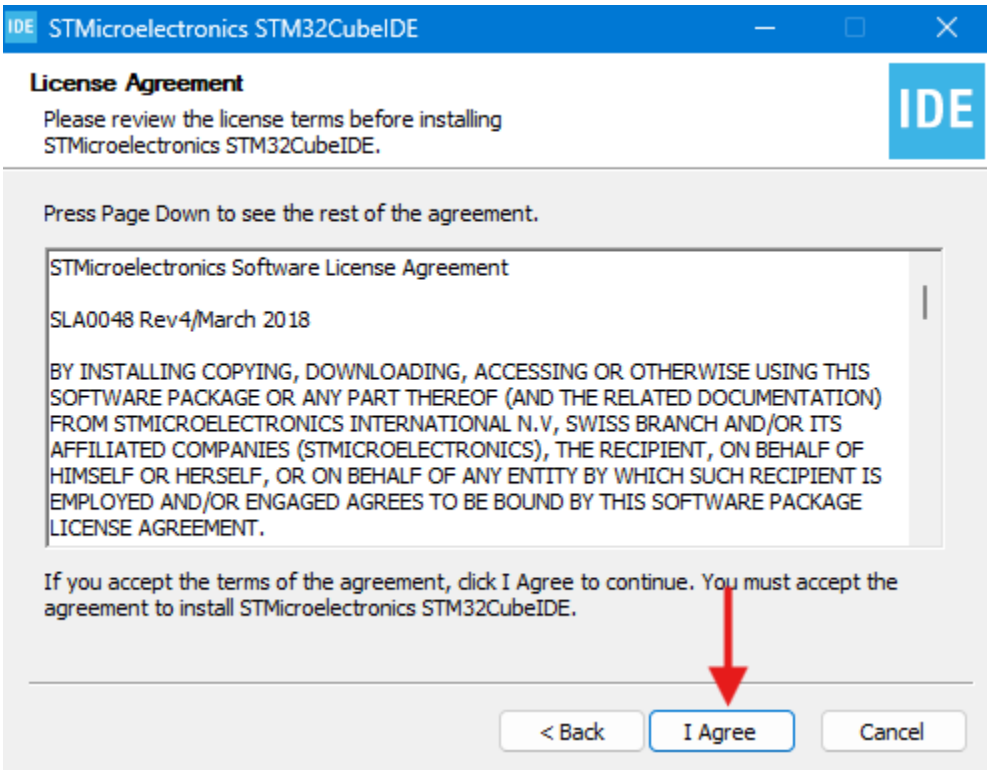


After decompression, You will obtain the installation file:

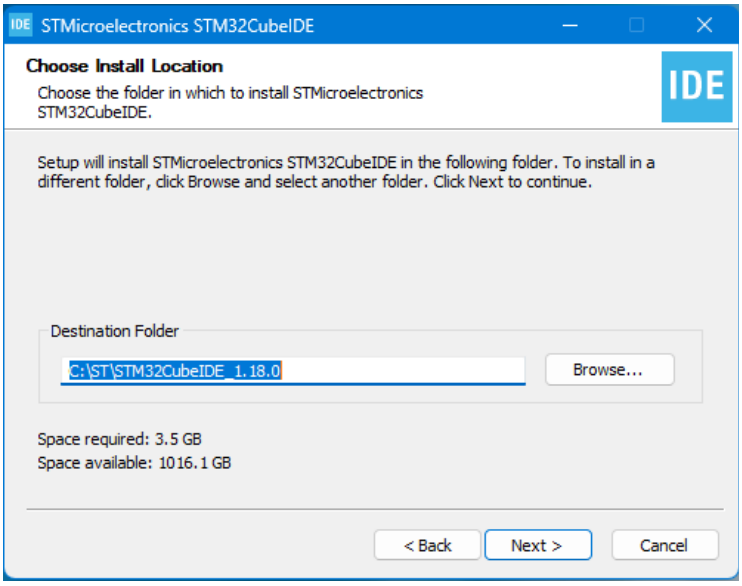


Double-click the installation file to open the setup and follow the steps to install STM32CubeIDE:

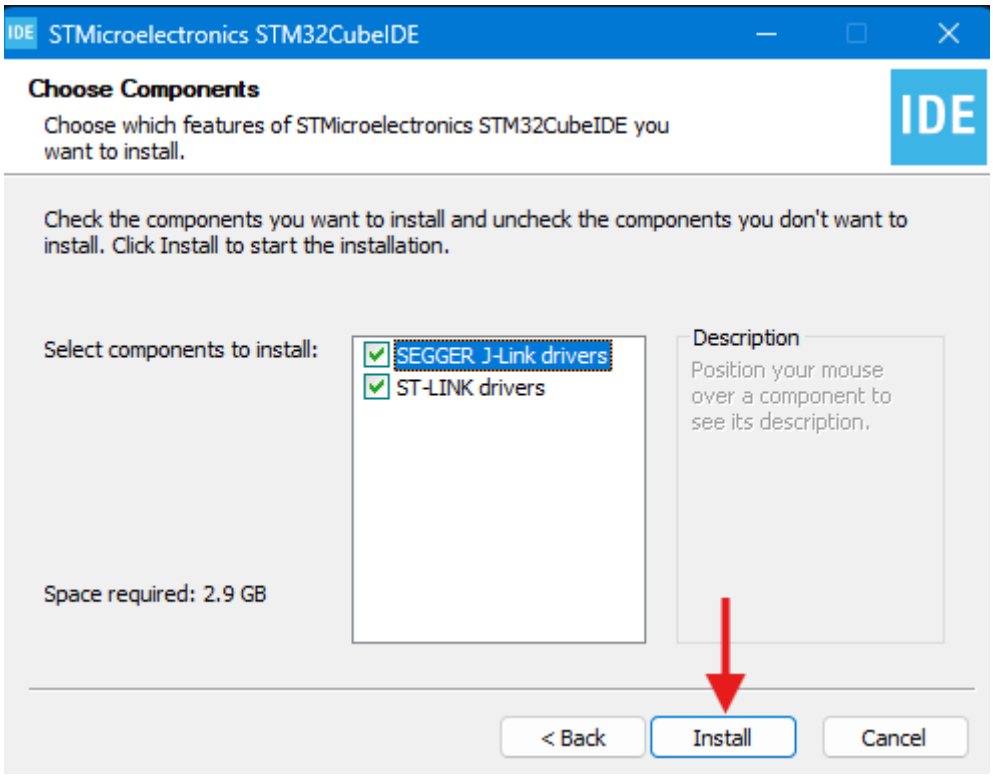




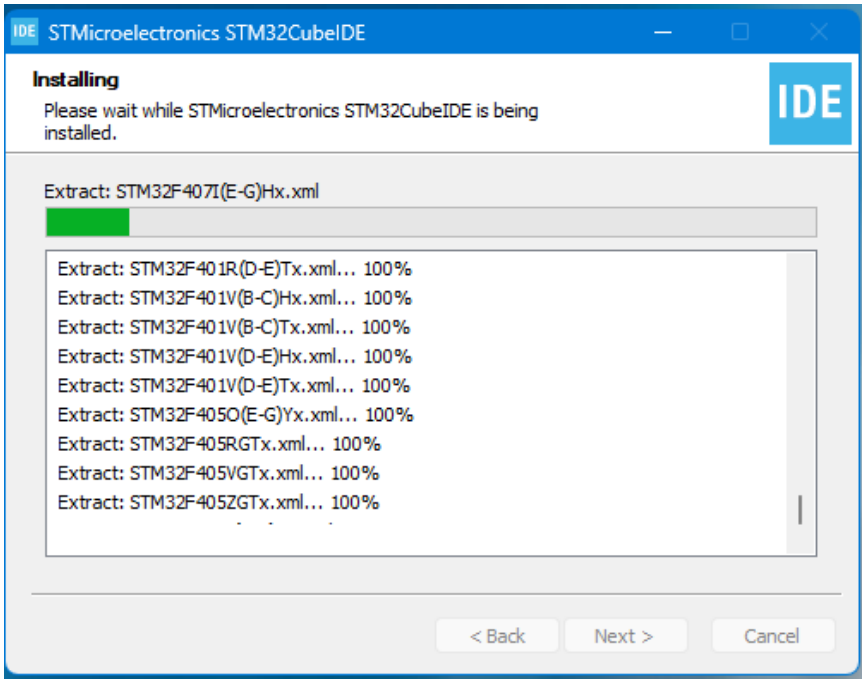
Select any location to install and click Next(preferably leave it at default path) :



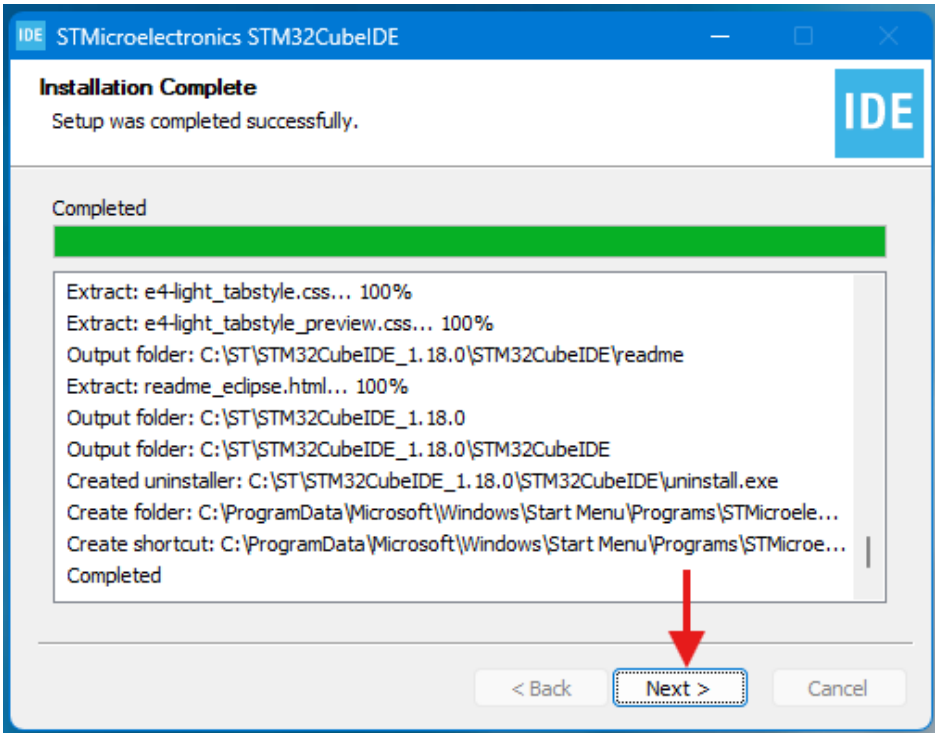
Leave both components checked and click Install:



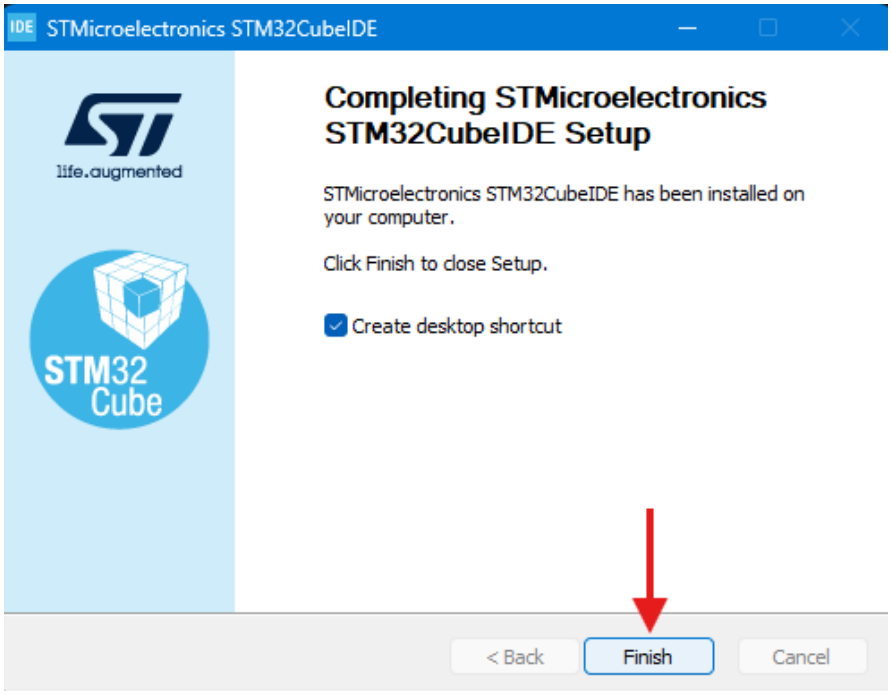
Wait for the installation to finish:



After the installation finishes, click on Next:



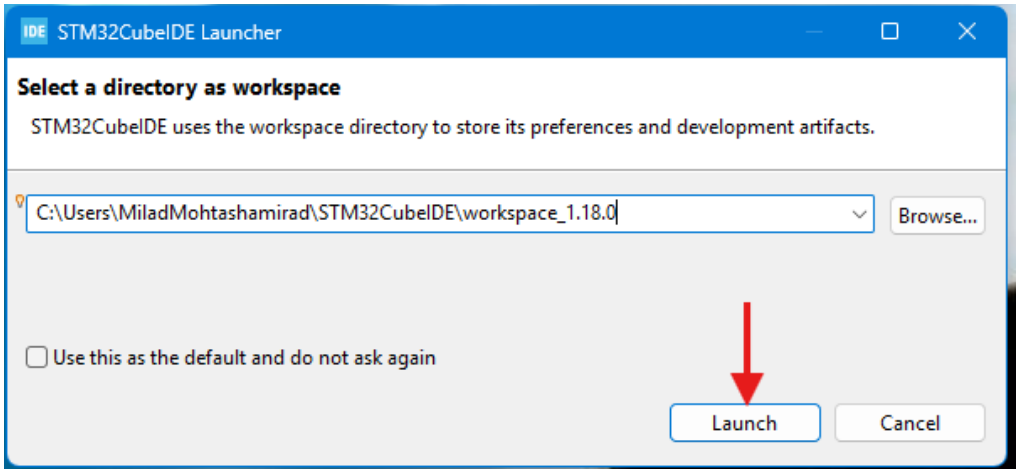
And click on Finish to close the installation window:



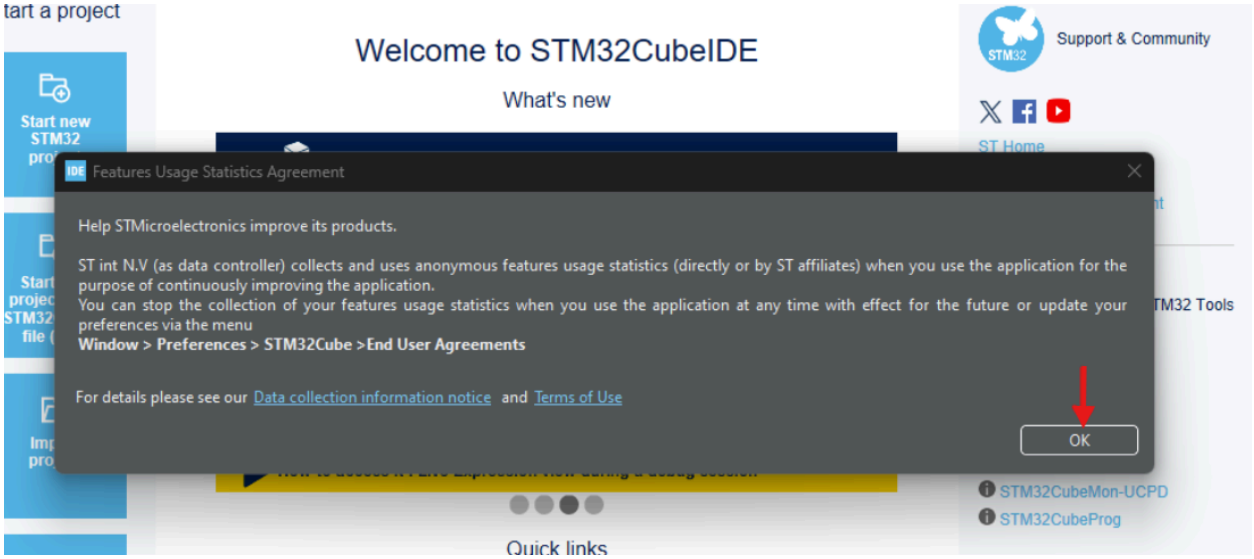
3 Starting STM32CubeIDE

Open the STM32CubeIDE by double clicking on its icon:

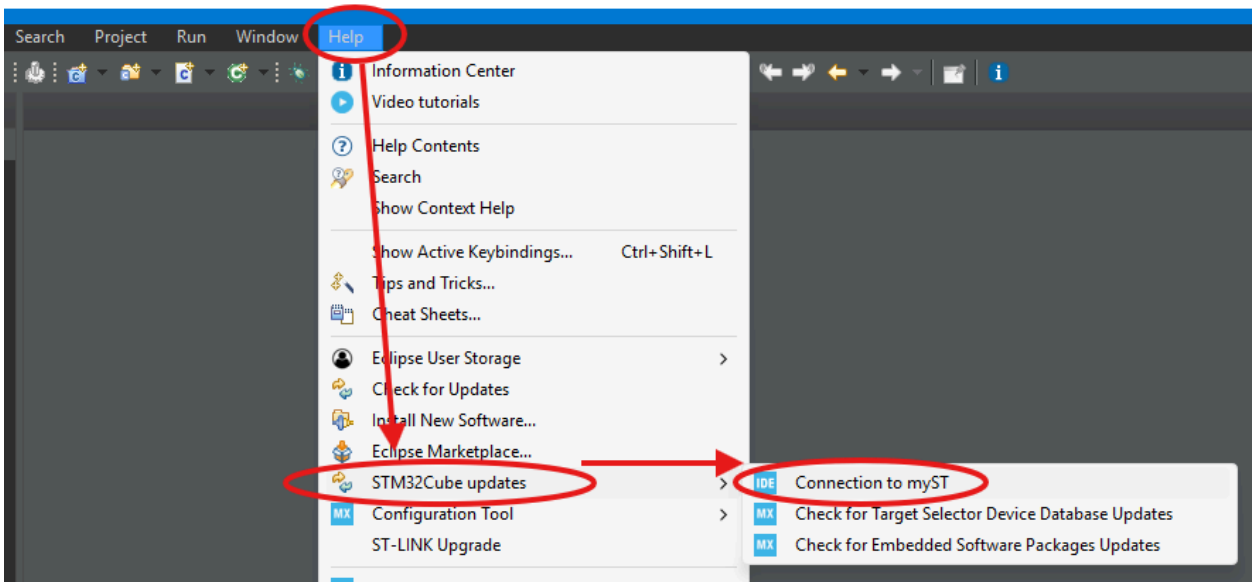
A dialog will open to ask for the workspace directory. Leave the default value and click Launch:



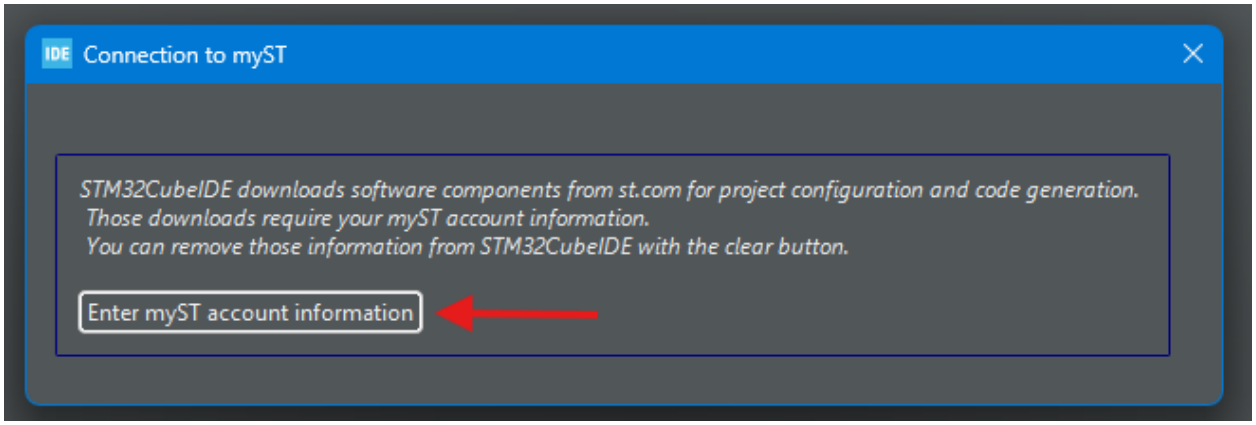
After loading, it might show you the Features Usage Statistics Agreement. Click OK to close it:



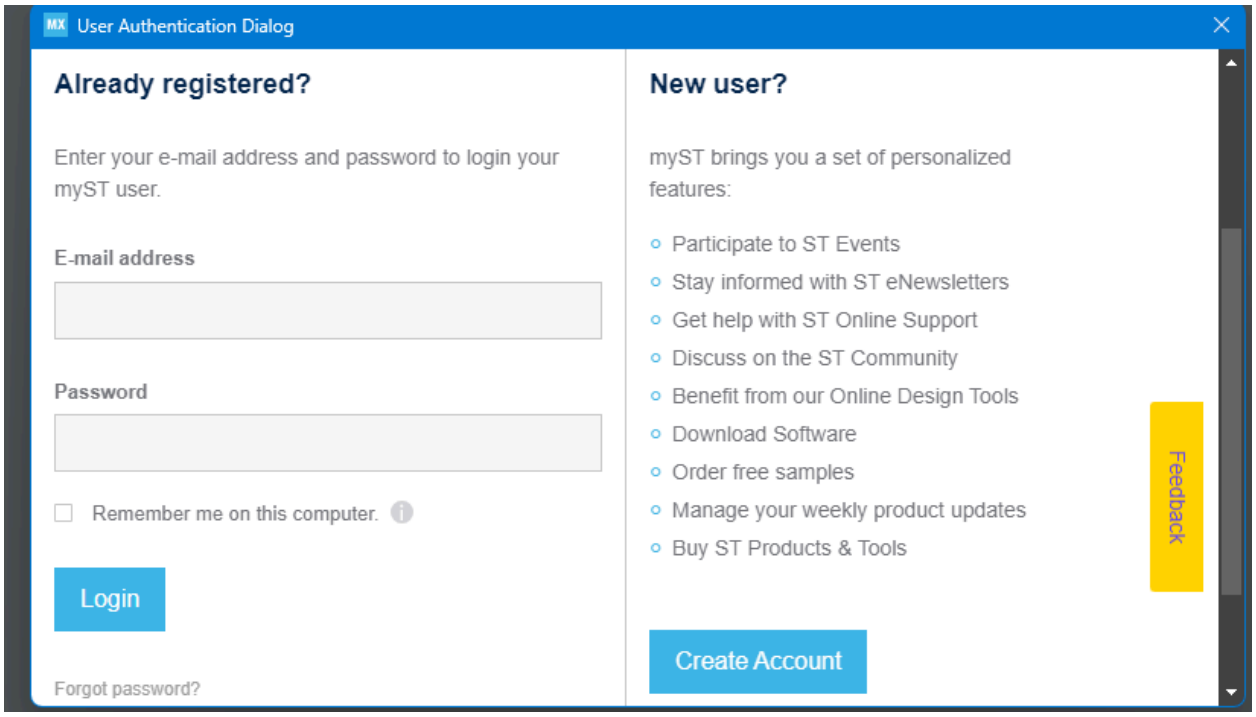
NOTE: You need to login with your account on STM32CubeIDE to be able to download the required packages for your projects. If you haven't already logged in, from **Help -> STM32Cube Updates** select **Connection to myST** :



And click **Enter myST account information** on the following dialog:



Now you need to enter your Email and Password to login:

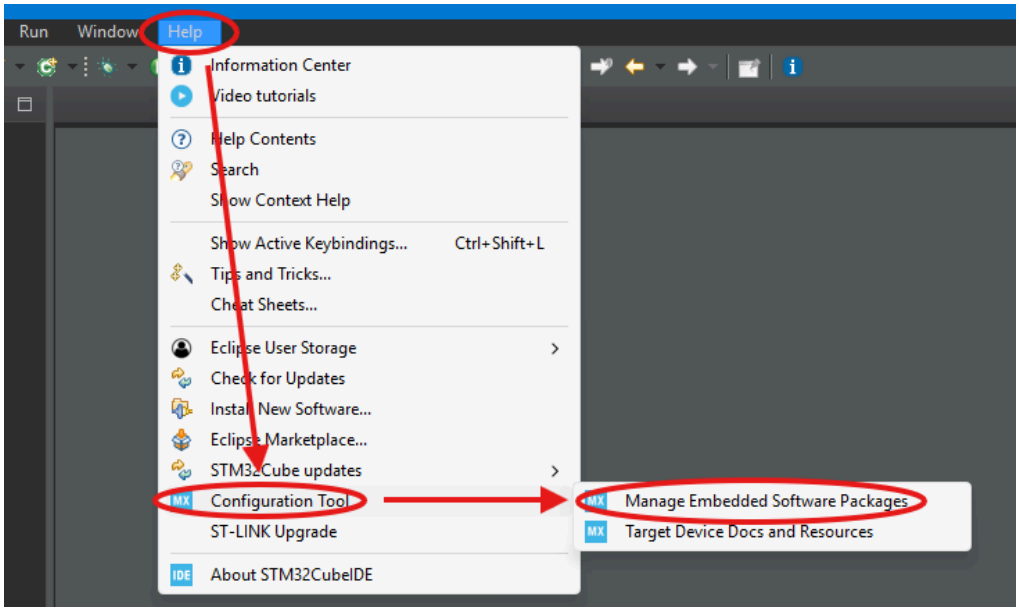


If your credentials are correct, the dialog will close and your account will stay active on STM32CubeIDE until you remove your account from it.

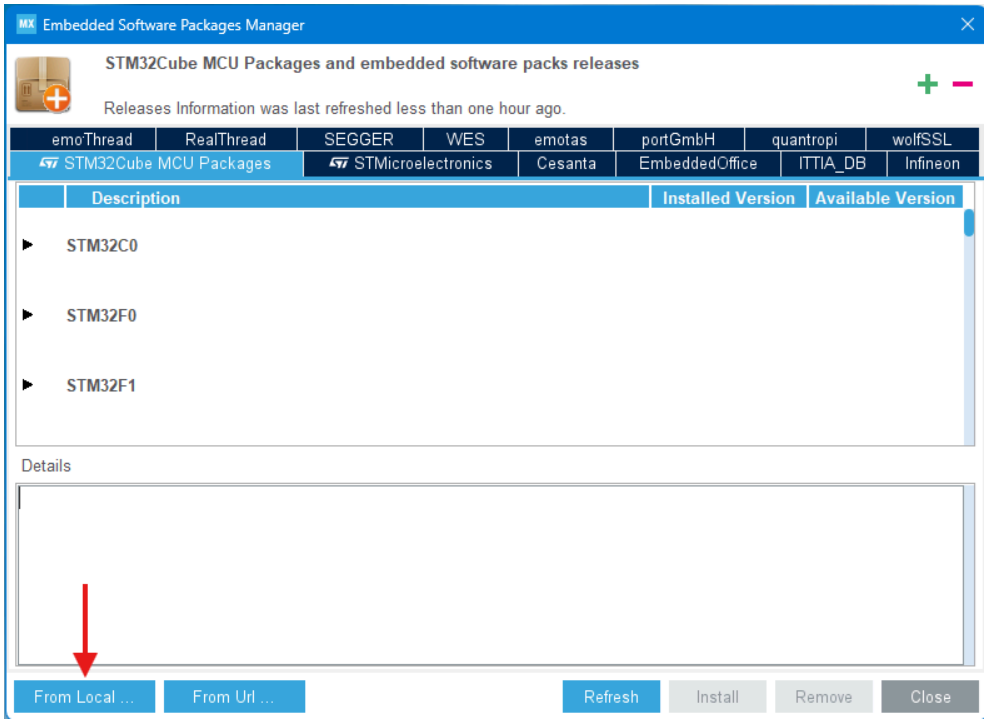
4 Installing Morse Micro CMSIS-Pack

If you haven't already downloaded Morse Micro CMSIS-Pack, you can download it from: https://github.com/MorseMicro/mm-iot-cmsis/releases

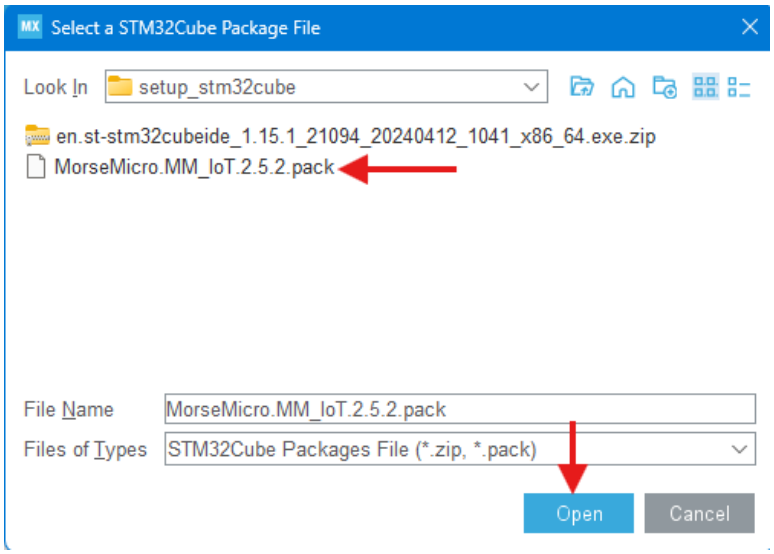
After Starting STM32CubeIDE, From Help menu, click on Manage Embedded Software Packages :



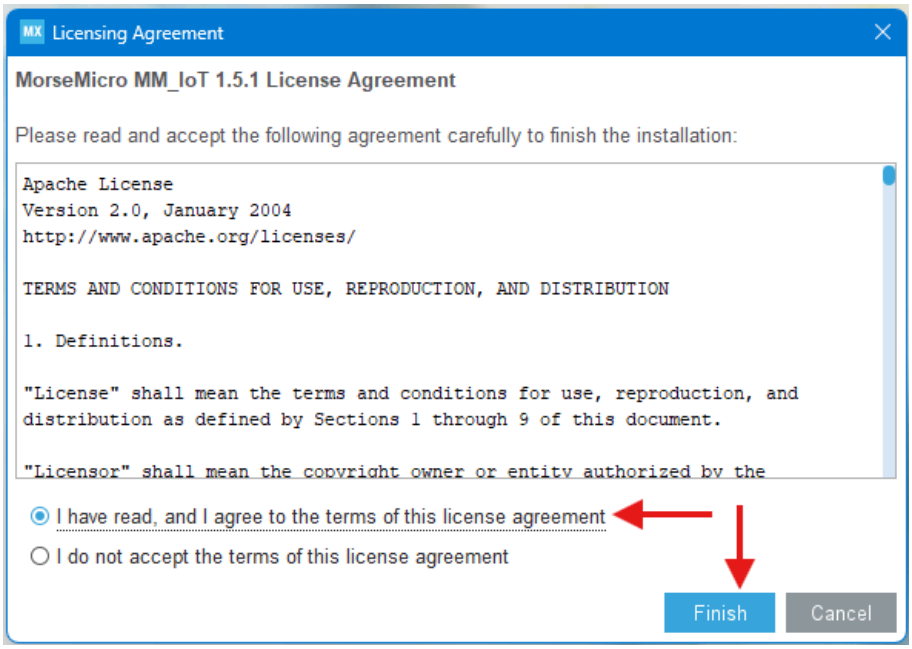
From the new window, click From Local … button:



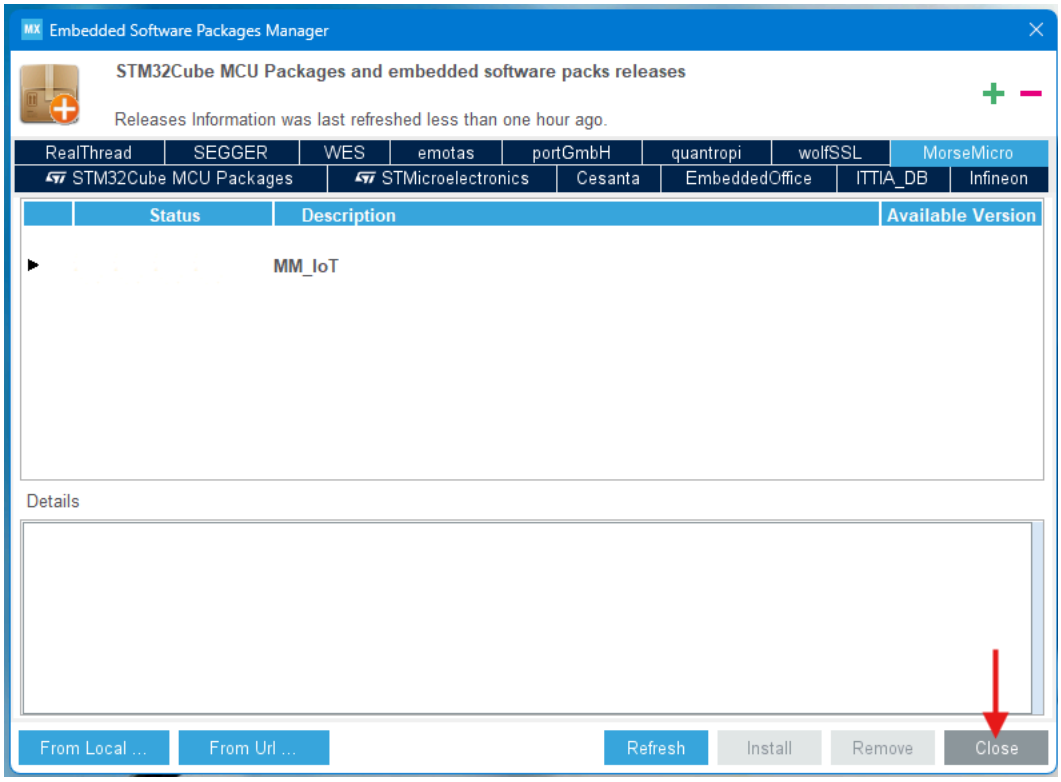
Navigate to the place that Morse Micro CMSIS-Pack is downloaded and select the package and click Open:



Read the License Agreement and select “I have read, and I agree …” and click Finish:

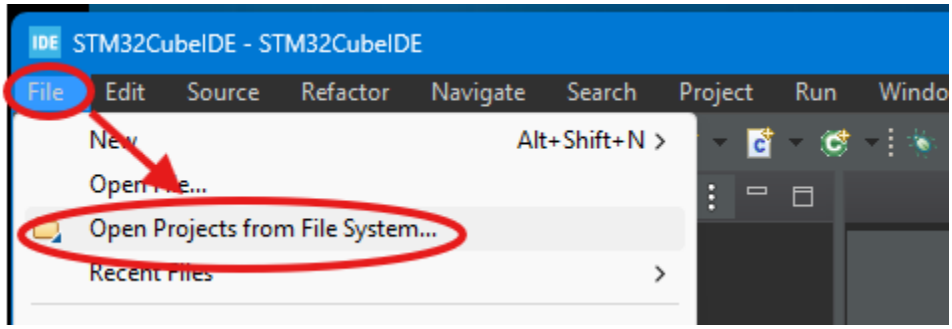


Now you should be able to see the MM_IoT package name. Under MorseMicro tab. Click Close to close package manager:

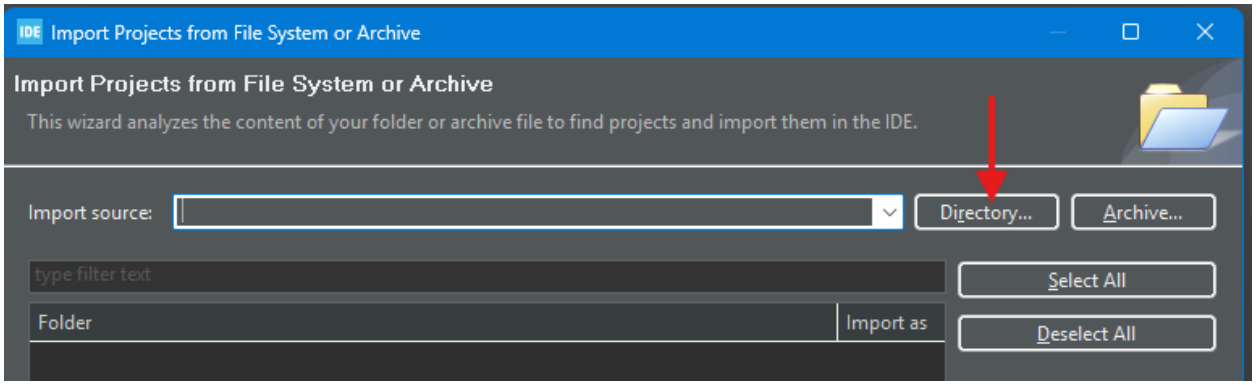


5 Trying HaLow_example in STM32CubeIDE

After installing Morse Micro CMSIS-Pack, you can open the example that is shipped with the pack. To open the example, from File menu, click on Open Projects from File System...:



From the Import Project windows, click on Directories:

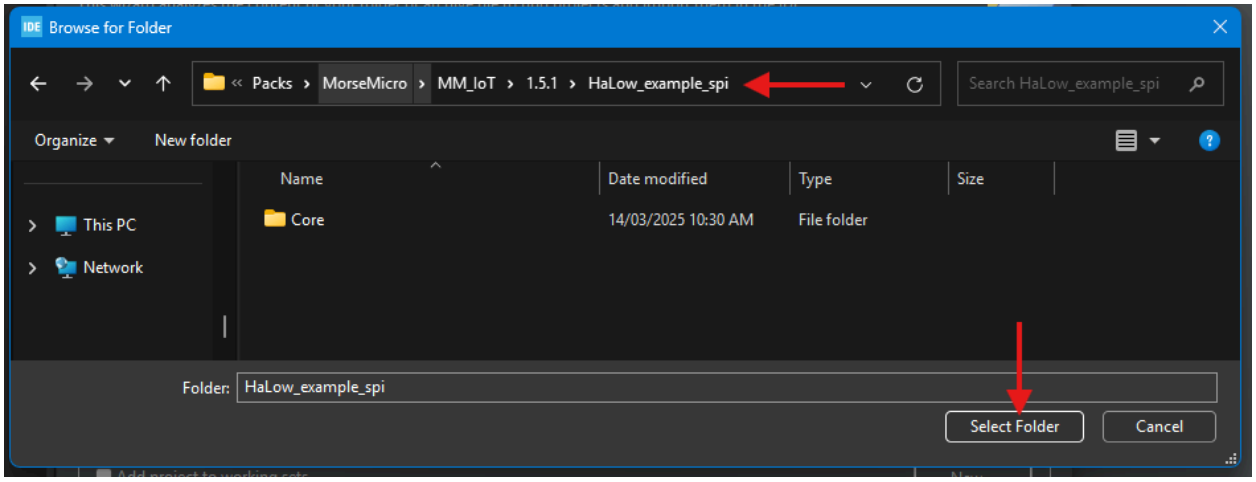


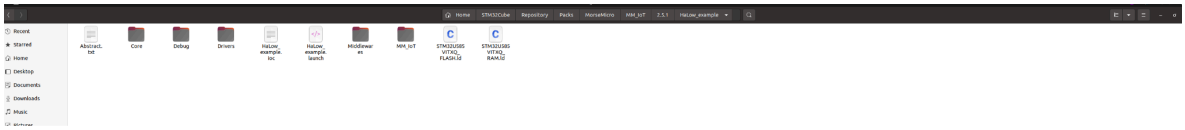
And Navigate to the place that the STM32 packages are installed into. In windows it would be

C:\Users\USER_NAME\STM32Cube\Repository\Packs\MorseMicro\MM_IoT\2.5.0\HaLow_example_spi

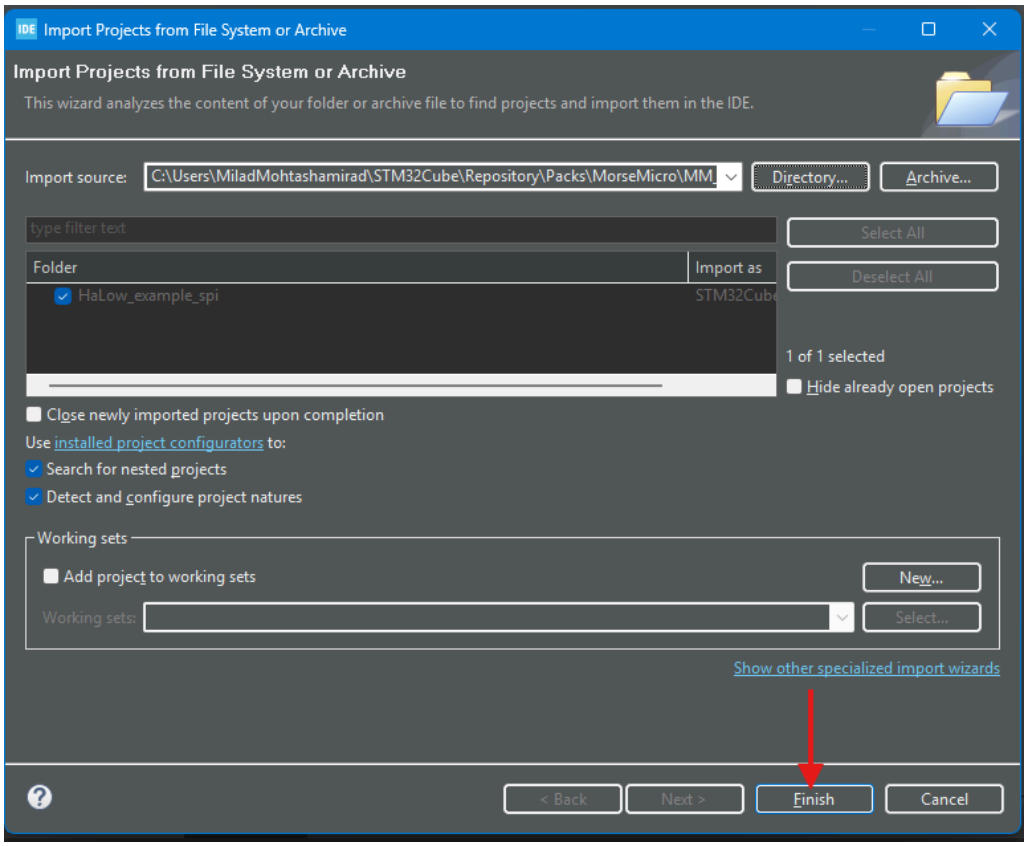
/home/USER_NAME/STM32Cube/Repository/Packs/MorseMicro/MM_IoT/2.5.1/HaLow_example_spi

(Or HaLow_example_sdio depending on your EKH05 resistor configuration) And click on Select Folder:

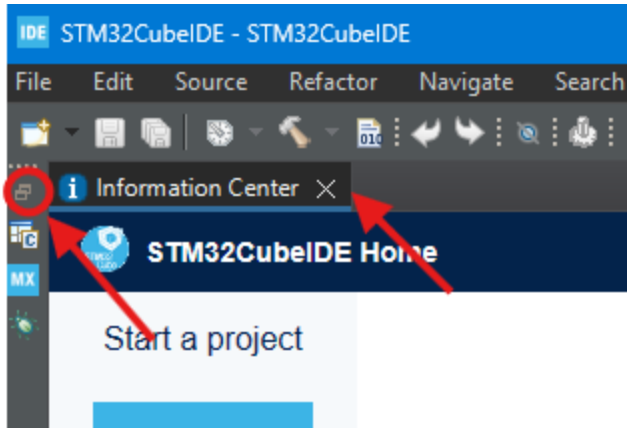




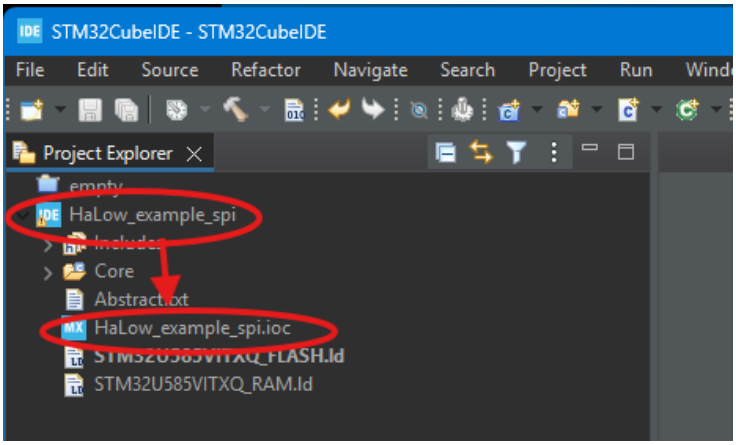
Then select Finish:



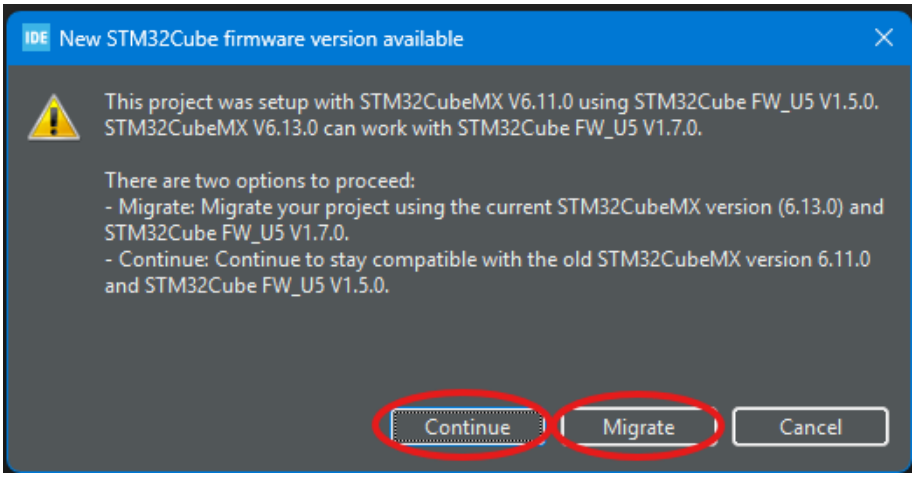
In order to see the project file tree, close the Information Center tab and click on restore icon on the left toolbar:



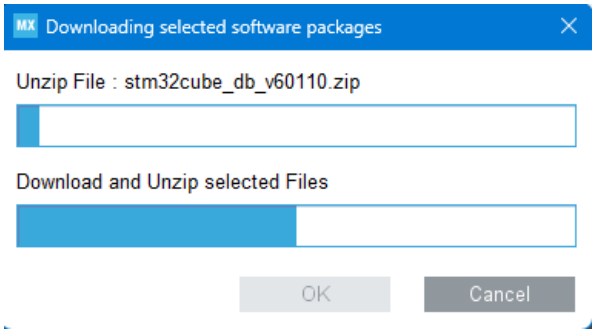
Expand Halow_example_spi project and double click on HaLow_example_spi.ioc to open cube configuration for the example project:



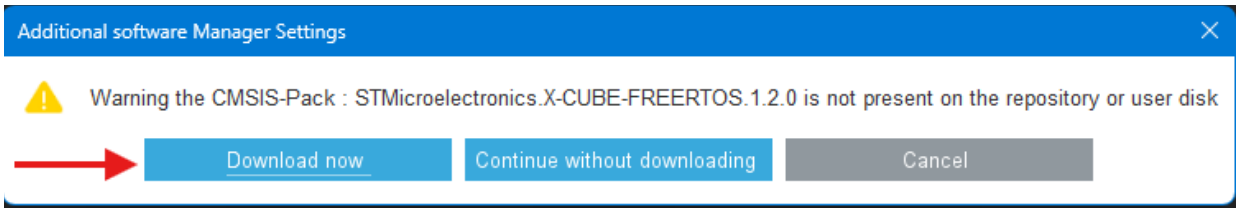
Normally you'll encounter a warning about the old version of the ioc file. You can choose either Continue or Migrate:



If it's the first time that you are opening this project STM32CubeMx will start to download some packages that are needed for the example project:

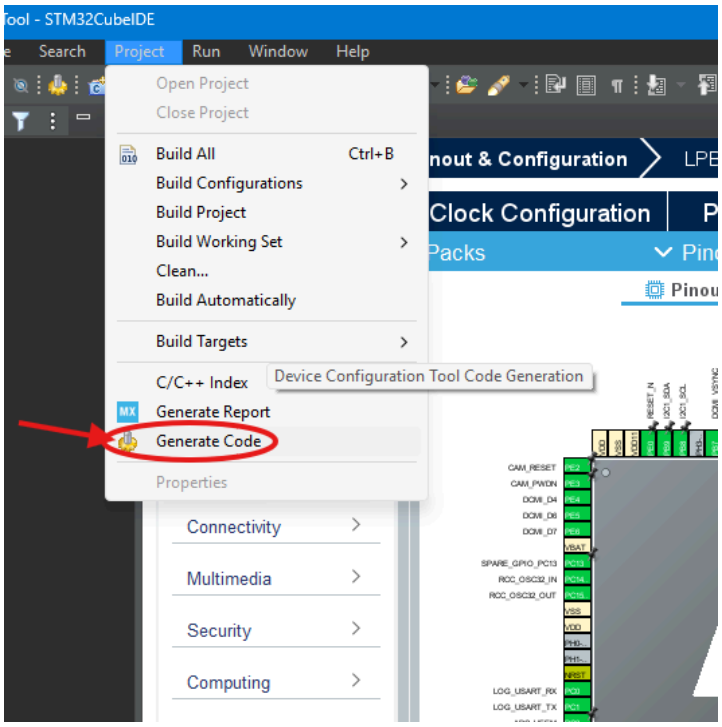


In case it asks for downloading more packages, select Download now to download required packages:

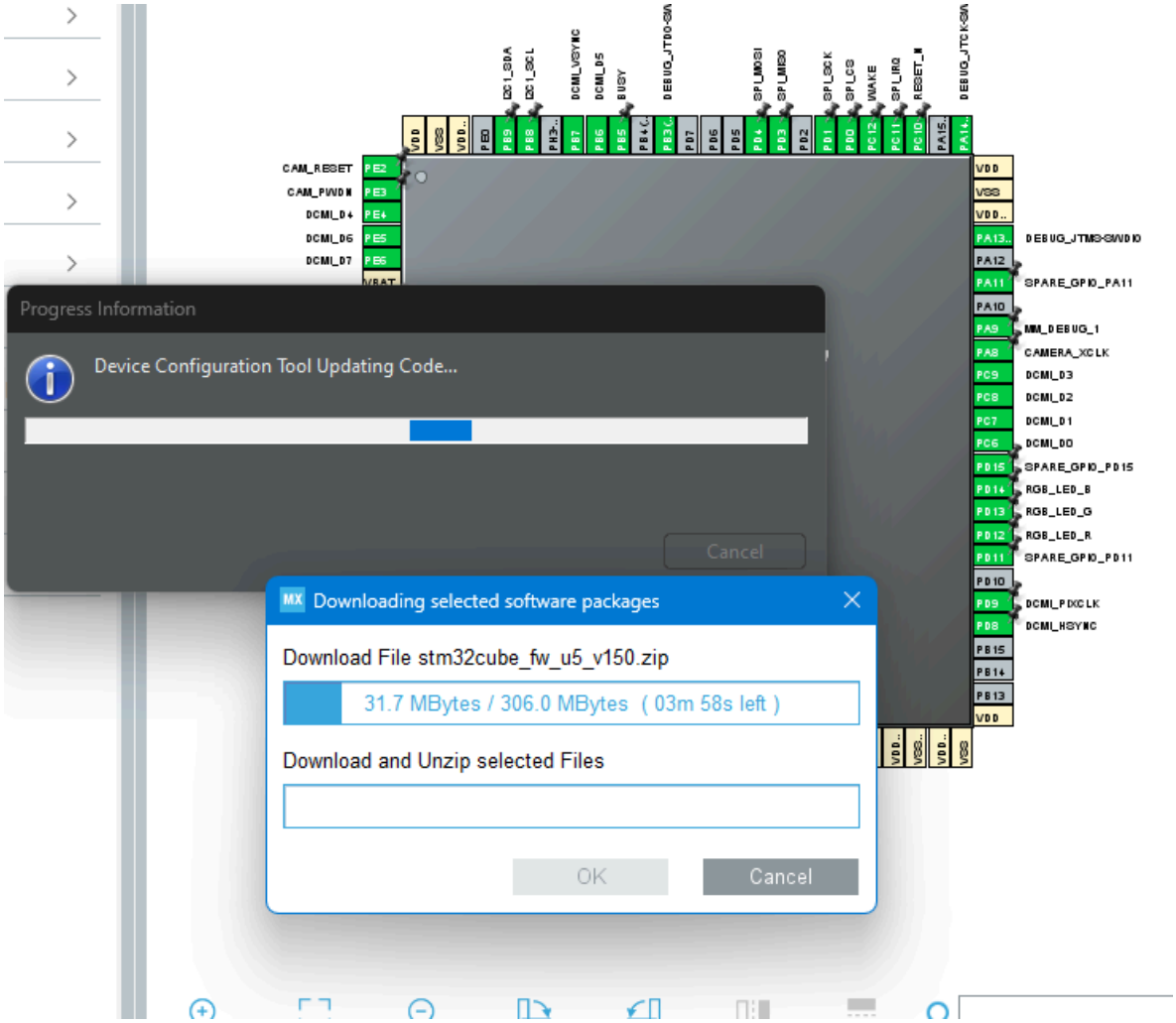


During the download and installation of the packages, it might ask you to agree with Terms and conditions a couple of times. Please Agree with all the terms and conditions and click Finish.

Now you should be able to see the cubeMx window. From Project menu, click on Generate Code:



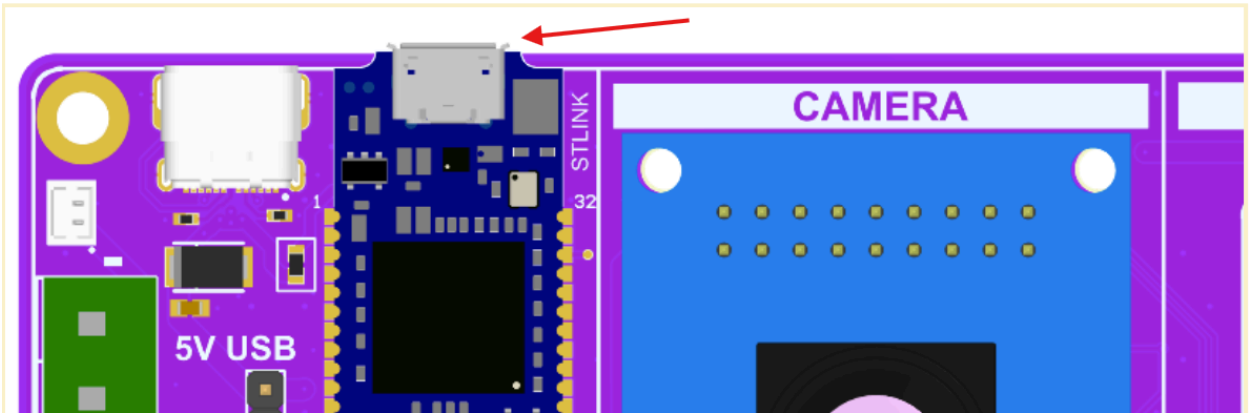
Probably it will start to download more packages:



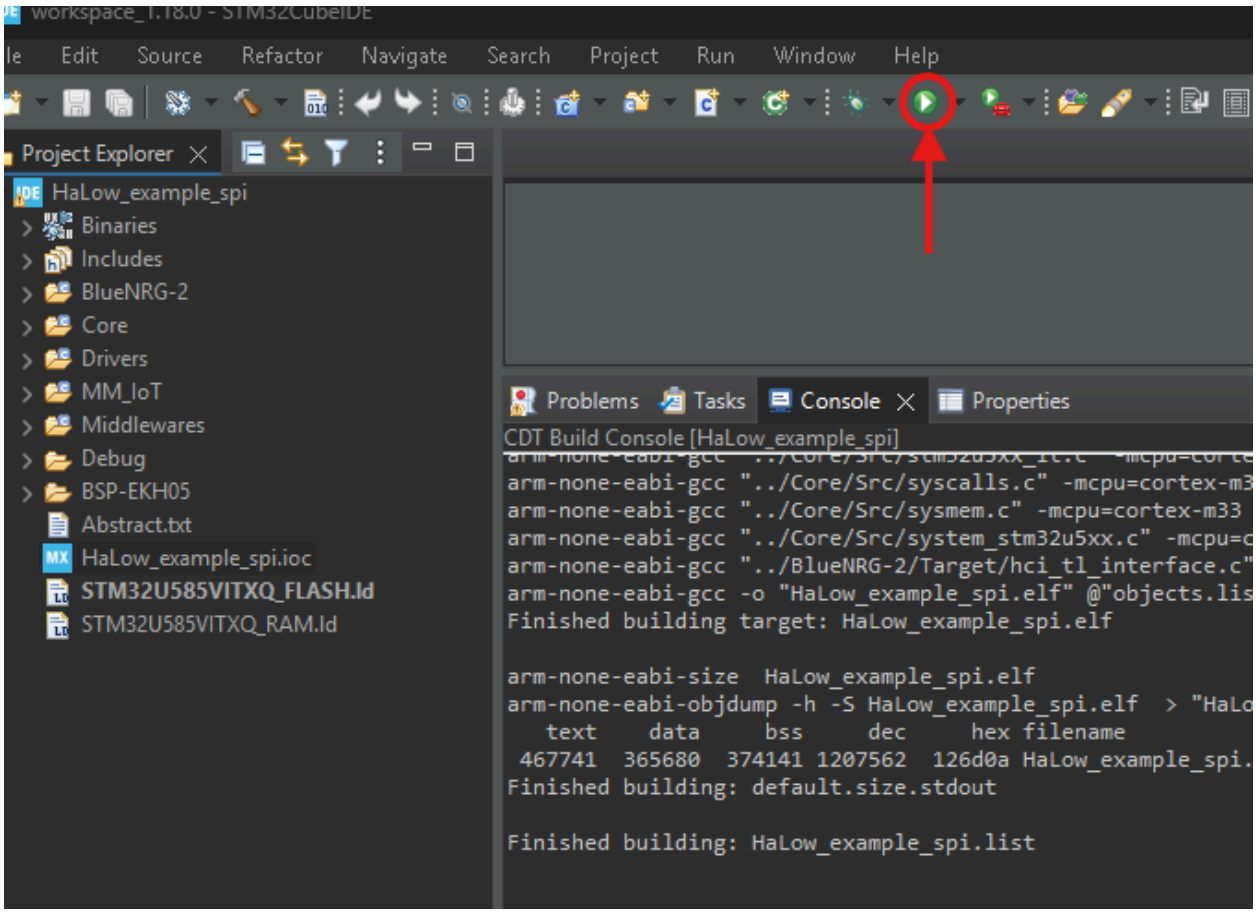
In case it shows license agreement dialog again, agree and click finish.

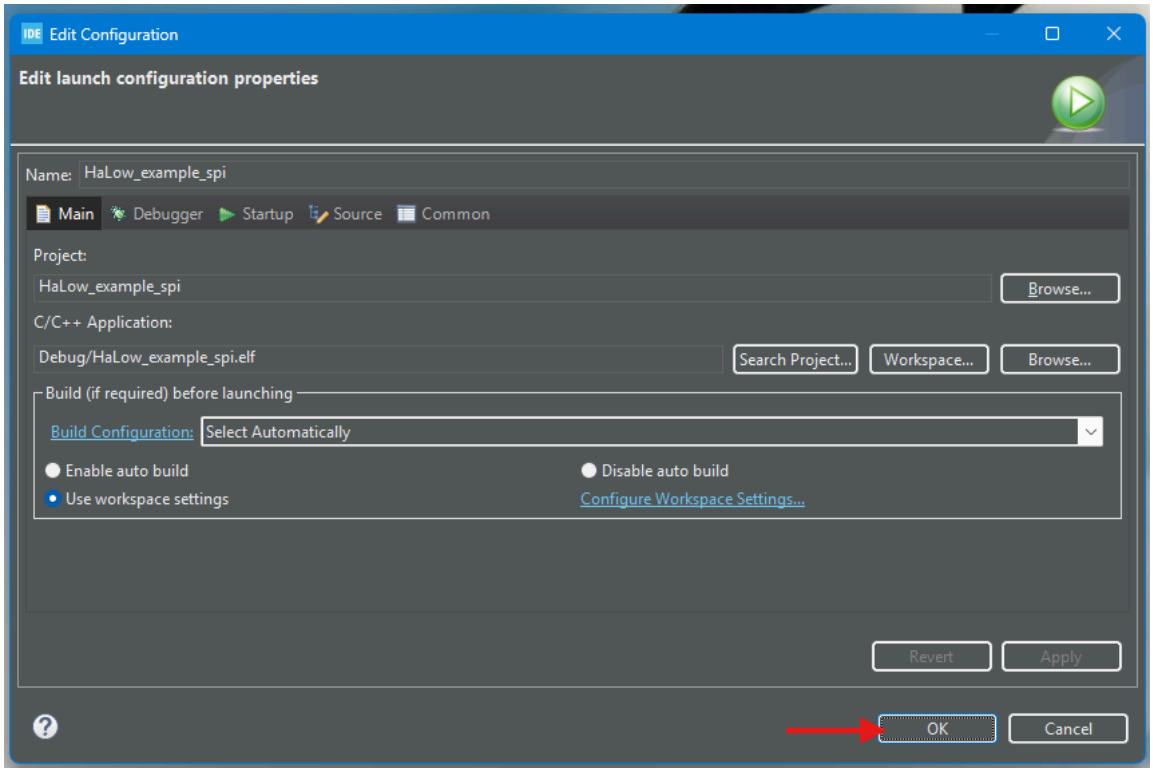
Now you can build and run the example. Connect the EKH05 board to your PC (using the

USB micro connector on ST-Link module):



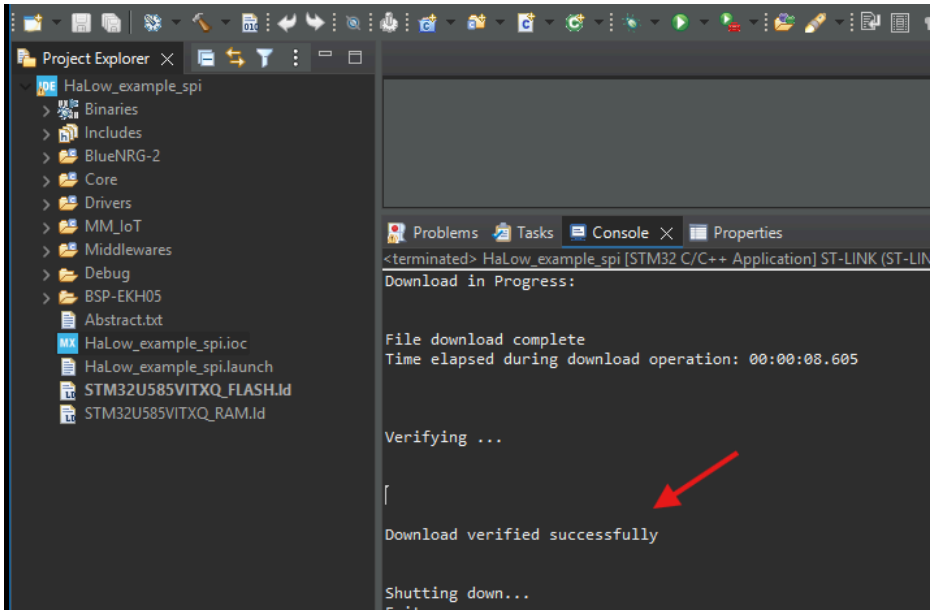
And click on Run button to build and run the example:





When you run the example for the first time, STM32CubeIDE will show you the Edit Configuration dialog to select the debugger. Leave everything with default settings and click OK:

You can see the programmers log. When it finishes programming successfully, you'll see the success output:

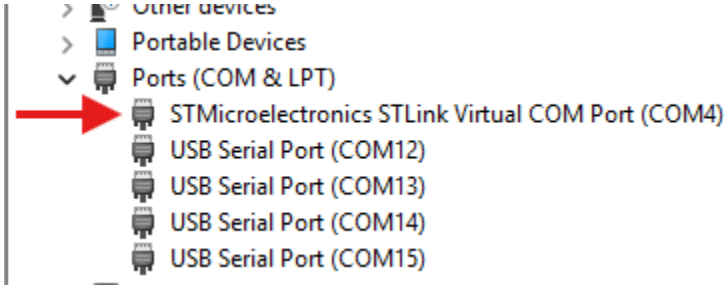


Now your EKH05 is running the IPerf program.

The following sections show how to connect to the serial port in Windows and Linux

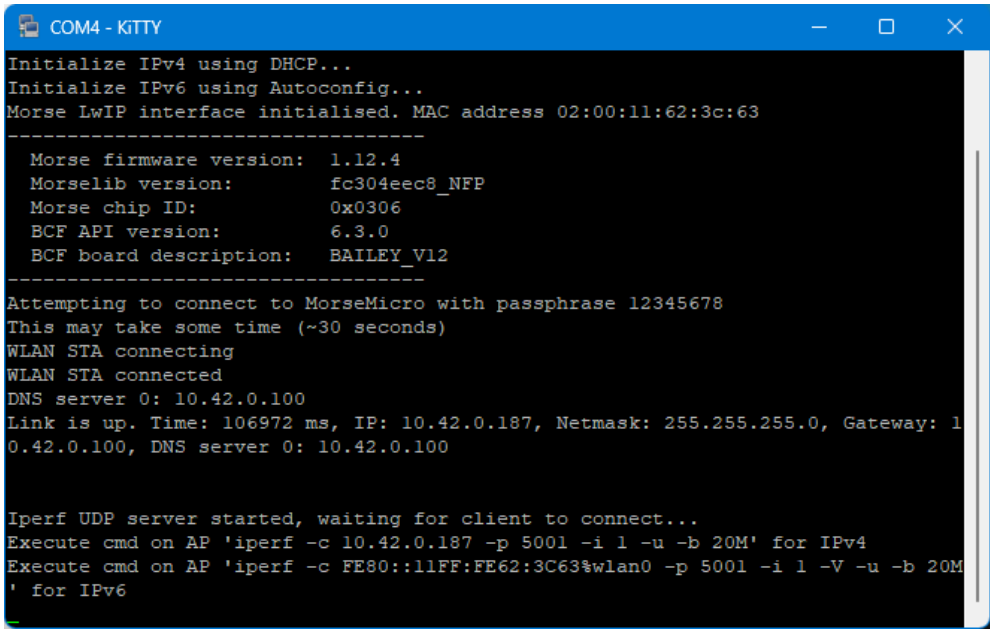
Windows

Open a serial port on the USB COM port of the ST-Link of the EKH05 with 115200 baudrate, no hardware flow control, no parity and 1 stop bit. You can find the correct COMx number from device manager:



You can use any serial terminal application. Here we are using PuTTy. After opening the serial port, press the STM RESET button on the board to restart the program. It will connect to your AP and you can try the throughput but running the commands that is shown in the

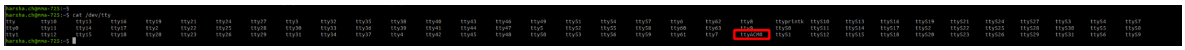
terminal:



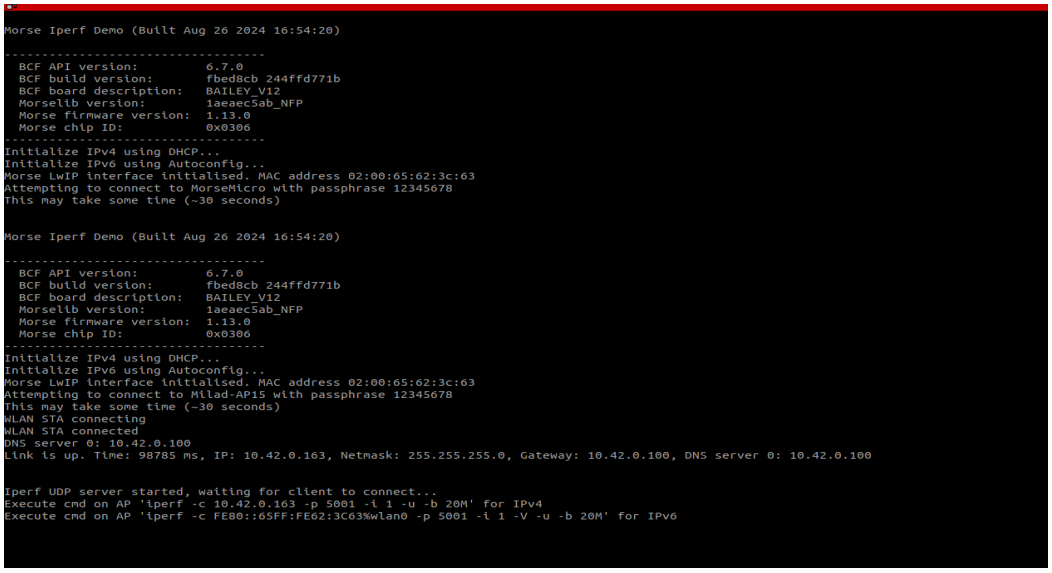
Linux

In Linux ,you can find the correct ttyACMx number from device manager(/dev). Example: /dev/ttyACM0

Use minicom/putty to open a serial port on the USB COM port of the ST-Link of the EKH05 with 115200 baudrate, no hardware flow control, no parity and 1 stop bit.

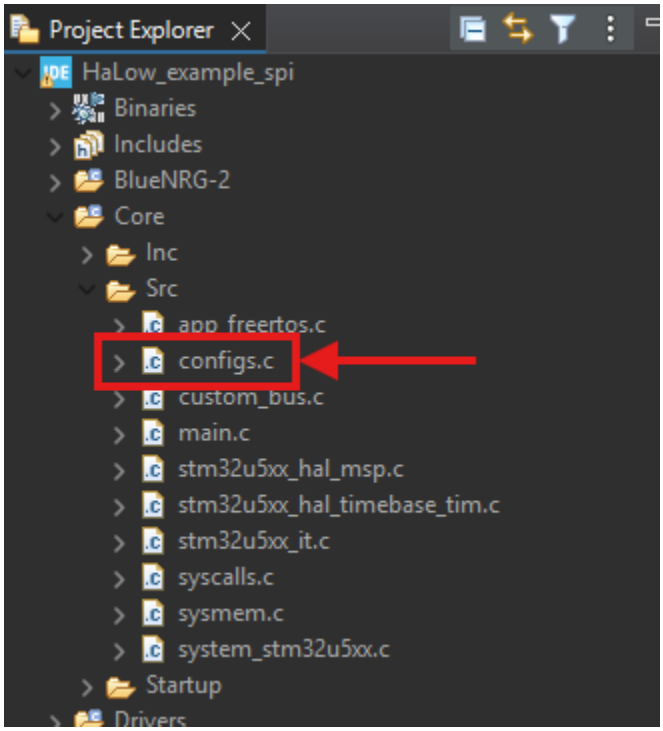


Minicom:

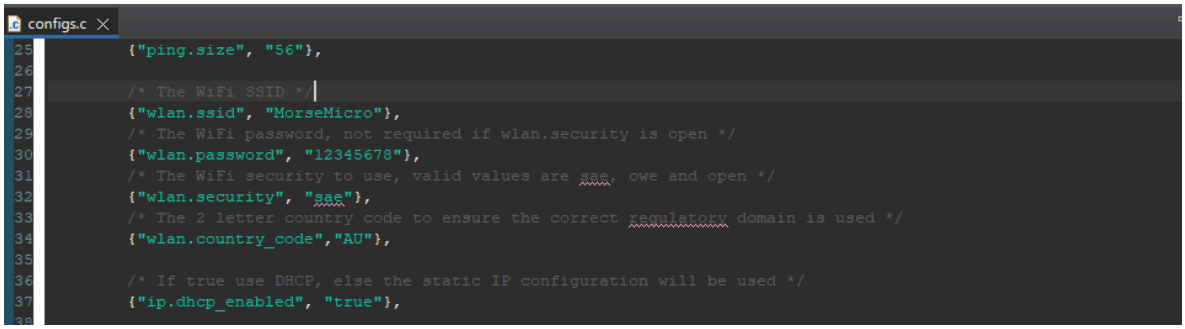

Morse Iperf Demo (Built Aug 26 2024 16:54:20)
BCF API version: 6.7.0
BCF build version: fbed8cb 244ffd771b
BCF board description: BAILEY_V12
Morselib version: 1aeaec5ab_NFP
Morse firmware version: 1.13.0
Morse chip ID: 0x0306
Initialize IPv4 using DHCP...
Initialize IPv6 using Autoconfig...
Morse LwIP interface initialised. MAC address 02:00:65:62:3c:63
Attempting to connect to MorseMicro with passphrase 12345678
This may take some time (~30 seconds)
Morse Iperf Demo (Built Aug 26 2024 16:54:20)
BCF API version: 6.7.0
BCF build version: fbed8cb 244ffd771b
BCF board description: BAILEY_V12
Morselib version: 1aeaec5ab_NFP
Morse firmware version: 1.13.0
Morse chip ID: 0x0306
Initialize IPv4 using DHCP...
Initialize IPv6 using Autoconfig...
Morse LwIP interface initialised. MAC address 02:00:65:62:3c:63
Attempting to connect to Mlad-AP15 with passphrase 12345678
This may take some time (~30 seconds)
WLAN STA connecting
WLAN STA connected
DNS server 0: 10.42.0.100
Link is up. Time: 98785 ms, IP: 10.42.0.163, Netmask: 255.255.255.0, Gateway: 10.42.0.100, DNS server 0: 10.42.0.100
Iperf UDP server started, waiting for client to connect...
Execute cmd on AP 'iperf -c 10.42.0.163 -p 5001 -i 1 -u -b 20M' for IPv4
Execute cmd on AP 'iperf -c FE80::65FF:FE62:3C63%wlan0 -p 5001 -i 1 -V -u -b 20M' for IPv6

6 Changing Example Configuration

In order to change configurations such as country, SSID and password, Open the configs.c from project tree:



You'll be able to change all the configurations in this file:



7 Custom BCF File

If you want to use a custom bcf mbin file, first you need to convert the mbin file into a c source file that contains bcf data. To do so run the following command:

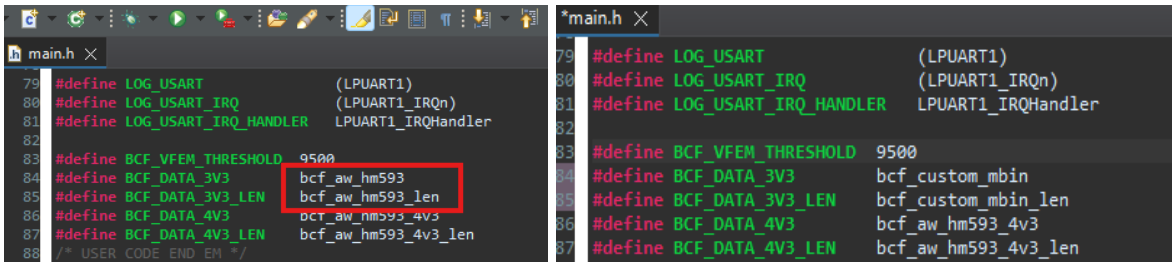
`$ xxd -i bcf_custom.mbin > bcf_custom.c`

That will create a c file containing an *unsigned char* array and an *unsigned int* variable. In the example above they will be:

`unsigned char bcf_custom_mbin[]`

`unsigned int bcf_custom_mbin_len;`

You need to replace the selected BCF in main.h with the newly generated one:

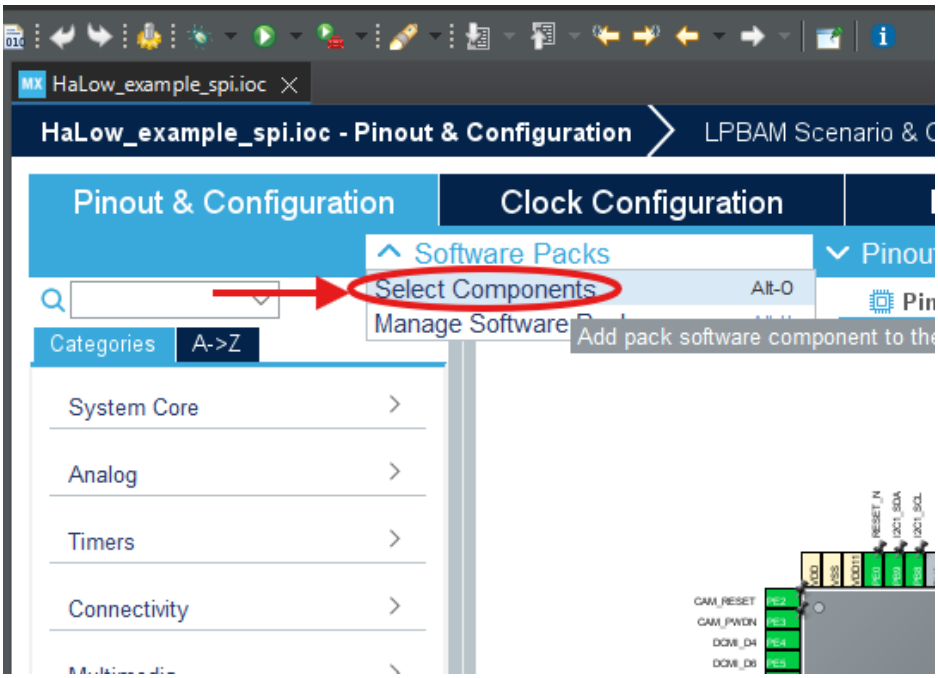


8 Build and run other example apps

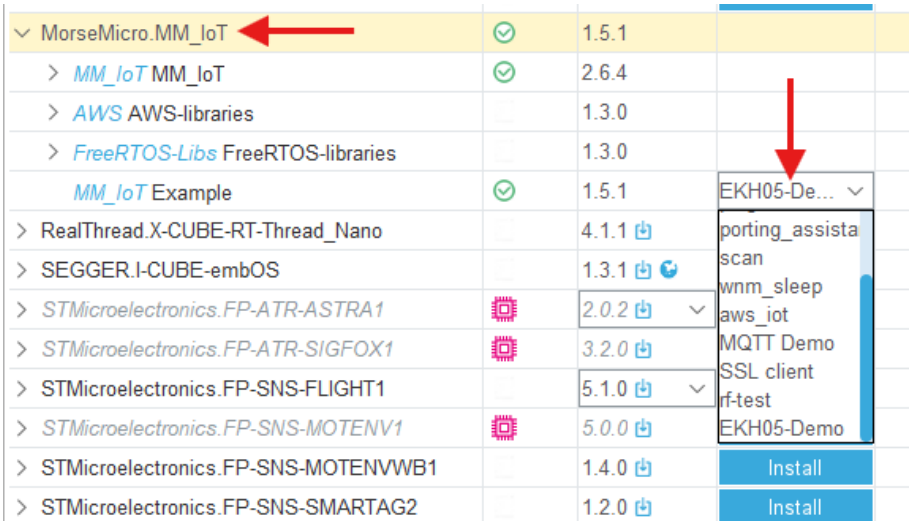
In MM-IoT CMSIS-Pack, the examples are provided as a software component with several variants.

Note: All the example variants in MM-IoT CMSIS-Pack (except EKH05-Demo example) are replicates of MM-IoT-SDK examples. To find more information on how to use a particular example, please refer to the "Example Applications" section of MM-IoT-SDK located at each release (https://github.com/MorseMicro/mm-iot-sdk/releases).

To build and run other examples, open HaLow_example.ioc file and open **Software Packs Component Selector dialog** from **Pinout & Configuration -> Software Packs -> Select Components** drop down menu:



Expand MorseMicro.MM_IoT component and select the desired example variant and click ok:



NOTE: Each example needs its own components. If you see a yellow warning sign in the Status column, it's because you need to select the dependency as well. The missing dependencies are displayed by selecting the warning icon.

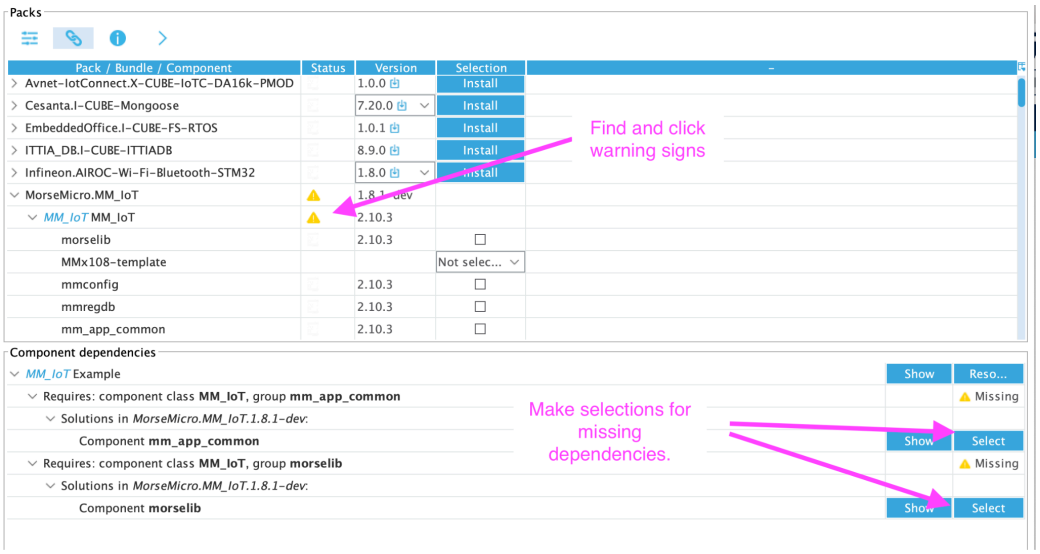
Common pitfall - Unnecessary components selected

When you change the chosen example application, the CMSIS tooling **does not** automatically select or deselect other components. For example, if changing from the EKH05-Demo example to the iperf example many components remain selected, but they will not compile as part of the "new" example. For example, seeing errors with mbedtls_x509_crt is common:

```
In file included from ../Middlewares/Third_Party/MorseMicro_MM_IoT_MM_IoT/mmagic/agent/core/mmagic_core_utils.h:8,
                 from ../Middlewares/Third_Party/MorseMicro_MM_IoT_MM_IoT/mmagic/agent/core/mmagic_core_ip.c:16:
../Middlewares/Third_Party/MorseMicro_FreeRTOS-Libs_FreeRTOS-libraries/freertos-libs-common/include/transport_interface.h:266:5:
  266 |     mbedtls_x509_crt_profile certProfile;    /**< Certificate security profile for this connection. */
      |     ^~~~~~~~~~~~~~~~~~~~~~~~~
../Middlewares/Third_Party/MorseMicro_FreeRTOS-Libs_FreeRTOS-libraries/freertos-libs-common/include/transport_interface.h:267:5:
  267 |     mbedtls_x509_crt rootCa;                 /**< Root CA certificate context. */
      |     ^~~~~~~~~~~~~~~~
../Middlewares/Third_Party/MorseMicro_FreeRTOS-Libs_FreeRTOS-libraries/freertos-libs-common/include/transport_interface.h:268:5:
  268 |     mbedtls_x509_crt clientCert;             /**< Client certificate context. */
      |     ^~~~~~~~~~~~~~~~
```

To resolve this issue you must remove unused components. If you ***do not know*** which components to remove, we recommend you:

1. Remove all components under the "MorseMicro.MM-IoT" pack
2. Use the yellow warnings to guide selecting all necessary components.
   This may require several passes to get all dependencies enabled.



NB: We recommend choosing mmLwIP over FreeRTOS-Plus-TCP, but both are supported as dependencies to mmipal. FreeRTOS-Plus-TCP will be deprecated in the next SDK release.

Common pitfall - CMSIS bundle deselected

If you have previously had all of the subcomponents from the “AWS AWS-Libraries” or “FreeRTOS-Libs FreeRTOS-libraries” bundles deselected then CubeMX will automatically deselect the bundle too. If you then switch back to enabling the subcomponents, CubeMX does not automatically re-select the bundle.

Some common build errors for this situation are:

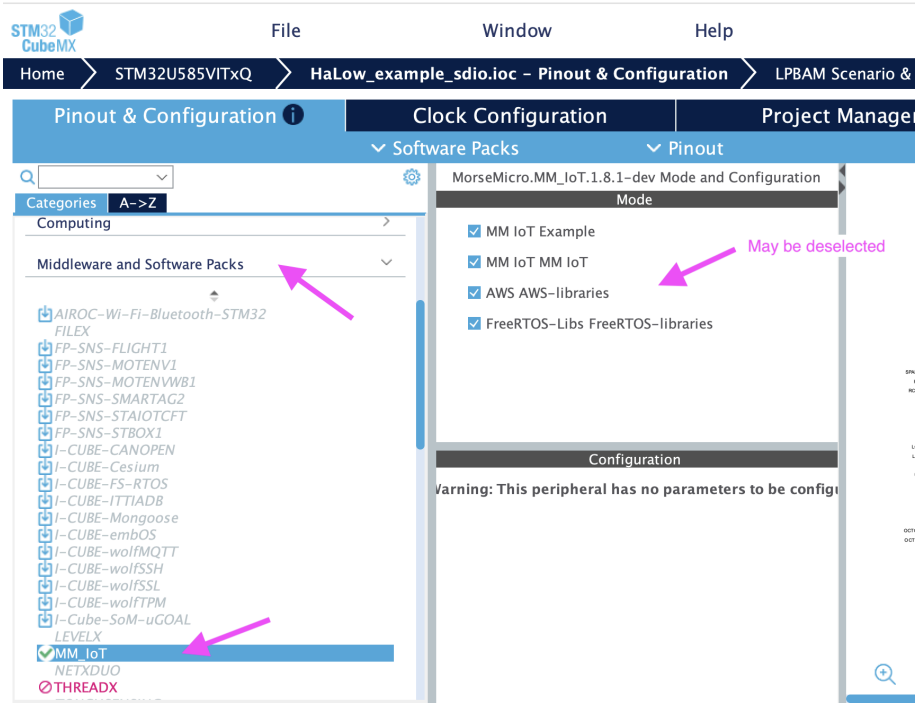
- Failing to find `shadow_device_task.h` - this likely means that the “AWS AWS-libraries” bundle is not selected.

```
arm-none-eabi-gcc "../Middlewares/Third_Party/MorseMicro_FreeRTOS-Libs_FreeRTOS-libraries/mqtt-agent-task/mqtt_agent_task.c" -mcpu=cortex-m33 -s
arm-none-eabi-gcc "../Middlewares/Third_Party/MorseMicro_FreeRTOS-Libs_FreeRTOS-libraries/freertos-libs-common/sntp_client-lwip.c" -mcpu=cortex-
../Middlewares/Third_Party/MorseMicro_MM_IoT/Example/aws_iot/aws_iot.c:514:10: fatal error: shadow_device_task.h: No such file or directory
  514 | #include "shadow_device_task.h"
      |          ^~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
```

- Failing to find `mqtt_agent_task.h` - this likely means that the “FreeRTOS-Libs FreeRTOS-libraries” bundle is not selected.

```
arm-none-eabi-gcc "../Middlewares/Third_Party/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/port.c" -mcpu=cortex-m33 -std=gnu11 -g3 -DDEBUG
../Middlewares/Third_Party/MorseMicro_MM_IoT/Example/aws_iot/aws_iot.c:512:10: fatal error: mqtt_agent_task.h: No such file or directory
arm-none-eabi-gcc "../Middlewares/Third_Party/FreeRTOS/Source/portable/GCC/ARM_CM33_NTZ/non_secure/portasm.c" -mcpu=cortex-m33 -std=gnu11 -g3 -DDE
  512 | #include "mqtt_agent_task.h"
      |          ^~~~~~~~~~~~~~~~~~~
compilation terminated.
```

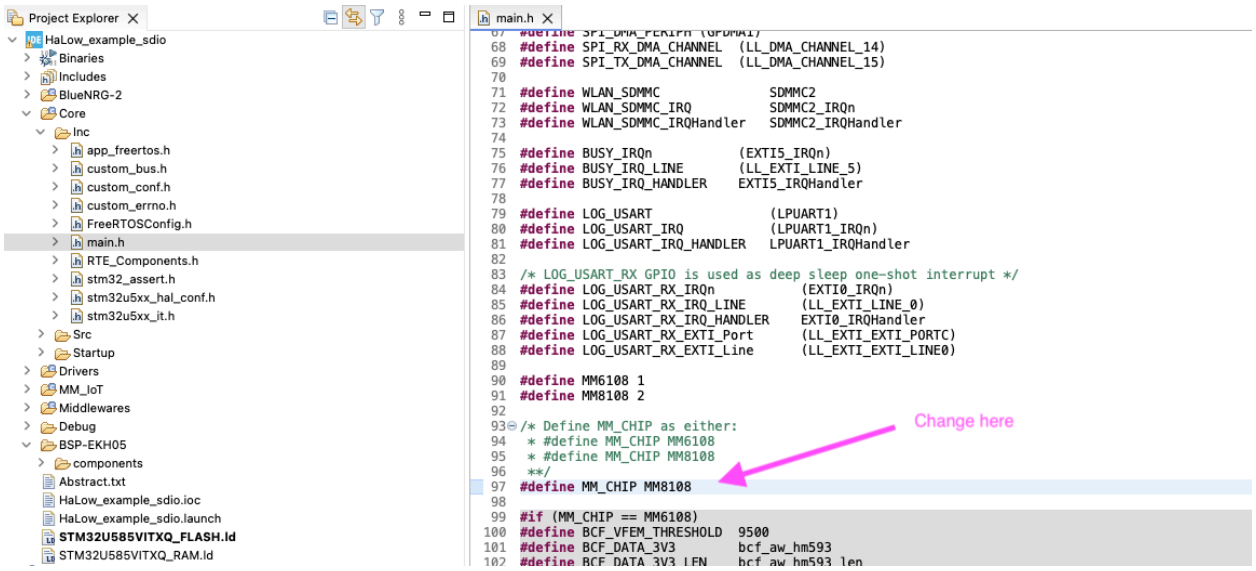
To resolve this issue, ensure that all items in the `Categories > Middleware and Software Packs > MM_IoT` option are selected. After selecting the bundle, ensure you “Generate code” to add source and components to the project, then clean and rebuild the project.



9 Switching a project between MM6108 and MM8108

The Morse Micro library binary (libmorse.a) provided with the CMSIS pack 1.8.0+ supports both MM6108 and MM8108 chips. The chip targeted is controlled by the MM_CHIP definition included in main.h.

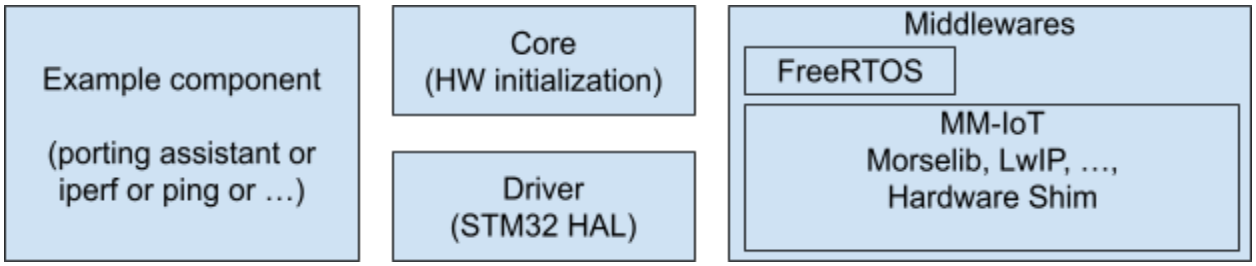
To change the chip used by a project, edit the file to specify the required chip:



10 Create a new HaLow project in CubeIDE

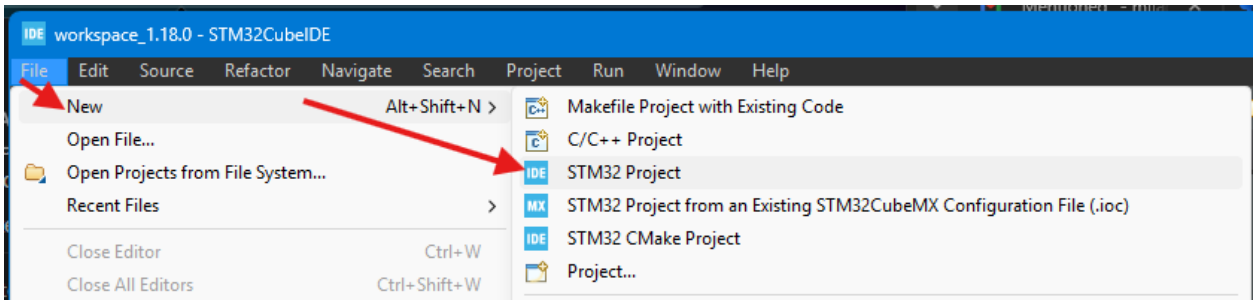
This section provides a walkthrough for creating a new project in STM32CubeIDE for STM32U585VIT6 target that is presented on EKH05 .

A high level overview of the code components of the project that we're going to create is as follows:

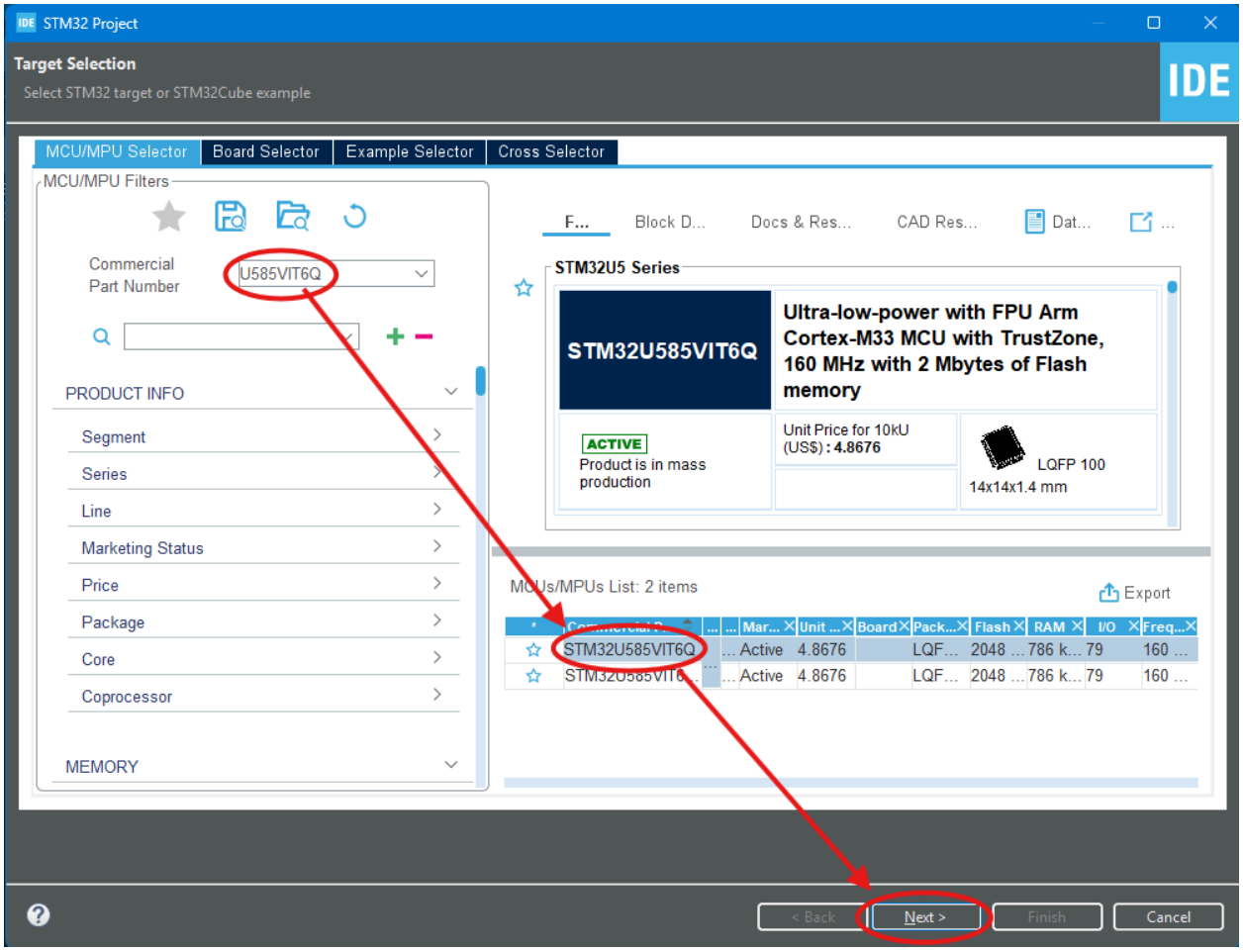


Most of this section will be explaining how to prepare the Core code that initializes the hardware peripherals.

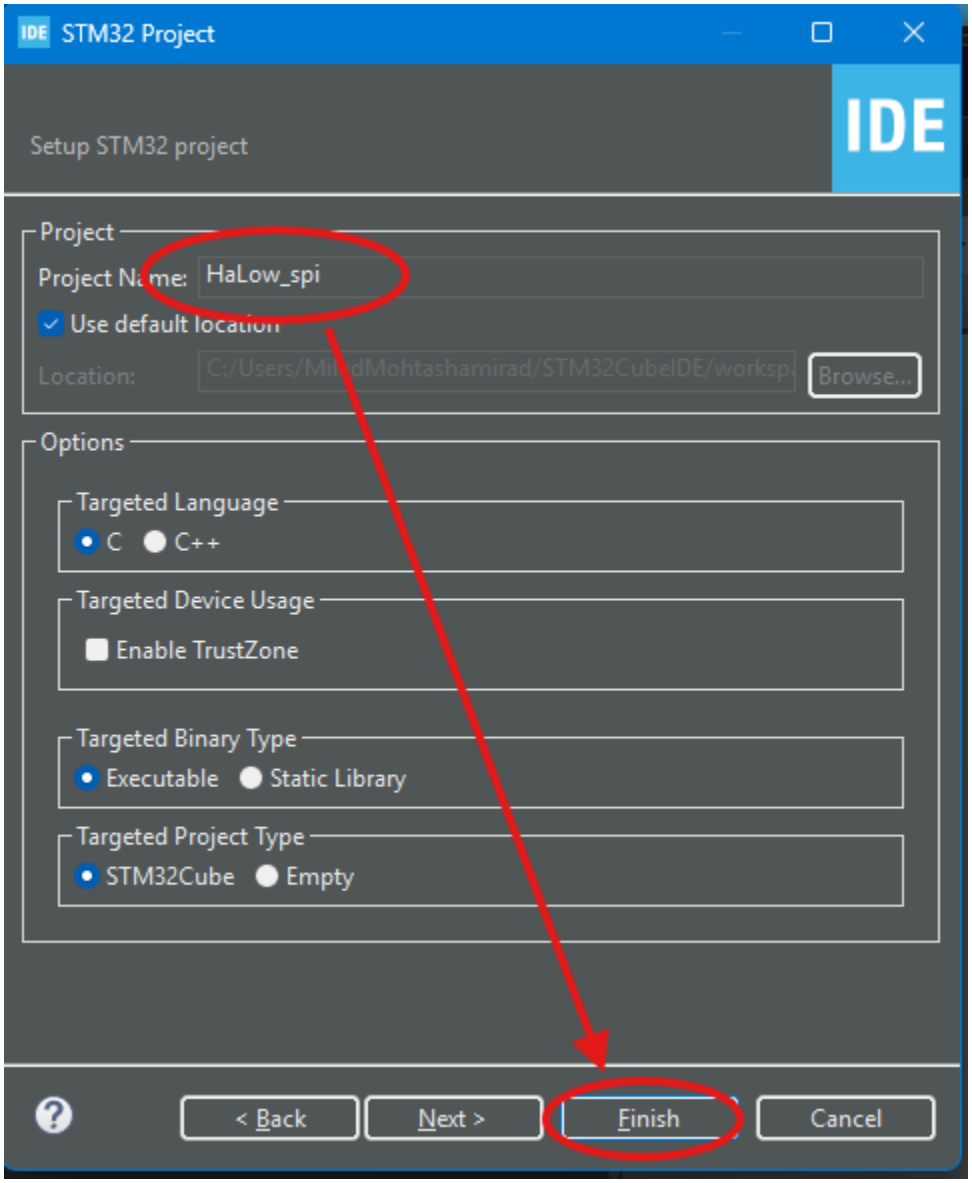
Open **STM32 Project** dialog from **File->New->STM32 Project** :



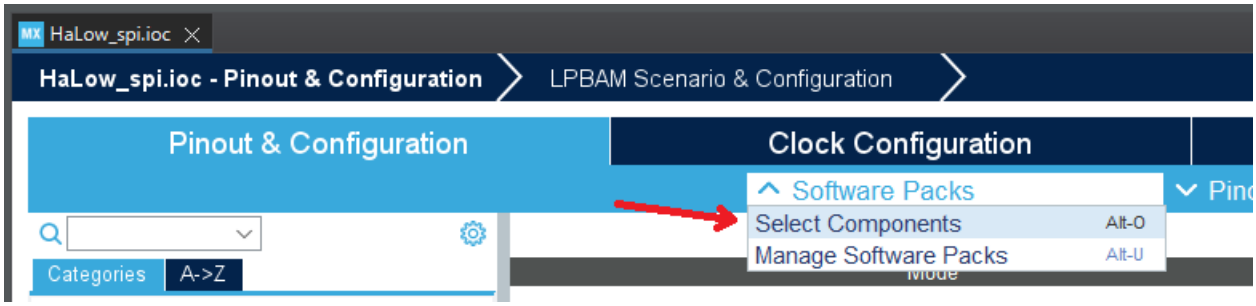
Select the desired component by searching in **Commercial Part Number** (Here we select STM32U585VIT6) and click next:



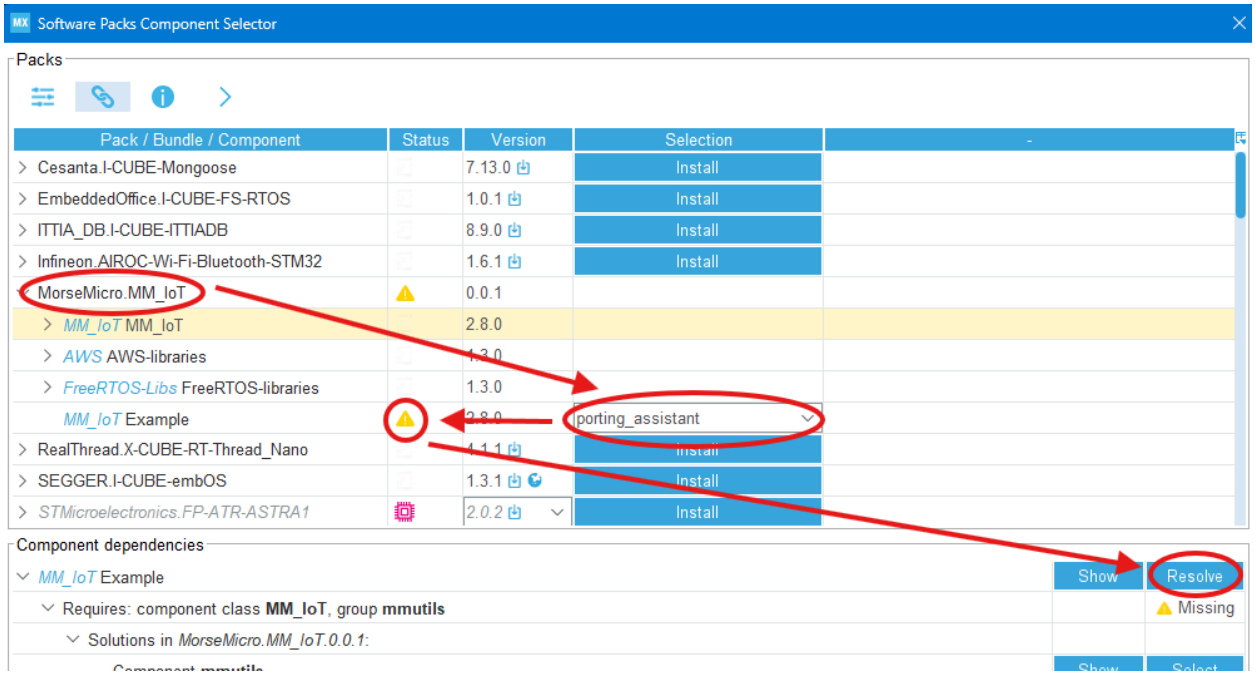
Select a name for project and click Finish:



This will create the project with only the cubemx file (ioc) in it where you can configure your peripherals. Open **Software Packs Components selector** from **Pinout & Configuration -> Software Packs -> Select Components.**

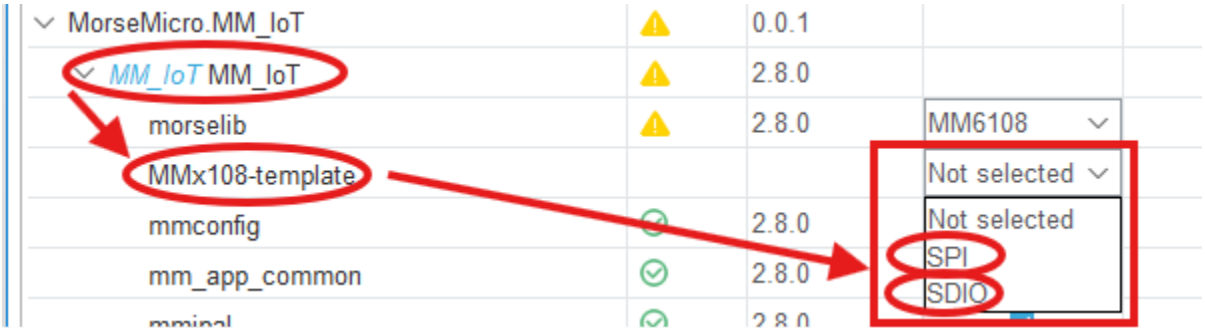


Then select **porting_assistant** from **MorseMicro.MM_IoT -> MM_IoT example** dropdown. Porting assistant example application, will run all the necessary hw tests (at the time of writing this document, **except for SPI_IRQ pin interrupt**) to make sure that your hardware and hardware abstraction layer are setup correctly.
Click on the yellow warning sign to highlight the missing components. You can click resolve to let the cubeMX select the missing components as much as it can.



You need to do the same for the rest of the components that have missing dependencies.
For morselib choose **BSP-EKH05/MMx108**/Variant **SPI** . If you are creating the project for a new hardware platform, select **MM_IoT** -> **MMx6108 -template**/Variant **SPI** or **SDIO** and skip the SPI settings of this chapter and setup your own SPI or SDIO settings. For more

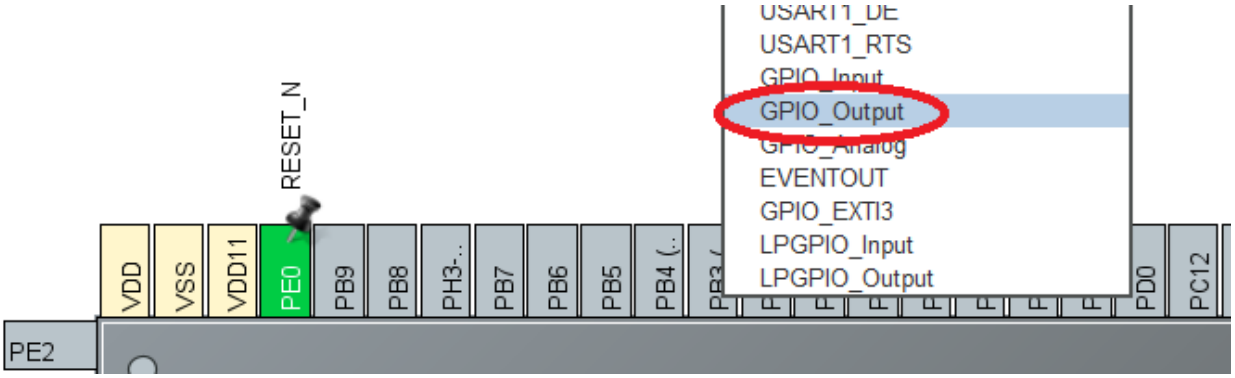
details about setting your own interface refer to **Porting a new platform** chapter.



For **mmipa**l, select **mmpktmem**/Variant **static** and **mmLwIP**. And close the components dialog.

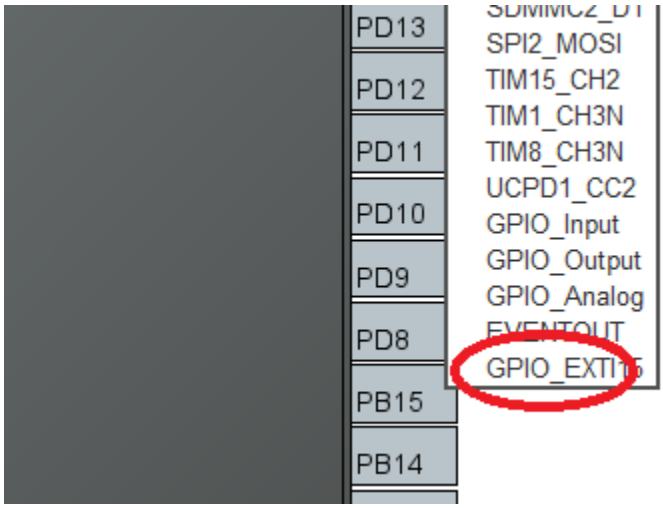
Proceed to configure the required pinouts and peripherals.
Here, we start from the easiest IOs. First select GPIO outputs and add user labels **WAKE, SPI_CS, RESET_N, GPIO_LED_GREEN, GPIO_LED_BLUE** , **GPIO_LED_RED, MM_DEBUG_0** and **MM_DEBUG_1** to them.

RESET_N	PE0
SPI_CS	PB4
WAKE	PD0
GPIO_LED_GREEN	PE7
GPIO_LED_BLUE	PE8
GPIO_LED_RED	PE11
MM_DEBUG_0	PA10
MM_DEBUG_1	PA9

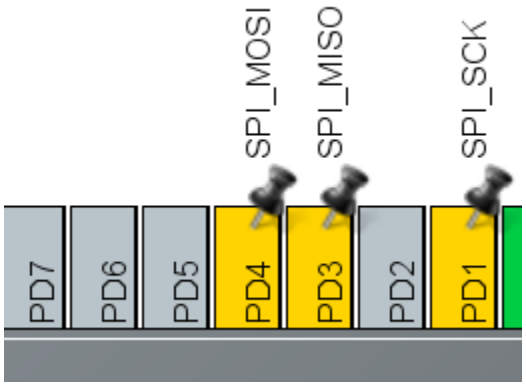


Select two pins to be GPIO_EXTI and label them as **SPI_IRQ** and **BUSY**.

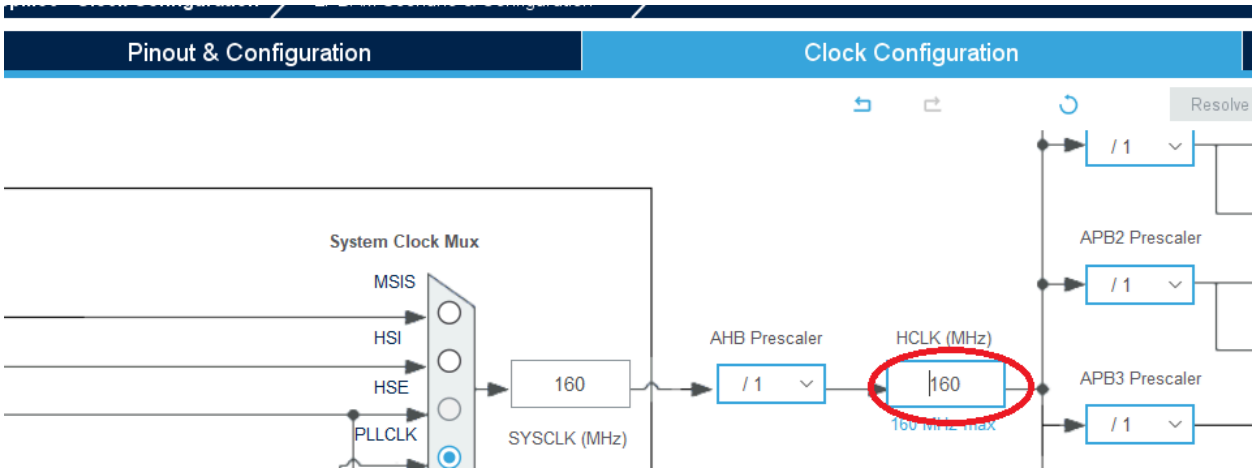
BUSY	PB5 (EXTI_LINE_5)
SPI_IRQ	PB15 (EXTI_LINE_15)



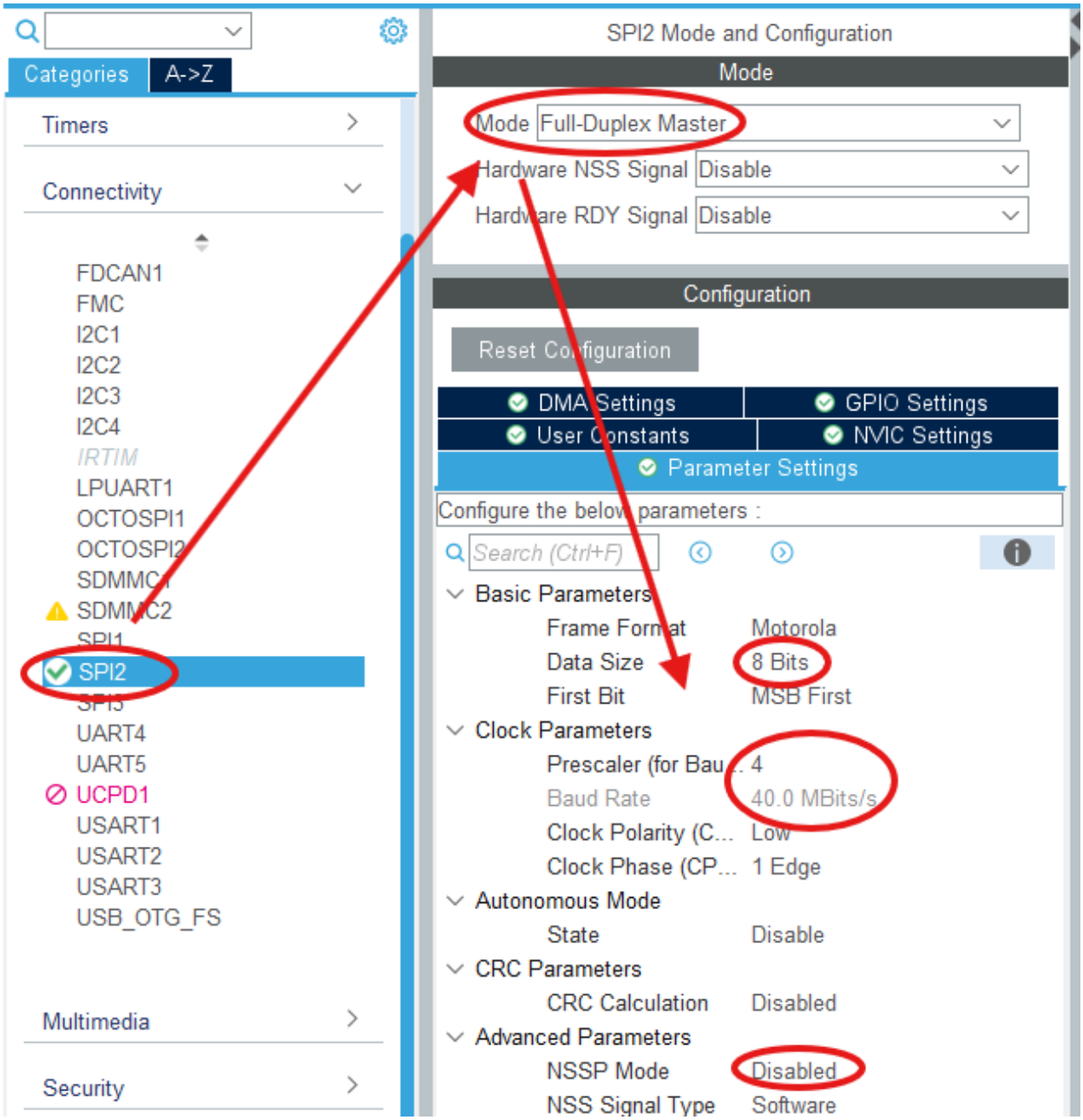
Select 3 SPI pins and label them **SPI_MOSI**, **SPI_MISO** and **SPI_SCK**:



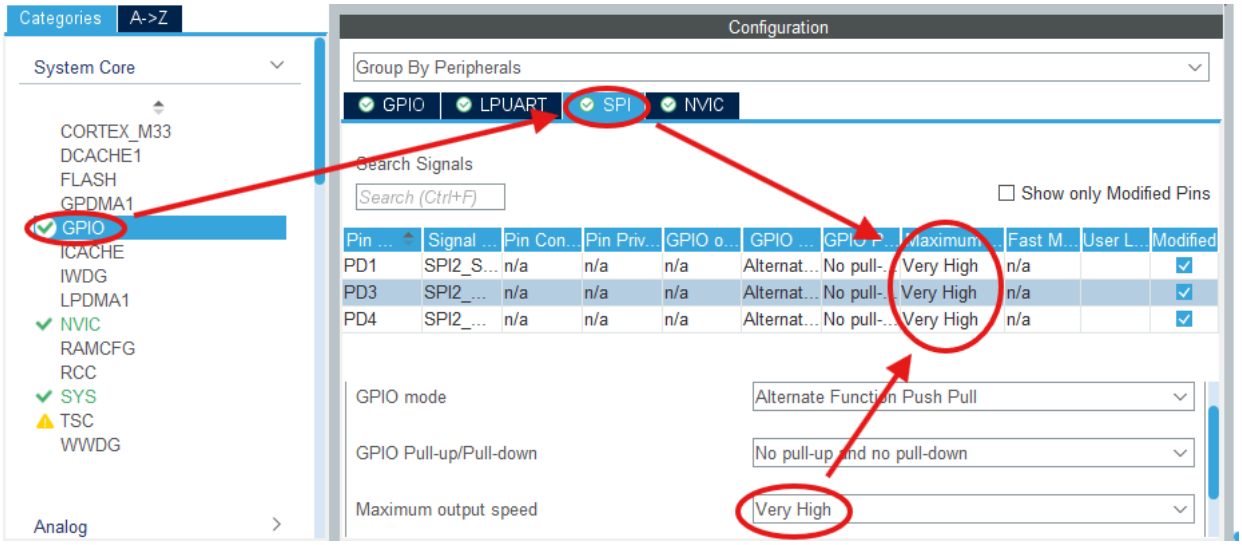
In Clock Configuration set the system clock (HCLK) to 160 MHz and press enter and let the CubeMX finds the best clock configuration for that frequency:



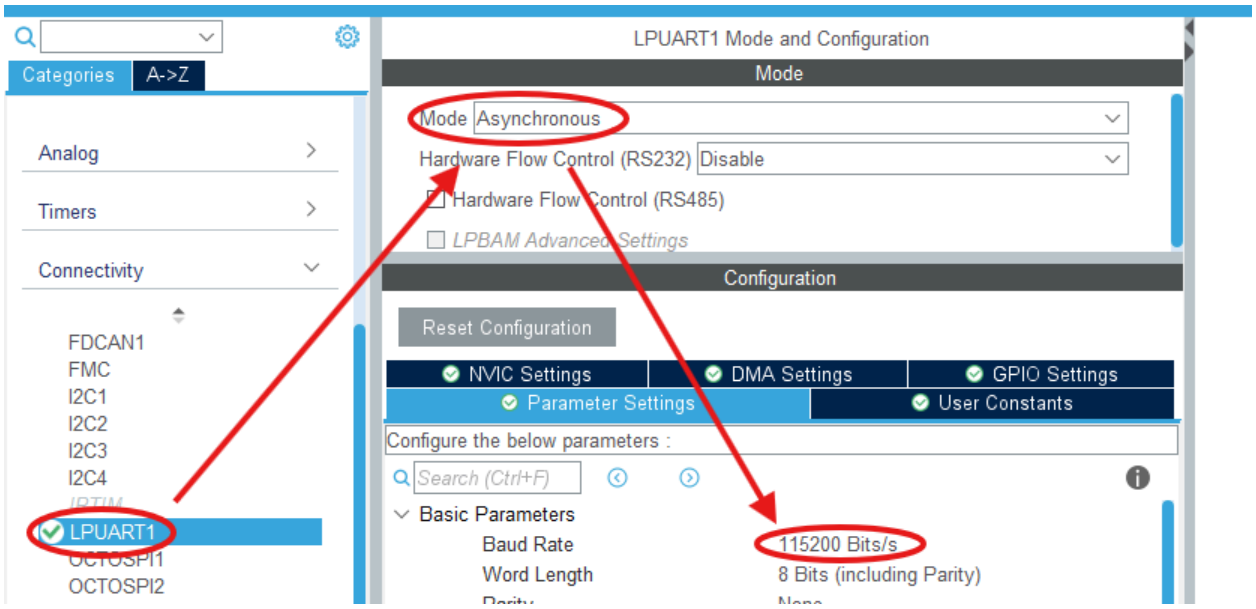
Now enable your SPI peripheral as **full-duplex master**, **8bits 50MHz or lower**, **Low Clock polarity** and **1 edge** Clock phase and **NSSP Mode Disabled**:



And set the SPI GPIOs' speed to **very high**:

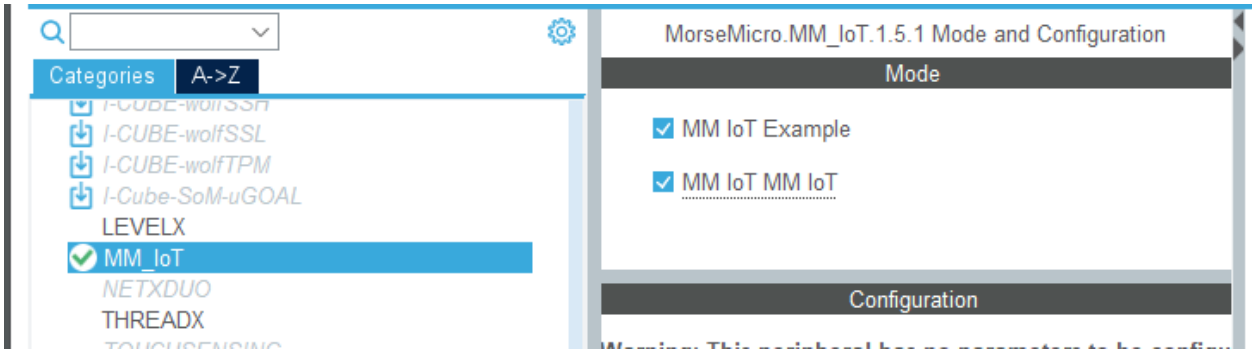


Enable a UART port for logs:

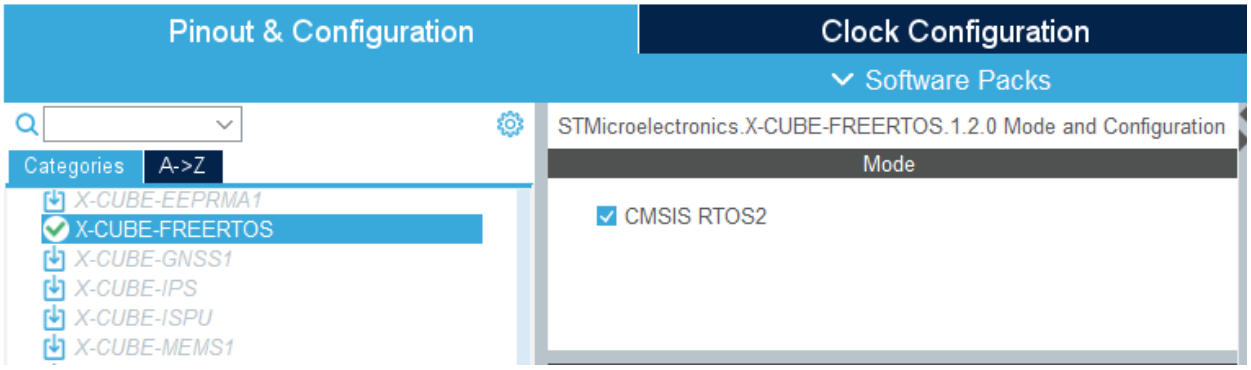


When enabling the UART make sure to label the pins as **LOG_USART_RX** and **LOG_USART_TX.**

Enable MM-IoT components from MM-IoT



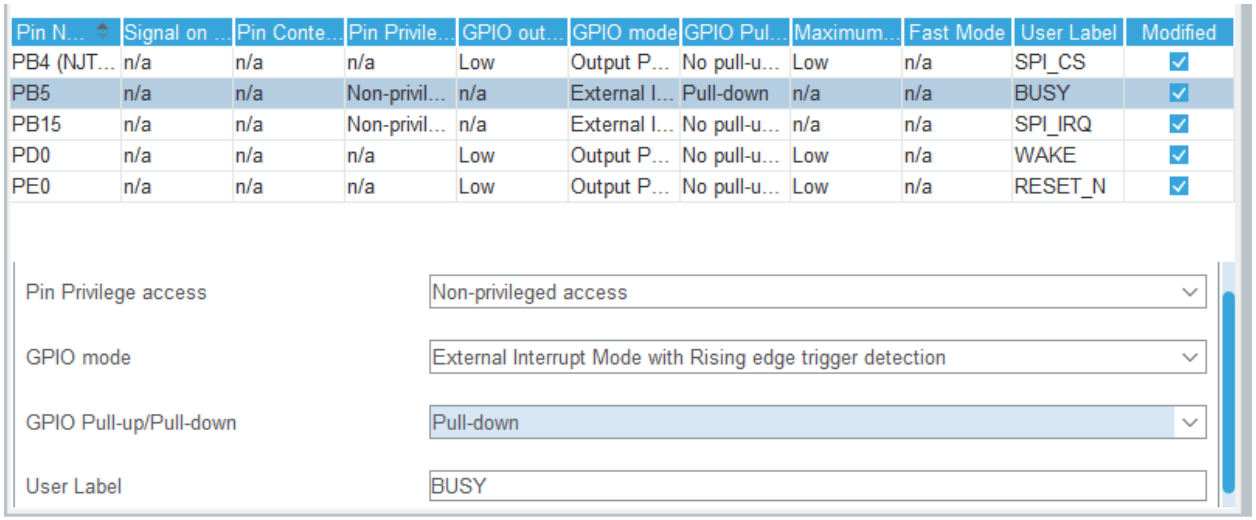
Enable **CMSIS RTOS2** from **X-CUBE-FREERTOS**



And set the the parameters as follow:

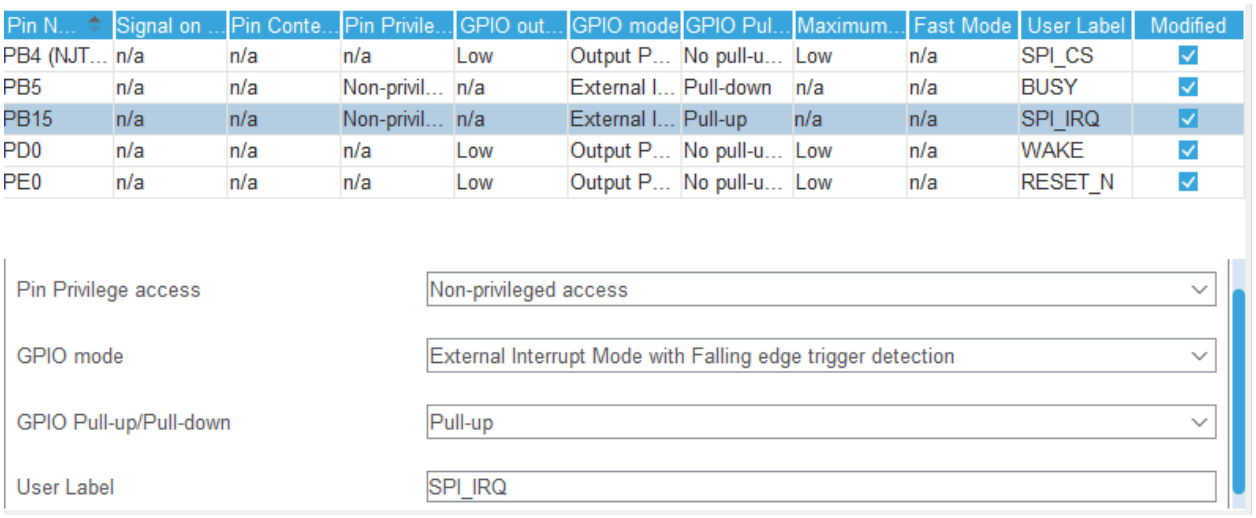
ENABLE_FPU	Enabled
MINIMAL_STACK_SIZE	256
USE_TICKLESS_IDLE	User defined functionality enabled
TOTAL_HEAP_SIZE	95*1024

From **GPIO** -> **GPIO**, select the row for **BUSY** pin and set it as **raising edge** interrupt with a **pull down**:



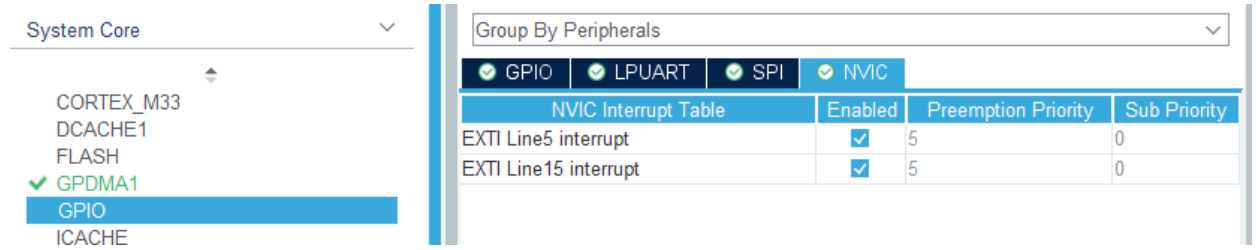
Pin N...	Signal on ...	Pin Conte...	Pin Privile...	GPIO out...	GPIO mode	GPIO Pul...	Maximum...	Fast Mode	User Label	Modified
PB4 (NJT...	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	SPI_CS	☑
PB5	n/a	n/a	Non-privil...	n/a	External I...	Pull-down	n/a	n/a	BUSY	☑
PB15	n/a	n/a	Non-privil...	n/a	External I...	No pull-u...	n/a	n/a	SPI_IRQ	☑
PD0	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	WAKE	☑
PE0	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	RESET_N	☑

And select the row for **SPI-IRQ** pin and set it as **falling edge** interrupt:

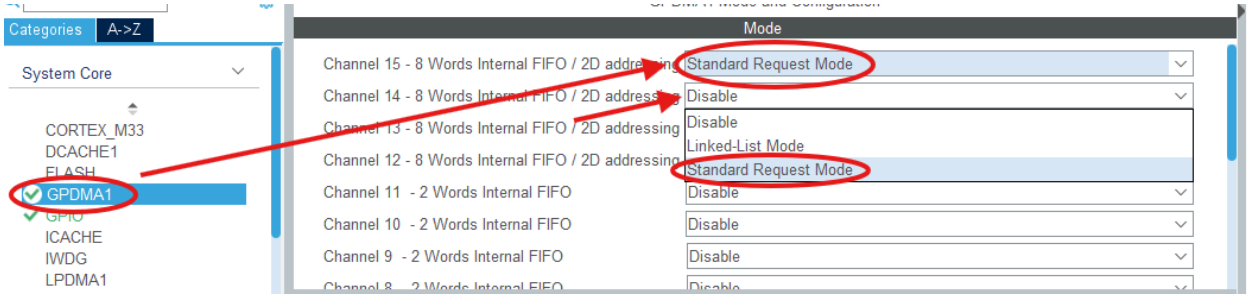


Pin N...	Signal on ...	Pin Conte...	Pin Privile...	GPIO out...	GPIO mode	GPIO Pul...	Maximum...	Fast Mode	User Label	Modified
PB4 (NJT...	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	SPI_CS	☑
PB5	n/a	n/a	Non-privil...	n/a	External I...	Pull-down	n/a	n/a	BUSY	☑
PB15	n/a	n/a	Non-privil...	n/a	External I...	Pull-up	n/a	n/a	SPI_IRQ	☑
PD0	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	WAKE	☑
PE0	n/a	n/a	n/a	Low	Output P...	No pull-u...	Low	n/a	RESET_N	☑

And enable interrupt for **EXTI 5** and **15** from **NVIC** tab

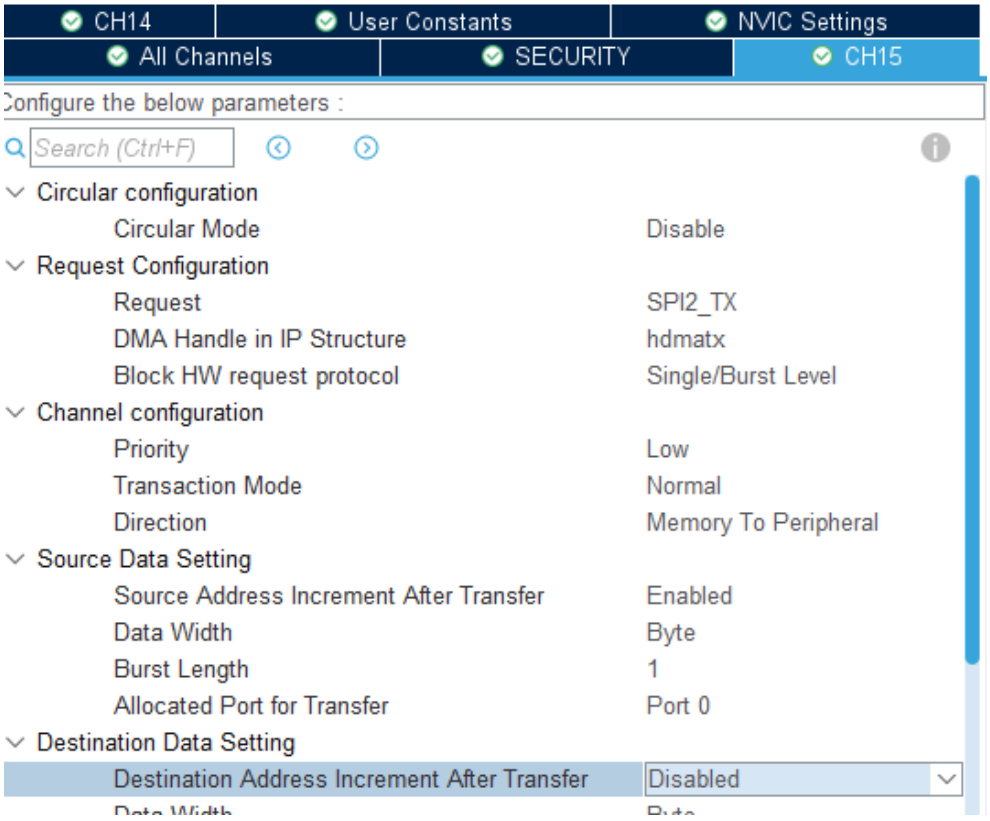


From **GPDMA1** -> **Mode** select **Standard Request Mode** for **channel 15** and **14** ,



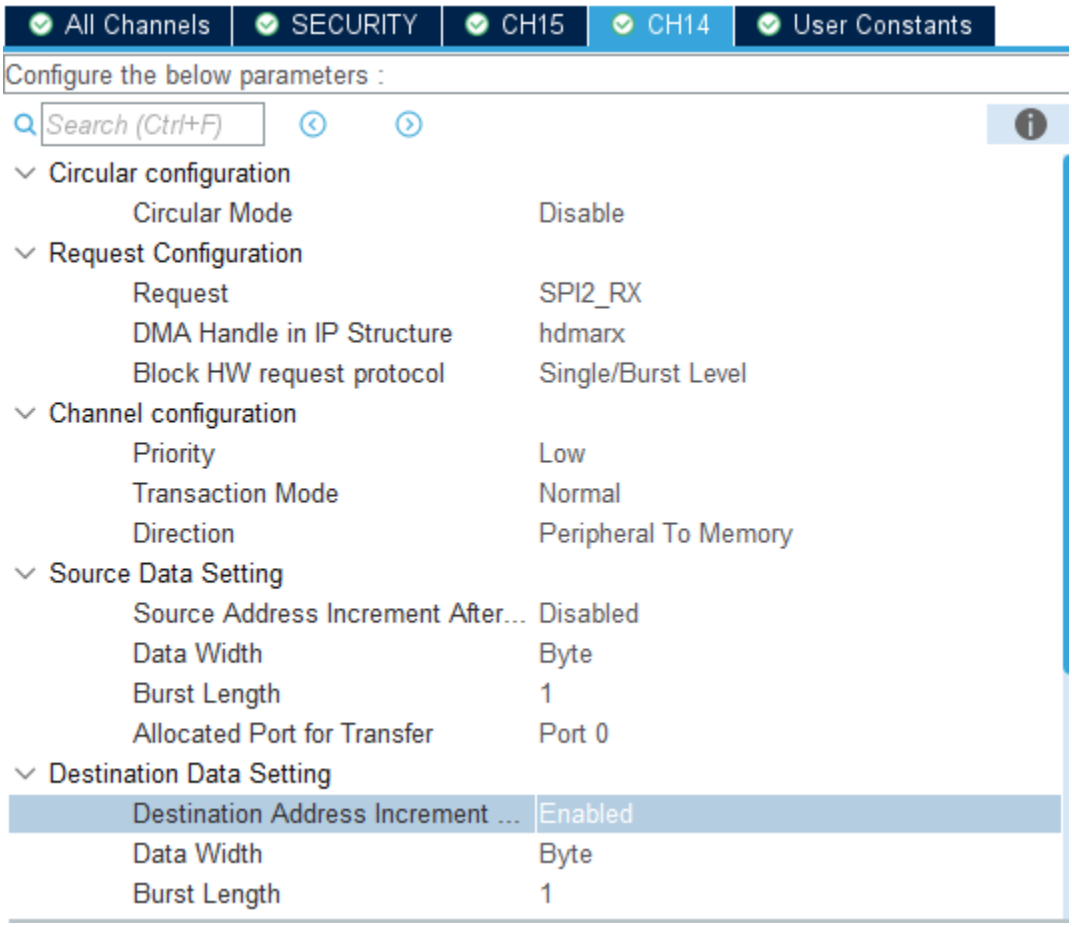
and then select **CH15** tab to setup the channel. Set the following parameters:

Request	SPI2_TX
Direction	Memory To Peripheral
Source Address Increment After Transfer	Enabled

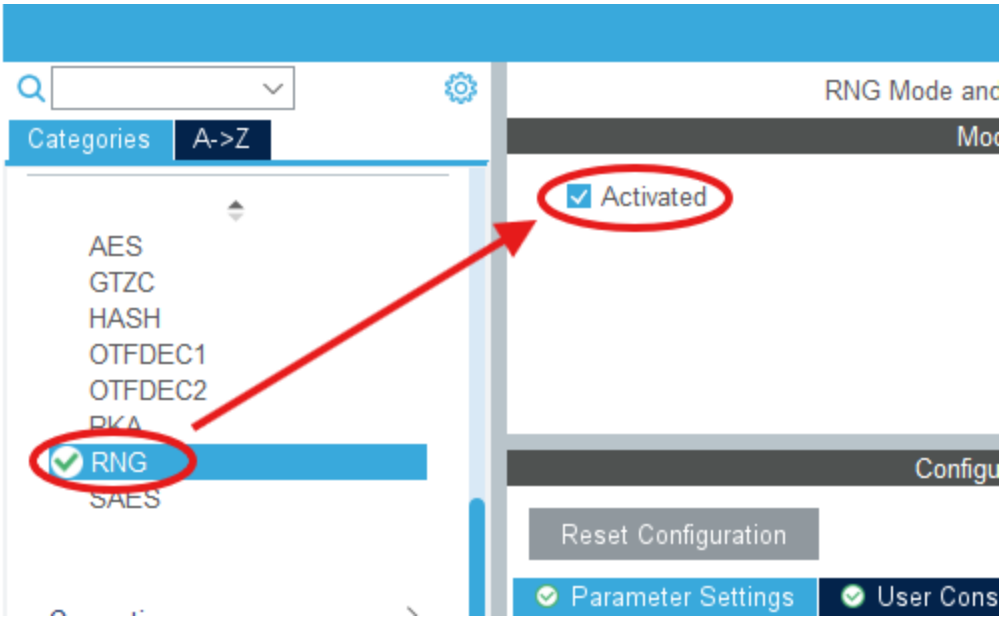


select **CH14** tab to setup the channel. Set the following parameters:

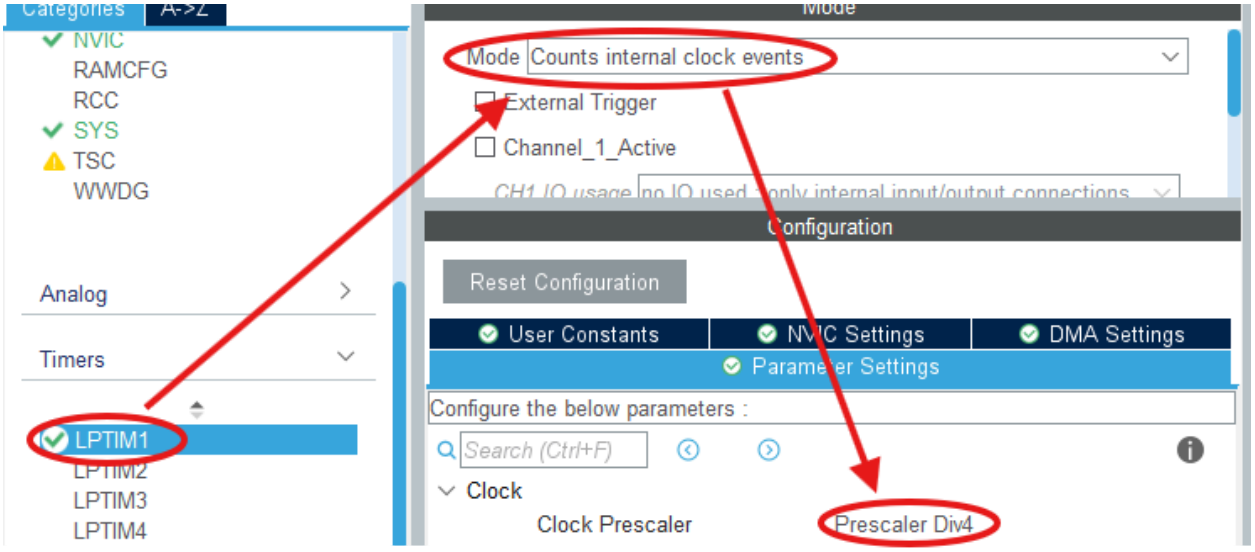
Request	SPI2_RX
Direction	Peripheral To Memory
Destination Address Increment After Transfer	Enabled



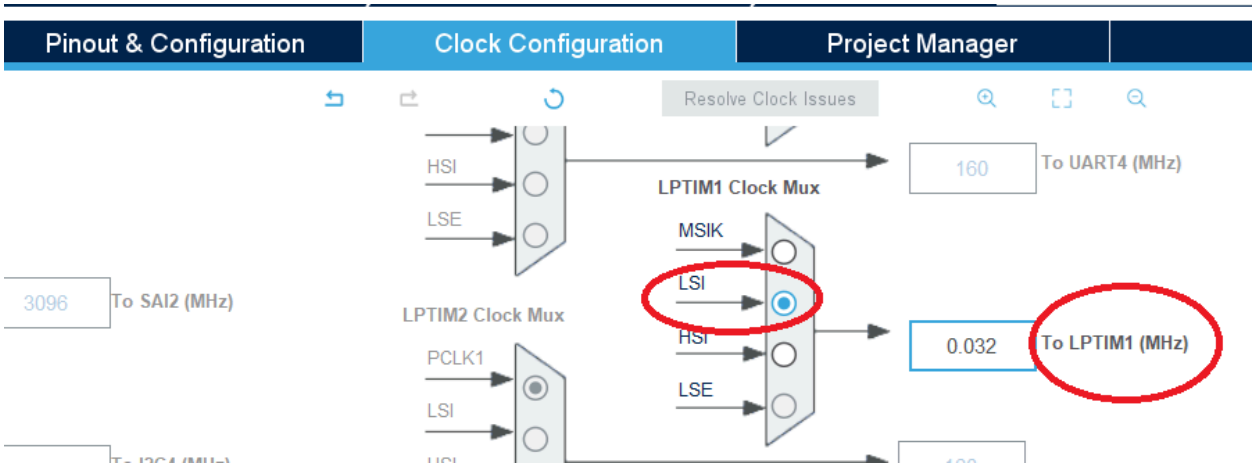
Enable **Random Number Generator** from **Security** -> **RNG**



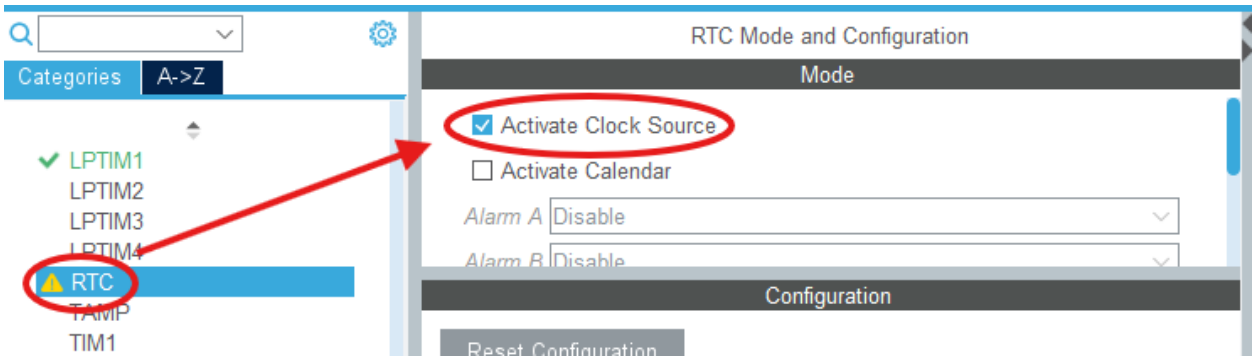
Enable a **low power timer** with internal clock source and set its **prescaler** to **dividing by 4**:



Change the LPTIM1 clock source to **LSI**:

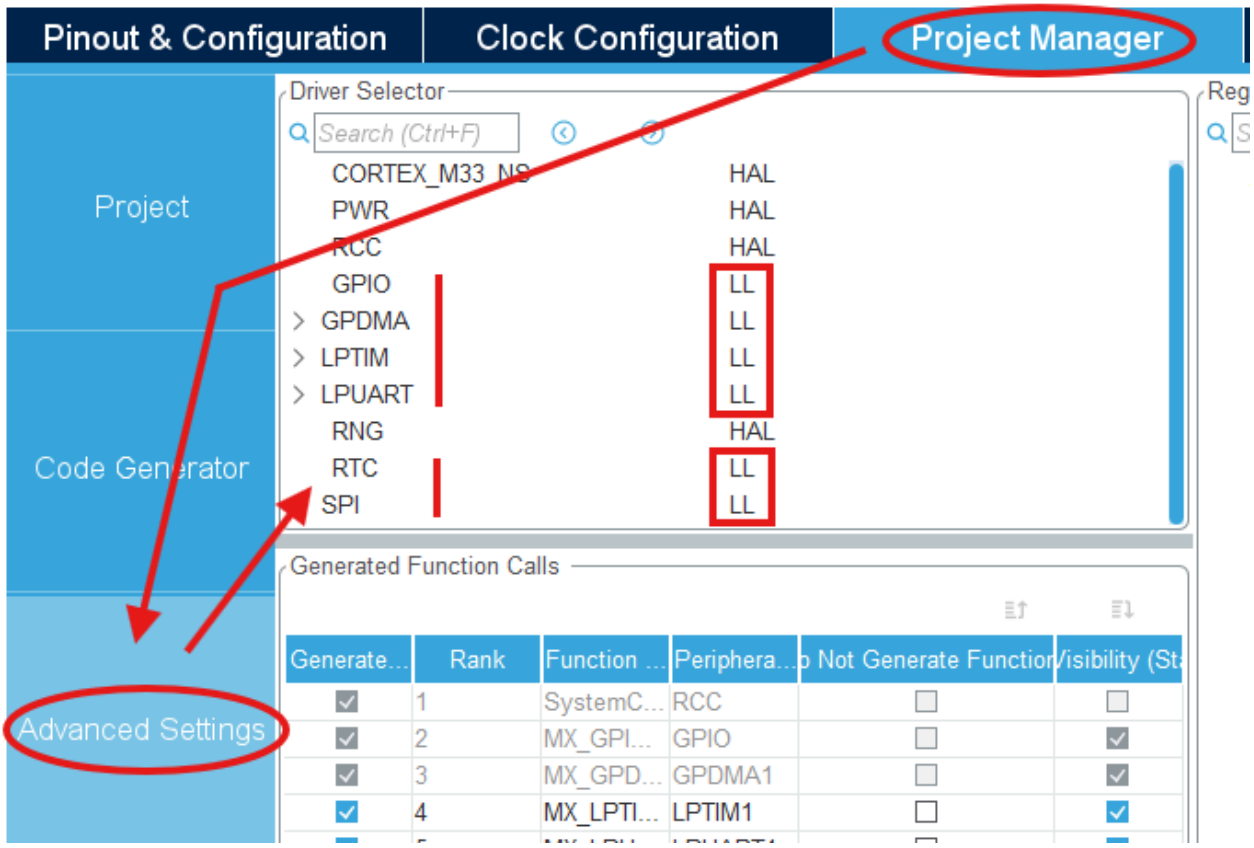


Enable RTC from Timers and leave everything at default:

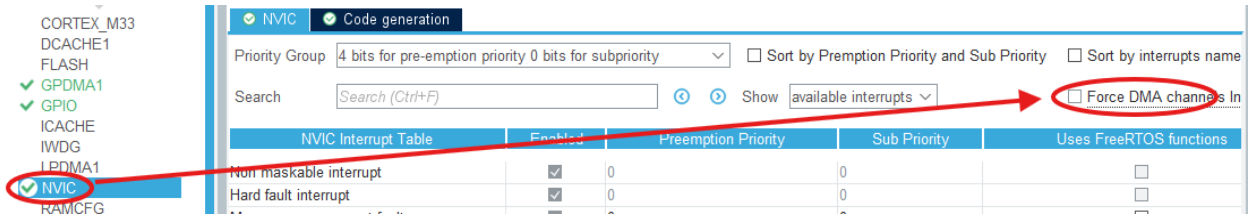


Now change the library type for **GPIO**, **LPUART**, **RTC**, **LPTIM**, **SPI** and **GPDMA** to **LL** from

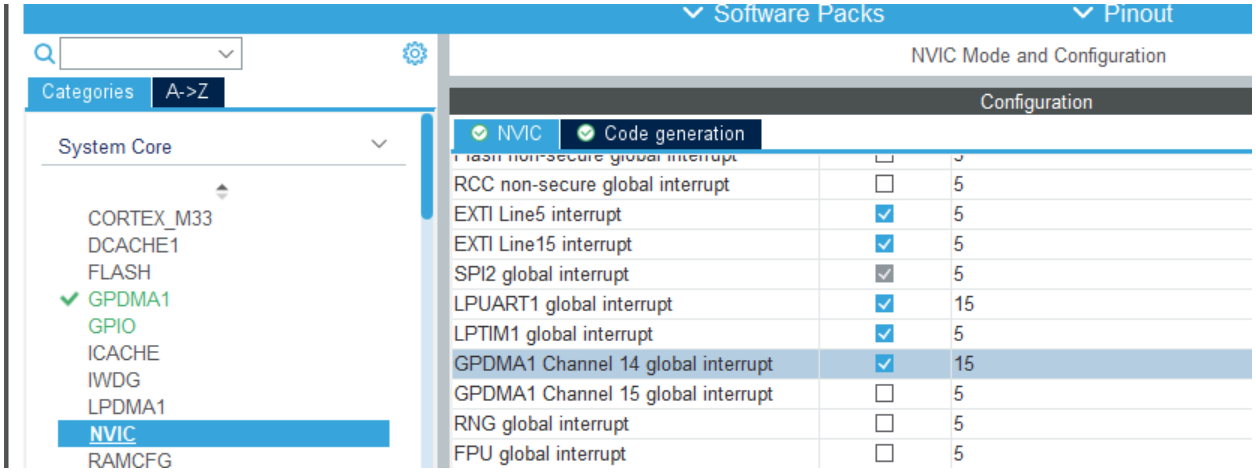
Project Manager -> **Advanced Settings**



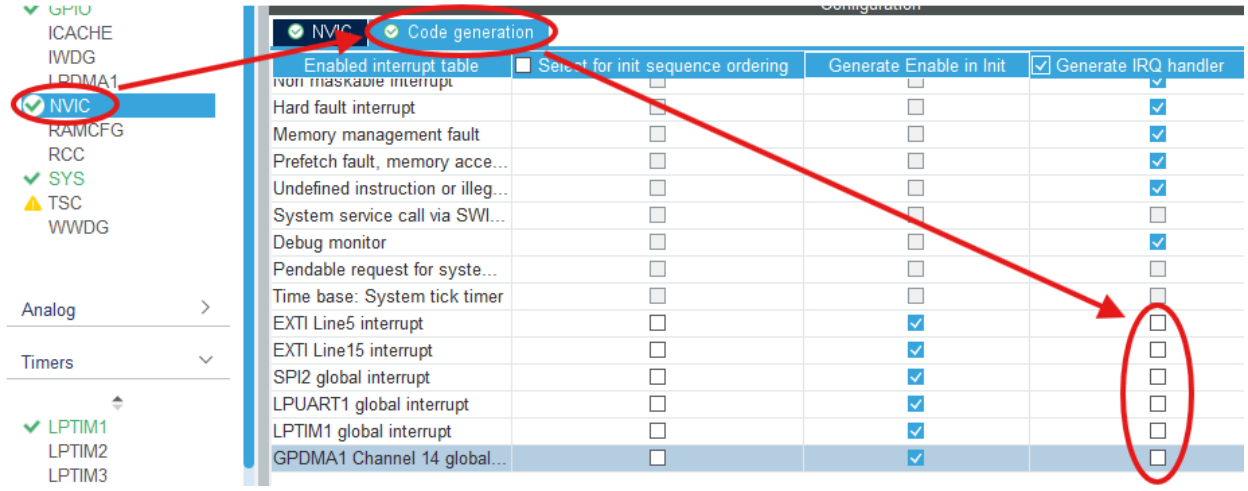
From **NVIC** -> **NVIC**, **uncheck** the tickbox for **Force DMA channel interrupts**



Disable GPDMA1 Channel 15 interrupt ,Enable LPUART1 and LPTIM1 interrupts and reduce the LPUART and GPDMA1 Channel 14 priority to 15.



From **NVIC -> Code generation** , **uncheck** IRQ handler generation for peripherals that we've configured and



Save the project and generate the code from menu **Project -> Generate Code**.

Open the linker script (**STM32U585VITXQ_FLASH.ld**), assign a part of the flash memory to the CONFIG section and add the definition of its boundaries:

```
/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,   LENGTH = 768K
  SRAM4  (xrw)    : ORIGIN = 0x28000000,   LENGTH = 16K
  FLASH  (rx)     : ORIGIN = 0x08000000,   LENGTH = 2032K
  CONFIG (r)      : ORIGIN = 0x081FC000,   LENGTH = 16K
}

/* Sections */
SECTIONS
{
  /* The startup code into "FLASH" Rom type memory */
  .isr_vector :
  {
    KEEP(*(.isr_vector)) /* Startup code */
  } >FLASH

  .mmconfig :
  {
    mmconfig_start   = ORIGIN(CONFIG);
    mmconfig_end     = ORIGIN(CONFIG) + LENGTH(CONFIG);
  }
```

Open the main.h file and add the following line to **USER CODE BEGIN/END EM** section:

```
/* USER CODE BEGIN EM */
#define SPI_PERIPH (SPI2)

#define SPI_DMA_PERIPH (GPDMA1)
#define SPI_RX_DMA_CHANNEL   (LL_DMA_CHANNEL_14)
#define SPI_TX_DMA_CHANNEL   (LL_DMA_CHANNEL_15)

#define SPI_IRQn             (EXTI15_IRQn)
#define SPI_IRQ_LINE         (LL_EXTI_LINE_15)
#define SPI_IRQ_HANDLER      EXTI15_IRQHandler

#define BUSY_IRQn            (EXTI5_IRQn)
#define BUSY_IRQ_LINE        (LL_EXTI_LINE_5)
#define BUSY_IRQ_HANDLER     EXTI5_IRQHandler

#define LOG_USART                (LPUART1)
#define LOG_USART_IRQ            (LPUART1_IRQn)
#define LOG_USART_IRQ_HANDLER    LPUART1_IRQHandler
```

```
#define BCF_DATA_3V3          bcf_aw_hm593
#define BCF_DATA_3V3_LEN      bcf_aw_hm593_len
/* USER CODE END EM */
```

Open main.h and add the mmosal.h header in **USER CODE Includes** section:

```
/* USER CODE BEGIN Includes */
#include "mmosal.h"
/* USER CODE END Includes */
```

Then add the app_init function declaration to **USER CODE FPF** section:

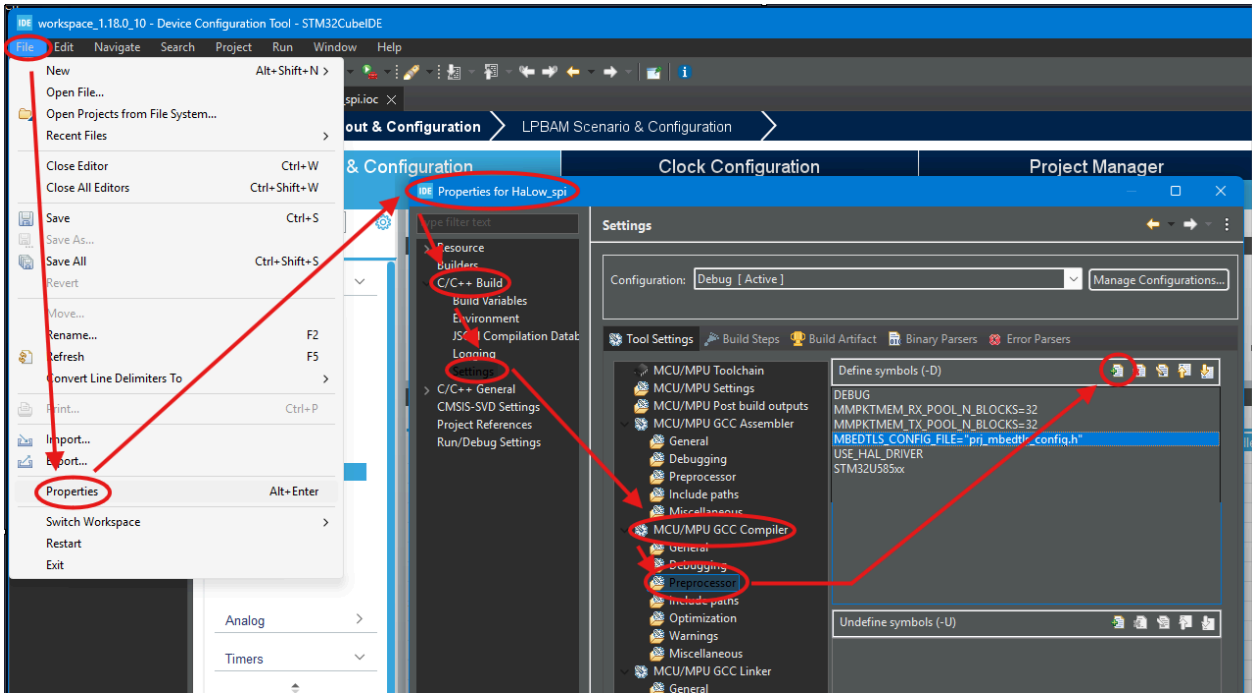
```
/* USER CODE BEGIN PFP */
void app_init(void);
/* USER CODE END PFP */
```

And add following call to **USER CODE 2** section:

```
 /* USER CODE BEGIN 2 */
  LL_DMA_EnableIT_TC(GPDMA1, LL_DMA_CHANNEL_14);
  LL_LPTIM_Enable(LPTIM1);
  LL_LPTIM_ClearFlag_DIEROK(LPTIM1);
  LL_LPTIM_EnableIT_CC1(LPTIM1);
  while (!LL_LPTIM_IsActiveFlag_DIEROK(LPTIM1))
  {
  }
  LL_LPTIM_StartCounter(LPTIM1, LL_LPTIM_OPERATING_MODE_CONTINUOUS);
  /* Enable LPTIM1 autonomous mode to wake up from stop modes. */
  LL_SRDAMR_GRP1_EnableAutonomousClock(LL_SRDAMR_GRP1_PERIPH_LPTIM1AMEN);
  mmosal_main(app_init);
 /* USER CODE END 2 */
```

Please note that this function will start the RTOS kernel and will never return from it.
At the end you need to add some global definitions to your project. Open the Project properties from **Project -> Properties** . In the shown dialog properties, go to **C/C++ build -> Settings -> MCU/MPU GCC Compiler -> Preprocessor** to open the symbols page. You can add global definitions by clicking on **Add...** button. Add the following definitions:

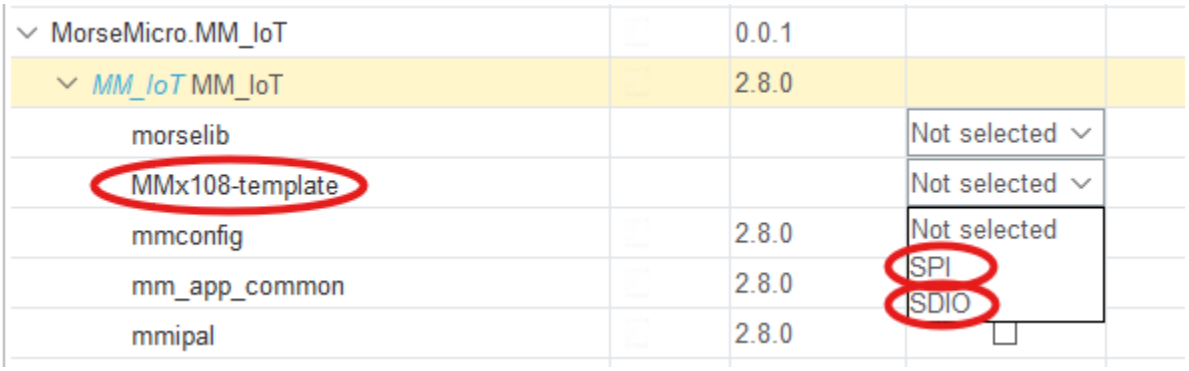
MBEDTLS_CONFIG_FILE="prj_mbedtls_config.h"
MMPKTMEM_TX_POOL_N_BLOCKS=32
MMPKTMEM_RX_POOL_N_BLOCKS=32


workspace_1.18.0_10 - Device Configuration Tool - STM32CubeIDE
File
Edit
Navigate
Search
Project
Run
Window
Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
LPBAM Scenario & Configuration
Clock Configuration
Project Manager
Properties for HaLow_spi
Resource
Builders
C/C++ Build
Build Variables
Environment
Logging
Settings
C/C++ General
CMSIS-SVD Settings
Project References
Run/Debug Settings
Settings
Configuration: Debug [Active]
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
MCU/MPU Toolchain
MCU/MPU Settings
MCU/MPU Post build outputs
MCU/MPU GCC Assembler
General
Debugging
Preprocessor
Include paths
MCU/MPU GCC Compiler
Preprocessor
Optimization
Warnings
Miscellaneous
MCU/MPU GCC Linker
Define symbols (-D)
DEBUG
MMPKTMEM_RX_POOL_N_BLOCKS=32
MMPKTMEM_TX_POOL_N_BLOCKS=32
USE_HAL_DRIVER
STM32U585xx
Undefine symbols (-U)
Analog
Timers

11 Porting a new platform

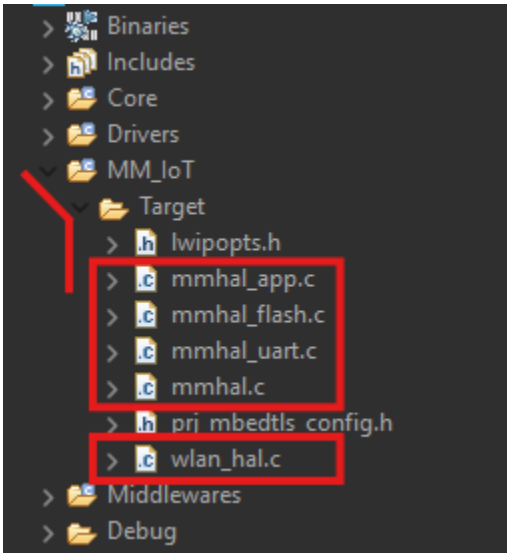
Note: The templates described in this section are available from CMSIS pack release 1.6.0 onwards.

If you are creating a new hardware platform support, You'll need to use the **MMx108-template** component from **MM-IoT CMSIS-Pack** and create your own hardware shim layer.



You can find several examples for different platforms at address below:
https://github.com/MorseMicro/mm-iot-sdk/tree/main/framework/src/platforms .

When using the **MMx108-template** component, you'll find **wlan_hal.c**, **mmhal_app.c, mmhal_uart.c, mmhal_flash.c** and **mmhal.c** files at **MM_IoT/Target** directory in your STM32CubeIDE project.



These files contain a set of empty weak functions (functions that will be replaced by a function with the same name if defined somewhere else) implementations that are required by morselib and the examples from MM-IoT CMSIS-Pack. Please copy these files from **MM_IoT/Target** directory to somewhere else in your project (for example Core directory) before modifying them, otherwise every time that you ask STM32CubeMX to generate code, it will replace these files with a clean file. After copying them, remove the weak attribute from all the functions, so they override the functions in original template files.

wlan_hal.c provides the hardware interface code that is responsible for handling SPI or SDIO read/write and interrupt handling. To find more details on how to implement each function, find the description for each function in **Middlewares \ Third_Party \ MorseMicro_MM_IoT_MM_IoT \ morselib \MMx108 \ with_wpa_supplicant \ include \ mmhal_wlan.h**

mmhal.c contains the code that morselib needs for handling system level functionality that are related to hardware such as sleeping. To find more details on how to implement each function, find the description for each function in **Middlewares \ Third_Party \ MorseMicro_MM_IoT_MM_IoT \ morselib \MMx108 \ with_wpa_supplicant \ include \ mmhal.h**.

mmhal_app.c handles the application related hardware functions such as buttons, leds, rtc, … . To find more details on how to implement each function find the description for each function, in **Middlewares \ Third_Party \ MorseMicro_MM_IoT_MM_IoT \ morselib \MMx108 \ with_wpa_supplicant \ include \ mmhal.h**

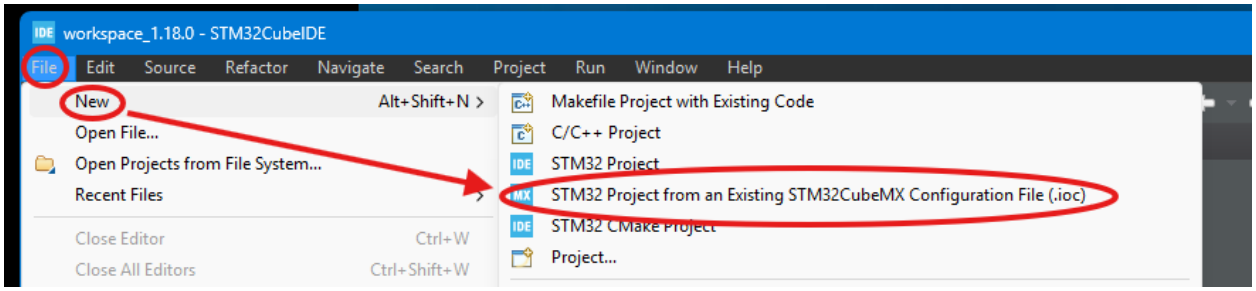
mmhal_uart.c is only needed if you need the rf-test example to work. This file provides the UARTmid layer to pass the data between the app and the UART. To find more details on how to implement each function find the description for each function in **Middlewares \ Third_Party \ MorseMicro_MM_IoT_MM_IoT \ morselib \MMx108 \ with_wpa_supplicant \ include \ mmhal_uart.h**

mmhal_flash.c is only needed if you need the mmconfig component to be able to change or write new configurations into a non-volatile memory. This file contains the implementation of read/write/erase functions of a non-volatile memory . To find more details on how to implement each function find the description for each function in **Middlewares \ Third_Party \ MorseMicro_MM_IoT_MM_IoT \ morselib \MMx108 **

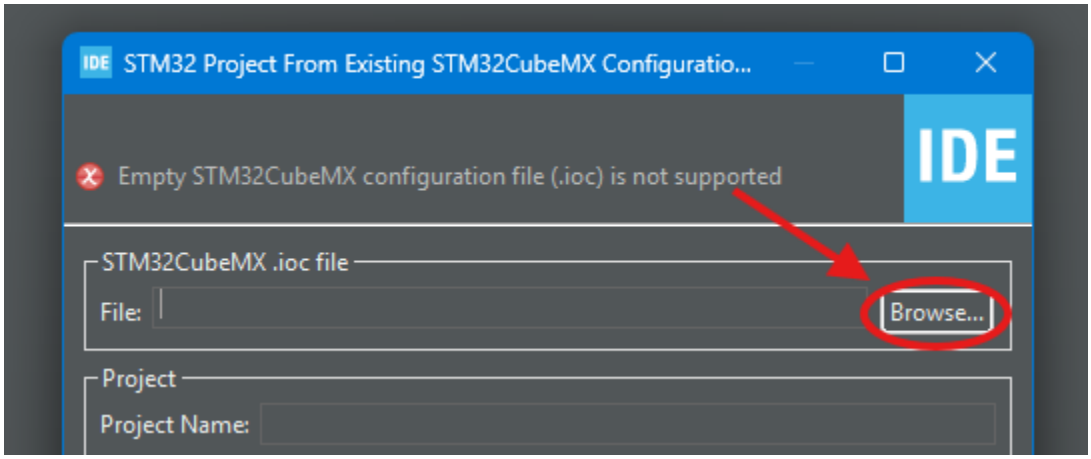
with_wpa_supplicant \ include \ mmhal_flash.h

12 Importing a platform from mm-iot-sdk

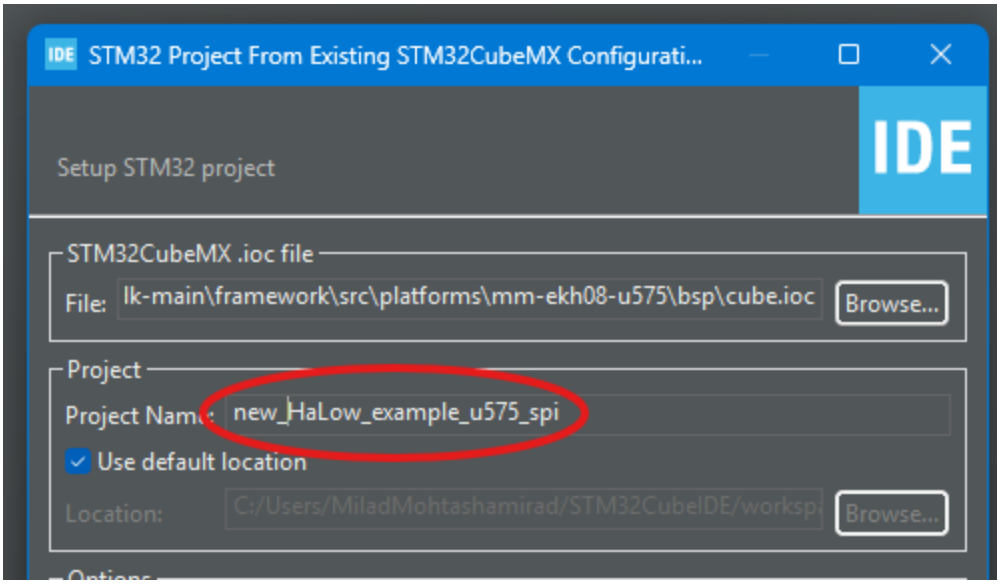
In the chapter the process to import a platform from https://github.com/MorseMicro/mm-iot-sdk/tree/main/framework/src/platforms to STM32cubeIDE to be used with CMSIS-Pack is explained. As an example, we import the mm-ekh08-u575 platform.
(Please make sure that the Morse Micro MM-IoT CMSIS-Pack is installed before proceding.)
Download https://github.com/MorseMicro/mm-iot-sdk to your computer.
Start a new project by selecting **STM32 Project from an Existing STM32CubeMx Configuration file (.ioc)** from **File -> New**



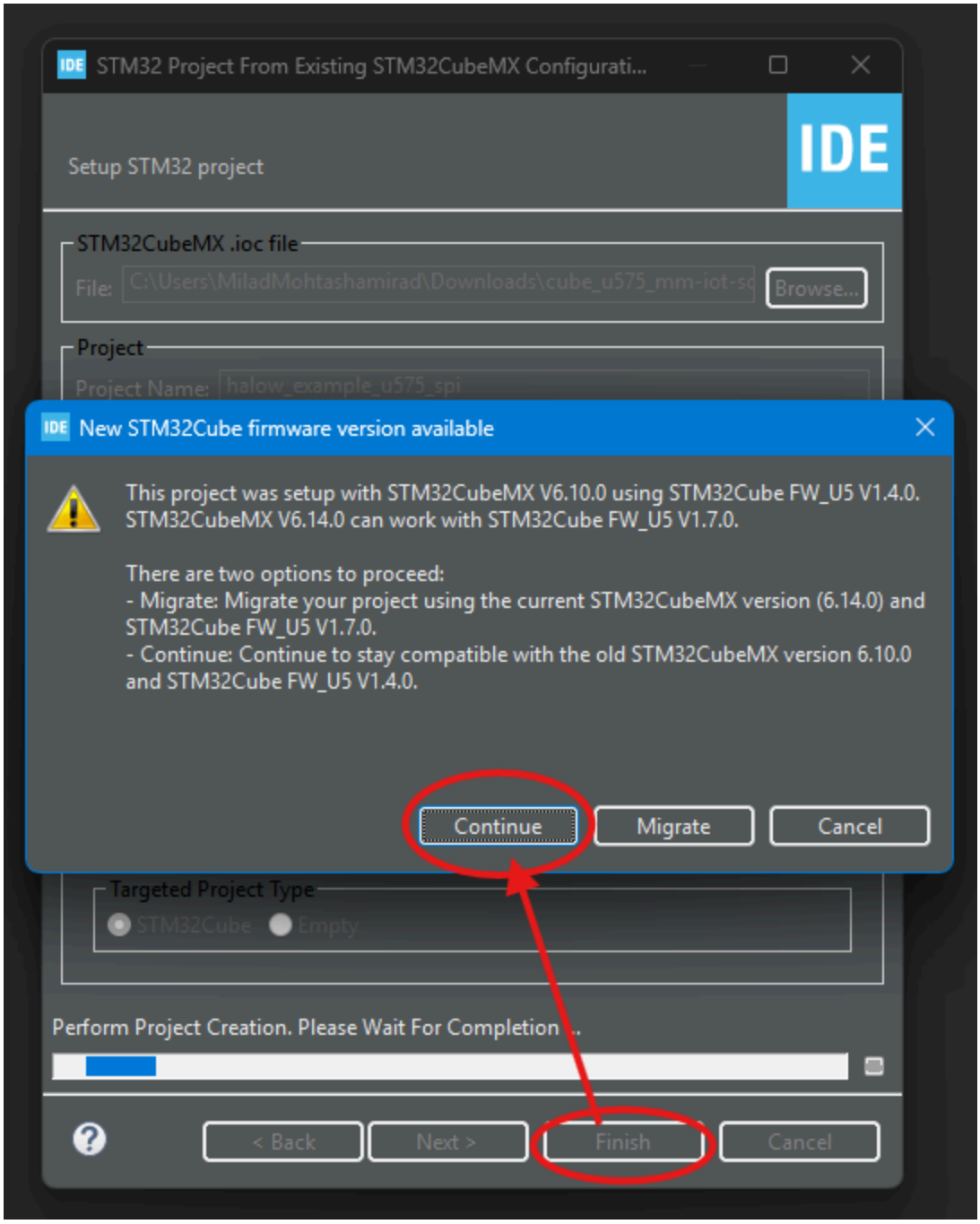
Click **Browse** and select the cubeMX project file at framework\src\platforms\mm-ekh08-u575\bsp\cube.ioc inside the mm-iot-sdk:



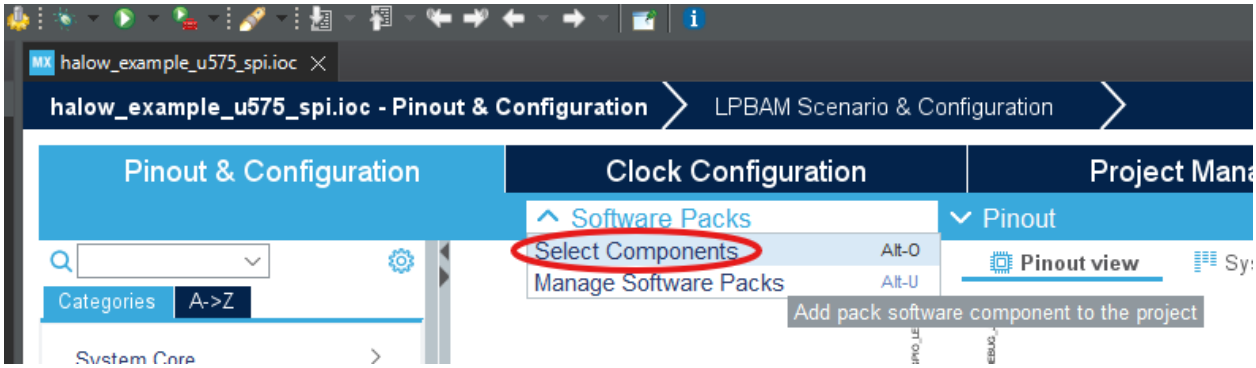
Choose a Project name and click **Finish:**



If you encounter a message regarding updating the versions of the packages, press **Continue** to keep the same versions:

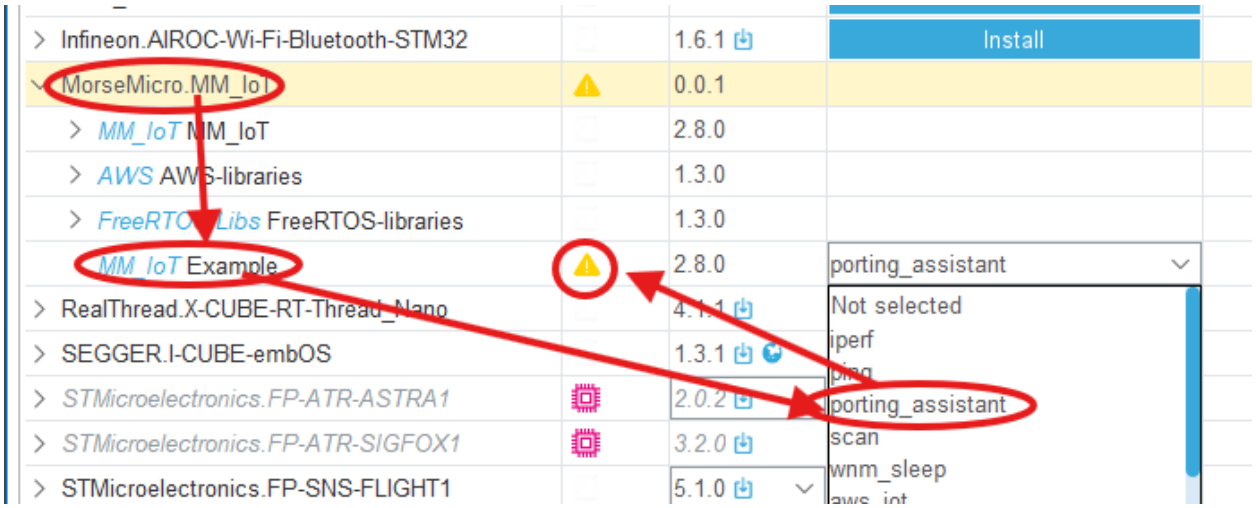


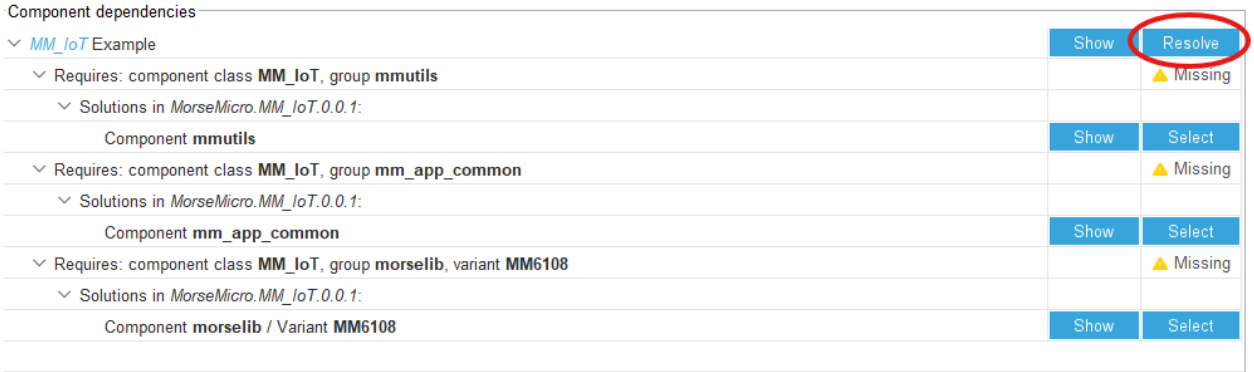
After the project is created, open the **Software Packs Component Selector** from **Software Packs -> Select Componets** or by pressing **Alt+O** keys:



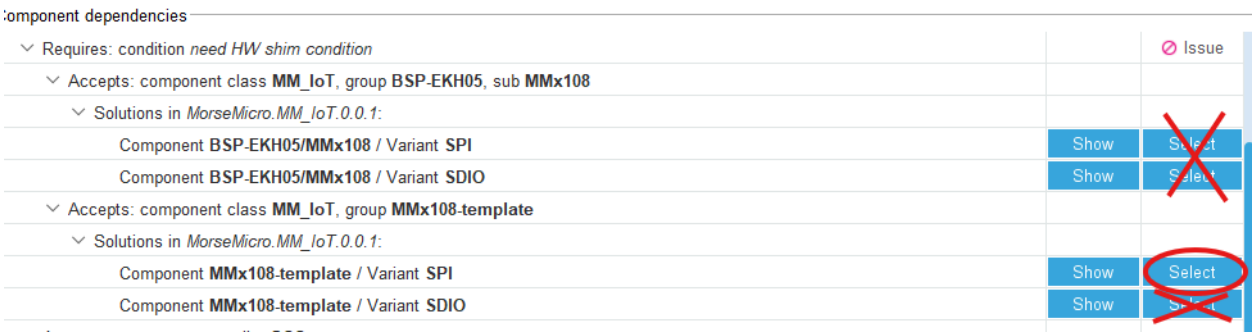
Then select **porting_assistant** from **MorseMicro.MM_IoT -> MM_IoT example** dropdown.

Click on the yellow warning sign to highlight the missing components. You can click resolve to let the cubeMX select the missing components as much as it can.

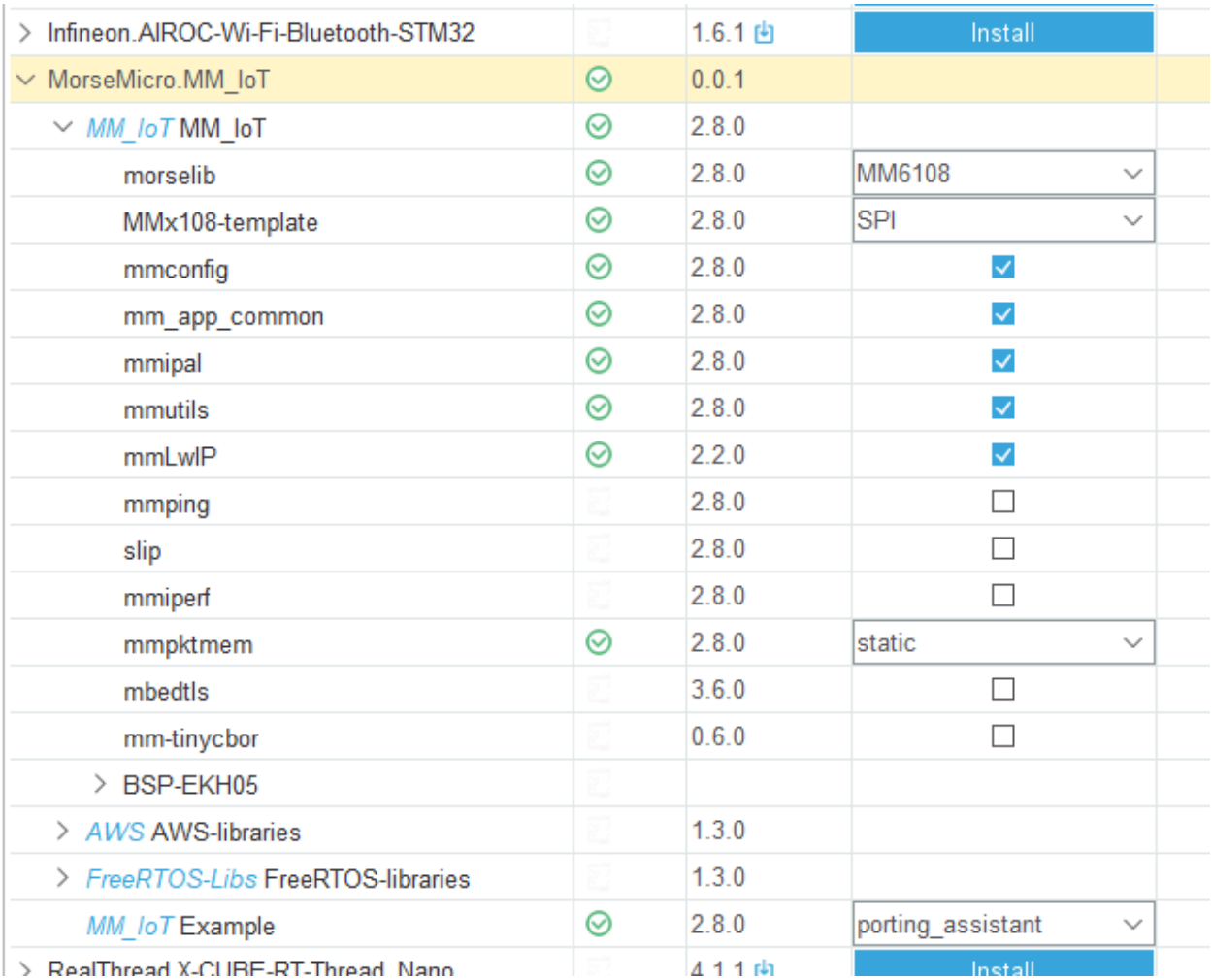




You need to do the same for the rest of the components that have missing dependencies. For morselib choose **MM_IoT** -> **MMx6108 -template**/Variant **SPI** .

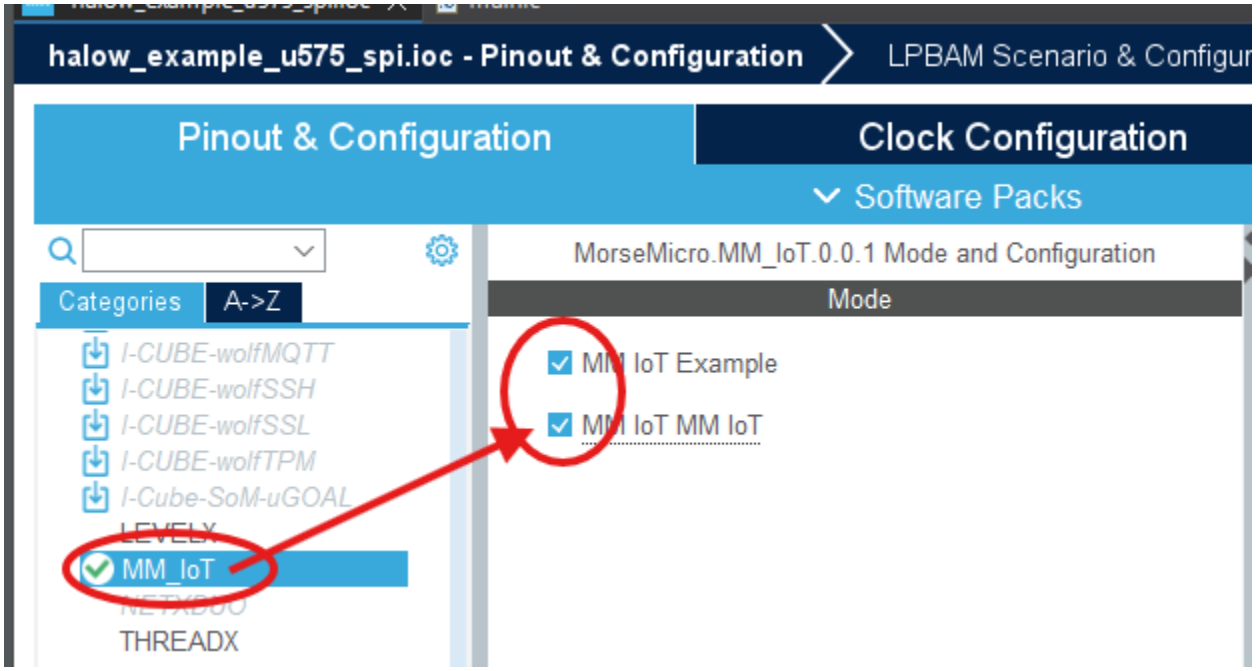


Your component selection in **MorseMicro.MM_IoT** should look like as follow:

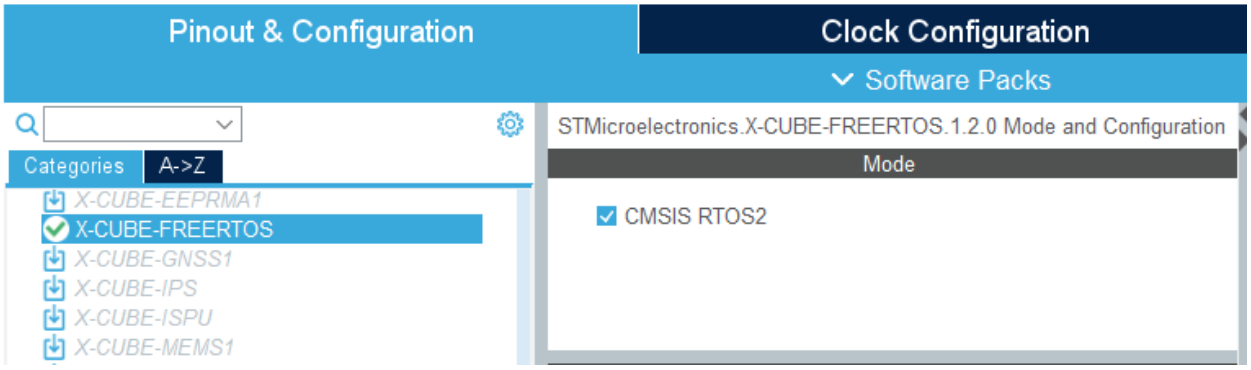


Click **OK** to close the component selector dialog.

Enable MM_IoT components by selecting all the check boxes from **Pinout & Configuration ->MM_IoT :**



Enable **CMSIS RTOS2** from **X-CUBE-FREERTOS**

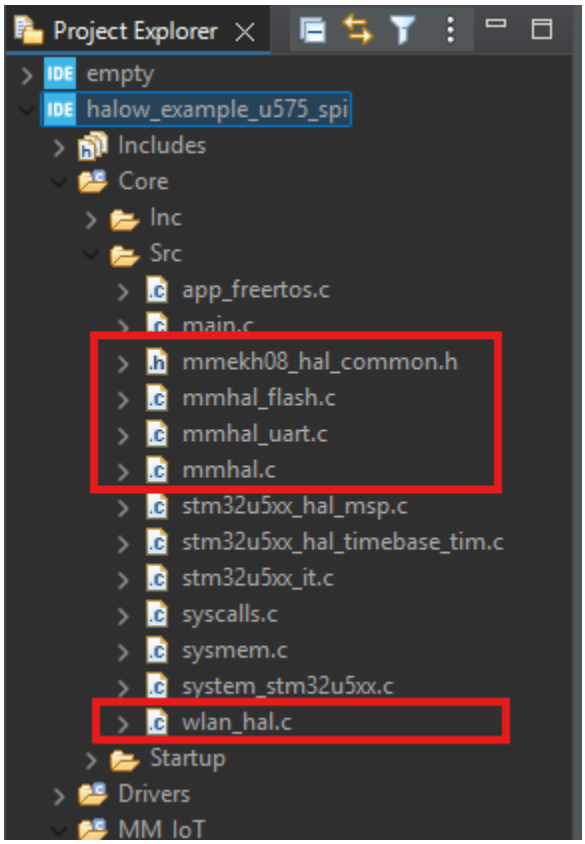


And set the the parameters as follow:

ENABLE_FPU	Enabled
MINIMAL_STACK_SIZE	256

USE_TICKLESS_IDLE	User defined functionality enabled
TOTAL_HEAP_SIZE	95*1024

From framework\src\platforms\mm-ekh08-u575\mm_shims inside mm-iot-sdk, copy **mmekh08_hal_common.h, mmhal_flash.c, mmhal_uart.c, mmhal.c** and **wlan_hal.c** into your project.



Also copy the lines inside USER CODE in main.c and main.h from mm-iot-sdk.

(Alternatively, replace main.c and main.h with ones from mm-iot-sdk to your project
RE-Generate the code.)

Open the linker script (STM32U575ZITXQ_FLASH.ld), assign a part of the flash memory to the CONFIG section and add the definition of its boundaries:

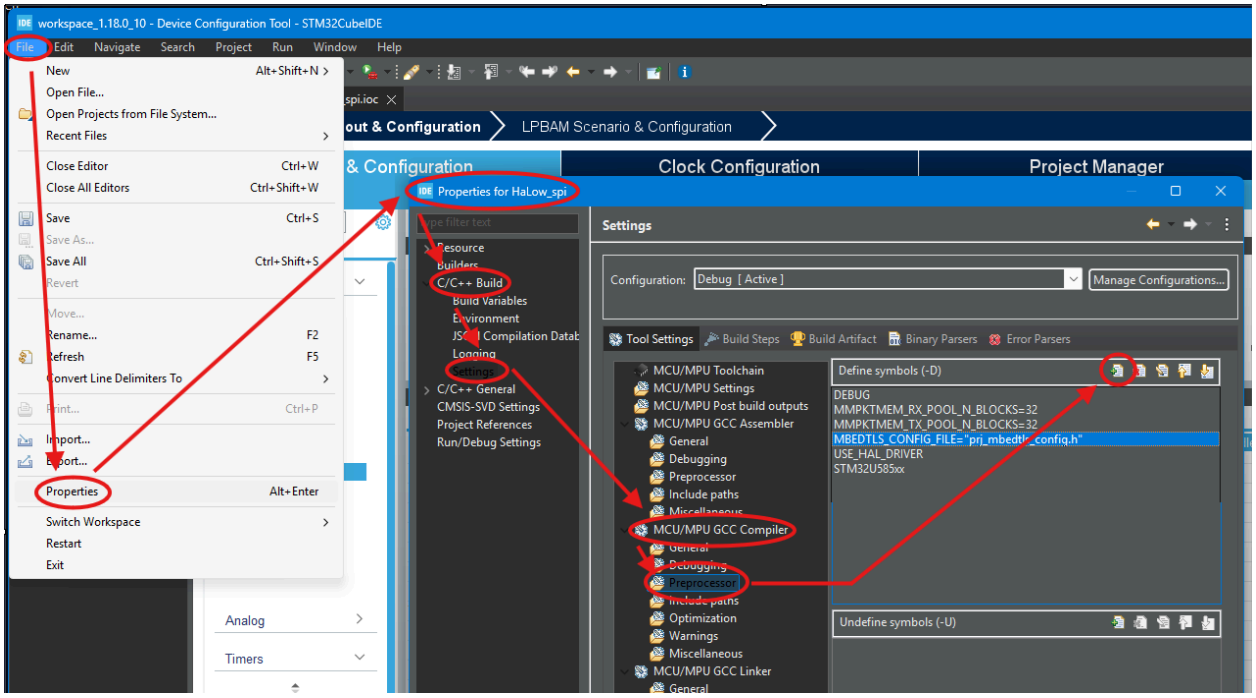
```
/* Memories definition */
MEMORY
{
  RAM    (xrw)    : ORIGIN = 0x20000000,  LENGTH = 768K
  SRAM4 (xrw)    : ORIGIN = 0x28000000,  LENGTH = 16K
  FLASH   (rx)    : ORIGIN = 0x08000000,  LENGTH = 2032K
  CONFIG (r)     : ORIGIN = 0x081FC000,  LENGTH = 16K
}

/* Sections */
SECTIONS
{
  /* The startup code into "FLASH" Rom type memory */
  .isr_vector :
  {
    KEEP(*(.isr_vector)) /* Startup code */
  } >FLASH

  .mmconfig :
  {
    mmconfig_start   = ORIGIN(CONFIG);
    mmconfig_end     = ORIGIN(CONFIG) + LENGTH(CONFIG);
  }
```

Add the necessary global definitions to your project. Open the Project properties from **File -> Properties** . In the shown dialog properties, go to **C/C++ build -> Settings -> MCU/MPU GCC Compiler -> Preprocessor** to open the symbols page. You can add global definitions by clicking on **Add...** button. Add the following definitions:

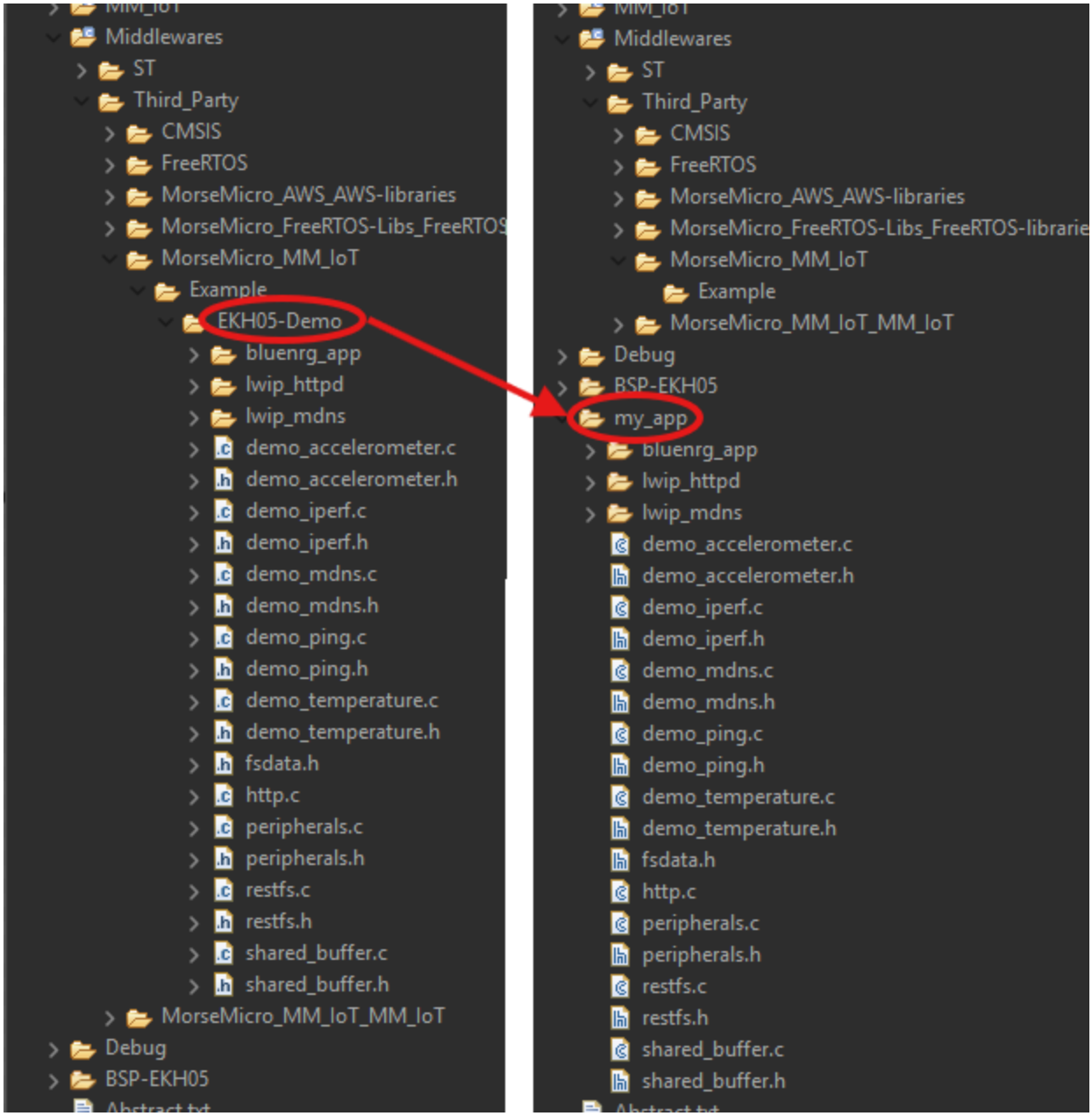
MBEDTLS_CONFIG_FILE="prj_mbedtls_config.h"
MMPKTMEM_TX_POOL_N_BLOCKS=32
MMPKTMEM_RX_POOL_N_BLOCKS=32



Save everything, re-generate the code and build the project.

13 Modifying an existing example to make a new application

If you need to modify an existing example to create a new application, the best way to do it is to copy the example code from the generated directories and put them somewhere else that doesn't get overwritten every time that you generate the code (for example in the root directory) . For example, imagine that you've opened the EKH05-Demo example in your workspace and you can build it and now you want to modify its code. From the project tree, copy the EKH05-Demo from **Middlewares/Third_Party/MorseMicro_MM_IoT/Example/EKH05-Demo** and paste it in the root directory and delete the original code. You can also rename the directory:



And then you need to disable the example component from component selector to avoid code duplication when you regenerate the code:


Infineon.AIROC-Wi-Fi-Bluetooth-STM32
1.6.1
Install
MorseMicro.MM_IoT
1.7.1
MM_IoT MM_IoT
2.9.3
AWS AWS-libraries
1.3.0
FreeRTOS-Libs FreeRTOS-libraries
1.3.0
MM_IoT Example
Not selected
RealThread.X-CUBE-RT-Thread_Nano
4.1.1
Not selected
iperf
SEGGER.I-CUBE-embOS
1.3.1

14 Revision History

Release Number	Release Date	Release Notes
04	05-Feb-2026	• Update for 1.8.2 release, including new method for switching between MM6108 and MM8108, and simpler changing between examples.
03	01-Sep-2025	• Update rf-test section. • Added a section for modifying the examples. • Added a section for cli example
02	14-May-2025	• Added section for switching between MM6108 and MM8108 • Added login instruction • Added github release page link • Add a note to refer to MM-IoT-SDK documentation for examples.
01	14-Apr-2025	• Initial Release

Morse Micro
reaching farther™

Morse Micro Pty. Ltd. - HQ
Level 8, 10-14 Waterloo Street,
Surry Hills, NSW 2010, AUSTRALIA

Morse Micro Inc. – USA
40 Waterworks Way
Irvine, CA 92618, UNITED STATES

Office Locations:
Bangalore, India
Cambridge, UK
Hangzhou, China
Irvine, CA - USA
Picton, NSW - Australia
Portola Valley, CA - USA
Shenzhen, China
Taipei, Taiwan

sales@morsemicro.com

www.morsemicro.com

About Morse Micro
Morse Micro is a leading Wi-Fi HaLow fabless semiconductor company based in Sydney, with global offices. As the world's premier Wi-Fi HaLow company, we pioneer next-gen IoT wireless connectivity solutions. Morse Micro is now sampling its Wi-Fi CERTIFIED HaLow MM6108 production silicon: the fastest, smallest, lowest power and longest-range Wi-Fi HaLow chip available in the market.

Morse Micro

reaching farther™