AIROC™ Wi-Fi/Bluetooth® STM32 Expansion pack user guide

About this document

Version

1.8.0

Scope and purpose

AIROC™ Wi-Fi/Bluetooth® [1] STM32 Expansion Pack from Infineon is an extension of the CMSIS-Pack standard established by Arm. The pack is compliant with the full CMSIS-Pack standard, with additional requirements/restrictions on the final pack to meet the STM standard. This pack uses libraries from the ModusToolbox™ environment. For more details, refer to https://www.infineon.com/cms/en/design-support/tools/sdk/modustoolbox-software. You can select and configure the pack in the STM32CubeMX tool, make choices appropriate for your design, such as which CYW43xxx/CYW55xxx device to use, and then generate a project from your selection.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus
Italics	Denotes file names and paths.
`Courier New`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
File > New	Indicates that a cascading sub-menu opens when you select a menu item

Abbreviations and definitions

The following define the abbreviations and terms used in this document:

- BSP – Board Support Package
- PAL – Platform Adaptation Layer
- WCM – Wi-Fi Connection Manager
- WHD – Wi-Fi Host Driver

[1] Bluetooth® is a registered trademark owned by Bluetooth SIG Inc.

Table of contents

Table of contents

1 Expansion pack contents

The following table shows the components and their versions included with the expansion pack:

Component name	Version	Details
abstraction-rtos	1.8.0	The RTOS abstraction layer provides simple RTOS services like threads, semaphores, mutexes, queues, and timers. It is not intended to be a full features RTOS interface, but they provide just enough support to allow for RTOS independent drivers and middleware.
btstack-integration	6.2.1	AIROC™ Bluetooth® host stack solution includes Bluetooth® stack library, Bluetooth® controller firmware and platform/OS porting layer. This component is compatible with Theadx as well.
btstack	4.1.4	BTSTACK is Infineon's Bluetooth® Host Protocol Stack implementation. The stack is optimized to work with Infineon Bluetooth® controllers. The BTSTACK supports Bluetooth® BR/EDR and Bluetooth® LE core protocols.
command-console	6.0.0	This library provides a framework to add command console to your application and support Wi-Fi, iPerf, Bluetooth® Low Energy commands.
connectivity-utilities	4.4.0	The connectivity utilities library is a collection of general-purpose middleware utilities such as: JSON parser, Linked list, String utilities, Network helpers, Logging functions, and Middleware Error codes. Several connectivity middleware libraries will depend on this library.
core-lib	1.3.1	The Core Library provides basic types and utilities that can be used between different devices. This allows different libraries to share common items between themselves to avoid reimplementation and promote consistency.
device	1.8.0	Selects appropriate CYW43xxx/CYW55xxx firmware and drivers for selected connectivity device.
lwIP	2.1.2	lwIP is a small independent implementation of the TCP/IP protocol suite. The focus of the lwIP TCP/IP implementation is to reduce the RAM usage while still having a full-scale TCP. This making lwIP suitable for use in embedded systems with tens of kilobytes of free RAM and room for around 40 kilobytes of code ROM.
pal	1.8.0	Infineon-STM32 Platform Adaptation Layer (PAL).
wifi-host-driver	4.1.0	The Wi-Fi host driver (WHD) is an independent, embedded driver that provides a set of APIs to interact with Infineon WLAN chips. The WHD is an independent firmware product that is easily portable to any embedded software environment. Therefore, the WHD includes hooks for RTOS and TCP/IP network abstraction layers.
wcm	3.6.0	The Wi-Fi Connection Manager (WCM) is a library which helps application developers to manage Wi-Fi Connectivity. The library provides a set of APIs that can be used to establish and monitor Wi-Fi connections on Infineon platforms that support Wi-Fi connectivity.
whd-bsp-integration	2.3.0	The WHD library provides some convenience functions for connecting to a Board Support Package (BSP) that includes a WLAN chip. This library initializes the hardware and passes a reference to the communication interface on the board into WHD. It also sets up the LwIP based network buffers to be used for sending packets back and forth.
netxduo-network-interface-integration	1.3.0	This library is an integration layer that links the NetXDuo network stack with the underlying WHD. This library interacts with ThreadX, NetXDuo TCP/IP stack, and WHD. It contains the associated code to bind these components together.
lwip-network-interface-integration	1.3.1	This library is an integration layer that links the lwIP network stack with the underlying WHD and Ethernet driver. This library interacts with FreeRTOS, lwIP TCP/IP stack, WHD, and Ethernet driver. It contains the associated code to bind these components together.

Expansion pack contents

Component name	Version	Details
lwip-freertos-integration	1.0.0	This library contains the FreeRTOS dependencies needed by the Lightweight open-source TCP/IP stack, version: 2.1.2 to execute. See the https://savannah.nongnu.org/projects/lwip/ web site for details.
wifi-mfg-test	3.3.0	The WLAN Manufacturing Test Middleware application is used to validate the WLAN firmware and radio performance of the Wi-Fi device. The mfg-test middleware repo can accept the serial input byte stream from the Mfg Test application and transform the contained commands into IOVAR/IOCTL messages to the WLAN firmware. It can get the response from the WLAN firmware (if expected) and transport it back to the 'wl tool' running on the host.
secure-sockets	3.0.0	The secure sockets library provides APIs to create software that can send and/or receive data over the network using sockets. This library supports both secure and non-secure sockets, and abstracts the complexity involved in directly using network stack and security stack APIs. This library supports both IPv4 and IPv6 addressing modes for UDP and TCP sockets.
stm32_mw_freertos	10.4.6	The stm32_mw_freertos MCU component repository is common to all STM32Cube MCU embedded software packages, providing the FreeRTOS Middleware part.
wpa3-external-supplicant	1.1.0	The WPA3 External Supplicant supports WPA3 SAE authentication using HnP (Hunting and Pecking Method) using RFC https://datatracker.ietf.org/doc/html/rfc7664 and H2E (Hash to Element Method) using RFC https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-10 and following 802.11 spec 2016.
mqtt	4.3.0	This library uses the AWS IoT device SDK MQTT client library and implements the glue layer that is required to work with Infineon connectivity platforms.
aws-iot-device-sdk-port	2.4.0	This library is a port layer implementation for the Infineon MQTT and HTTP Client libraries to work with the AWS-IoT-Device-SDK-Embedded-C library on Infineon connectivity-enabled MCU platforms. These library APIs are not expected to be called by the application directly. See the MQTT and HTTP Client library documentation for more details.
aws-iot-device-sdk-embedded-C	202103	The AWS IoT Device SDK for Embedded C (C-SDK) is a collection of C source files under the MIT open source license that can be used in embedded applications to securely connect IoT devices to the AWS IoT Core. It contains MQTT client, HTTP client, JSON Parser, AWS IoT Device Shadow, AWS IoT Jobs, and AWS IoT Device Defender libraries.

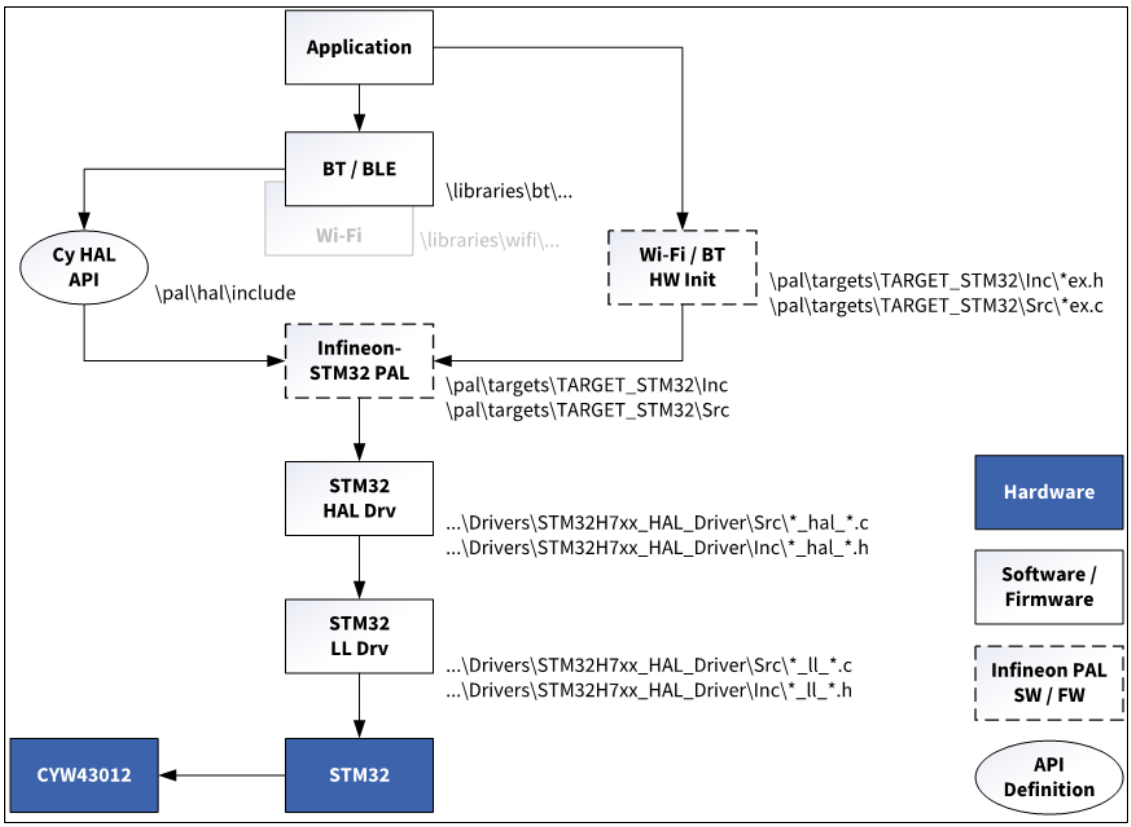
1.1 Infineon-STM32 Platform Adaptation Layer (PAL)

The Infineon-STM32 PAL is based on the STM32 Driver MCU Component HAL, and it offers the minimum set of (required) APIs for Infineon-STM32 PAL. The supported HAL versions are:

STM32Cube HAL package	STM32Cube MCU verified package version
STM32H7 Series	1.12.1
STM32L5 Series	1.5.1
STM32U5 Series	1.8.0
STM32H5 Series	1.5.1
STM32N6 Series	1.2.0

Expansion pack contents

The PAL integrates the STM32 HAL APIs underneath the Infineon HAL APIs expected by the Infineon Connectivity Libraries. The following figure shows the architectural intent of the Infineon-STM32 PAL:



We created the Infineon-STM32-PAL to meet the following guidelines:

- Developers will continue to use STM32CubeMX and/or STM32 HAL APIs to configure STM32 MCU hardware.
- Developers will communicate to the PAL what STM32 hardware that they have selected and configured for communicating with a CYW43xxx/CYW55xxx via an initialization API.
- Infineon-STM32 PAL adapts only the minimum set of Infineon HAL APIs to STM32 HAL in order to communicate and control Infineon's CYW43xxx/CYW55xxx connectivity device(s).
- The Infineon PAL layer behaves like the Infineon HAL as much as possible to minimize impact to the Infineon libraries. The Infineon PAL adapts the following STM32 HAL Drivers:
  - GPIO
  - LPTimer
  - SDIO
  - SPI
  - TRNG
  - UART

1.2 Supported STM32 MCUs

- STM32H7xx
- STM32L5xx
- STM32U5xx
- STM32H5xx

Expansion pack contents

- STM32N6xx

1.3 Supported STM32 boards

- STM32H747I-DISCO Discovery kit
- STM32L562E-DK
- STM32U575I-EV
- NUCLEO-H563ZI
- STM32N6570-DK

1.4 Supported connectivity modules

Infineon's AIROC™ CYW43xxx/CYW55xxx Wi-Fi-Bluetooth® combo chip family:

- CYW43012
- CYW43022
- CYW43439
- CYW4373 / CYW4373/E
- CYW55500
- CYW55572

1.5 Compatible software

- STM32 CubeMX 6.15.0
- STM32 CubeIDE 1.19.0
- STM32 CubeProgrammer 2.20.0 (mandatory to generate trusted binary for STM32N6)
- IAR EWARM 9.30.1

2 Download/install/import expansion pack

2.1 Downloading the pack

Download the expansion pack from GitHub:

https://github.com/Infineon/AIROC-Wi-Fi-Bluetooth-STM32/releases/tag/release-v1.8.0

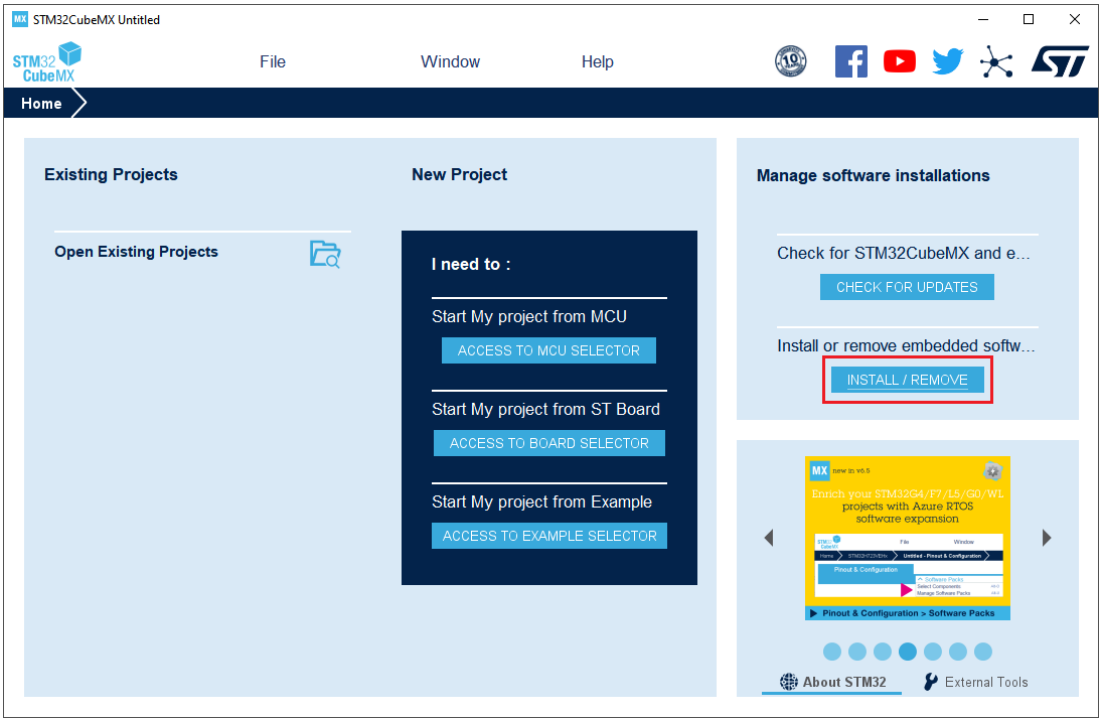
For the AIROC™ CYW5553x device, please contact the FAE to download the latest release.

2.2 Installing/importing the pack

2.2.1 Add from local file

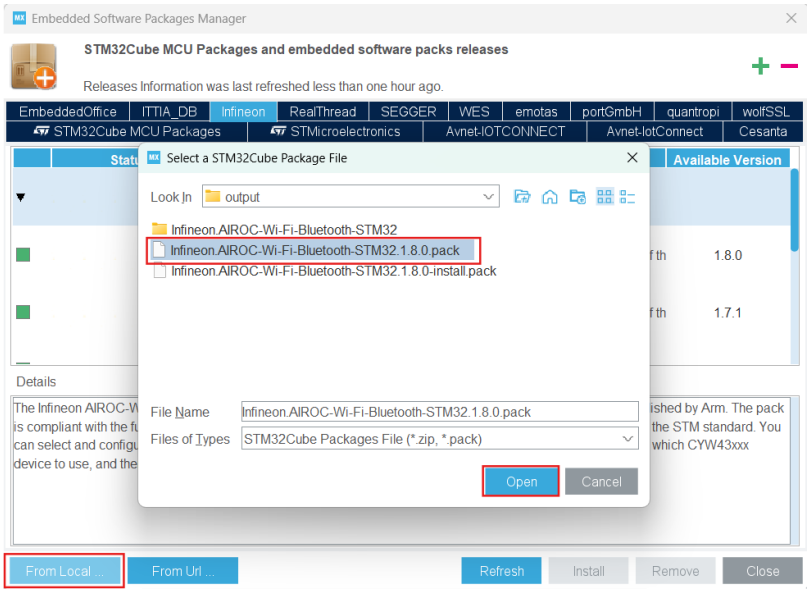
Perform these steps to add the expansion pack to the STM32 development environment:

1. Run the STM32CubeMX tool.
2. Navigate to **Home > Manage software installations** and select **Install/Remove**.

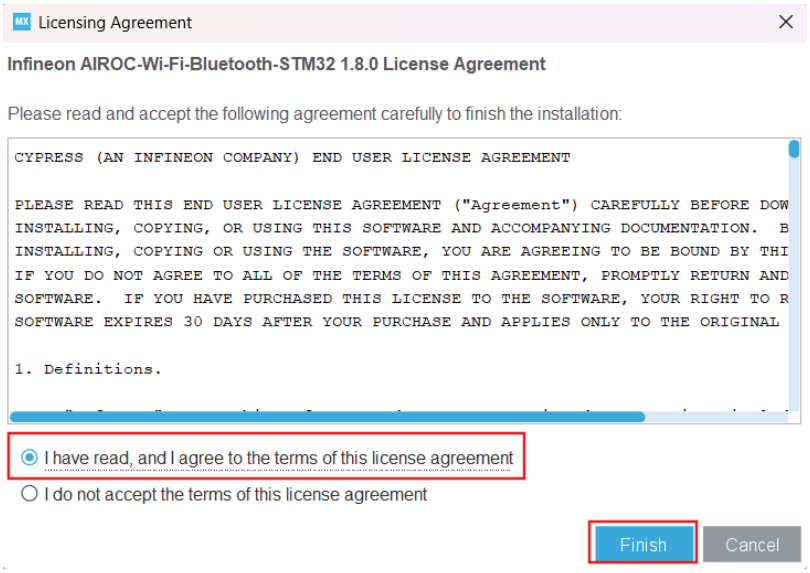


Download/install/import expansion pack

3. Select **From Local...**, navigate to the downloaded pack file, and select **Open**.

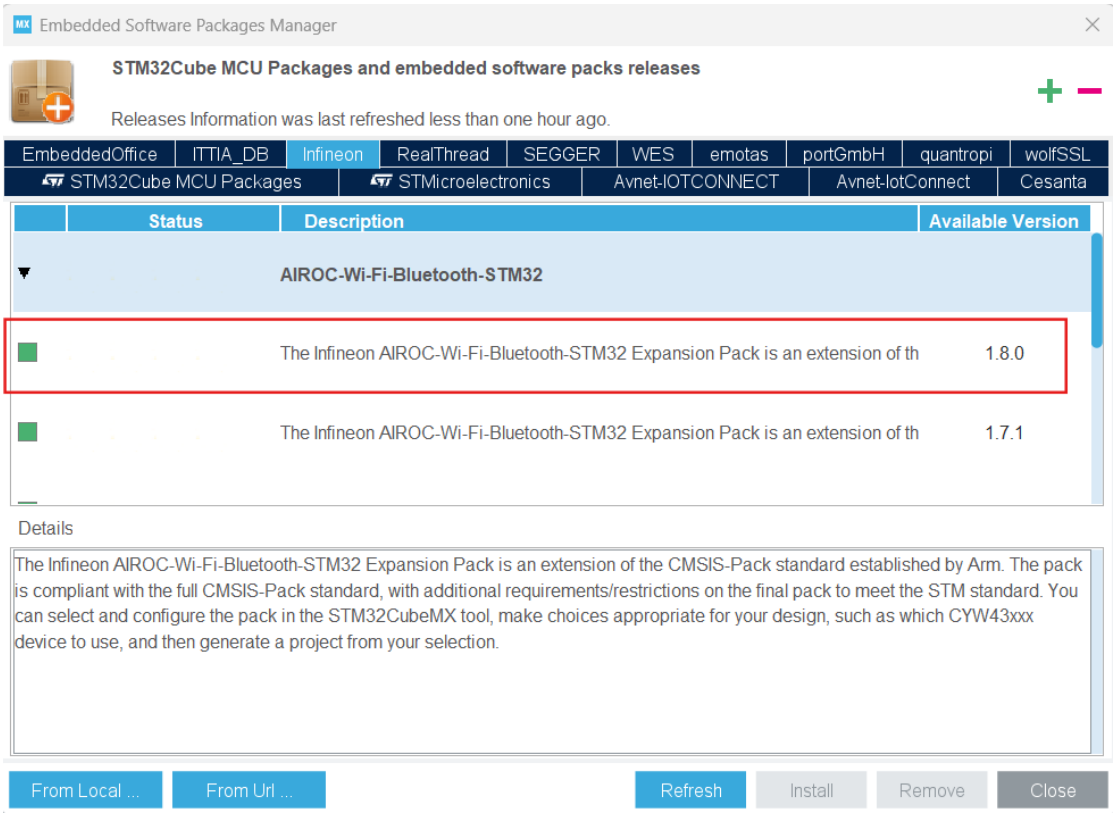


4. Accept the license agreement and select **Finish**.



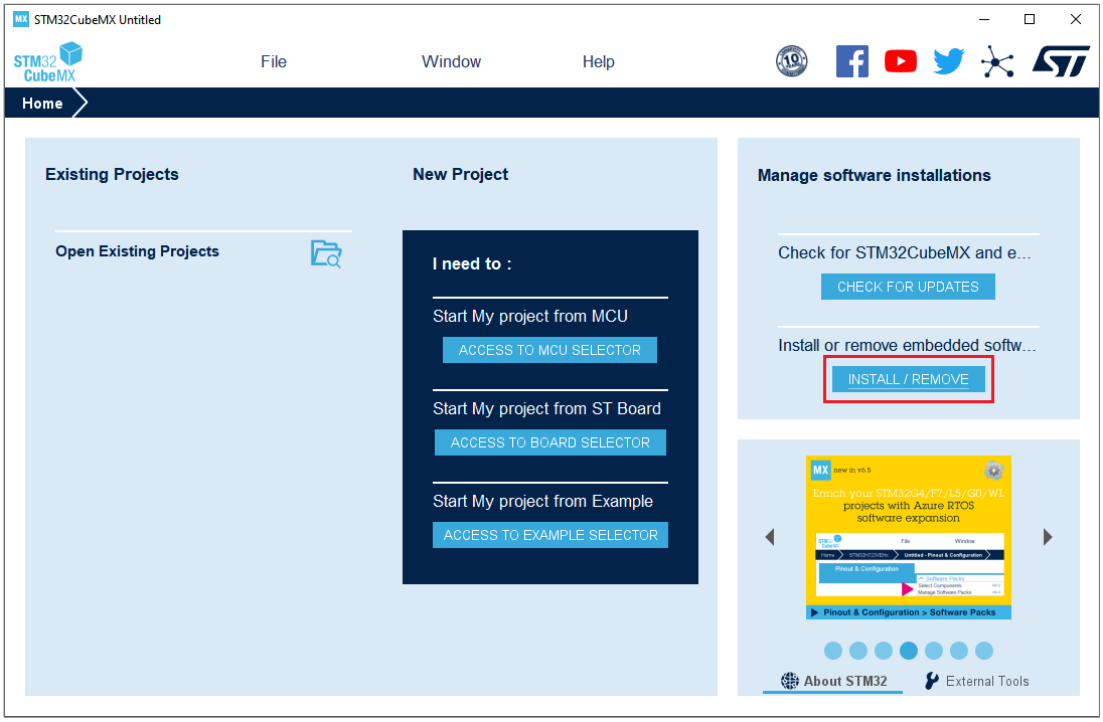
5. The tool shows an **Infineon** tab with the installed Expansion Pack displayed. Click **Close**.

Download/install/import expansion pack



2.2.2 Add the Pack from URL

1. Run the STM32CubeMX tool.
2. Navigate to **Home > Manage software installations** and select **Install/Remove**.



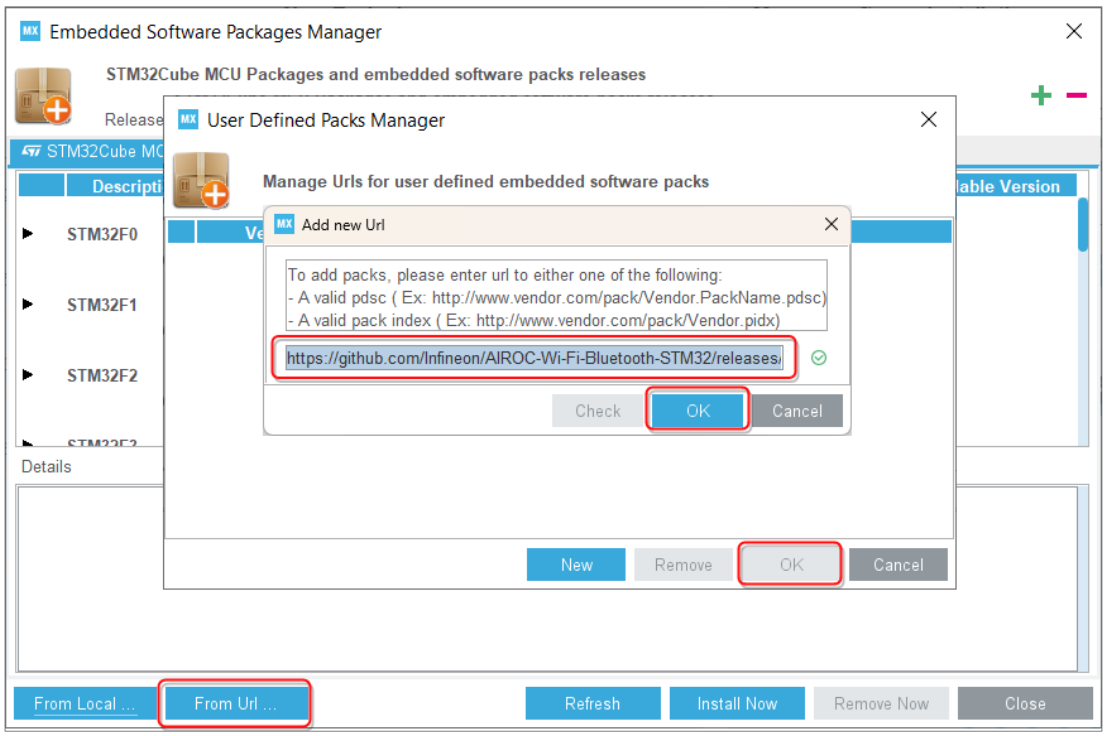
3. Select From URL…
4. Select **New (URL)**.

Download/install/import expansion pack

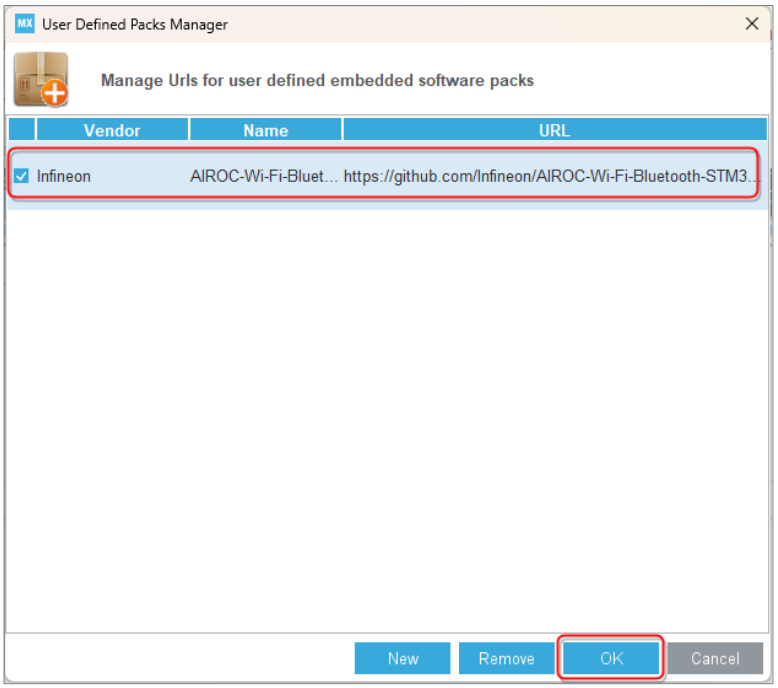
5. Input the GitHub URL to PDSC-file:

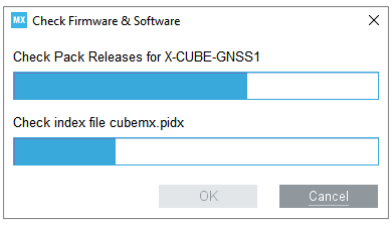
   https://github.com/Infineon/AIROC-Wi-Fi-Bluetooth-STM32/releases/download/packs/Infineon.AIROC-Wi-Fi-Bluetooth-STM32.pdsc

6. Click **Check** and **OK** if check is successful.



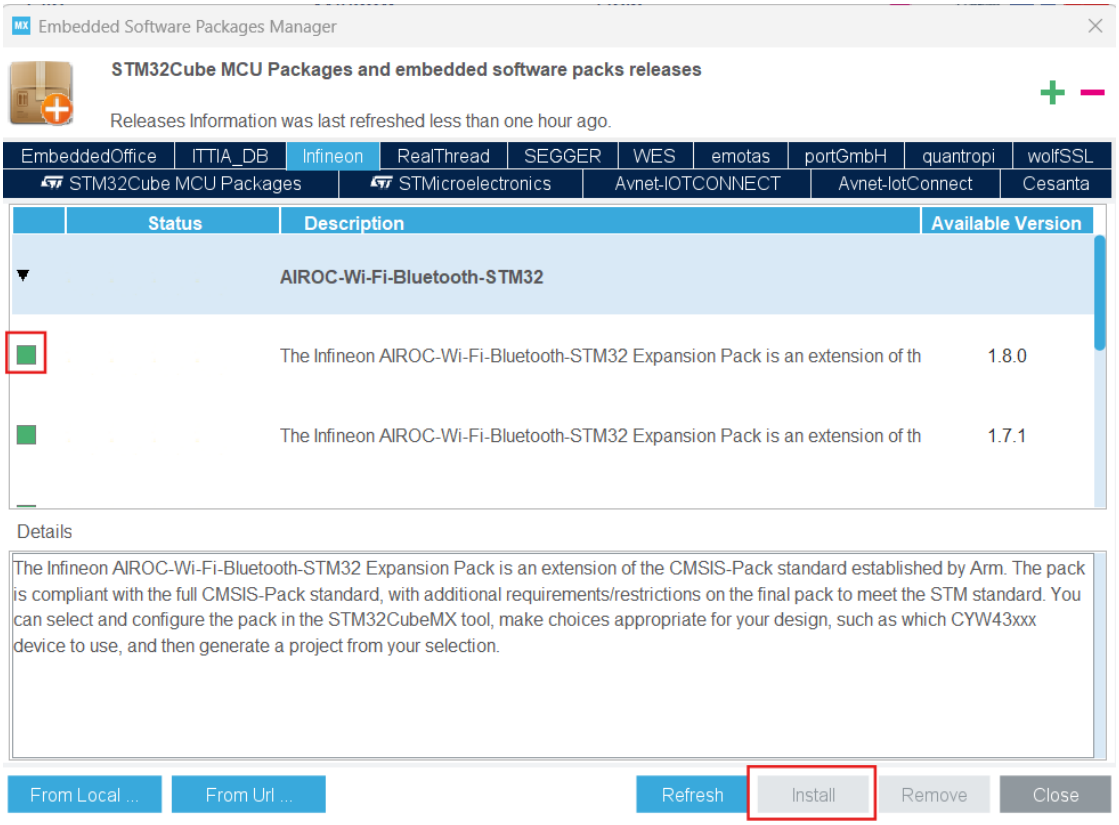
7. Select the just added URL and confirm with **OK** button.





8. In the Software Package Manager select the pack and click **Install Now** to start online installation.

Download/install/import expansion pack



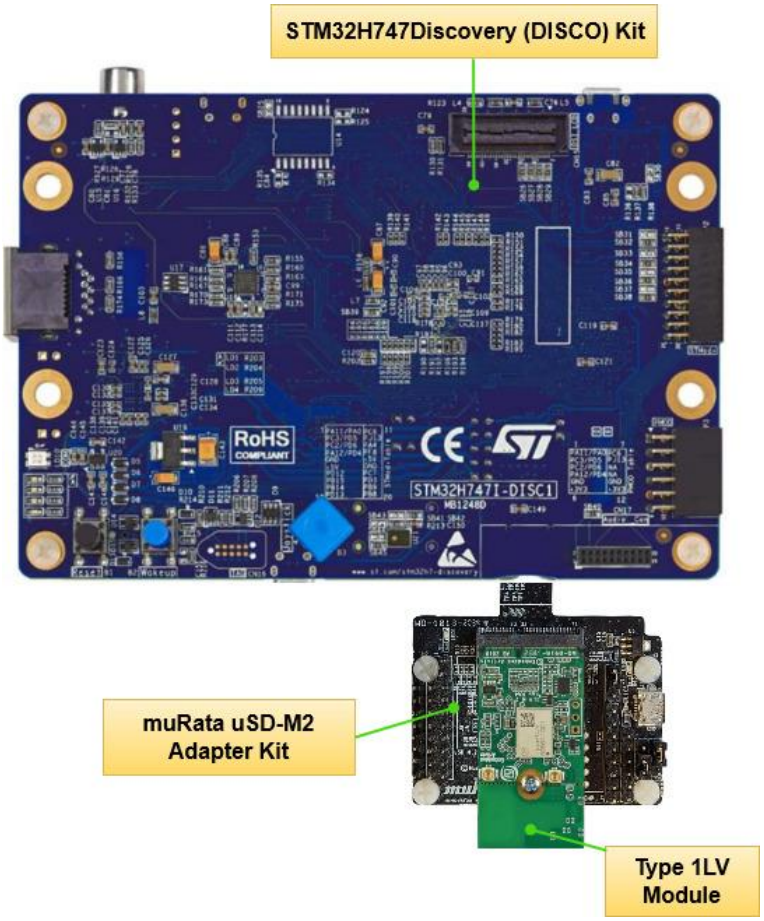
3 Hardware setup

3.1 Using STM32H747 DISCO kit

STM32H747 Disco Kit setup requires three discrete boards to create a setup where an STM32H747 hosts Infineon's CYW43xxx/CYW55xxx connectivity device. The three boards and links are:

- STM32H747 Discovery (DISCO) Kit: The STM32H747I-DISCO Discovery kit is a complete demonstration and development platform for STMicroelectronics STM32H747XIH6 microcontroller, designed to simplify user application development.
- muRata uSD-M2 Adapter Kit (rev B1): muRata's uSD-M.2 Adapter Kit with Embedded Artists' Wi-Fi/Bluetooth® M.2 Modules enable users with a simple plug-in solution. The Embedded Artists' Wi-Fi/Bluetooth® M.2 Modules are based on Murata modules using Infineon's Wi-Fi/Bluetooth® chipsets.

  Current Wi-Fi/Bluetooth® EVB support include
  - Murata Type 1DX M.2 (CYW4343W)
  - Type 1MW (CYW43455)
  - Type 1LV M.2 (CYW43012)
- Embedded Artists 1LV M.2 Module: Embedded Artists Type 1LV M.2 EVB is designed to work with the Murata uSD-M.2 Adapter.



3.1.1 Set up type 1LV M.2 module

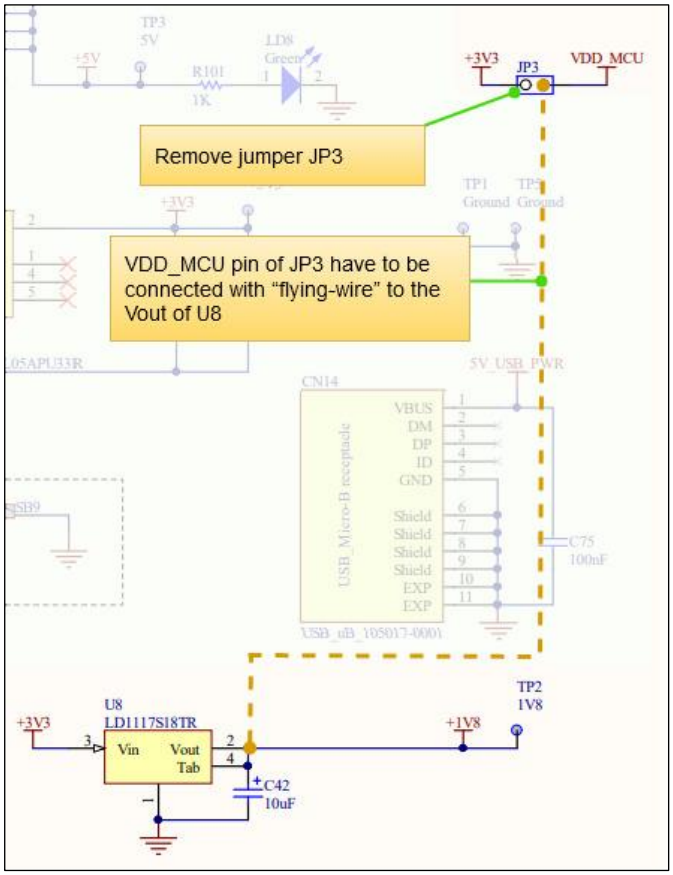
Model	Embedded Artists 1LV M.2 Module
Features	• 802.11 a/b/g/n/ac-friendly™ and Bluetooth/LE 5.0 • SDIO 3.0 interface, SDR40@80MHz • Chipset: Infineon CYW43012
Datasheet	1LV M.2

3.1.1.1 Board preparations

The 1LV module operates at 1.8 V VIO only (chipset limitation). The following preparation on STM32H747 DISCO Kit and muRata uSD-M2 Adapter are required:

9. Modify STM32H747 Disco Kit to operate on 1.8 V.

   Remove the jumper JP3 and connect the VDD_MCU pin of JP3 with "flying-wire" to the Vout of U8 linear voltage regulator (which is effectively a 1.8 V source).

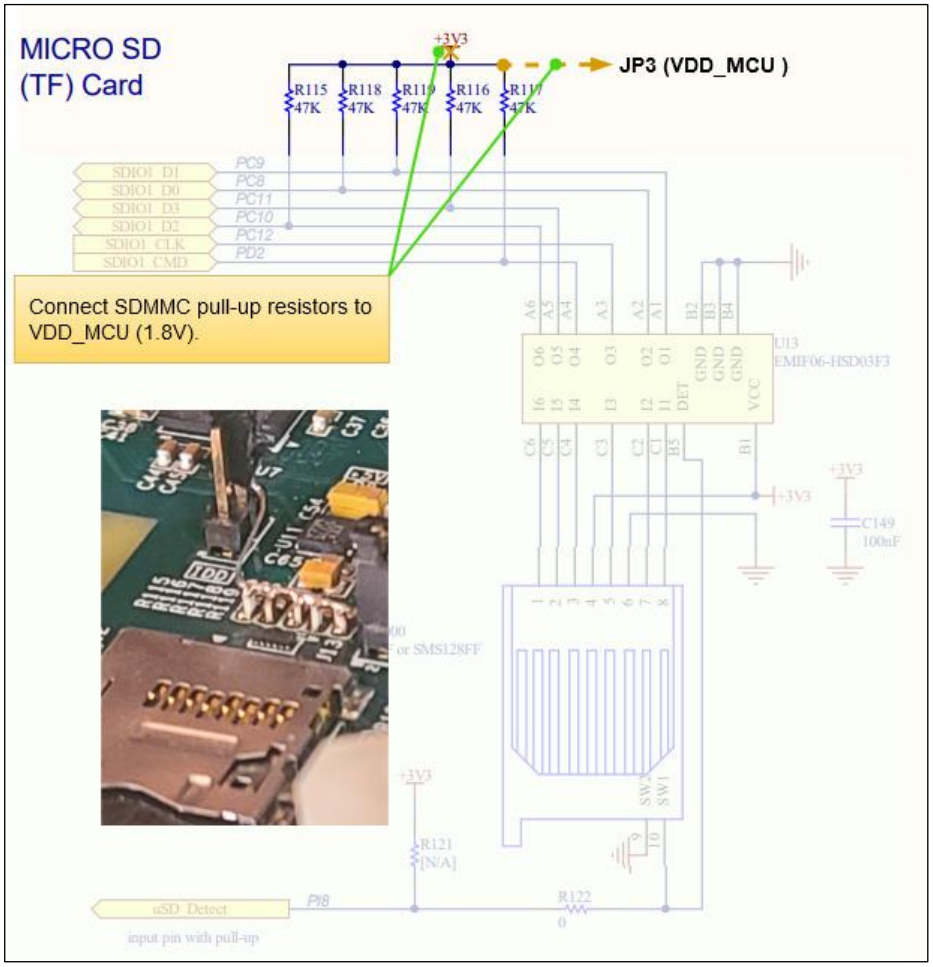


Note: Switching STM32H747 Disco Kit to operate on 1.8 V affects the functionality of external flash (MT25QL512ABB8ESF).

Hardware setup

10. Connect SDMMC pull-up resistors to VDD_MCU (1.8V) on STM32H747 DISCO Kit.

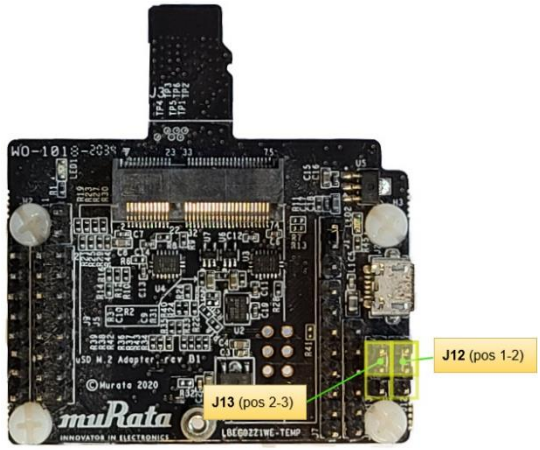
    SDMMC pull-up resistors R115-R119 must be unsoldered from the 3.3 V point and soldered vertically. The tops of these resistors have to be soldered to "flying-wire" and connected to JP3 at the side of VDD_MCU.



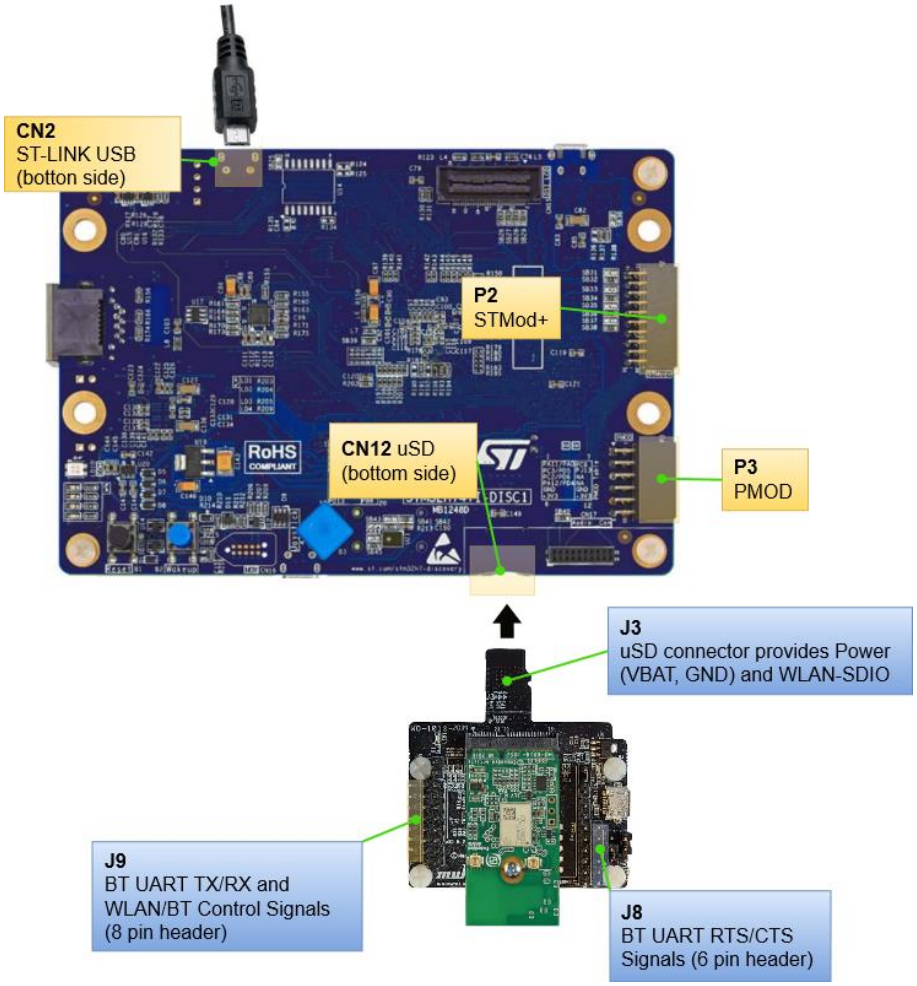
11. Modify muRata uSD-M2 Adapter to operate on 1.8V.

    To switch muRata uSD-M2 Adapter to 1.8V the following jumpers have to be configured:

    - J1 to pos 2-3 to powered USD_3V3 from uSD VCC
    - J12 to pos 1-2 (M2 IO Voltage for 1.8V VDDIO)
    - J13 to pos 2-3 (Host IO Voltage for 1.8V)



3.1.1.2 Wire connections



Connection	Operation	STM32H747 Disco Kit		muRata uSD-M2 Adapter	Note
		Connector	STM32 GPIO		
VBAT (3.3V)	VCC	CN12		J3	VBAT, GND connected via microSD connector
GND	GND				
WL_REG_ON_HOST	Wi-Fi	P3.7 (PMOD#11)	PC6	J9.3	Enables/Disables WLAN core: Active High
WL_HOST_WAKE_HOST	Wi-Fi	P3.8 (PMOD#12)	PJ13	J9.5	WLAN Host Wake: Active Low (OOB IRQ)
SDIO	Wi-Fi	CN12	PC8, PC9, PC10, PC11, PC12, PD2	J3	uSD connector pin provides Power (VBAT, GND) and WLAN-SDIO (DATA1, DATA2, DATA3, Clock and Command)
UART RX	Bluetooth	P3.1 (PMOD#1)	PA11	J9.1	UART
UART TX	Bluetooth	P3.4 (PMOD#4)	PA12	J9.2	
UART CTS	Bluetooth	P2.8 (STmod+)	PB15	J8.3	
UART RTS	Bluetooth	P2.9 (STmod+)	PB14	J8.4	
BT_REG_ON	Bluetooth	P2.10 (STmod+)	PD13	J9.4	Enables/Disables Bluetooth® core: Active High

3.1.2 Set up type 1DX M.2 module

Model	Embedded Artists 1DX M.2 Module
Features	• 802.11 b/g/n and Bluetooth/LE 4.2 • SDIO 2.0 interface, SDR25@50MHz • Chipset: Infineon CYW4343W
Datasheet	1DX M.2

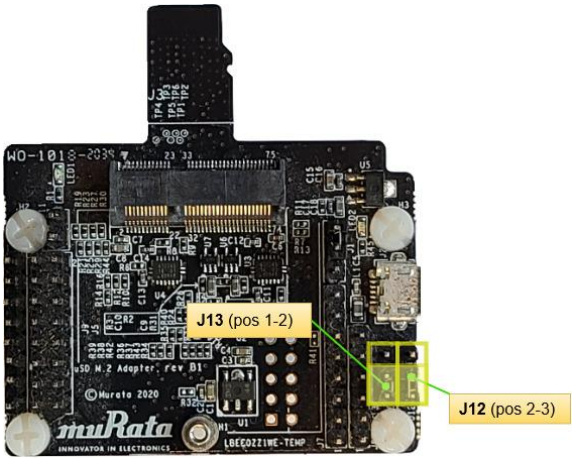
3.1.2.1 Board preparations

This module does not require the host to provide 1.8 V on the SDIO/UART GPIO. It can operate on 3.3V/1.8V. This makes board preparation simpler. Please see the following sections

1. Modify muRata uSD-M2 Adapter to operate on 3.3V.

   To switch muRata uSD-M2 Adapter to 3.3V the following jumpers have to be configured:

   - J1 to pos 2-3 to powered USD_3V3 from uSD VCC
   - J12 to pos 2-3 (M2 IO Voltage for 3.3V VDDIO)
   - J13 to pos 1-2 (Host IO Voltage for 3.3V VDDIO)



3.1.2.2 Wire connections

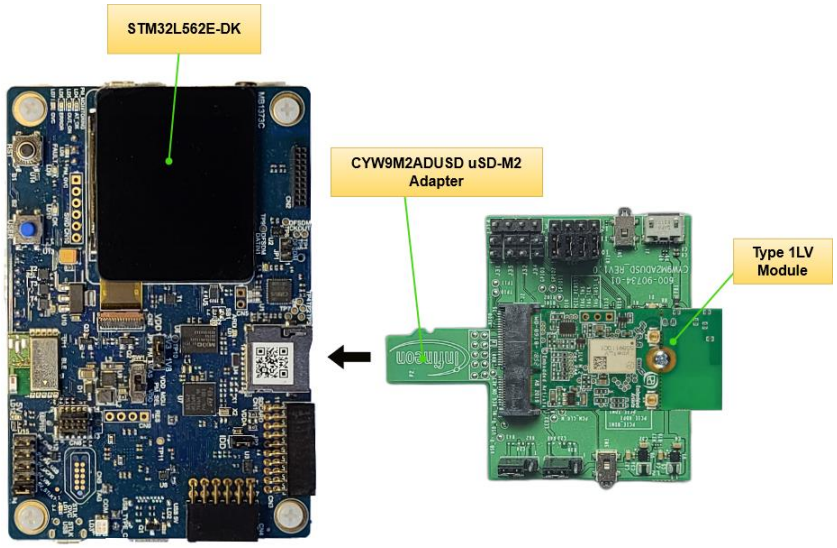
The Type 1DXM module uses the same wire connections as Type 1LV modules. Refer to the Wire connections section for Type 1LV Modules.

3.2 Using STM32L562E-DK

3.2.1 Set Up M.2 Module + CYW9M2ADUSD Adapter Kit for Wi-Fi and Bluetooth® Connectivity

STM32L562E DK Kit setup for Bluetooth® connectivity requires three discrete boards to create a setup where an STM32L562E hosts Infineon's CYW43xxx/CYW55xxx connectivity device. The three boards and links are:

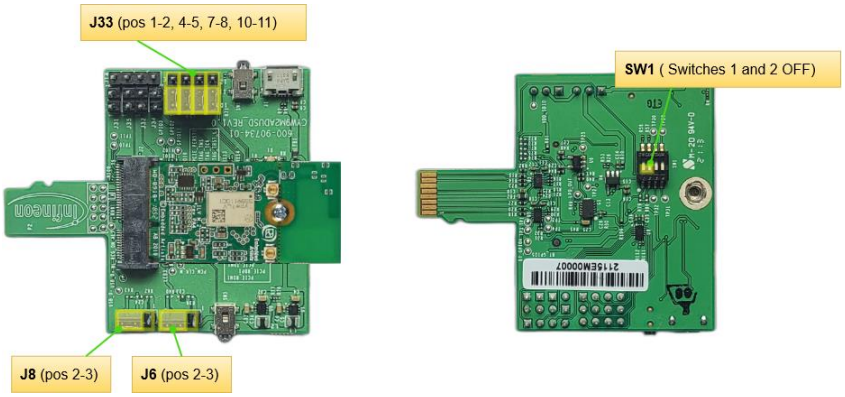
- STM32L562E-DK Discovery kit is a complete demonstration and development platform for Arm® Cortex®-M33 with Arm® TrustZone® and ARMv8-M mainline security extension core-based STM32L562QEI6QU microcontroller, with 512 Kbytes of Flash memory and 256 Kbytes of SRAM.
- CYW9M2ADUSD Adapter Kit : adapter which allows you to connect M.2-based CYW43x connectivity modules into SD-card slot of a various DVKs and EVKs. Please contact sales for order questions.
- Embedded Artists 1LV M.2 Module: Embedded Artists Type 1LV M.2 EVB is designed to work with the Murata uSD-M.2 Adapter.



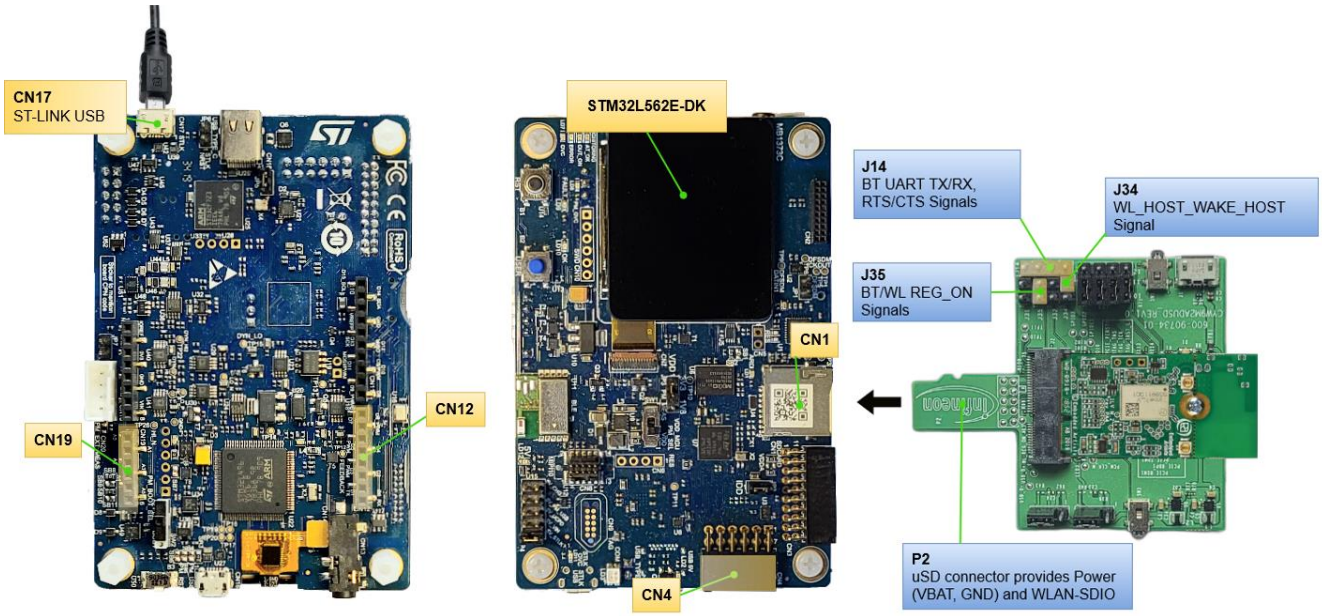
3.2.1.1 Board preparation

CYW9M2ADUSD Adapter requires to configure the following jumpers:

- J6 and J8 to pos 2-3 (use 3.3V from VDD_SDIO)
- J33 to use 1.8V level shifters for UART
- SW1 – switches 1 and 2 in OFF position



3.2.1.2 Wire connections



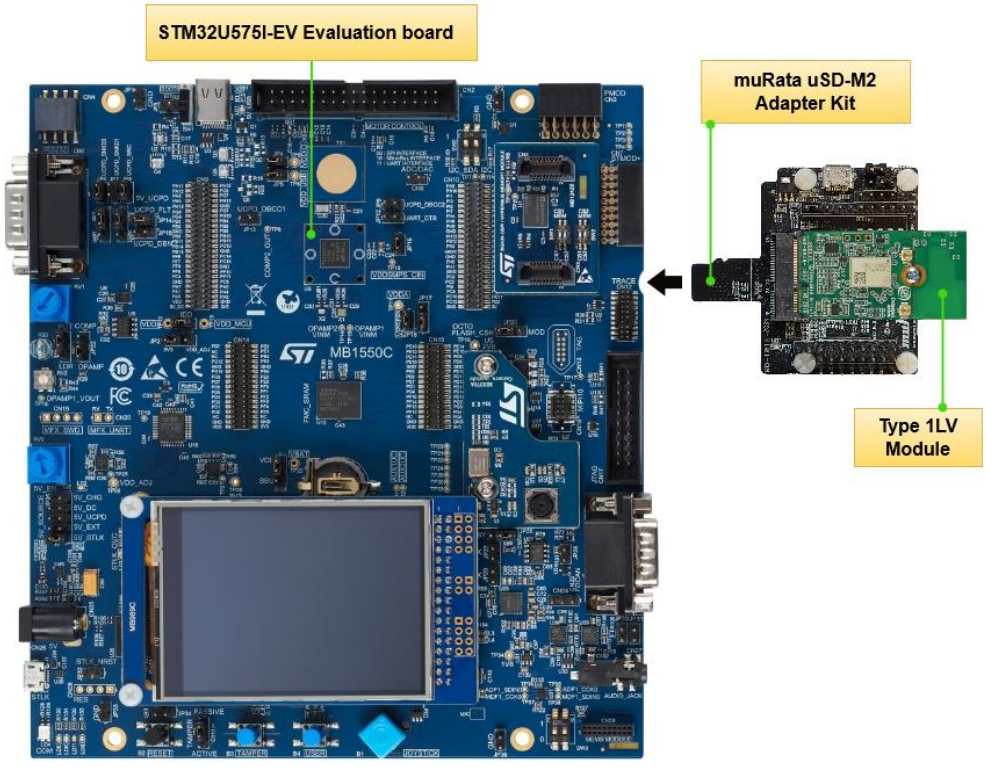
Connection	Operation	STM32L562E-DK		CYW9M2ADUSD Adapter	Note
		Connector	STM32 GPIO		
VBAT (3.3V)	VCC	CN1		P2 (uSD Connection)	VBAT, GND connected via microSD connector
GND	GND				
WL_REG_ON_HOST	Wi-Fi	CN4.7	PF5	J35.1	Enables/Disables WLAN core: Active High
WL_HOST_WAKE_HOST	Wi-Fi	CN4.1	PB13	J34.1	WLAN Host Wake: Active Low (OOB IRQ)
SDIO	Wi-Fi	CN1	PC8, PC9, PC10, PC11, PC12, PD2	P2 (uSD Connection)	uSD connector pins: provides Power (VBAT, GND) and WLAN-SDIO (DATA0, DATA1, DATA2, DATA3, Clock and Command)
BT_REG_ON	Bluetooth	CN12.5	PF4	J35.2	Enables/Disables Bluetooth® core: Active High
UART RX	Bluetooth	CN19.6	PC5	J14.2 (TX)	UART (USART3)
UART TX	Bluetooth	CN12.1	PB10	J14.1 (RX)	
UART CTS	Bluetooth	CN12.3	PD11	J14.4 (RTS)	
UART RTS	Bluetooth	CN12.4	PD12	J14.3 (CTS)	

3.3 Using STM32U575I-EV Evaluation board

The STM32U575I-EV Evaluation board setup requires three discrete boards to enable the STM32U575 board to host Infineon's CYW43xxx/CYW55xxx connectivity device. The three boards and links are:

- STM32U575I-EV Evaluation board: This board is a complete demonstration and development platform for STMicroelectronics STM32U575AII6Q microcontroller, designed to simplify user application development.
- muRata uSD-M2 Adapter Kit (rev B1): muRata's uSD-M.2 Adapter Kit with Embedded Artists' Wi-Fi/Bluetooth® M.2 Modules enable users with a simple plug-in solution. The Embedded Artists' Wi-Fi/Bluetooth® M.2 Modules are based on Murata modules using Infineon's Wi-Fi/Bluetooth® chipsets.

  Current Wi-Fi/Bluetooth® EVB support include:
  - Murata Type 1DX M.2 (CYW4343W)
  - Type 1MW (CYW43455)
  - Type 1LV M.2 (CYW43012)
- Embedded Artists 1LV M.2 Module: Embedded Artists Type 1LV M.2 EVB is designed to work with the Murata uSD-M.2 Adapter.



3.3.1 Set up type 1LV M.2 module

3.3.1.1 Board preparations

The 1LV module operates at 1.8 V VIO only (chipset limitation). The following preparation on STM32U575I-EV Evaluation board and muRata uSD-M2 Adapter are required:

1. Modify the STM32U575I-EV Evaluation board to operate on 1.8 V.

   By default, the STM32U575I-EV Evaluation board is configured with VDD_MCU at 3.3 V. To switch the board to 1.8V:

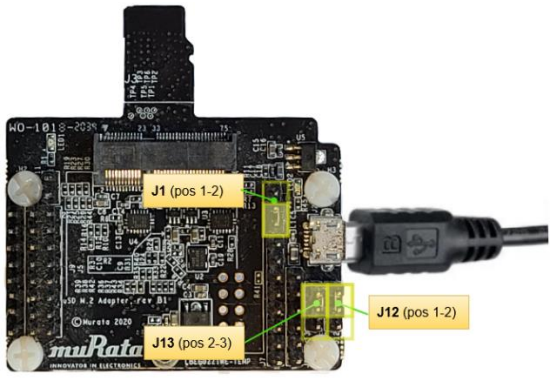
   - Use a potentiometer RV3 to adjust VDD_ADJ to 1.8V. You can use TP29 as test point for the Voltmeter connection.
   - Configure JP23 to pos 2-3. It switches VDD/VDD_MCU to VDD_ADJ instead of 3.3V

Note: Switching the STM32U575I-EV Evaluation board to operate on 1.8 V affects the functionality of external flash (MT25QL512ABB8ESF) and external SRAM (IS61WV102416BLL-10MLI).

2. Modify the muRata uSD-M2 Adapter to operate on 1.8 V.

   To switch the muRata uSD-M2 Adapter to 1.8 V, configure the following jumpers:

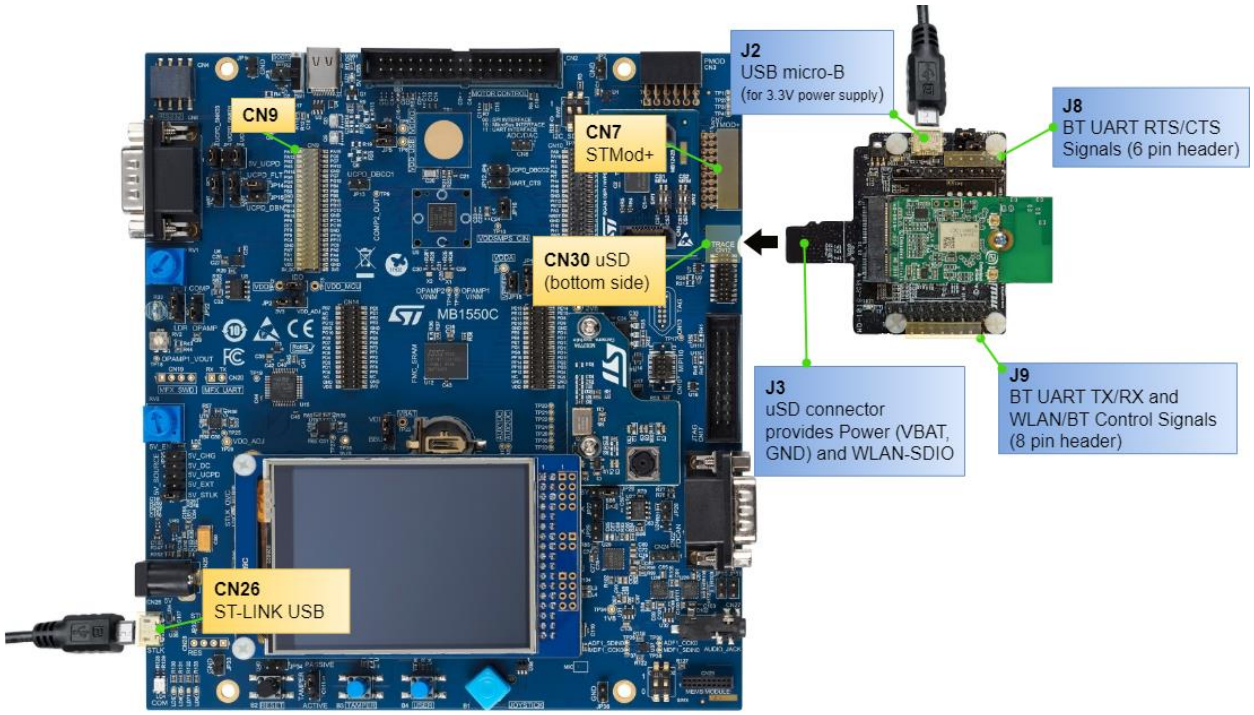
   - J1 to pos 1-2 to powered USD_3V3 from micro USB (J2)
   - J12 to pos 1-2 (M2 IO Voltage for 1.8V VDDIO)
   - J13 to pos 2-3 (Host IO Voltage for 1.8 V)
   - Micro USB (J2) should be plugged in.



3. Configure jumpers on the STM32U575I-EV Evaluation board:

   - Remove JP10
   - Remove JP11
   - Remove JP12
   - Remove SB38 is shorted (default)

3.3.1.2 Wire connections



Connection	Operation	STM32U575I-EV		muRata uSD-M2 Adapter	Note
		Connector	STM32 GPIO		
VBAT (3.3V)	VCC	CN30		J3	VBAT, GND connected via microSD connector
GND	GND				
WL_REG_ON_HOST	Wi-Fi	CN7.9 (STmod+)	PB4	J9.3	Enables/Disables WLAN core: Active High
WL_HOST_WAKE_HOST	Wi-Fi	CN7.8 (STmod+)	PB5	J9.5	WLAN Host Wake: Active Low (OOB IRQ)
SDIO	Wi-Fi	CN30	PC8, PC9, PC10, PC11, PC12, PD2	J3	uSD connector pin provides Power (VBAT, GND) and WLAN-SDIO (DATA1, DATA2, DATA3, Clock and Command)
UART RX	Bluetooth	CN9.13	PG8	J9.1	UART (LPUART1)
UART TX	Bluetooth	CN9.12	PG7	J9.2	
UART CTS	Bluetooth	CN9.24	PB13	J8.3	
UART RTS	Bluetooth	CN9.11	PG6	J8.4	
BT_REG_ON	Bluetooth	CN7.10 (STmod+)	PB11	J9.4	Enables/Disables Bluetooth® core: Active High

3.3.2 Set up type 1DX M.2 module

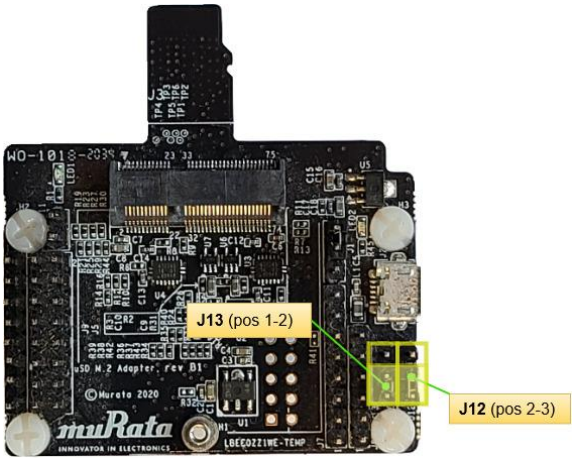
3.3.2.1 Board preparations

This module does not require the host to provide 1.8 V on the SDIO/UART GPIO. It can operate on 3.3 V/1.8 V. This makes board preparation simpler.

1. Modify the muRata uSD-M2 Adapter to operate on 3.3 V.

   To switch the muRata uSD-M2 Adapter to 3.3V, configure the following jumpers:

   - J12 to pos 2-3 (M2 IO Voltage for 3.3V VDDIO)
   - J13 to pos 1-2 (Host IO Voltage for 3.3V VDDIO)



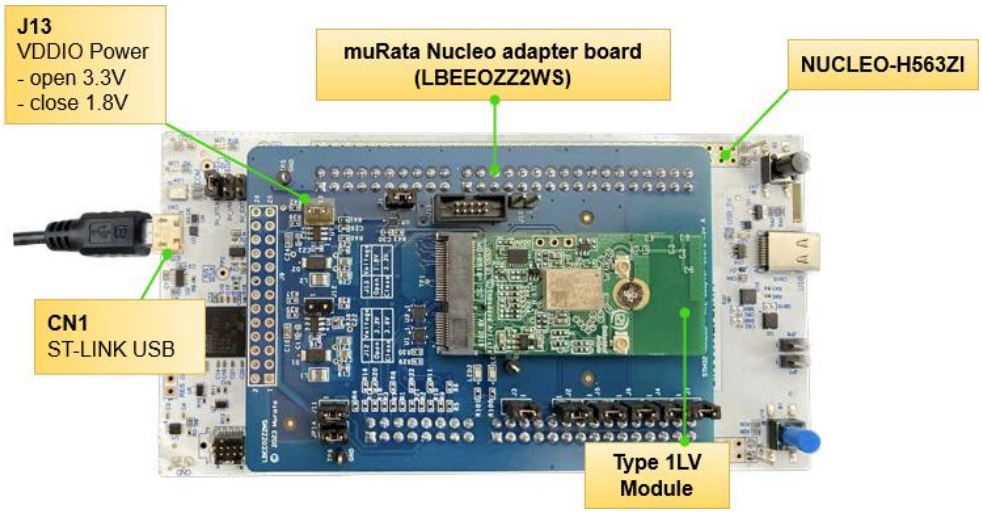
3.3.2.2 Wire connections

The Type 1DXM module uses the same wire connections as the Type 1LV modules. Refer to the Wire connections section (3.3.1.2) for Type 1LV Modules.

3.4 Using NUCLEO-H563ZI board with MuRata Type2WS

The NUCLEO-H563ZI board setup requires three discrete boards to enable the STM32H5xx board to host Infineon's CYW43xxx/CYW55xxx connectivity device. The three boards and links are:

- MuRata Nucleo Adapter board (Type2WS): Adapter board (Type2WS) is available for interfacing STM32 Nucleo-144 and M.2 board.
- NUCLEO-H563ZI board: This board is a complete demonstration and development platform for STMicroelectronics STM32U575AII6Q microcontroller, designed to simplify user application development. Murata STM32 NUCLEO M.2 Adapter Board (LBEEOZZ2WS): This Board enables users to connect M.2 Module to NUCLEO-H563ZI Board. The LBEEOZZ2WS board is early ES sample and will be coming soon to be ordered.
- Embedded Artists 1LV M.2 Module: Embedded Artists Type 1LV M.2 EVB is designed to work with the Murata uSD-M.2 Adapter.



This setup does not require any wires to be connected. The default pin mapping is described in the following section.

Note: Ensure you set 1.8V for VDDIO (J13 must be shorted) when using only 1.8V compatible radio.

3.4.1 Pin mapping

Connection	Operation	Nucleo-H563ZI		muRata Nucleo Adapter	Note
		Connector	STM32 GPIO		
WL_REG_ON_HOST	Wi-Fi	CN9.25	PD0	CN3.25	Enables/Disables WLAN core: Active High
WL_HOST_WAKE_HOST	Wi-Fi	CN9.27	PD1	CN3.27	WLAN Host Wake: Active Low (OOB IRQ)
SDIO	Wi-Fi	CN8	PC8, PC9, PC10, PC11, PC12, PD2	CN1	WLAN-SDIO (DATA1, DATA2, DATA3, Clock and Command)
UART RX	Bluetooth	CN9.4	PD6	CN3.4	UART (USART)
UART TX	Bluetooth	CN9.6	PD5	CN3.6	
UART CTS	Bluetooth	CN9.10	PD3	CN3.10	
UART RTS	Bluetooth	CN9.8	PD4	CN3.8	

Hardware setup

Connection	Operation	Nucleo-H563ZI		muRata Nucleo Adapter	Note
		Connector	STM32 GPIO		
BT_REG_ON	Bluetooth	CN8.16	PG3	CN1.16	Enables/Disables Bluetooth® core: Active High

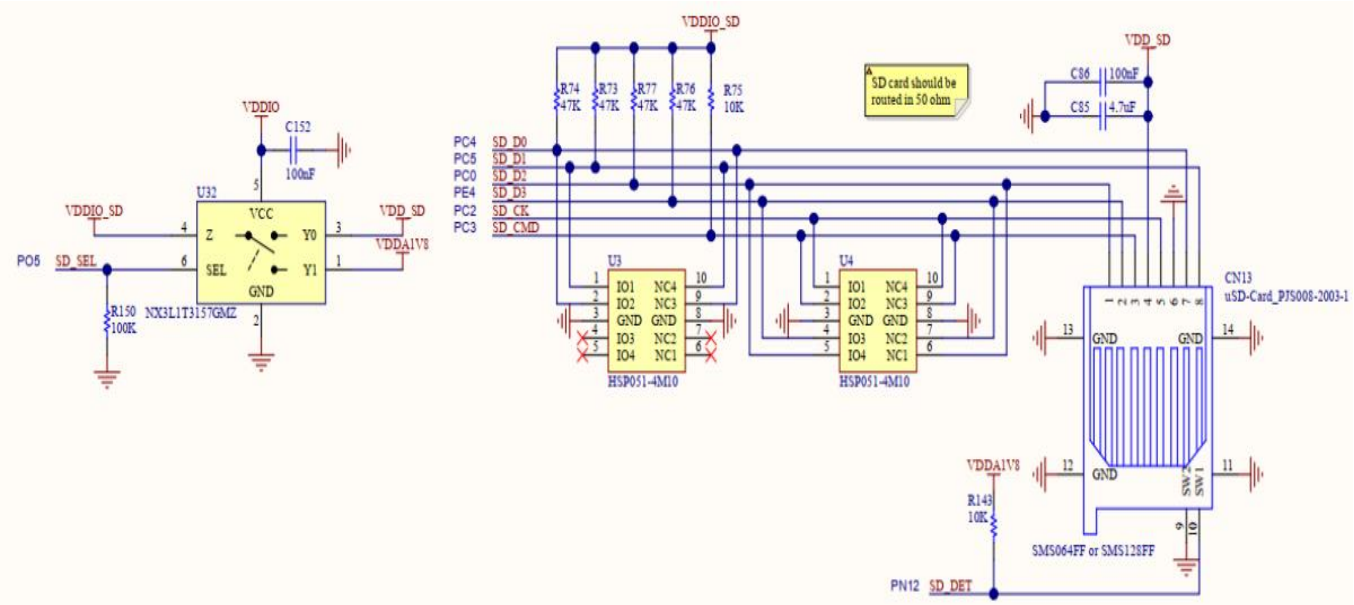
3.5 Using STM32N6570-DK board with Infineon CYW55513

3.5.1.1 Board preparations

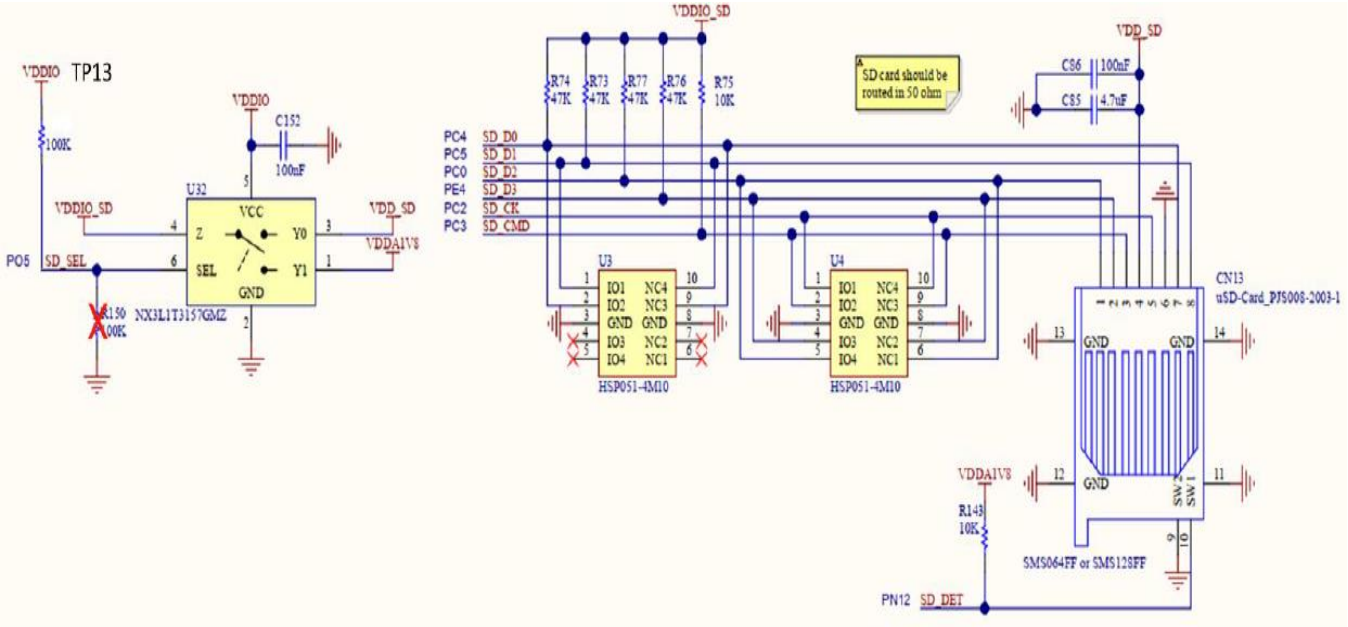
The CYW55513 module operates at 1.8 V VIO only (chipset limitation). The following preparation on STM32N6570-DK Evaluation board and muRata uSD-M2 Adapter are required:

1. Modify STM32N6570-DK Evaluation board to operate on 1.8 V.
   - Replace R150 with a 100kOhm pull-up resistor to VDDIO (1v8) on TP13
   - U32 will now set VDDIO_SD to VDDA1V8(1v8) by default via pullup resistor

   (Before rework)



   After rework

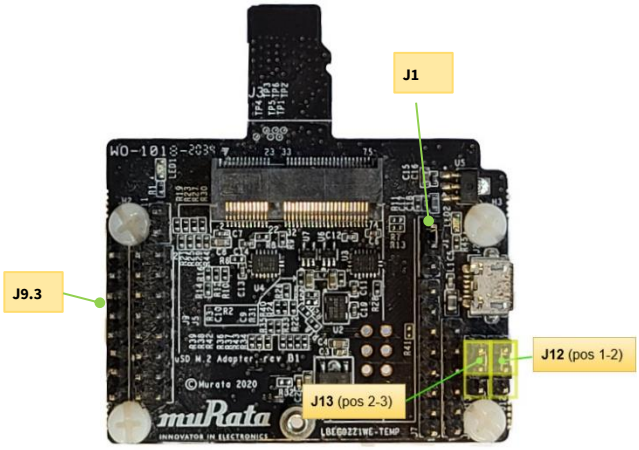


2. Modify the muRata uSD-M2 Adapter to operate on 1.8 V.

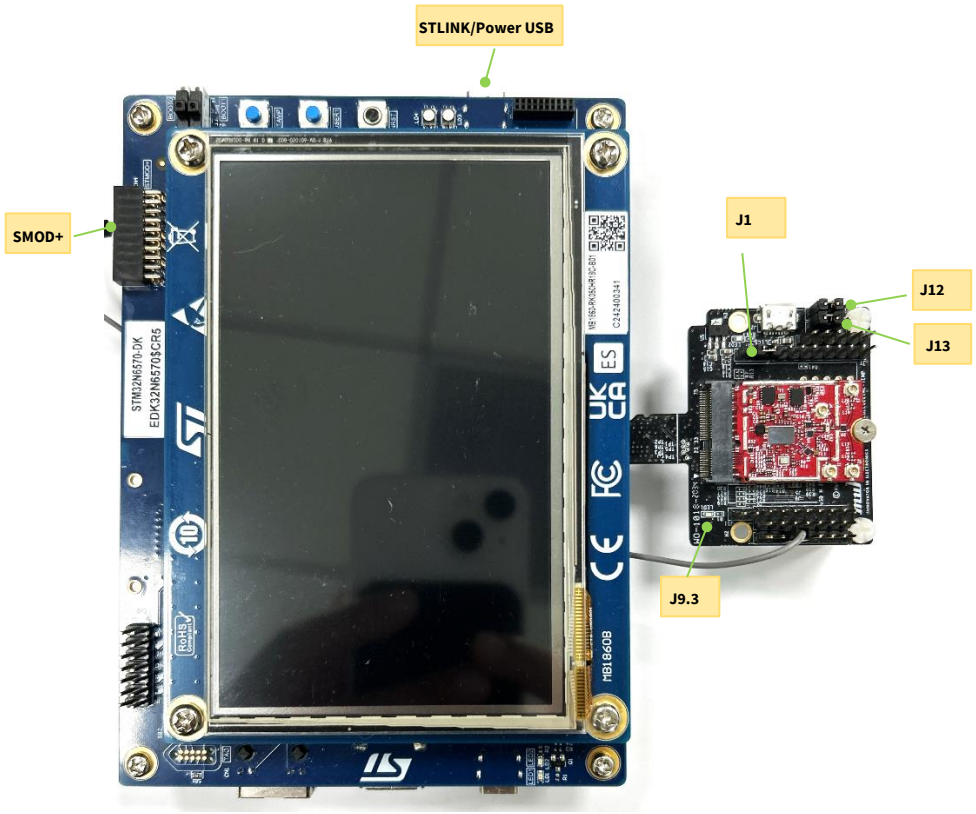
Hardware setup

To switch the muRata uSD-M2 Adapter to 1.8 V, configure the following jumpers:

- J1 to pos 1-2 to powered USD_3V3 from micro USB (J2)
- J12 to pos 1-2 (M2 IO Voltage for 1.8V VDDIO)
- J13 to pos 2-3 (Host IO Voltage for 1.8 V)
- Connect J9.3=WL_REG_ON to STMod+ Pin17 of STM32N6570KD board



3.5.1.2 Wire connections



3.5.1.3 Pin Mappings

Connection	Operation	STM32N6570-DK		muRata uSD-M2 Adapter	Note
		Connector	STM32 GPIO		
VBAT (3.3V)	VCC	CN3		J4 (uSD Connection)	VBAT, GND connected via microSD connector
GND	GND				
WL_REG_ON_HOST	Wi-Fi	CN4.17	PD13	J9.3	Enables/Disables WLAN core: Active High
SDIO	Wi-Fi	CN1		P2 (uSD Connection)	uSD connector pins: provides Power (VBAT, GND) and WLAN-SDIO (DATA0, DATA1, DATA2, DATA3, Clock and Command)
BT_REG_ON	Bluetooth	CN4.18 (CN4=STMod+)	PF1	J9.4	Enables/Disables Bluetooth® core: Active High
USART2 RX	Bluetooth	CN4.3	PF6	J9.1 (TX)	UART (USART3)
USART2 TX	Bluetooth	CN4.2	PD5	J9.2 (RX)	
USART2 CTS	Bluetooth	CN4.1	PG5	J8.3 (RTS)	
USART2 RTS	Bluetooth	CN4.4	PG14	J8.4 (CTS)	

4 Using example projects

We provide the following example projects to get started using the pack:

- Wi-Fi Scan
- Wi-Fi onboarding with Bluetooth® LE
- Azure RTOS NetXDuo Wi-Fi UDP echo server
- Bluetooth® LE Hello Sensor
- Wi-Fi TCP keepalive offload
- Wi-Fi MQTT client

4.1 Wi-Fi Scan

This example demonstrates how to configure different scan filters provided in the Wi-Fi Connection Manager (WCM) middleware and scan for the available Wi-Fi networks.

The example initializes the Wi-Fi device and starts a Wi-Fi scan without any filter and prints the results on the serial terminal. The example starts a scan every three seconds after the previous scan completes.

This example demonstrates how an STM32H7 can be used to host CYW43xxx/CYW55xxx connectivity devices.

4.1.1 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

4.1.2 Other software

Install a terminal emulator if you do not have one. Instructions in this document use Tera Term.

4.1.3 Project components

The following are the only components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/LwIP
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)

4.1.4 Example project start/import

You can open the Wi-Fi Scan example by copying the example from the Pack to an appropriate location. Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the following steps:

1. Copy the code example from the pack directory to your local directory.

   The default path for installed packs is:

   C:\Users\<USER>\STM32Cube\Repository\Packs

   Copy the wifi_scan example from the appropriate directory. For instance, for STM32H747I-DISCO:

   C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_scan

   Paste into your working folder. For example:

   C:\Users\<USER>\STM32Cube\Example

2. Open *wifi_scan.ioc* file in the root folder of project.

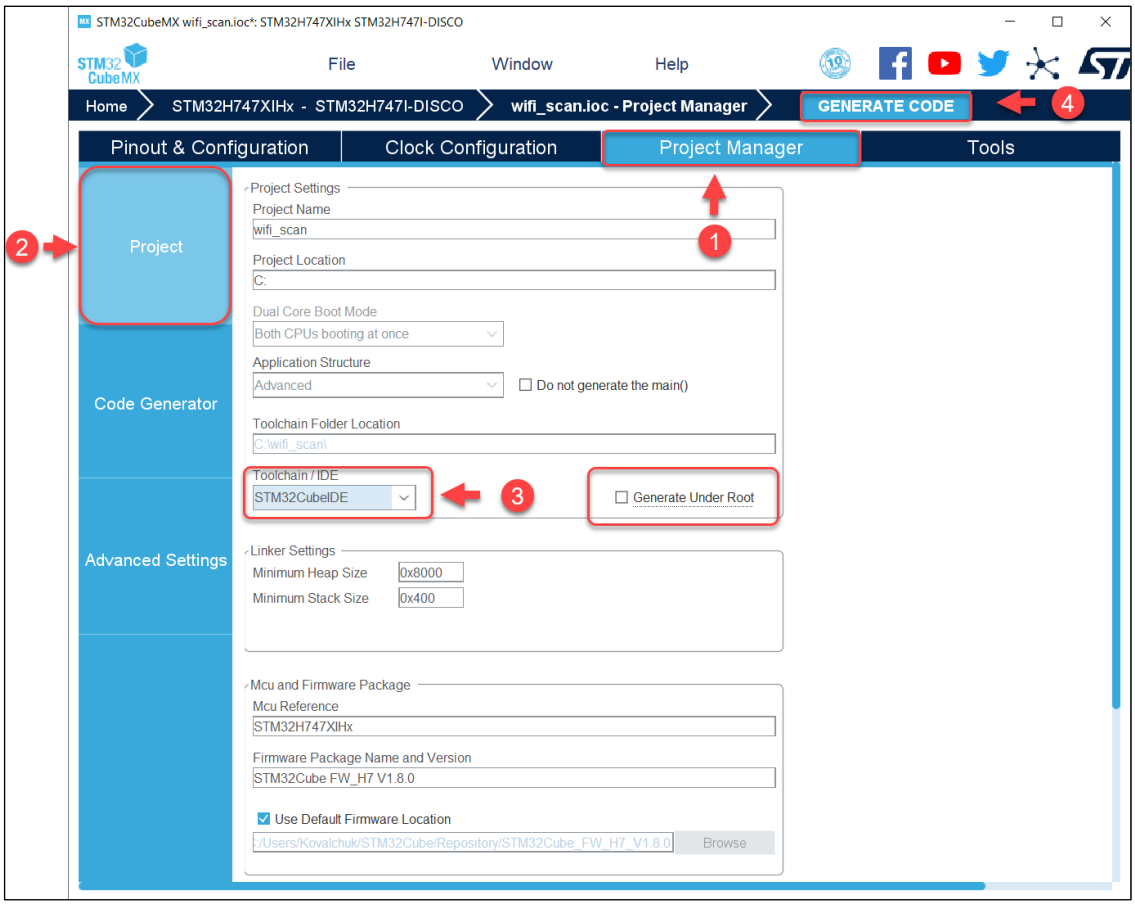
   C:\Users\<USER>\STM32Cube\Example\wifi_scan\wifi_scan.ioc

3. Click **OK** to accept.

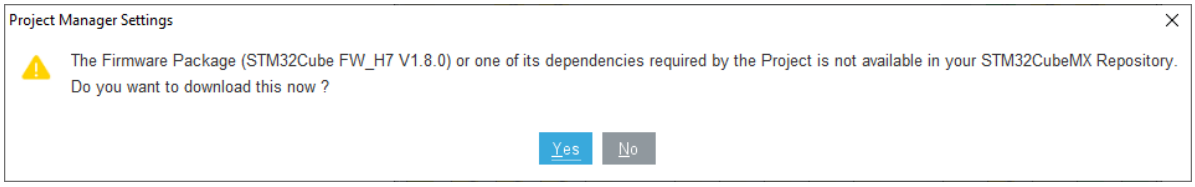
4.1.5 Generate code

Follow these steps to generate code:

1. Select **the Project Manager** tab.
2. Select **Project.**
3. Select the appropriate option under **Toolchain / IDE** and select the **Generate Under Root** check box.
4. Click **GENERATE CODE**.



If a message displays about missing packages, select **Yes**:

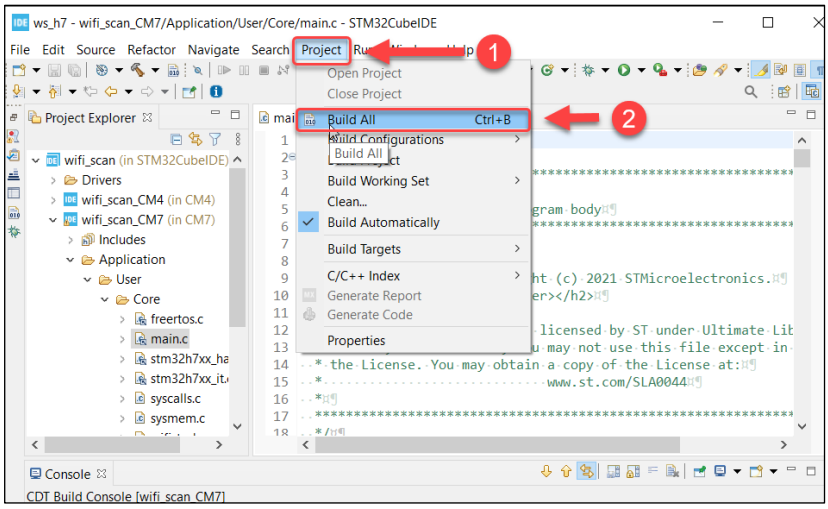


5. After the code is generated, you will see this dialog. Select **Open Project**.

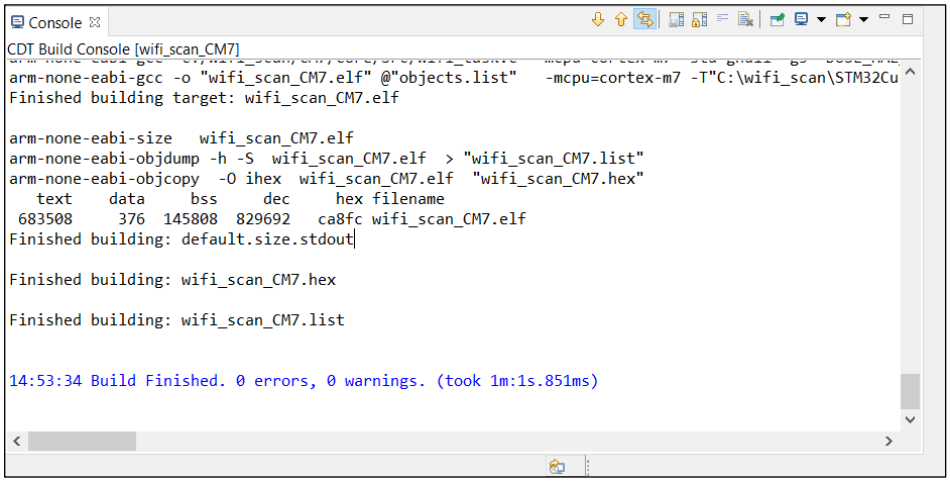
4.1.6 Build the project

The build step and expected output are illustrated here for each IDE.

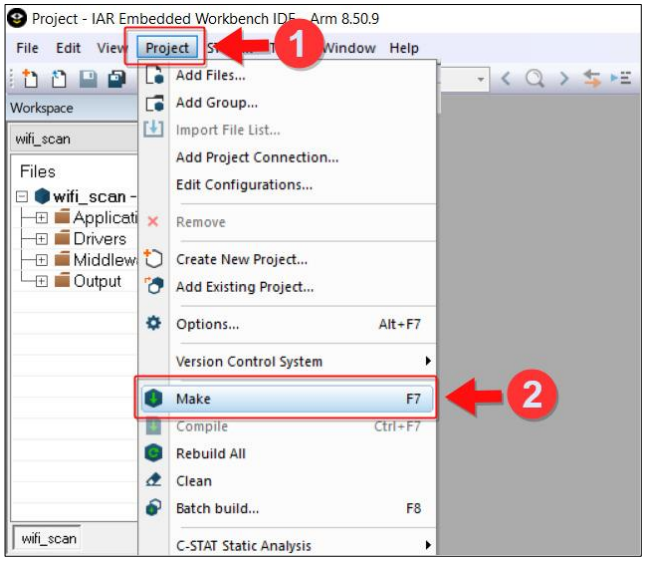
4.1.6.1 STM32CubeIDE:



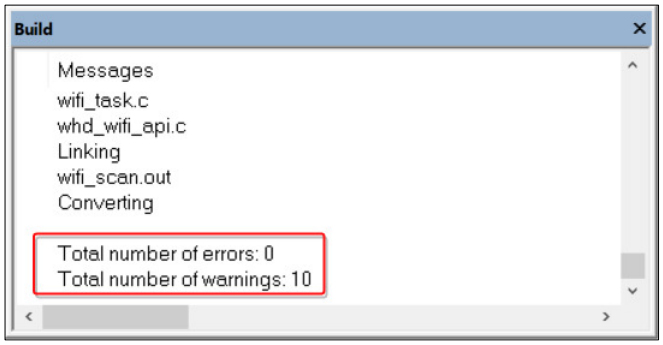
Example output from a successful build:



4.1.6.2 IAR EWARM:



The project should build without errors. There are 10 warnings in the lwIP library.



4.1.7 Project hardware setup

Refer to section Hardware Setup.

4.1.8 Terminal display

The terminal display is used by the application to provide status and network information.

You will need a terminal emulator such as Tera Term (https://github.com/TeraTermProject/teraterm/releases) to display the output.

4.1.8.1 Serial terminal setup

The terminal interface is a virtual COM port which is part of the ST-LINK (CN2) USB connection. Terminal emulator configuration:

- Baud Rate: 115200
- Data Length: 8 Bits
- Stop Bit(s): 1
- Parity: None
- Flow control: None

4.1.8.2 Example output

```
****************** WiFi-Scan app ******************

    Insert CYW43xxx/CYW55xxx into microSD card slot

    Push blue button to continue...

    CYW43xxx/CYW55xxx detected

WLAN MAC Address : E8:E8:B7:9F:CC:EAWLAN Firmware : wl0: Sep  9 2020 01:22:10 version 13.10.271.253
(c4c4c7c CY) FWID 01-79301becWLAN CLM            : API: 18.2 Data: 9.10.0 Compiler: 1.36.1 ClmImport:
1.34.1 Creation: 2020-09-09 01:19:03 WHD VERSION  : v1.93.0 : v1.93.0 : IAR 8050009 : 2020-12-21
13:24:03 +0530
-----------------------------------------------------------------------------------------------
  #                 SSID                 RSSI    Channel       MAC Address              Security
-----------------------------------------------------------------------------------------------
  1   Private                             -72      11      1C:AF:F7:26:8D:A8         WPA2_MIXED_PSK
  2   Private                             -73      11      74:DA:88:29:F2:27         WPA2_MIXED_PSK

-----------------------------------------------------------------------------------------------
  #                 SSID                 RSSI    Channel       MAC Address              Security
-----------------------------------------------------------------------------------------------
  1   Private                             -68      11      74:DA:88:29:F2:27         WPA2_MIXED_PSK
  2   Private                             -73      11      1C:AF:F7:26:8D:A8         WPA2_MIXED_PSK
```

4.2 Wi-Fi onboarding with Bluetooth® LE

This example uses the STM32H7 MCU to communicate with the CYW43xxx/CYW55xxx combo devices and control the Wi-Fi and Bluetooth® LE functionality. It uses Bluetooth® LE on the combo device to help connect the Wi-Fi to the AP.

In this example, Bluetooth® LE provides a mechanism for the device to connect to a Wi-Fi AP by providing the Wi-Fi SSID and password in a secure manner. The Wi-Fi credentials are stored in EEPROM so that the device can use this data upon reset to connect to an AP without requiring Bluetooth® LE intervention. Note that the data stored in the EEPROM is unencrypted.

The Wi-Fi SSID and password are exchanged using custom GATT service and characteristics. There is a third custom characteristic, which gives the command to connect and disconnect. The Wi-Fi password is write-only; the device needs to be paired before this characteristic can be written.

Bluetooth® LE GATT Custom Service This example uses custom GATT service and characteristics to communicate with the Bluetooth® LE GATT client. The files cycfg_gatt_db.c and cycfg_gatt_db.h contain the GATT DB.

The following custom characteristics are used in this example:

- **Wi-Fi SSID:** Provides the Wi-Fi SSID from Bluetooth® LE GATT client to the server. The maximum size is 32 as Wi-Fi limits the SSID name to 32 characters.
- **Wi-Fi Password:** Provides the Wi-Fi password from the Bluetooth® LE GATT client to the server. The minimum size is 8 because Wi-Fi encryption requires a minimum of 8 characters for password.
- **Wi-Fi Connect:** A Boolean characteristic that is used to connect and disconnect from the Wi-Fi AP. This has a Client Characteristic Configuration Descriptor (CCCD) attached with it. Whenever there is a successful connection, it will send a notification value of 1 otherwise it will send a notification value of 0 if notifications are enabled.

4.2.1 Hardware

Refer to section the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

4.2.2 Other software

This code example requires two devices: Host (Mobile Phone or PC) and a Target (STM32H747 DISCO Kit).

1. For the Host, download and install the AIROC™ Bluetooth® Connect App for iOS or Android. Scan the following QR codes from your mobile phone to download the AIROC™ Bluetooth® Connect App:

2. Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

4.2.3 Project components

The following are the components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/LwIP
- Bluetooth/btstack
- Bluetooth/btstack-integration
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)

4.2.4 Example project start/import

You can open the Wi-Fi Onboarding with Bluetooth® LE example by copying the example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\Bluetooth® LE_wifi_onboarding

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

4.2.5 Project hardware setup

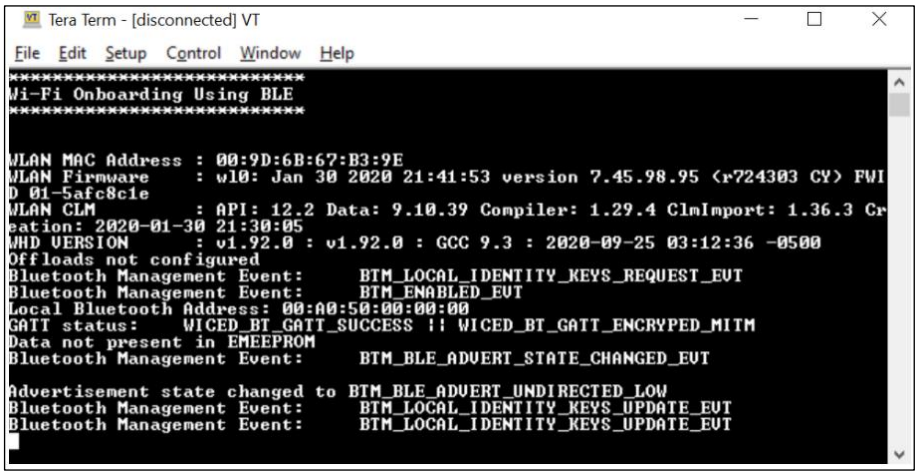
Refer to section Hardware Setup.

4.2.6 Operation

1. Connect the STM32H747 DISCO Kit to your PC.
2. Use your favorite serial terminal application and connect to the ST-LINK (CN2) COM port. Configure the terminal application to access the serial port using the following settings.

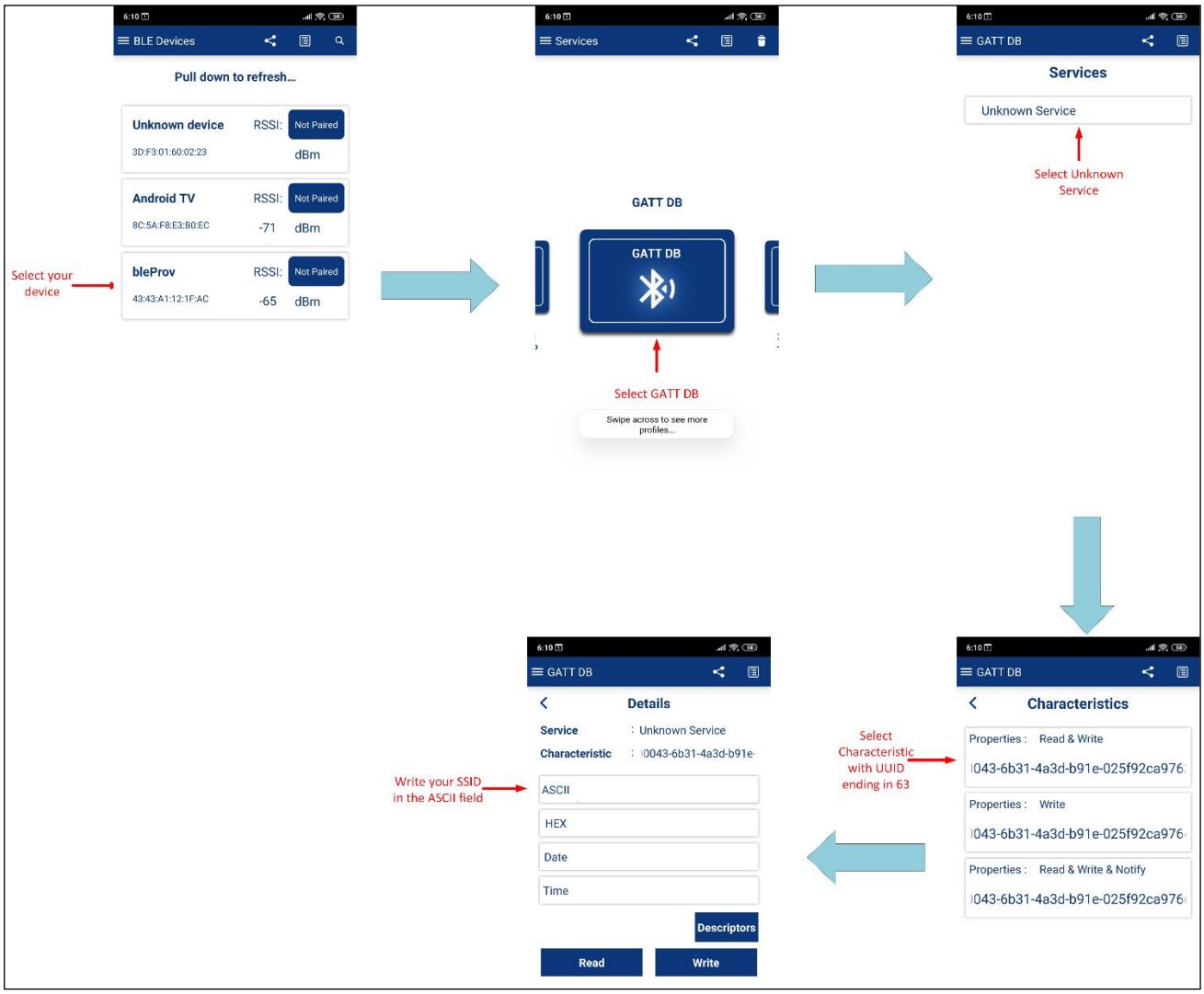
   Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control: None; New line for receive data: Line Feed (LF) or Auto setting.
3. Program the board.

   After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for the device to make all the required connections.

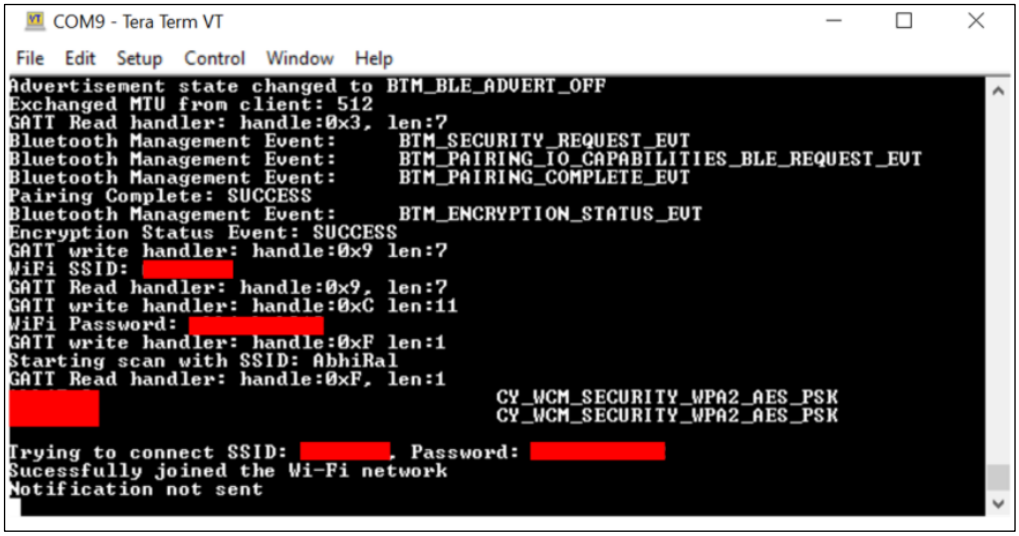


4. To test using the AIROC™ Bluetooth® Connect App mobile app, do the following:
   a. Turn ON Bluetooth® on your Android or iOS device.
   b. Launch the app.
   c. Press the reset switch on the kit to start sending advertisements.
   d. Swipe down on the app home screen to start scanning for Bluetooth® LE Peripherals. Your device ("bleProv") appears in the app home screen. Select your device to establish a Bluetooth® LE connection.
   e. Select the **GATT DB** Profile from the carousel view then select **Unknown Service**.
   f. Select the attribute with the UUID ending in 63. In the ASCII field, type your Wi-Fi SSID in string format. Do the same for password UUID ending in 64) as described above.

Using example projects



5. Select the attribute with the UUID ending in 65. Select **Notify**. Write hex value 1 to this characteristic to connect to the Wi-Fi network. If the connection is successful, then the server will send a notification with the value 1 or with the value 0.



Once the Wi-Fi SSID and password are provided by the client it is stored in the EEPROM. To delete this data the user needs to press the User Button.

4.3 Azure RTOS NetXDuo Wi-Fi UDP echo server

This application provides an example of Azure RTOS NetX/NetXDuo stack usage. It shows you how to develop a NetX UDP server to communicate with a remote client using the NetX UDP socket API.

This example demonstrates how an STM32H7 can be used to host CYW43xxx/CYW55xxx connectivity devices.

4.3.1 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

4.3.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

Download echotool utility.

4.3.3 Project components

The following are the only components used in this project:

- Wifi/network-interface (configured as NetXDuo)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Bluetooth/btstack
- Bluetooth/btstack-integration
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the ThreadX kernel)
- Platform/device (configured as CYW43012)

4.3.4 Example project start/import

You can open this example by copying the example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_netxduo

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

4.3.5 Project hardware setup

- Refer to section Hardware Setup.

4.3.6 Operation

1. Connect the board to your PC using the provided USB cable through the ST-Link USB connector.
2. Modify the `WIFI_SSID` and `WIFI_PASSWORD` macros in *Application/User/NetXDuo/console_task.c* to match with those of the Wi-Fi network that you want to connect to.
3. Update the `DEFAULT_PORT` macro in *Application/User/NetXDuo/console_task.c*.
4. Open a terminal program and select the **ST-Link COM** port. Set the serial port parameters to 8N1 and 115200 baud.
5. Program the board using STM32CubeIDE or EWARM.

   After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for the device to make the required connections.
6. Run the echotool utility on a windows console as following:

   ```
   # echotool.exe  <board IP address> /p udp  /r  <DEFAULT_PORT> /n 10 /d "Hello World"
   ```

   Example usage:

   ```
   echotool.exe 192.168.1.2 /p udp /r 6000 /n 10 /d "Hello World"
   ```

4.4 Bluetooth® LE Hello Sensor

This code example demonstrates the implementation of a simple Bluetooth® Stack functionality in GAP Peripheral role. During initialization the app registers with LE stack to receive various notifications including bonding complete, connection status change and peer write. Peer device can also write to the client configuration descriptor of the notification characteristic.

4.4.1 Features demonstrated

- GATT database and Device configuration initialization
- Registration with LE stack for various events
- Sending data to the client
- Processing write requests from the client

4.4.2 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

4.4.3 Other Software

This code example requires two devices: Host (Mobile Phone or PC) and a Target (STM32H747 DISCO Kit).

1. For the Host, download and install the AIROC™ Bluetooth® Connect App for iOS or Android. Scan the following QR codes from your mobile phone to download the AIROC™ Bluetooth® Connect App:

2. Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

4.4.4 Project Components

The following are the only components used in this project:

- Bluetooth/btstack
- Bluetooth/bluetooth-freertos
- Platform/pal
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device
- connectivity-utilities
- pal (minimum interface to ST HAL to enable connectivity)

4.4.5 Example Project Start/Import

You can open the Bluetooth® Hello Sensor example by copying the example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\BLE_hello_sensor

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

4.4.6 Project Hardware Setup

Refer to section Hardware Setup.

4.4.7 Operation

1. Connect the STM32H747 DISCO Kit to your PC.
2. Use your favorite serial terminal application and connect to the ST-LINK (CN2) COM port. Configure the terminal application to access the serial port using the following settings.

Baud rate: 115200 bps; Data: 8 bits; Parity : None; Stop : 1 bit; Flow control : None; New line for receive data : Line Feed(LF) or Auto setting

3. Program the board.

4. After programming, the application starts automatically. Observe the messages on the UART terminal. Use the ST-LINK (CN2) COM port to view the Bluetooth® stack and application trace messages in the terminal window as shown below:

```
[0] Hello Sensor Start
[0] wiced_bt_stack_init()
[515] bt_post_reset_cback()
[515] bt_post_reset_cback(): Change baudrate (3000000) for FW downloading
[516] bt_update_controller_baudrate(): 3000000
[521] bt_baudrate_updated_cback(): Baudrate is updated for FW downloading
[522] bt_update_platform_baudrate(): 3000000
[722] bt_start_fw_download(): FW ver = [1428] bt_patch_download_complete_cback():
status=1
[1428] bt_fw_download_complete_cback(): Reset baudrate to 115200
[1429] bt_update_platform_baudrate(): 115200
[1630] bt_fw_download_complete_cback(): Changing baudrate to 3000000
[1630] bt_update_controller_baudrate(): 3000000
[2065] bt_baudrate_updated_cback(): Baudrate is updated for feature
```

5. To test using the mobile app, do the following:

   a. Turn ON Bluetooth® on your Android or iOS device.

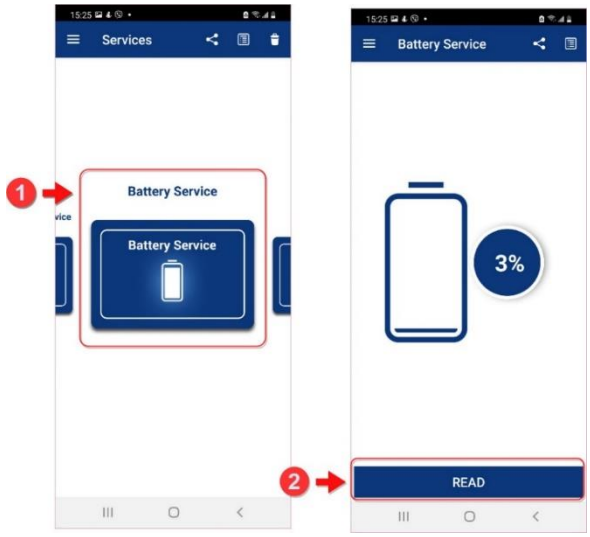
   b. Launch the app on your Phone.

   c. Swipe down on the app home screen to start scanning for Bluetooth® LE Peripherals; your device ("hello") appears in the app home screen. Select your device to establish a Bluetooth® LE connection.

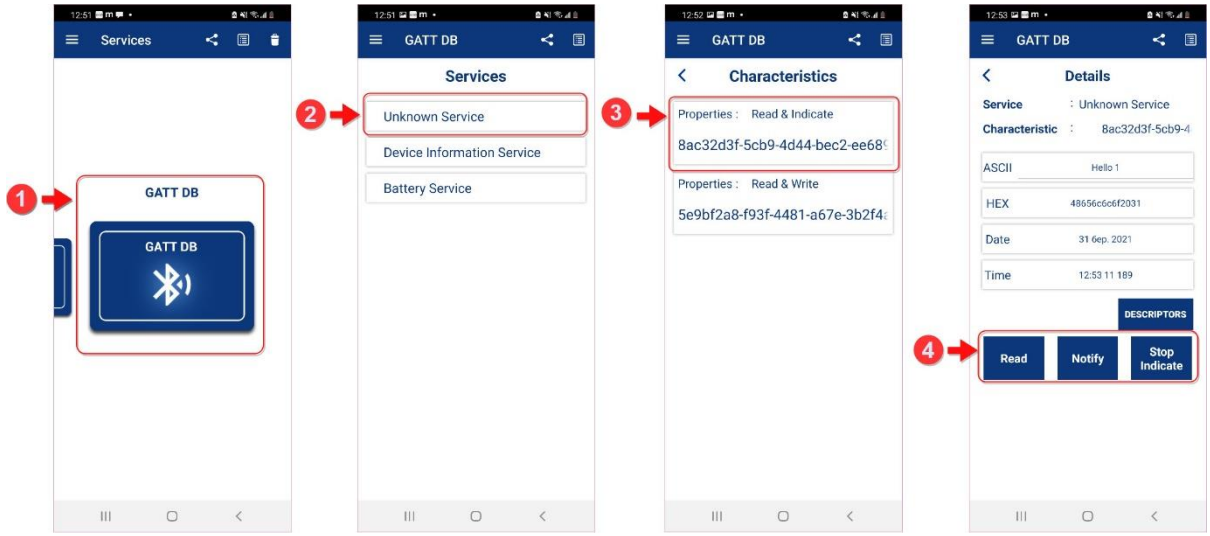
   d. Read Battery.

   - Select the 'Battery' Profile from the carousel view.
   - Press Read button.

Using example projects

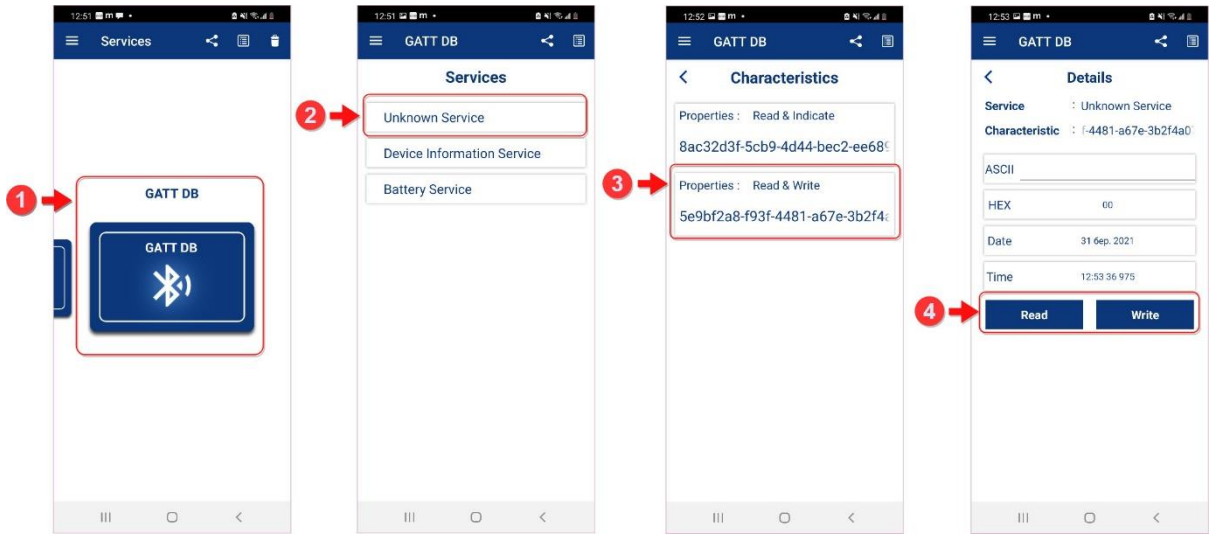


e. Enable Sensor notification/indication as shown in the following images.



The notification "Hello **N**" appears every 10 seconds.

f. Read / Write the Sensor characteristic.

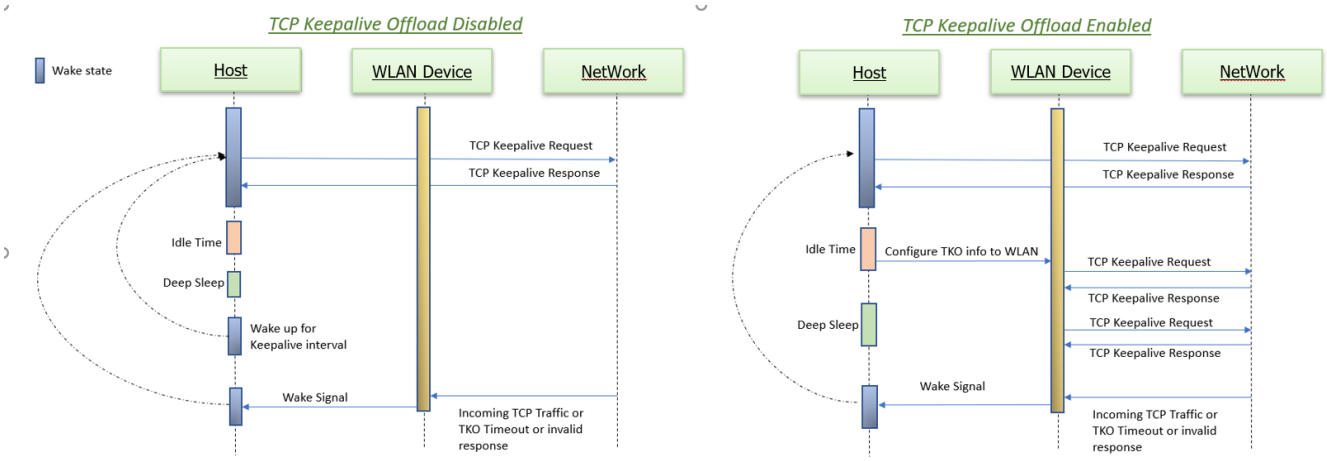


4.5 Wi-Fi TCP keepalive offload

The TCP keepalive offload part of the Low Power Assistant (LPA) improves the power consumption of your connected system by reducing the time the Host needs to stay awake to support the TCP keepalive request. This example describes how to enable TCP keepalive offload and configure four different sockets for TCP keepalive that can be incorporated into your project from LPA middleware.

TCP keepalive maintains idle TCP connections by periodically passing packets between the client and server. If either the client or server does not respond to a packet, the connection is considered inactive and is terminated. This helps in pruning dead connections. Typically, TCP keepalives are sent every 45 or 60 seconds on an idle TCP connection, and the connection is dropped after 3 sequential ACKs are missed. This means the Host MCU has to wake up periodically to send a TCP keepalive packet to maintain the TCP connection during idle state.

TCP keepalive offload helps in moving this functionality to WLAN firmware so that the Host MCU does not need to wake up periodically to send/receive TCP keepalive packets. This functionality is offloaded only when the Host MCU goes to sleep and the network stack is suspended.



4.5.1 Hardware

Refer to section the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

4.5.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

4.5.3 Project components

The following are the components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/secure-sockets

Using example projects

- Wifi/LwIP
- Wifi/mbedtls
- Wifi/lpa
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)
- Platform/module(configured as MURATA-1V)

4.5.4 Example project start/import

You can open the Wi-Fi TCP keepalive offload example by copying the example from the Pack to an appropriate location:

> ***C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_tko***

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps described in the Wi-Fi Scan section (sections 4.1.4 - 4.1.6).

4.5.5 Project hardware setup

Refer to section Hardware Setup.

4.5.6 Operation

1. Connect the STM32H747 DISCO Kit to your PC.
2. Open **app_config.h** and modify the **WIFI_SSID**, **WIFI_PASSWORD**, and **WIFI_SECURITY_TYPE** macros to match the Wi-Fi network credentials that you want to connect to. All possible security types are defined in the cy_wcm_security_t structure in cy_wcm.h file.
3. Ensure that your computer is connected to the same Wi-Fi access point (AP) that you configured in Step 2 and Setup a TCP server and the server starts listening for incoming TCP connections.
4. Open a **cycfg_connectivity_wifi.c** and modify cy_tko_ol_cfg_0, ports, r**emote_port** and **remote_ip** to match the TCP server that be set up on your computer;
5. Use your favorite serial terminal application and connect to the ST-LINK (CN2) COM port. Configure the terminal application to access the serial port using the following settings.

   Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control: None; New line for receive data: Line Feed (LF) or Auto setting
6. Program the board.

Note: For dual cores MCU (e.g STM32H7) the both cores must be programmed.

7. After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for the device to make all the required connections. application trace messages in the terminal window as shown:

```
WLAN MAC Address : 00:A0:50:45:13:81
WLAN Firmware    : wl0: Apr 12 2022 20:39:36 version 13.10.271.287 (760d561 CY) FWID 01-b438e2a0
```

```
WLAN CLM         : API: 18.2 Data: 9.10.0 Compiler: 1.36.1 ClmImport: 1.34.1 Creation: 2021-04-26
04:01:15
WHD VERSION      : v2.4.0 : v2.4.0 : GCC 10.3 : 2022-08-04 17:12:02 +0800
Info: Wi-Fi initialization is successful
Info: Join to AP: SM9500
Info: Successfully joined wifi network SM9500
Info: Assigned IP address: 192.168.43.124
Info: Taking TCP Keepalive configuration from the Generated sources.
Info: Socket[0]: Created connection to IP 192.168.43.228, local port 3353, remote port 3360
Info: Skipped TCP socket connection for socket id[1]. Check the TCP Keepalive configuration.
Info: Skipped TCP socket connection for socket id[2]. Check the TCP Keepalive configuration.
Info: Skipped TCP socket connection for socket id[3]. Check the TCP Keepalive configuration.
whd_tko_toggle: Successfully enabled

Network Stack Suspended, MCU will enter DeepSleep power mode
Resuming Network Stack, Network stack was suspended for 31867ms

=====================================================
WHD Stats..
tx_total:73, rx_total:74, tx_no_mem:0, rx_no_mem:0
tx_fail:0, no_credit:0, flow_control:0
Bus Stats..
cmd52:2430, cmd53_read:393, cmd53_write:596
cmd52_fail:7, cmd53_read_fail:0, cmd53_write_fail:0
oob_intrs:0, sdio_intrs:484, error_intrs:0, read_aborts:0
=====================================================
Network is active. Resuming network stack
whd_tko_toggle: Successfully disabled
whd_tko_toggle: Successfully enabled

Network Stack Suspended, MCU will enter DeepSleep power mode
Resuming Network Stack, Network stack was suspended for 4142ms
```

4.6 Wi-Fi MQTT client

This code example demonstrates implementing an MQTT client using the MQTT library. The library uses the AWS IoT device SDK Port library and implements the glue layer that is required to work with Infineon connectivity platforms.

In this example, the MQTT client RTOS task establishes a connection with the configured MQTT broker, and creates two tasks: publisher and subscriber. The publisher task publishes messages on a topic when the user button is pressed on the kit. The subscriber task subscribes to the same topic and controls the user LED based on the messages received from the MQTT broker. In case of unexpected disconnection of MQTT or Wi-Fi connection, the application executes a re-connection mechanism to restore the connection.

Sequence of operation

1. Press the user button.
2. The GPIO interrupt service routine (ISR) notifies the publisher task.
3. The publisher task publishes a message on a topic.
4. The MQTT broker sends back the message to the MQTT client because it is also subscribed to the same topic.

5. When the message is received, the subscriber task turns the LED ON or OFF. As a result, the user LED toggles every time you press the button.

4.6.1 Hardware

Refer to section the STM32 hardware configuration descriptions as appropriate:

- Using STM32U575I-EV Evaluation board

4.6.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

4.6.3 Project components

The following are the components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/secure-sockets
- Wifi/LwIP
- Wifi/mbedtls
- Wifi/lpa
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)
- Platform/module(configured as MURATA-1V)
- Wifi/aws-iot-device-sdk-embedded-C
- Wifi/aws-iot-device-sdk-port
- Wifi/mqtt

4.6.4 Example project start/import

You can open the Wi-Fi MQTT client example by copying the example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32U575I-EV\Applications\ wifi_mqtt_client

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps described in the Wi-Fi Scan section (sections 4.1.4 - 4.1.6).

4.6.5 Project hardware setup

Refer to section Hardware Setup.

4.6.6 Operation

1. Connect the STM32U575I-EV Kit to your PC.
2. Wi-Fi configuration: Set the Wi-Fi credentials in Core/Src/wifi_config.h: Modify the macros WIFI_SSID, WIFI_PASSWORD, and WIFI_SECURITY to match with that of the Wi-Fi network that you want to connect.
3. MQTT configuration: Set up the MQTT client (AWS IoT MQTT) and configure the credentials in Core*/Src/mqtt_client_config.h*. Some of the important configuration macros are as follows:
   - MQTT_BROKER_ADDRESS: Hostname of the MQTT broker.
   - MQTT_PORT: Port number to be used for the MQTT connection. As specified by IANA (Internet Assigned Numbers Authority), port numbers assigned for MQTT protocol are 1883 for non-secure connections and 8883 for secure connections. However, MQTT brokers may use other ports. Configure this macro as specified by the MQTT broker. For AWS IoT MQTT, use port as 8883.
   - MQTT_SECURE_CONNECTION: Set this macro to 1 if a secure (TLS) connection to the MQTT broker is required to be established; else 0.
   - MQTT_USERNAME and MQTT_PASSWORD: User name and password for client authentication and authorization, if required by the MQTT broker. However, note that this information is generally not encrypted and the password is sent in plain text. Therefore, this is not a recommended method of client authentication.
   - CLIENT_CERTIFICATE and CLIENT_PRIVATE_KEY: Certificate and private key of the MQTT client used for client authentication. Note that these macros are applicable only when MQTT_SECURE_CONNECTION is set to 1.
   - ROOT_CA_CERTIFICATE: Root CA certificate of the MQTT broker.
   - MQTT_PUB_TOPIC: MQTT topic to which the messages are published by the Publisher task to the MQTT broker.
   - MQTT_SUB_TOPIC: MQTT topic to which the subscriber task subscribes to. The MQTT broker sends the messages to the subscriber that are published in this topic (or equivalent topic).
   - For AWS IoT MQTT, configure the following mandatory fields:

     ```
     CLIENT_CERTIFICATE - xxxxxxxxxx.cert.pem
     CLIENT_PRIVATE_KEY - xxxxxxxxxx.private.key
     ROOT_CA_CERTIFICATE - Root CA certificate
     ```
4. Use your favorite serial terminal application and connect to the ST-LINK (CN2) COM port. Configure the terminal application to access the serial port using the following settings.

   Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control: None; New line for receive data: Line Feed (LF) or Auto setting.
5. Program the board.

Note: For dual cores MCU (e.g STM32H7) both cores must be programmed.

6. After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for the device to make all the required connections. Application trace messages in the terminal window as shown:

```
WLAN MAC Address : E8:E8:B7:9F:D6:E0
WLAN Firmware    : wl0: Jul 31 2023 06:07:24 version 13.10.271.305 (f2b5c53 CY)
FWID 01-e6b954e
WLAN CLM         : API: 18.2 Data: Murata.1LVindoorSTA Compiler: 1.36.1 ClmImport:
1.36.3 Customization: v2 191015 Creation: 2020-01-22 06:19:41
WHD VERSION      : 3.2.1.21799 : master dev-v3.2.1 : GCC 11.3 : 2023-10-19
03:48:22 -0500
Offloads not configured
Wi-Fi Connection Manager initialized.
Wi-Fi Connecting to 'WIFI_SSID'
Successfully connected to Wi-Fi network 'WIFI_SSID'.
IPv4 Address Assigned: 192.168.1.8
MQTT library initialization successful.
'stm32-mqtt-client8462' connecting to MQTT broker 'a33jl9z28enc1q-ats.iot.us-east-
2.amazonaws.com'...
MQTT connection successful.
MQTT client subscribed to the topic 'ledstatus' successfully.
Press the user button (SW2) to publish "TURN ON"/"TURN OFF" on the topic
'ledstatus'...
```

7. Once the initialization is complete, confirm that the message 'Press the user button (SW2) to publish "TURN ON"/"TURN OFF" on the topic', which is configured in 'MQTT_PUB_TOPIC" and printed on the UART terminal. This message may vary depending on the MQTT topic and publish messages that are configured in the *mqtt_client_config.h* file.

8. Press the user button on the kit to publish "TURN ON" message to MQTT client.

9. Confirm that the messages received on the subscribed topic are printed on the UART terminal.

```
Publisher: Publishing 'TURN ON' on the topic 'ledstatus'
Subsciber: Incoming MQTT message received:
    Publish topic name: ledstatus
    Publish QoS: 1
    Publish payload: TURN ON

Publisher: Publishing 'TURN OFF' on the topic 'ledstatus'
Subsciber: Incoming MQTT message received:
    Publish topic name: ledstatus
    Publish QoS: 1
    Publish payload: TURN OFF
```

10. Publish and Subscribe functionalities of the MQTT client can be individually verified if the MQTT broker supports a test MQTT client like the AWS IoT.

    - To verify the subscribe functionality: Using the test MQTT client, publish messages such as "TURN ON" and "TURN OFF" on the topic specified by the MQTT_PUB_TOPIC macro in *mqtt_client_config.h* and check the messages on the terminal.
    - To verify the publish functionality: From the Test MQTT client, subscribe to the MQTT topic specified by the MQTT_SUB_TOPIC macro and confirm that the messages published by the kit (when the user button is pressed) are displayed on the test MQTT client's console.

4.7 Wi-Fi Enterprise Security

This application provides a console interface to test enterprise Wi-Fi commands. This application integrates the command console library.

4.7.1 Hardware

Refer to section the STM32 hardware configuration descriptions as appropriate:

- Using STM32U575I-EV Evaluation board

4.7.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

4.7.3 Project components

The following are the components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/enterprise-security
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/secure-sockets
- Wifi/LwIP
- Wifi/mbedtls
- Wifi/lpa
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)
- Platform/module(configured as MURATA-1LV)
- MfgTools/command-console

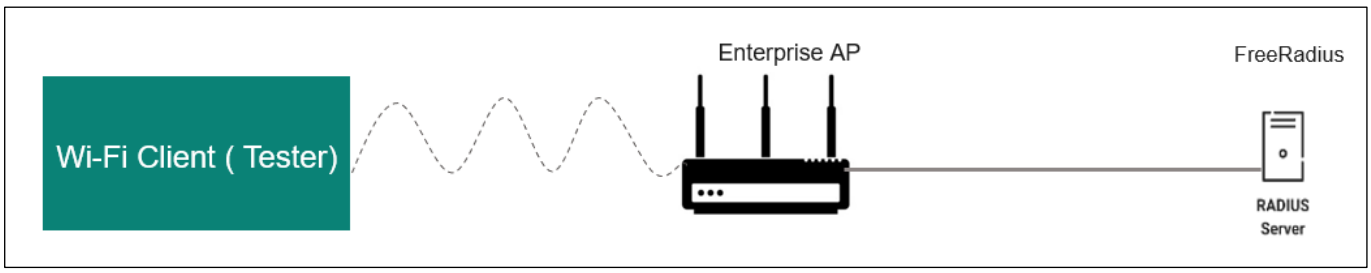
4.7.4 Example project start/import

You can open the Wi-Fi MQTT client example by copying the example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\NUCLEO-H745ZI-Q\Applications\wifi_ent_sec

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps described in the Wi-Fi Scan section (sections 4.1.4 - 4.1.6).

4.7.5 Project hardware setup



- RADIUS Server: Linux PC + FreeRadius
- Enterprise AP: WPA2/WPA3-Enterprise support
- Wi-Fi Client

Configuration of RADIUS Server and Enterprise AP are outside the scope of this document.

4.7.6 Operation

1. Connect the STM32U575I-EV Kit to your PC.
2. Wi-Fi configuration: Modify macros in *Core/Src/certificate.h*:
   - WIFI_ROOT_CERTIFICATE_STRING / WPA3_192BIT_ROOT_CERTIFICATE_STRING: Root CA certificate
   - WIFI_USER_PRIVATE_KEY_STRING / WPA3_192BIT_USER_PRIVATE_KEY_STRING: Private Key of Client (not protected by password)
   - WIFI_USER_CERTIFICATE_STRING / WPA3_192BIT_USER_CERTIFICATE_STRING: Certificate of client

Note: Private key is certificate are provided from network administrator. But you create them by yourself for test purpose. See section 4.7.7.

3. Use your favorite serial terminal application and connect to the ST-LINK (CN2) COM port. Configure the terminal application to access the serial port using the following settings.

   Baud rate: 115200 bps; Data: 8 bits; Parity: None; Stop: 1 bit; Flow control: None; New line for receive data: Line Feed (LF) or Auto setting.

4. Program the board.

Note: For dual-core MCUs (e.g., STM32H7) both cores must be programmed.

After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for command prompt.

5. Enterprise commands:

   - `join_ent`: This command will connect the device to enterprise Wi-Fi network.

     ```
     join_ent <ssid> <eap_protocol> [username] [password] <wifi auth_type>
     ```

     Example:

     ```
     join_ent WIFI_SSID eap_tls user pass wpa2_aes
     ```

     Note: WPA2/3 Enterprise command auth types:

     - *wpa2_aes: WPA2 Enterprise only mode*
     - *wpa3_aes_ccmp: WPA3 Transition mode*
     - *wpa3_aes_gcm: WPA3 Enterprise only mode*
     - *wpa3_192bit: WPA3-Enterprise 192-bit mode*

     Note: EAP security protocols :

     - *eap_tls : EAP TLS*
     - *peap: PEAPv0 with MSCHAPv2*
     - *eap_ttls: EAP-TTLS with EAP-MSCHAPv2*

   - `leave_en` : Leave from the connected enterprise Wi-Fi network.

4.7.7 Creation of Private Key and Certificate of Root CA, Radius Server(FreeRadius) and Client

1. Generate Root Key rootKey.pem.

   ```
   openssl ecparam -name prime256v1 -genkey -noout -out rootKey.pem
   ```

2. Generate Root Certificate: This shall be updated in WIFI_ROOT_CERTIFICATE_STRING / WPA3_192BIT_ROOT_CERTIFICATE_STRING. Also update the root certificate in radius server.

   ```
   openssl req -new -key rootKey.pem -x509 -nodes -days 365 -out rootCert.pem
   ```

   This command will prompt for necessary information like country code, State etc.

3. Generate Client Key and certificate.

   ClientKey.pem shall be updated in WIFI_USER_PRIVATE_KEY_STRING / WPA3_192BIT_USER_PRIVATE_KEY_STRING.

   client.crt shall be updated in WIFI_USER_CERTIFICATE_STRING / WPA3_192BIT_USER_CERTIFICATE_STRING.

   a. Create Client Key ClientKey.pem:

      ```
      openssl ecparam -name prime256v1 -genkey -noout -out clientKey.pem
      ```

   b. Create CSR :

      ```
      openssl req -new -sha256 -key clientKey.pem -out client.csr
      ```

   c. Create Client Certificate client.crt:

      ```
      openssl x509 -req -days 365 -in client.csr -CA rootCert.pem -CAkey rootKey.pem
      -CAcreateserial -out client.crt
      ```

4. Generate Server Key and certificate. These shall be loaded in radius server:

   a. Create Server Key serverKey.pem:

      ```
      openssl ecparam -name prime256v1 -genkey -noout -out serverKey.pem
      ```

   b. Create CSR :

      ```
      openssl req -new -sha256 -key serverKey.pem -out server.csr
      ```

   c. Create Server Certificate server.crt:

      ```
      openssl x509 -req -days 365 -in server.csr -CA rootCert.pem -CAkey rootKey.pem
      -CAcreateserial -out server.crt
      ```

5 Manufacture tools

The following manufacture tool projects are included in the pack:

- Tester - Wi-Fi Bluetooth® Console
- WLAN manufacturing test application (Wifi-Mfg-Tester) for FreeRTOS
- Bluetooth® Manufacturing Test Application for FreeRTOS

5.1 Tester - Wi-Fi Bluetooth® Console

This application integrates the command console library including Wi-Fi iPerf and Bluetooth® Low Energy functionality. You can use this application to characterize the Wi-Fi/Bluetooth® LE functionality and performance.

This example demonstrates how an STM32H7 can be used to host CYW43xxx/CYW55xxx connectivity devices.

5.1.1 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

5.1.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

Setting up iPerf on the host:

- iPerf 2.0.13 (supported on Ubuntu, macOS, and Windows)
- Go to the iPerf installation directory and launch the terminal (command prompt for Windows, terminal shell for macOS or Ubuntu).

5.1.3 Project components

The following are the only components used in this project:

- Wifi/network-interface (configured as LWIP)
- Wifi/wifi-host-driver (WHD)
- Wifi/wcm
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/secure-sockets
- Wifi/LwIP
- Bluetooth/btstack
- Bluetooth/btstack-integration
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)
- MfgTools/command-console

5.1.4 Example project start/import

You can open this example by copying the example from the Pack to an appropriate location:

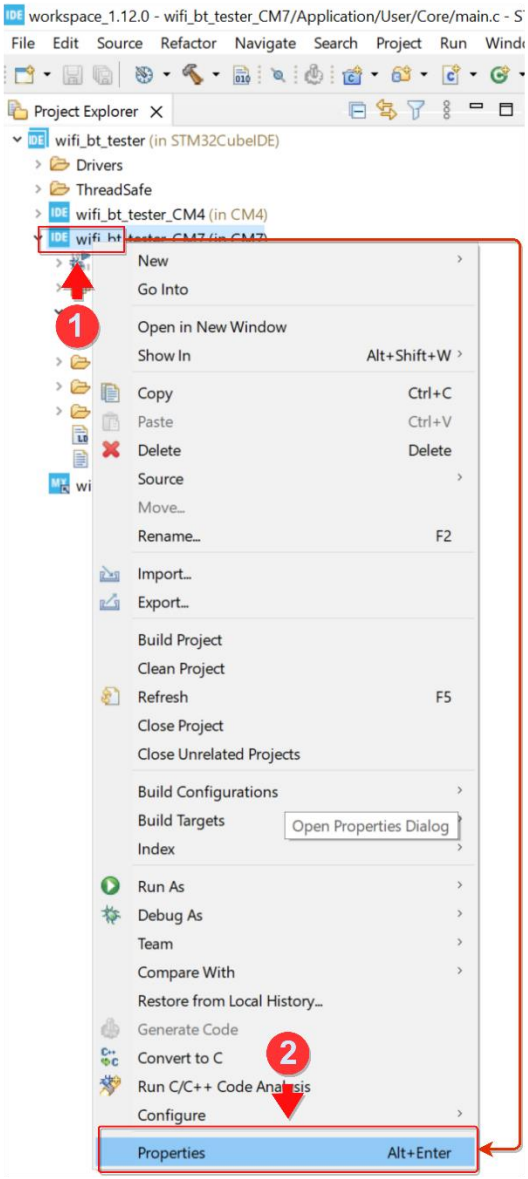
C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_bt_tester

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

Also check in your project workspace that **MCU Settings** and **Preprocessor** are configured correctly.

Open Properties

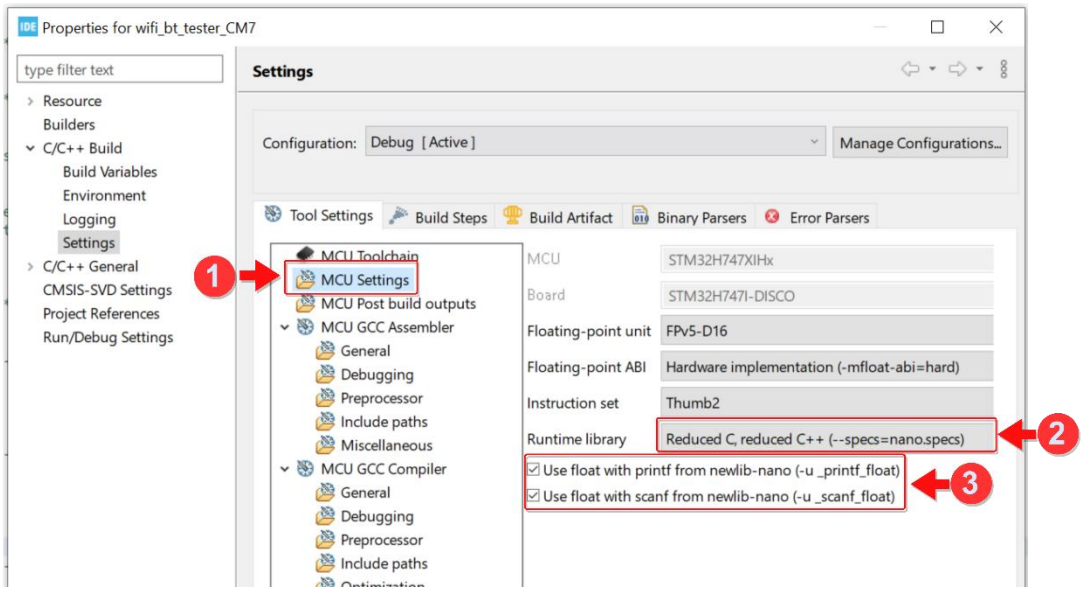
1. Right-click on a project.
2. Select **Properties**



Manufacture tools

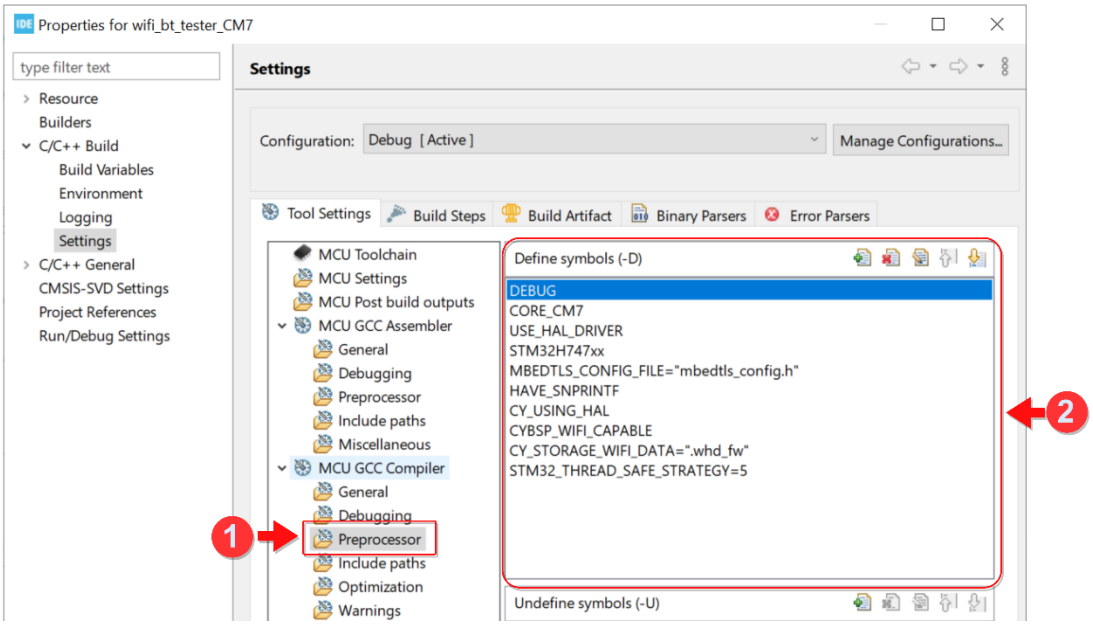
MCU Settings

1. In project Properties select C**/C++ Build > Settings > MCU Settings**.
2. Check **Runtime library** should be Reduced C, reduced C++ (--specs=nano.specs)
3. Set tick for **Use float with printf** and **Use float with scanf**



Preprocessor

1. In project **Properties**, select **C/C++ Build > Settings > MCU GCC Compiler > Preprocessor** .
2. Check if these defines exist:
   - HAVE_SNPRINTF
   - CY_USING_HAL
   - CYBSP_WIFI_CAPABLE
   - CY_STORAGE_WIFI_DATA=".whd_fw"



5.1.5 Project hardware setup

Refer to section Hardware Setup.

5.1.6 Operation

Connect the board to your PC using the provided USB cable through the ST-Link USB connector.

Modify the WIFI_SSID and WIFI_KEY macros in Application/User/Core/console_task.c to match with those of the Wi-Fi network that you want to connect to.

1. To join a Wi-Fi network of a specific band, update the **WIFI_BAND** macro in *Application/User/Core/console_task.c* as follows:

   CY_WCM_WIFI_BAND_2_4GHZ: 2.4-GHz band **CY_WCM_WIFI_BAND_5GHZ**: 5-GHz band

   Configure the TCP window size in iPerf before building the application. See the command console library's README.md for instructions on how to configure the TCP window size.

2. Open a terminal program and select the ST-Link COM port. Set the serial port parameters to 8N1 and 115200 baud.

   Program the board using STM32CubeIDE or EWARM. After programming, the application starts automatically. Observe the messages on the UART terminal, and wait for the device to make the required connections.

3. The application connects to the configured Wi-Fi access point (AP) and obtains the IP address. When the device is ready, the > prompt appears.

4. Run iPerf commands (client and server) against a remote peer device.

   See Running iPerf client and server against a remote peer device.

5. Run Bluetooth® LE commands against a remote peer device.

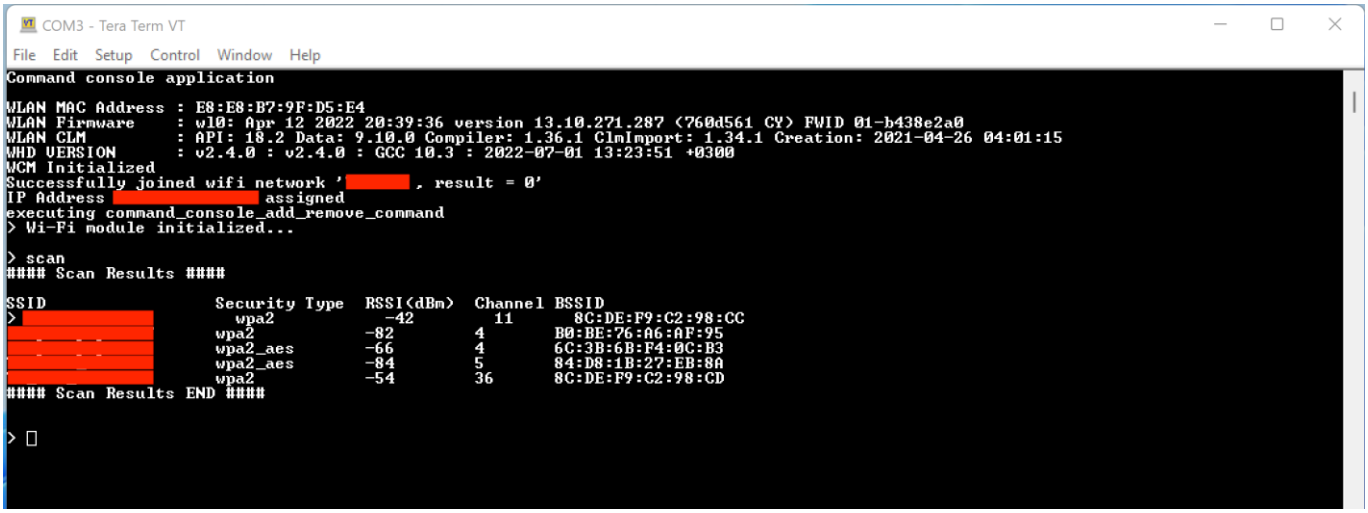
5.1.7 Serial terminal setup

The terminal interface is a virtual COM port which is part of the ST-LINK (CN2) USB connection.

Terminal emulator configuration:

- Baud Rate: 115200
- Data Length: 8 Bits
- Stop Bit(s): 1
- Parity: None
- Flow control: None

5.1.8 Example output

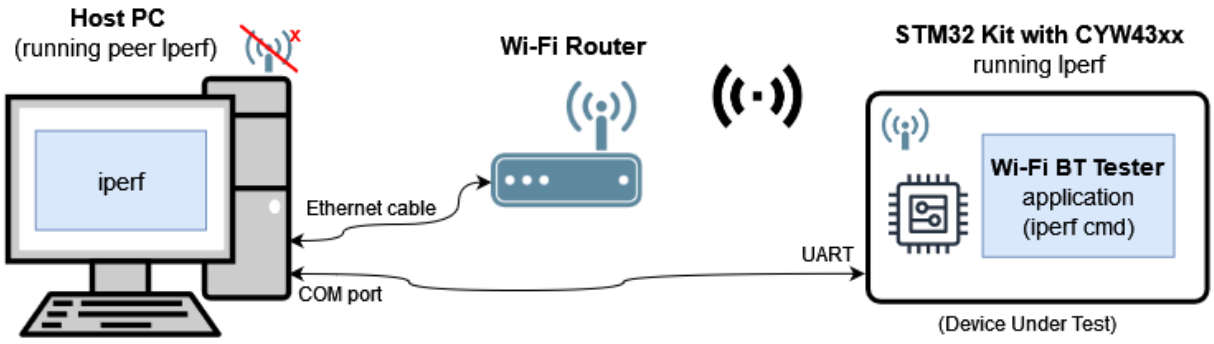


5.1.9 iPerf measurement

iPerf commands are used for measuring the Wi-Fi performance/throughput. The iperf tool sends TCP/UDP data between two peer devices to compute the Wi-Fi performance/throughput.

5.1.9.1 iPerf setup

The following diagram shows the exact setup that should be used for measuring the Wi-Fi performance/throughput of a STM32 device using iperf.



5.1.9.2 iPerf commands for Wi-Fi throughput measurement

Enter the following commands on the STM32 device (DUT) after the device boots up and connects to the Wi-Fi network. This section provides only the commands to be run on the DUT. When the 'client iperf command' runs on the DUT, the 'server iperf command' should run on the host PC (as shown in the iPerf Setup diagram), and vice versa.

1. Start iPerf as a TCP server:

```
iperf -s
```

Note: On the peer iPerf device (host PC), start iPerf as a TCP client to send the TCP data.

2. Start iPerf as a TCP client:

Manufacture tools

```
iperf -c <server_ip_addr> -t <time in sec>
```

Sample command:

```
iperf -c 192.168.0.100 -t 60
```

Note: On the peer iPerf device (host PC), start iPerf as a TCP server.

3. Start iPerf as a UDP server:

```
iperf -s -u
```

Note: On the peer iPerf device (host PC), start iPerf as a UDP client to send the UDP data.

4. Start iPerf as a UDP client:

```
iperf -c <server_ip_addr> -t <time in sec> -u -b <band width>
```

Sample command:

```
iperf -c 192.168.0.100 -t 60 -u -b 50M
```

Note: On the peer iPerf device (host PC), start iPerf as a UDP server.

5.1.9.3 Results

STM32H747 DISCO + CYW43012

TCP/ UDP	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	35.3	42.7	iperf -c <ip> -t 60
TCP RX	35.7	39.2	iperf -s
UDP TX	52.4	52.4	iperf -c <ip> -t 60 -u -b 50M
UDP RX	50.0	50.0	iperf -s -u

Test configuration: Iperf app run on STM32H747 CM7/400Mhz, GCC, Wi-Fi router: Asus RT-AX56U.

STM32L5-DK + CYW43012

TCP/ UPD	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	20.2	20.5	iperf -c <ip> -t 60
TCP RX	20.6	20.8	iperf -s
UDP TX	31.0	31.1	iperf -c <ip> -t 60 -u -b 50M
UDP RX	25.7	24.7	iperf -s -u

Test configuration: Iperf app run on STM32L5 CM33/110Mhz, GCC, Wi-Fi router: Asus RT-AX56U.

STM32U575I-EV + CYW43012

TCP/ UPD	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	26.5	27.4	iperf -c <ip> -t 60
TCP RX	25.3	26.1	iperf -s
UDP TX	36.5	36.6	iperf -c <ip> -t 60 -u -b 50M
UDP RX	33.8	33.9	iperf -s -u

Test configuration: Iperf app run on STM32U5 CM33/160Mhz, GCC, Wi-Fi router: Asus RT-AX56U.

STM32H747 DISCO + CYW43022

TCP/ UDP	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	32,6	36,9	iperf -c <ip> -t 60
TCP RX	33,4	37,1	iperf -s
UDP TX	52,1	52,2	iperf -c <ip> -t 60 -u -b 80M
UDP RX	57,3	64,9	iperf -s -u

Test configuration: Iperf app run on STM32H747 CM7/400Mhz, GCC, Wi-Fi router: NEC Aterm WX7800 (11ax).

STM32H747 DISCO + CYW55500

TCP/ UDP	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	44,7	48,7	iperf -c <ip> -t 60

Manufacture tools

TCP/ UDP	Throughput, Mbit/s		Command
	2.4G	5G	
TCP RX	30,1	39,3	iperf -s
UDP TX	78,8	80	iperf -c <ip> -t 60 -u -b 80M
UDP RX	62,3	63,7	iperf -s -u

Test configuration: Iperf app run on STM32H747 CM7/400Mhz, GCC, Wi-Fi router: NEC Aterm WX7800 (11ax).

STM32H747 DISCO + CYW55572

TCP/ UDP	Throughput, Mbit/s		Command
	2.4G	5G	
TCP TX	53,4	57,1	iperf -c <ip> -t 60
TCP RX	37,5	40,3	iperf -s
UDP TX	82,9	83,1	iperf -c <ip> -t 60 -u -b 80M
UDP RX	64,8	64,8	iperf -s -u

Test configuration: Iperf app run on STM32H747 CM7/400Mhz, GCC, Wi-Fi router: NEC Aterm WX7800 (11ax).

STM32N6570DK + Murata 2FY (CYW55513) with external power

TCP/ UDP	Throughput, Mbit/s			Command
	2.4G	5G	6G	
TCP TX	44,3	46	46,3	iperf -c <ip> -t 60
TCP RX	40,3	38,9	39,9	iperf -s
UDP TX	102	98,7	103	iperf -c <ip> -t 60 -u -b 100M
UDP RX	78,5	76,6	80,1	iperf -s -u

Test configuration: Iperf app run on STM32N6570 CM55/600Mhz, GCC, Wi-Fi router: Asus AXE11000.

5.2 WLAN manufacturing test application (Wifi-Mfg-Tester) for FreeRTOS

The Wifi-Mfg-Tester is used to validate the WLAN firmware and radio performance of Wi-Fi chips.

The Wifi-Mfg-Tester acts as a transport layer between the host "wl tool" and the WLAN firmware. It receives the commands from the wl tool and forwards them to the WLAN firmware using IOVAR/IOCTL commands. The Tester also relays the response received back from the WLAN firmware.

The wl tool binaries for testing the WLAN firmware are also included in this application repository.

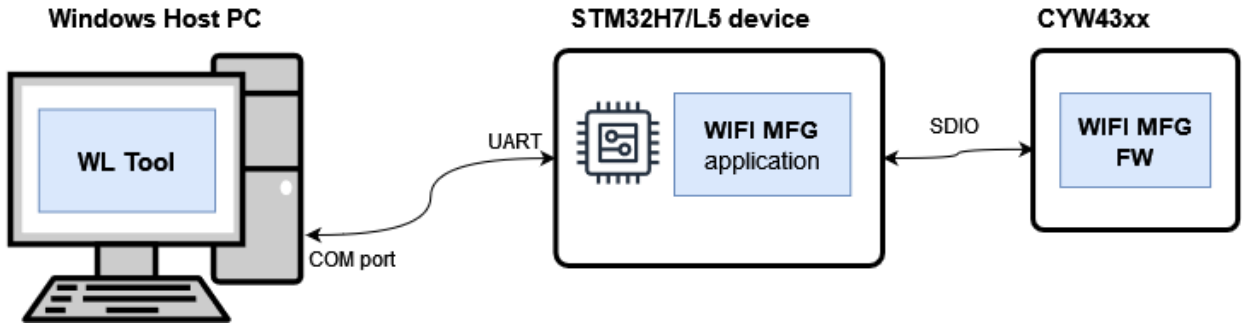
This example demonstrates how an STM32H7 can be used to host CYW43xxx/CYW55xxx connectivity devices.

5.2.1 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate: Using STM32H747 DISCO Kit

Manufacture tools

Test setup is shown below:



5.2.2 Other software

Install a terminal emulator if you don't have one. Instructions in this document use Tera Term.

This application requires the WL tool running on a Windows PC. The pre-built executables for the WL tool are available in the *wl-tool-bin*.

5.2.3 Project components

The following list shows the only components used in this project:

- Wifi/wcm
- Wifi/wifi-mw-core
- Wifi/wifi-host-driver (WHD)
- Wifi/whd-bsp-integration
- Wifi/connectivity-utilities
- Wifi/LwIP
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device
- MfgTools/wifi-mfg-test

5.2.4 Example project start/import

You can open the example by copying this example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_mfg_tester

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

5.2.5 Project hardware setup

Refer to section Hardware Setup.

5.2.6 Operation

1. Go to the WL tool directory:

   ```
   # cd wl-tool-bin
   ```

2. Reset the board by pressing the Reset button.

3. Run the command on Windows host for the WLAN chip on the target board:

   ```
   wl.exe --serial <port> ver
   ```

   For example:

   ```
   #wl.exe --serial 5 ver
   cmd resp: 7/19/2017 build 0
   wl0: Jan 11 2022 21:32:24 version 13.10.271.280 (c32ff79 CY WLTEST) FWID 01-
   3566e923
   ```

4. Observe the output of the command.

   The list of WL commands which can be retrieved by typing `--help`. Partial output of the command and display is as follows:

   ```
   # wl.exe --serial 5 -help

   Usage: wl.exe [-a|i <adapter>] [-h] [-d|u|x] <command> [arguments]
       -h          this message and command descriptions
       -h [cmd]    command description for cmd
       -a, -i      adapter name or number
       -d          output format signed integer
       -u          output format unsigned integer
       -x          output format hexadecimal
       ver          get version information
       cmds         generate a short list of available commands

       ioctl_echo check ioctl functionality
       up       reinitialize and mark adapter up (operational)
       down     reset and mark adapter down (disabled)
       out      mark adapter down but do not reset hardware(disabled)
                On dual-band cards, cards must be band-locked before use.
   ```

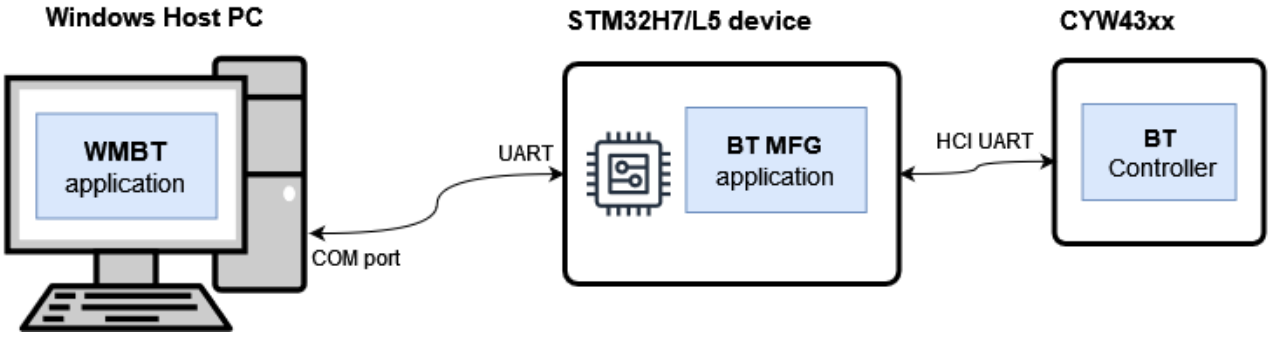
5.3 Bluetooth® Manufacturing Test Application for FreeRTOS

The Bluetooth® Manufacturing Test Application is used to validate the Bluetooth® Firmware and RF performance of Cypress SoC Bluetooth® BR/EDR/LE devices.

The Bluetooth® MFG Application acts as a transport layer between the host "WMBT tool" and Bluetooth® Firmware. Mfg Test Application receive commands from the WMBT tool and forwards them to the Bluetooth® firmware. The Bluetooth® MFG Application also relays the response received back from Bluetooth® firmware.

This example demonstrates how an STM32H7 can be used to host CYW43xxx/CYW55xxx connectivity devices.

Test setup is shown below:



5.3.1 Hardware

Refer to the section on the STM32 hardware configuration descriptions as appropriate:

- Using STM32H747 DISCO Kit

5.3.2 Other software

- This application requires the WMBT Tool running on a windows PC and uses a UART port for communication with the target. The pre-built executables for the WMBT Tool are available in *bt_mfg_tester/wmbt-tool-bin* directory, which sync from btsdk-utils. The user guide is in Bluetooth® Manufacturing Test Tool.
- Use the IQxel tool as a transmitter to send a fixed count test packet to ensure whatever is sent from the transmitter is received without any error.
- Use a Sniffer to ensure that whatever is included in the test packet is in same transmit channel, packet length and data patterns from the transmitter. Better to test it in the shield room to avoid air interference.

5.3.3 Project components

The following are the only components used in this project:

- Bluetooth/btstack
- Bluetooth/btstack-integration
- Platform/pal (PAL, HAL, core-lib)
- Platform/abstraction-rtos (configured for the FreeRTOS kernel)
- Platform/device (configured as CYW43012)

5.3.4 Example project start/import

You can open the example by copying this example from the Pack to an appropriate location:

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\bt_mfg_tester

Once you have copied the example, you can then open it in STM32CubeMX and export to your IDE using the steps from the Wi-Fi Scan example: Example project start/import, Generate code, Build the project.

5.3.5 Project hardware setup

Refer to section Hardware Setup.

5.3.6 Operation

1. Go to WMBT tool directory
2. Reset the Board by pressing Reset button
3. Run the command on Windows Host for the proper BT Chip on target board.
4. Observe the output of the command

   List of wmbt commands with Bluetooth® LE function which can be retrieved by typing --help Partial output of the command and display is below.

   ```
   wmbt reset COMx
   ```

5.3.6.1 Example output

```
# wmbt.exe reset COM5

cmd resp: MBT_BAUD_RATE:  115200
TRANSPORT_MODE: 0 (HCI)

Opened COM5 at speed: 115200
Close Serial Bus
Opened COM5 at speed: 115200

Sending HCI Command:
0000 < 01 03 0C 00 >

Received HCI Event:
0000 < 04 0E 04 01 03 0C 00 >

Success
Close Serial Bus
```

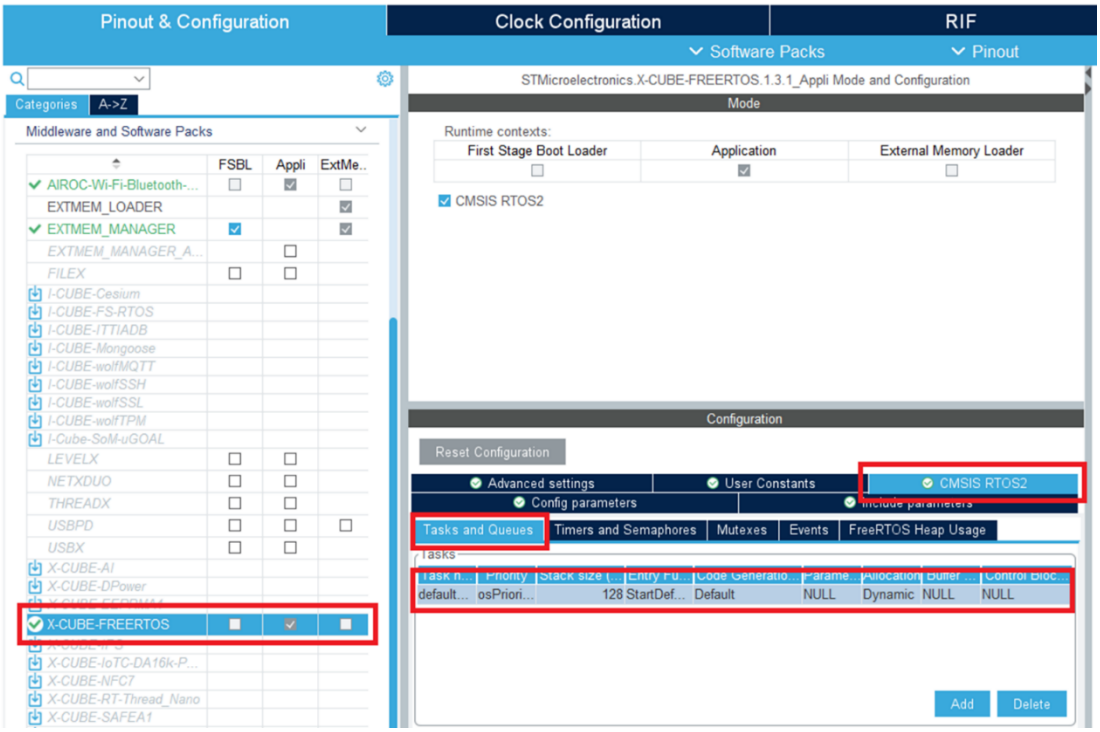
6 Build and program sample project for STM32N6570-DK board

The procedure to build and program a sample application for the STM32N6570-DK board is different from other STM32 variants. Follow these instructions.

6.1 Generating code on STM32CubeMX

Currently, the STM32CubeMX has a bug that it cannot save the default FreeRTOS task configuration in a .ioc file. You have to modify the default FreeRTOS task as follows before generating code.

1. Open the .ioc file using STM32CubeMX.
2. Select **Pinout & Configuration** tab **> Middleware and Software Packs > X-CUBE-FREERTOS > CMSIS RTOS2 > Tasks and Queues > default** Task.



3. Modify the task configuration as follows:
   - Task name: Change Task name per the following table
   - Stack size: Change to 2048
   - Entry Function: Change Entry Function name per the following table
   - Code Generation Option: Change to “As weak”

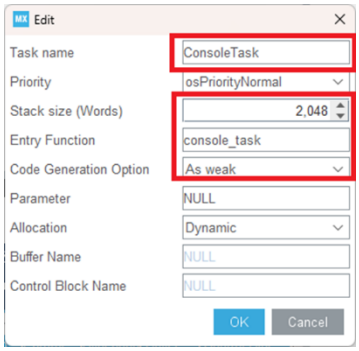
The following table shows the Task name and Entry Function for each application.

Application	Task name	Entry function
wifi_scan	WiFi_Task	WiFiTask
wifi_join_wpa3	WiFi_Task	WiFiTask
wifi_bt_tester	ConsoleTask	console_task

Build and program sample project for STM32N6570-DK board

Application	Task name	Entry function
wifi_mfg_tester	mfg_test_client	mfg_test_client_task
wifi_mqtt_client	mqtt_client	mqtt_client_task

The following image shows the Edit dialog used to modify a task configuration.



6.2 Code Fix after Generating Code (only for Bluetooth)

Currently, STM32CubeMX cannot generate BT/DMA related code correctly. Most of the code are added as used define code. So, you don't need to care of them.

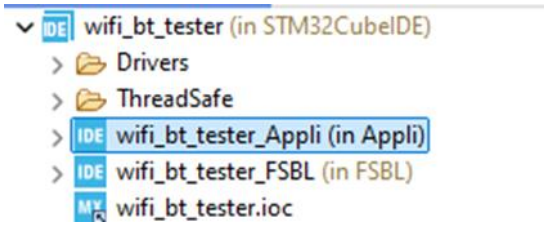
But you need to modify one line manually as follows.

[Appli\Core\Src\stm32n6xx_hal_msp.c]

```
void HAL_UART_MspInit(UART_HandleTypeDef* huart)
{
(...)
 else if(huart->Instance==USART2)
 {
(...)
   /* USART2 interrupt Init */
   HAL_NVIC_SetPriority(USART2_IRQn, 5, 0);   // <====== change priority from 0 to 5
   HAL_NVIC_EnableIRQ(USART2_IRQn);
   /* USER CODE BEGIN USART2_MspInit 1 */
```

6.3 Build and Program STM32N6 sample project

The STM32N6 Project consists of two binary images. You have to build and program both of them.



- **<sample project name>_FSBL**: Boot loader. Stored at 0x7000'0000. It just load Appli from XSPI FLASH to SRAM.

Build and program sample project for STM32N6570-DK board

- **<sample project name>_Appli**: Main body of application. Stored at 0x7010'0000. It's loaded by FSBL.

In addition, STM32N6 is different from other STM32 variants for the following two steps:

- Two images must be signed by the ST tool after compile for trusted/secure boot.
- When programming two images, the external loader is used. Run/Debug configuration is not used for programming image to N6.

In the default N6 sample projects, these two steps are configured as "Post-build steps" in project file. You don't need to take care of them. However, when you change device/modules, additional steps are needed. See the details in Change Device/Module of Sample Project.

The steps to build and program FSBL and Appli are as follows:

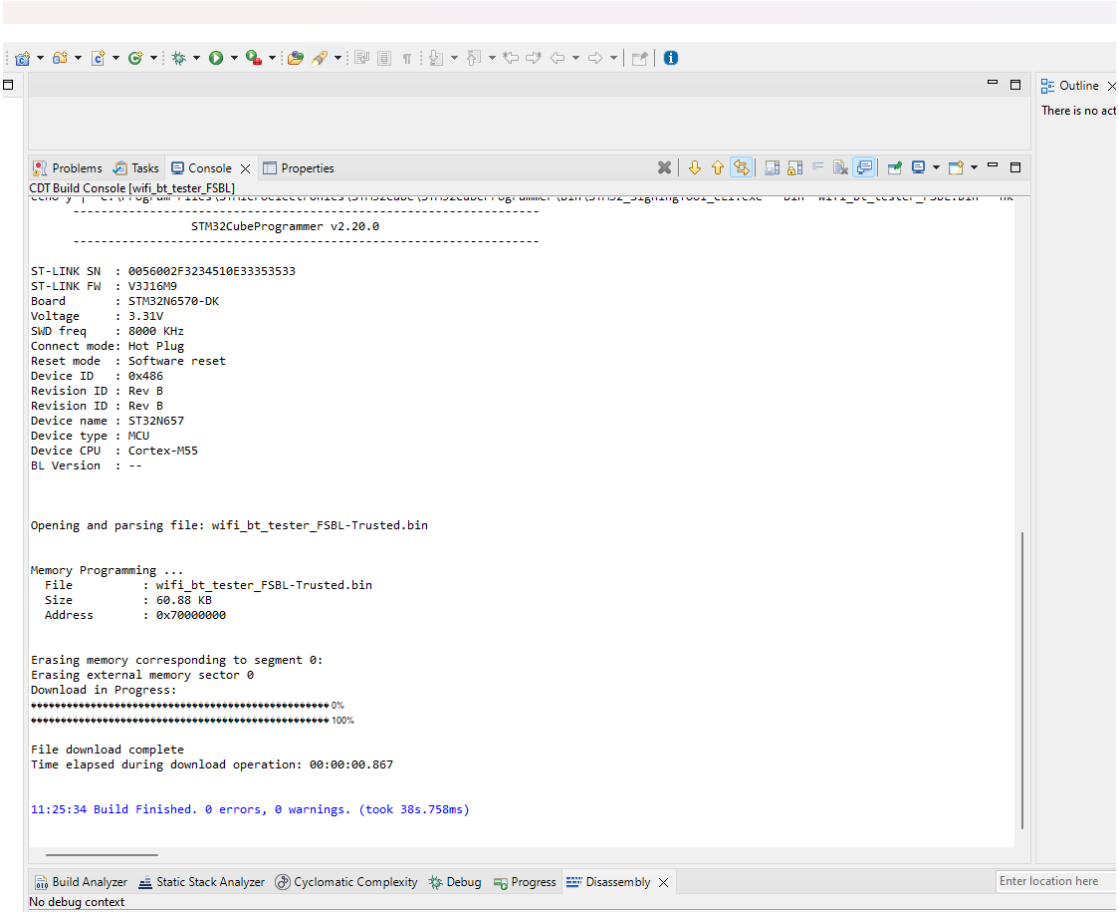
1. Set the STM32N6570-DK board to program mode.

   Set the BOOT1 switch of the N6 board to high and push the reset button.

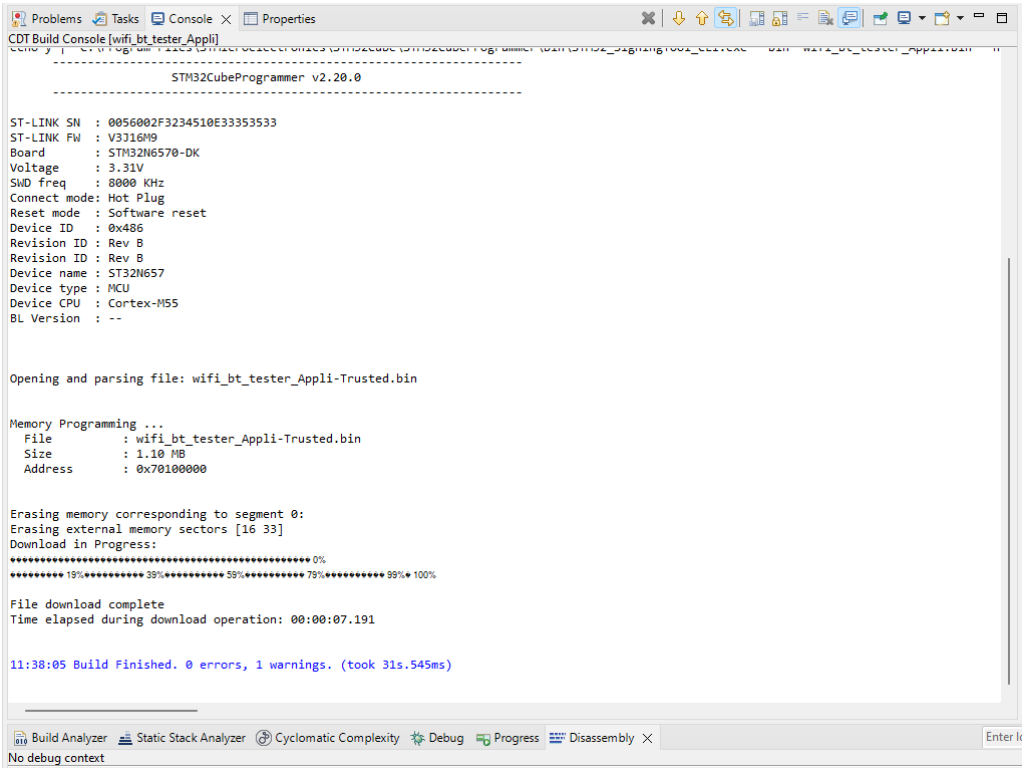
2. Click Build for FSBL project. It builds wifi_bt_tester_FBSL. Then, the post-build steps add the signature to the .bin file and program it using the external loader.

   You will see the following log if programming was successful.



3. Click Build for the Appli project. It builds wifi_bt_tester_Appli. Then, the post-build steps add the signature to .bin file and program it using external loader.

   You will see the following log if programming was successful.



4. Set the STM32N6570-DK board to normal mode.

   Set the BOOT1 switch of the N6 board to low and push the reset button.

6.4 Generate trusted binary for STM32N6

If you add manually a signature to the binary instead of using the “Post-build steps” described in the Build and program STM32N6 sample sroject section, follow this procedure:

Open a command prompt and type the following command:

```
C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_SigningTool_CLI.e
xe -bin <Project Name>.bin -nk -of 0x80000000 -t fsbl -o <Project Name>-
Trusted.bin -hv 2.3 -dump <Project Name>-Trusted.bin
```

<Project Name>.bin is a binary after after being compiled. The two files are located as follows:

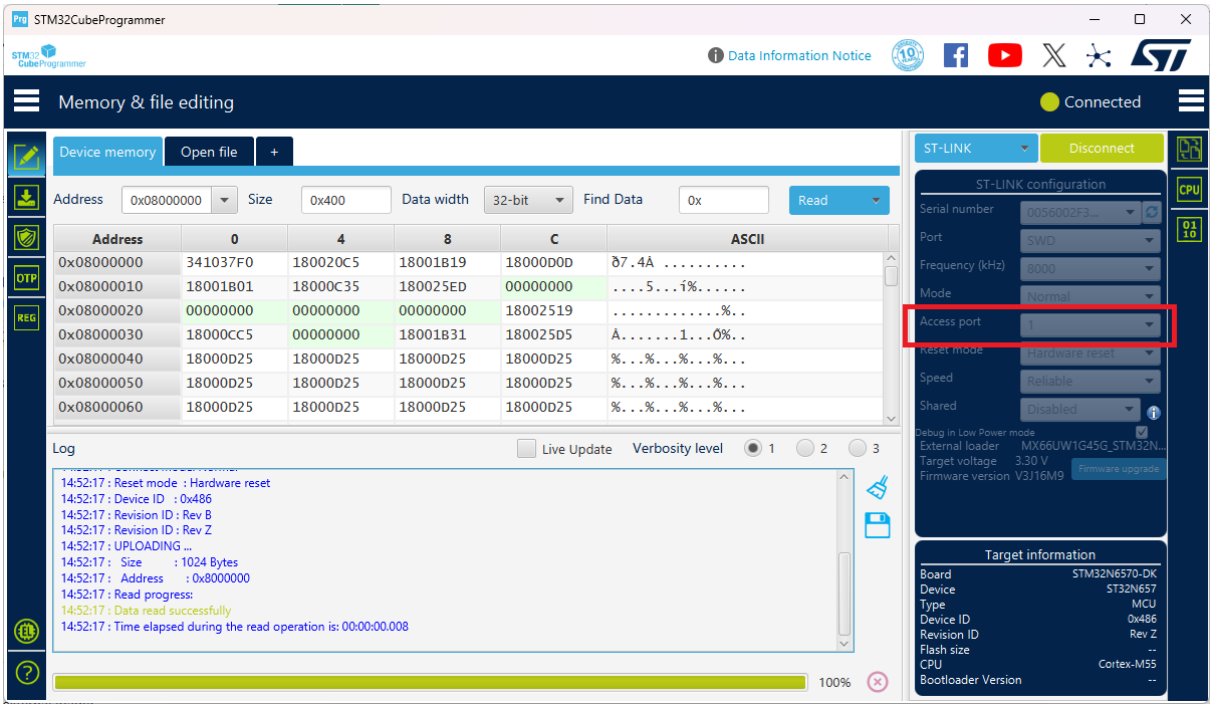
- FSBL: STM32CubeIDE\FSBL\Debug\<Project Name>_FSBL.bin
- Appli: STM32CubeIDE\Appli\Debug\<Project Name>_Appli.bin

<Project Name>-Trusted.bin is a binary after being signed. Generate a signed binary for both FSBL and Appli and program them as described in the Program signed binary using STM32CubeProgrammer section.

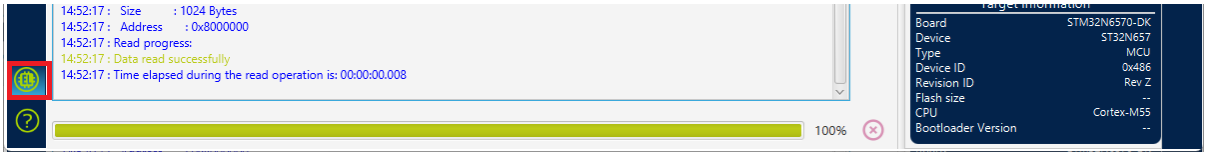
6.5 Program signed binary using STM32CubeProgrammer

If you program a signed binary file manually instead of using “Post-build steps” described in the Build and Program STM32N6 Sample Project section, follow this procedure:

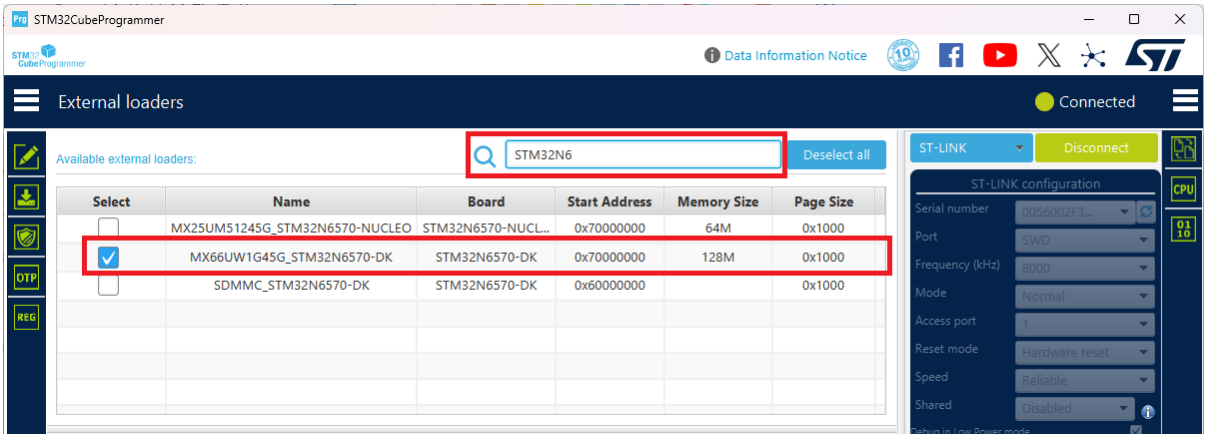
1. Set the BOOT1 switch of the N6 board to high and push the reset button.
2. Run STM32CubeProgrammer.
3. Set **Access Port** to 1 and click the **Connect** button.



4. Select the **External Loaders** tab.

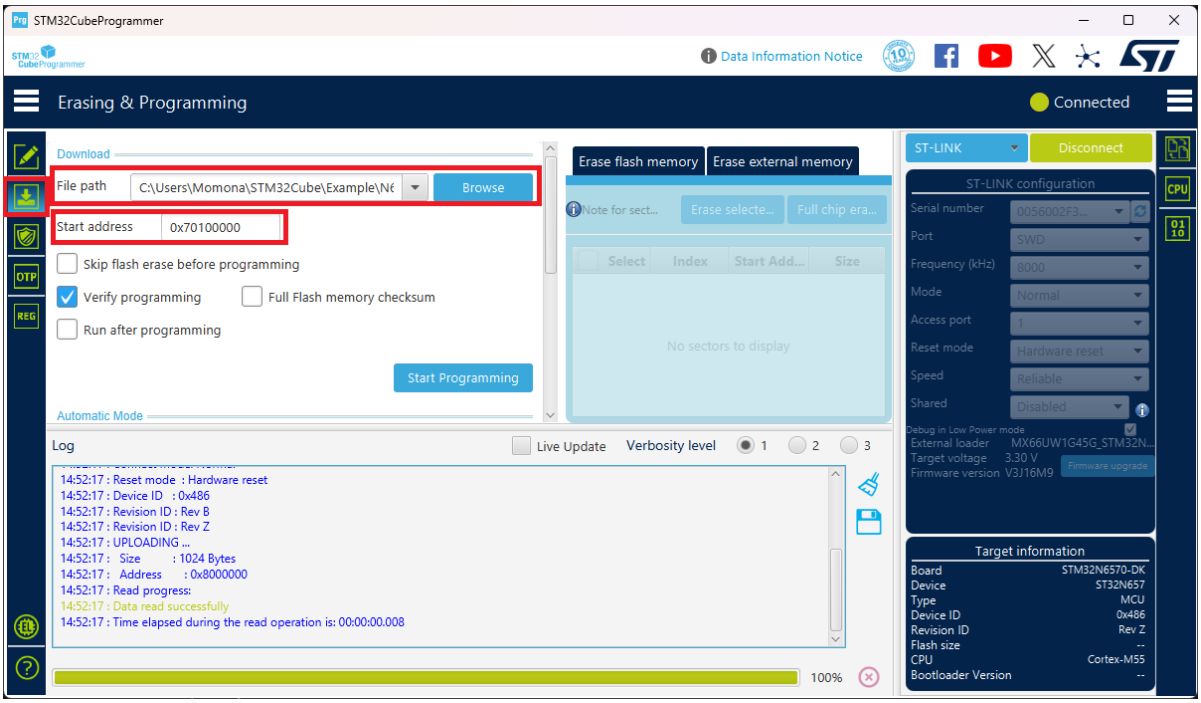


5. Search for STM32N6.
6. Select **MX66UW1G45G_STM32N6570-DK** as the external loader.



Build and program sample project for STM32N6570-DK board

7. Select the **Erasing & Programming** tab.
8. Select the FSBL binary in **File path**.
9. Set the **Start address** to 0x7000'0000.
10. Select **Verify programming**. Deselect other options.
11. Click the **Download** button.
12. Select Appli binary in **File Path**.
13. Set **Start address** to 0x70100'0000
14. Select **Verify programming**. Deselect other options.
15. Click the **Download** button.



16. Click the **Disconnect** button.
17. Set the BOOT1 switch to low and push the reset button on the STM32N6-DK board.

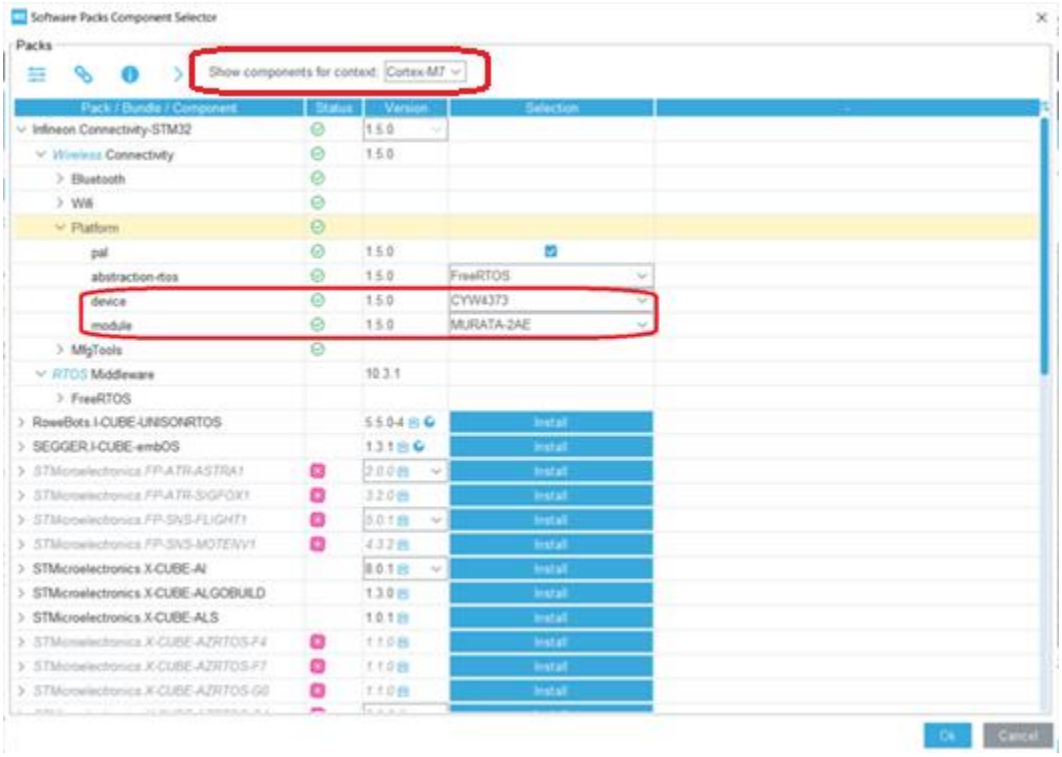
7 Change device/module of sample project

When you build a sample application for a different device/module, follow these steps.

7.1 Before generating code on STM32CubeMX

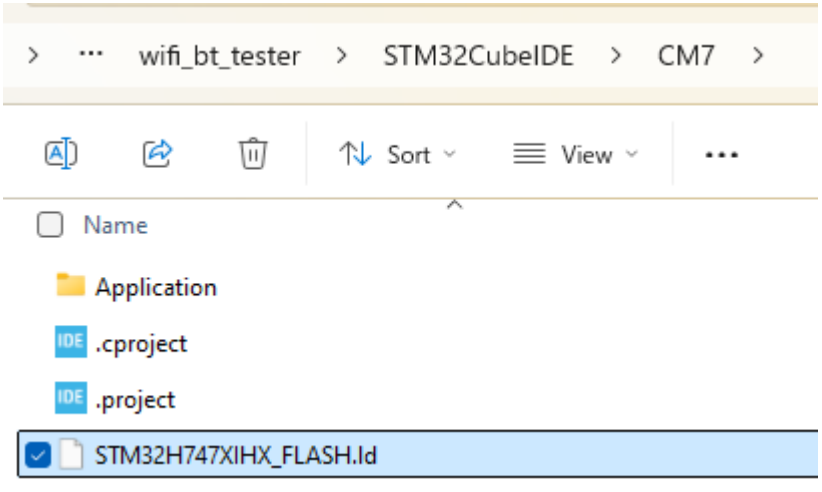
1. Change device/module on the STM32CubeMX as follows.

   - Select **Pinout & Configuration > Software Packs**
   - Select **Show components for context** and then the appropriate option for multi-core: (e.g., Cortext-M7 for STM32H7, Appli for STM32N6)
   - Expand **Infineon.Connectivity-STM32 > Wireless Connectivity > Platform**



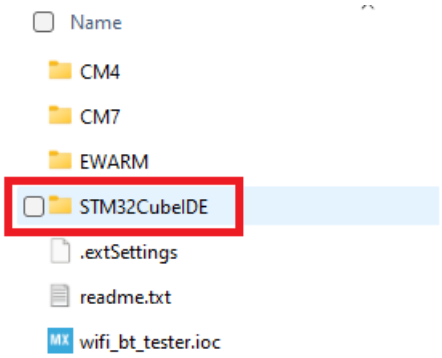
2. Back up the linker script for the multi-core device (STM32H7, Nucleo-H7).

   Go to STM32CubeIDE/CM7/STM32H7. Under the sample project folder is a .FLASH.ld linker script for STM32H7 that has custom settings. Back up this file.

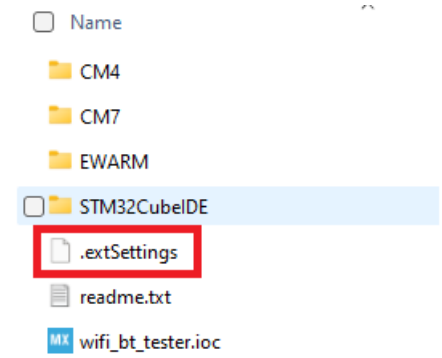


Change device/module of sample project

3. Remove the STM32CubeIDE folder under sample project folder in order to remove previous device/module settings.



4. Modify the build option in the .extSettings file under the sample project folder according to the selected device.



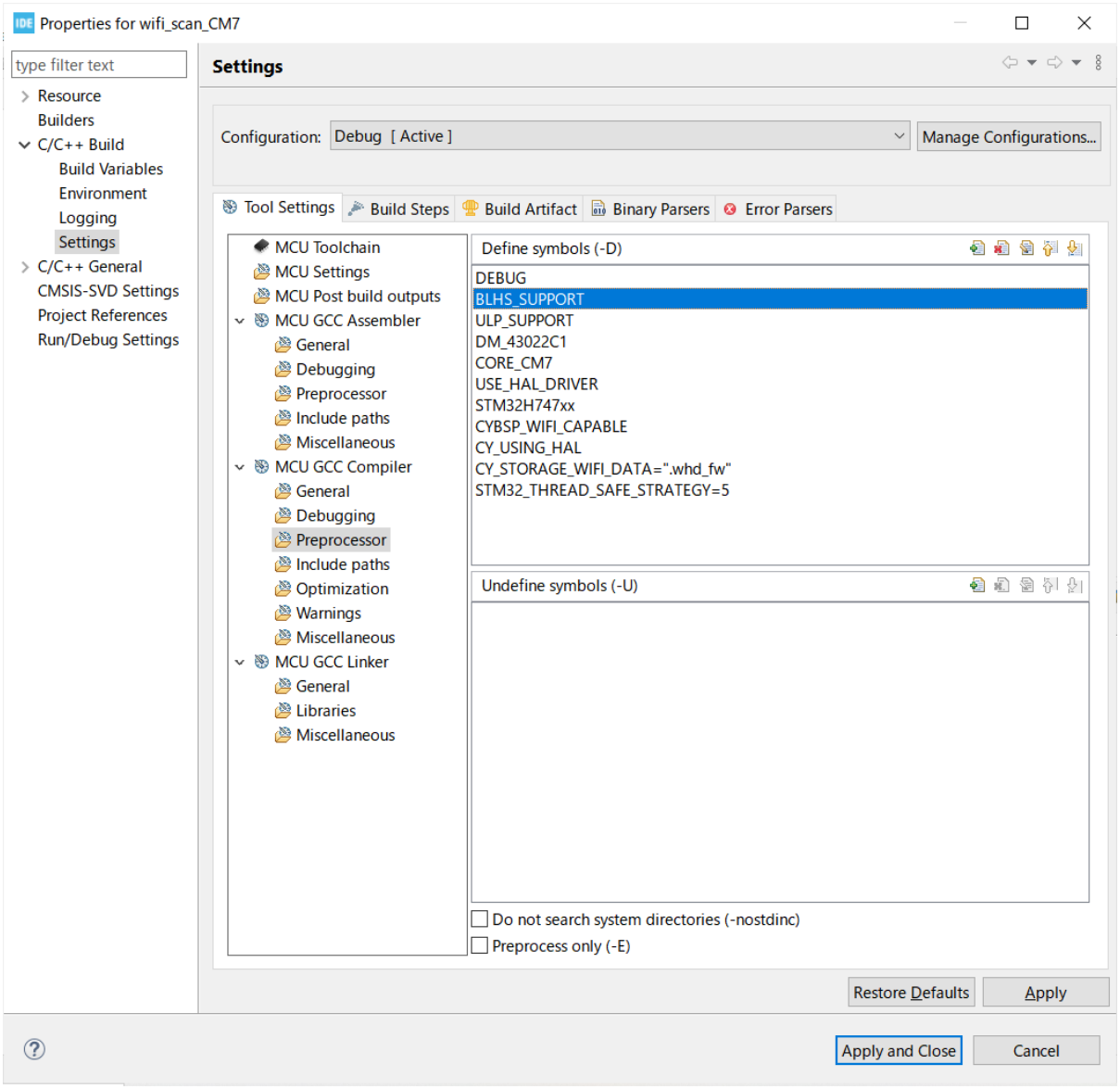
   The .extSettings has a line `Define=` that is used as the build option/macro of the sample application. Edit this line as follows.

   - CYW43012/43439/4373: Must **NOT** include `BLHS_SUPPORT` and `DM_43022C1`. If they exist, remove both.
   - CYW555x1/5557x: Must include `BLHS_SUPPORT`. Must **NOT** include `DM_43022C1`.
   - CYW43022: Must include both `BLHS_SUPPORT` and `DM_43022C1`.

Change device/module of sample project

Note: *You can confirm these settings in the STM32CubeIDE after generating code by STM32CubeMX.*

*In STM32CubeIDE, select **Project > Properties > C/C++ Build > Settings > MCU/GCC Compiler > Preprocessor**.*

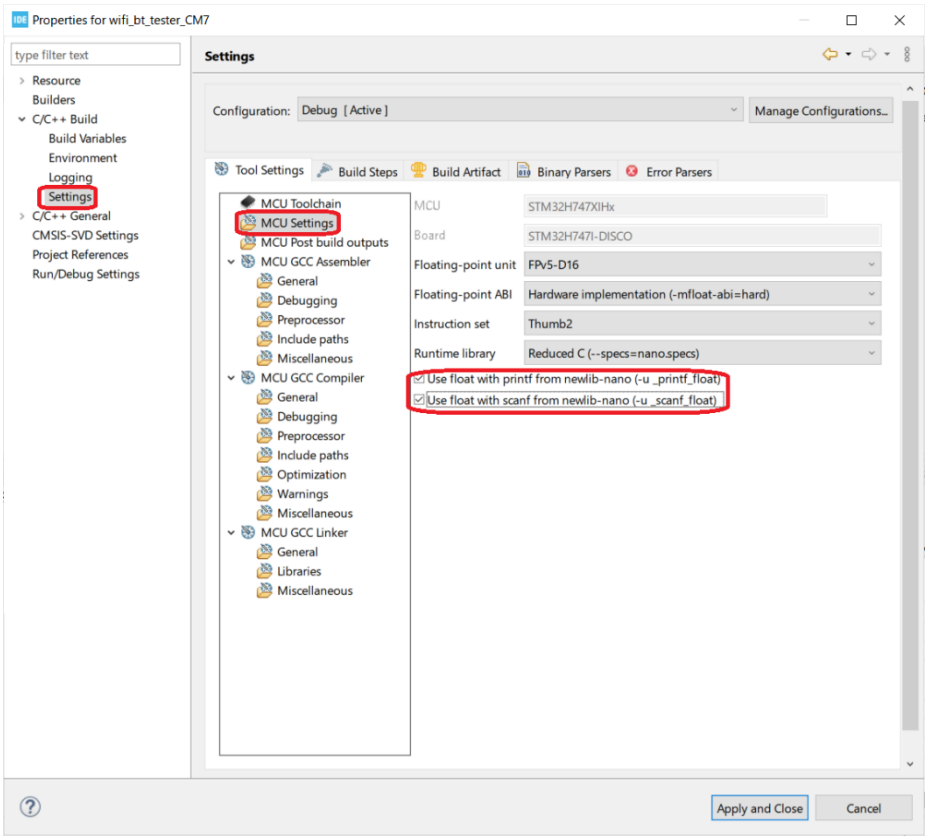


5. Click the **Generate Code** button.
6. Restore the linker script that was backed up in step 2 to the newly generated STM32CubeIDE folder.

7.2 Restore project settings in STM32CubeIDE

If the sample project uses floating point variables with printf/scanf (e.g., wifi_bt_tester), enable floating point with printf/scanf library as follows:

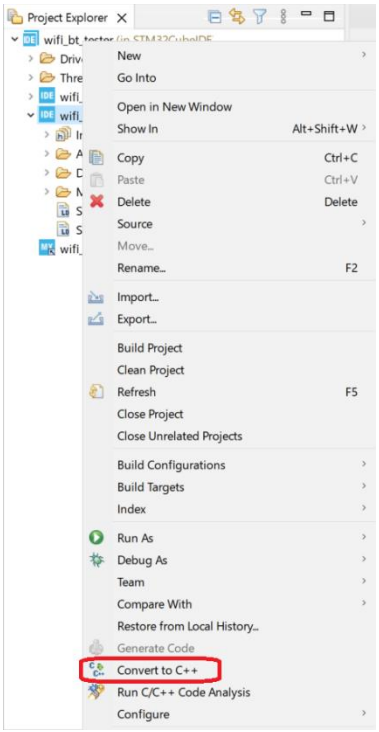
1. Select **Project > Properties > Settings > MCU Settings > Use float with printf/scanf**.



Change device/module of sample project

2. If the application includes C++ source code (e.g., wifi_bt_tester), convert the project from C to C++.

   Right click on the project and select **Convert to C++**.



3. (This is STM32N6 only) Restore the post-build steps.

   Add post-build command to the wifi_bt_tester_FSBL project.

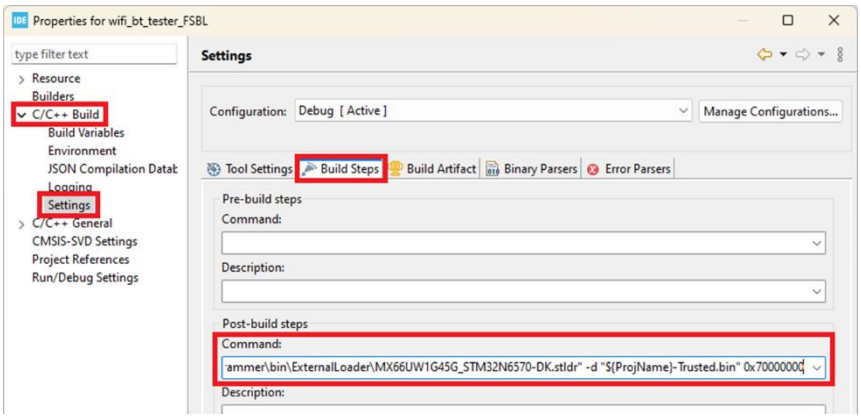
   a. Right click on the **<sample project> _FSBL** project and select **Properties**.

   b. On the dialog, select **C/C++ build -> Settings -> Build Steps -> Post-Build Steps -> Commands**.

   c. Add the following command:

```
cd "${ProjDirPath}/Debug" && echo y | "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_SigningTool_CL
I.exe" -bin "${ProjName}.bin" -nk -of 0x80000000 -t fsbl -o "${ProjName}-
Trusted.bin" -hv 2.3 -dump "${ProjName}-Trusted.bin" | "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI
.exe" -c port=SWD mode=HOTPLUG AP=1 -el "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader\MX66U
W1G45G_STM32N6570-DK.stldr" -d "${ProjName}-Trusted.bin" 0x70000000
```

   d. Click **Apply and Close**.



Change device/module of sample project

Add post-build command to wifi_bt_tester_Appli project.

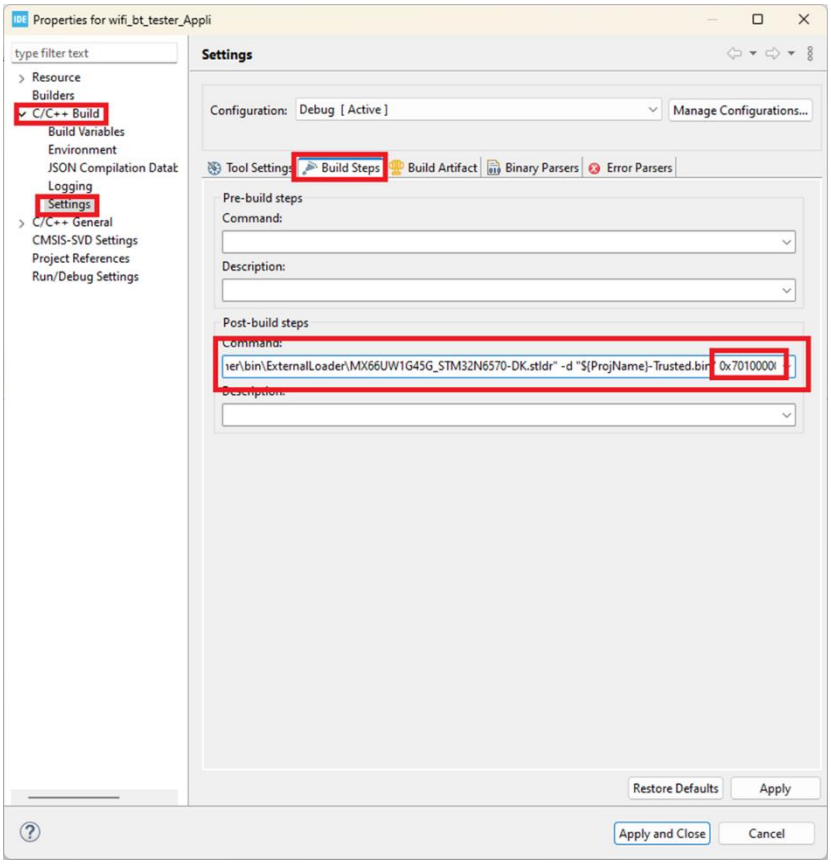
a. Right click on the **<sample project> _Appli** project and select **Properties**.

b. On the dialog, select **C/C++ build -> Settings -> Build Steps -> Post-Build Steps -> Commands**.

c. Add the following command:

```
cd "${ProjDirPath}/Debug" && echo y | "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_SigningTool_CL
I.exe" -bin "${ProjName}.bin" -nk -of 0x80000000 -t fsbl -o "${ProjName}-
Trusted.bin" -hv 2.3 -dump "${ProjName}-Trusted.bin" | "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\STM32_Programmer_CLI
.exe" -c port=SWD mode=HOTPLUG AP=1 -el "C:\Program
Files\STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader\MX66U
W1G45G_STM32N6570-DK.stldr" -d "${ProjName}-Trusted.bin" 0x70100000
```

d. Click **Apply and Close**.



8 Special options and setup

8.1 STM32H7xx – using serial flash

There may be a need for extra internal Flash space when running applications on STM32H7xx. A significant amount of internal Flash can be saved if the Wi-Fi stack is placed on an external Serial Flash memory module. The STM32H747I-DISCO board has MT25QL512ABB8ESF-0SIT memory IC present for this purpose.

- STM32H747I-DISCO has serial Flash in dual-bank Quad-SPI mode
- STM32H7 has QSPI HW block

Additional settings are needed to enable placing the Wi-Fi stack firmware on external memory:

1. Linker script (*.ld) has external memory address defined:

```
QSPI    (rx)    : ORIGIN = 0x90000000,       LENGTH = 131072K
```

2. Linker script has section name defined where Wi-Fi stack will be located during linkage:

```
.whd_fw :
{
__whd_fw_start = .;
KEEP(*(.whd_fw))
__whd_fw_end = .;
} > OSPI
```

3. Preprocessor macro name added:

```
CY_STORAGE_WIFI_DATA=".whd_fw"

BSP-files have to be added:
BSP\stm32h747i_discovery_qspi.c
BSP\stm32h747i_discovery_qspi.h
BSP\Components\mt25tl01g\mt25tl01g.c(*.h)
BSP\Components\mt25tl01g\mt25tl01g.c(*.h)
BSP\Components\mt25tl01g\mt25tl01g_conf.h
```

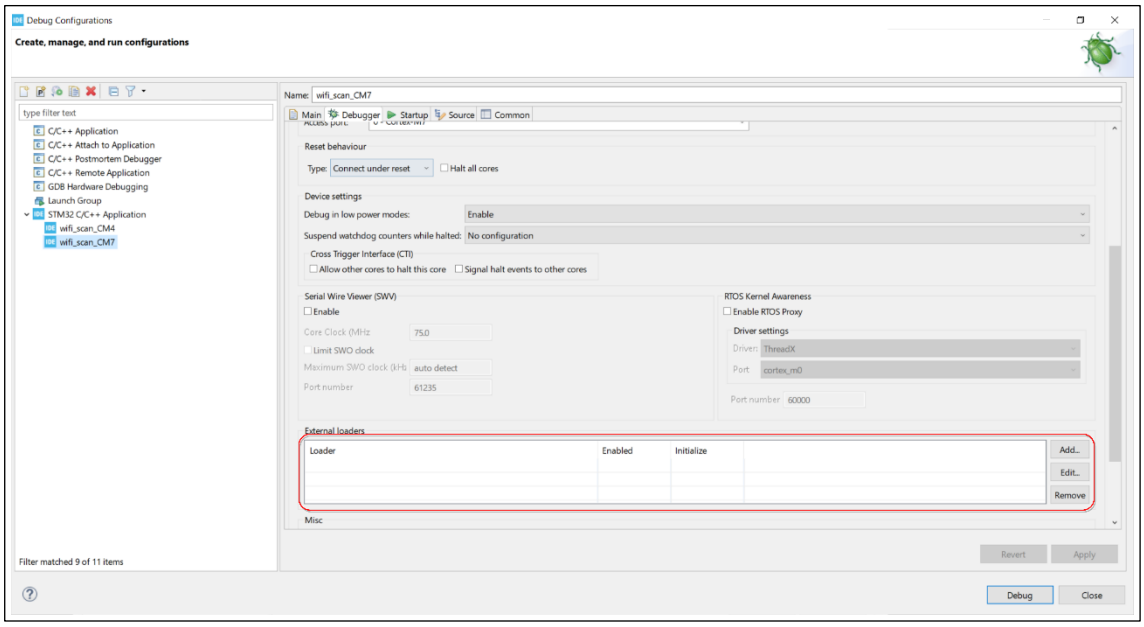
4. BSP Initialization routine call have to be added:

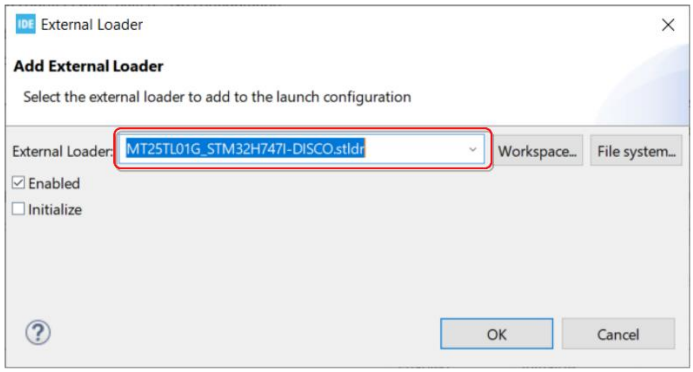
```
/* Configure External Memory to Memory Mapped Mode*/
/* QSPI info structure */
BSP_QSPI_Info_t pQSPI_Info;
uint8_t status;
/*##-1- Configure the QSPI device ###########################################*/
/* QSPI device configuration */
BSP_QSPI_Init_t init ;
init.InterfaceMode=MT25TL01G_QPI_MODE;
init.TransferRate= MT25TL01G_DTR_TRANSFER ;
init.DualFlashMode= MT25TL01G_DUALFLASH_ENABLE;
status = BSP_QSPI_Init(0,&init);
if (status != BSP_ERROR_NONE)
{
     printf("\r\n    ERROR: BSP_QSPI_Init() failed \r\n");
     Error_Handler();
}
/*##-2- Read & check the QSPI info ##########################################*/
/* Initialize the structure */
pQSPI_Info.FlashSize          = (uint32_t)0x00;
pQSPI_Info.EraseSectorSize    = (uint32_t)0x00;
pQSPI_Info.EraseSectorsNumber = (uint32_t)0x00;
pQSPI_Info.ProgPageSize       = (uint32_t)0x00;
```

Special options and setup

```
pQSPI_Info.ProgPagesNumber    = (uint32_t)0x00;
/* Read the QSPI memory info */
BSP_QSPI_GetInfo(0,&pQSPI_Info);
/*##-6-Memory Mapped Mode ##################################################*/
status = BSP_QSPI_EnableMemoryMappedMode(0);
if (status != BSP_ERROR_NONE)
{
     printf("\r\n    ERROR: BSP_QSPI_EnableMemoryMappedMode() failed \r\n");
     Error_Handler();
}
```

5. Programming of the Serial Flash should be performed with appropriate Flash Loader selection:





Note: External flash (MT25QL512ABB8ESF) requires 3.3 V for normal operation.

8.2 STM32H7xx – using internal flash (BANK2) to store Wi-Fi FW

The internal flash space on STM32H7xx is divided into two banks: BANK1 (1M) is used for CM7, BANK2 (1M) is used for CM4. The steps to reuse part of BANK2 to store Wi-Fi firmware images:

1. Update the linker script (*.ld):

   a. Add WIFI_FLASH memory definition to the MEMORY section of the linker script:

```
WIFI_FLASH (rx) : ORIGIN = 0x08140000, LENGTH = 768K /* ORIGIN address of
 BANK2 (0x08100000) with 768K offset */
```

b. Define the whd_fw section where the Wi-Fi FW will be located during linkage:

```
.whd_fw :
{
__whd_fw_start = .;
KEEP(*(.whd_fw))
__whd_fw_end = .;
} >WIFI_FLASH
```

2. Add the reprocessor macro name:

```
CY_STORAGE_WIFI_DATA=".whd_fw"
```

8.3 STM32L562 – using serial flash

The Wi-Fi application can't fit STM32L5x internal flash due to size constraints. MCU has 512kB of area when connectivity firmware reaches over 1MB.

To resolve this external memory module is used, present on STM32L562E-DK board.

The project has the following additional settings made to enable placing WiFi stack firmware on external memory:

1. Linker script (*.ld) has external memory address defined:

```
OSPI(rx)    : ORIGIN = 0x90000000,       LENGTH = 65536K
```

2. Linker script has section name defined where WiFi stack will be located during linkage:

```
.whd_fw :
{
__whd_fw_start = .;
KEEP(*(.whd_fw))
__whd_fw_end = .;
} > OSPI
```

3. Preprocessor macro name added:

```
CY_STORAGE_WIFI_DATA=".whd_fw"
```

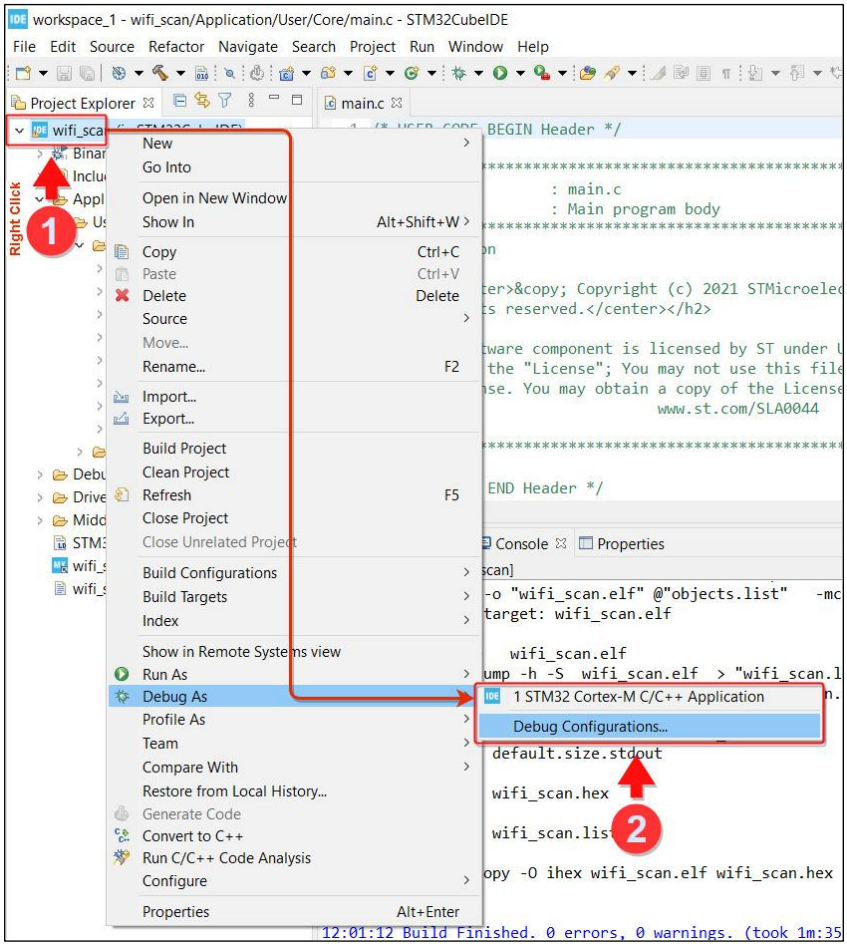
With given setup, the compiler and linker will split a resulting image into two pieces, which will reside in both – internal and external memory of an STM32L562E-DK.

8.4 STM32L562 – serial flash programming

8.4.1 Using STM32CubeMX IDE

To program resulting image into the target device an appropriate Flash loader has to be selected:

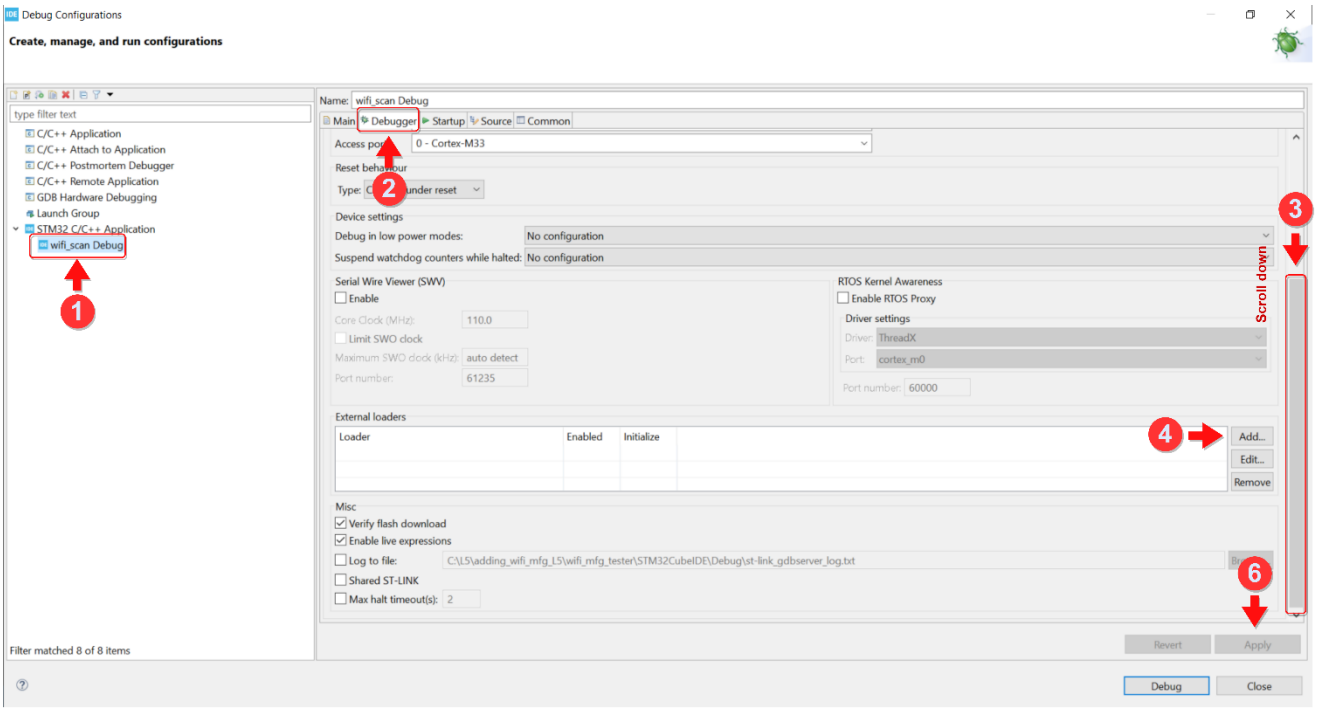
Right-click on a project, select **Debug As > Debug Configurations...**



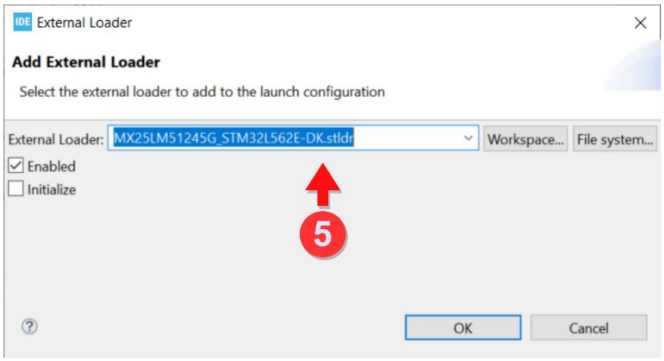
Special options and setup

Select the external loader (see steps illustrated in the following image).

- Select "wifi_scan Debug" and select the **Debugger** tab.
- Scroll down and find the **External Loader**.
- Click **Add** button to add external loader.



- Enter the appropriate loader in the External loader dialog:
  MX25LM51245G_STM32L562E-DK.stldr



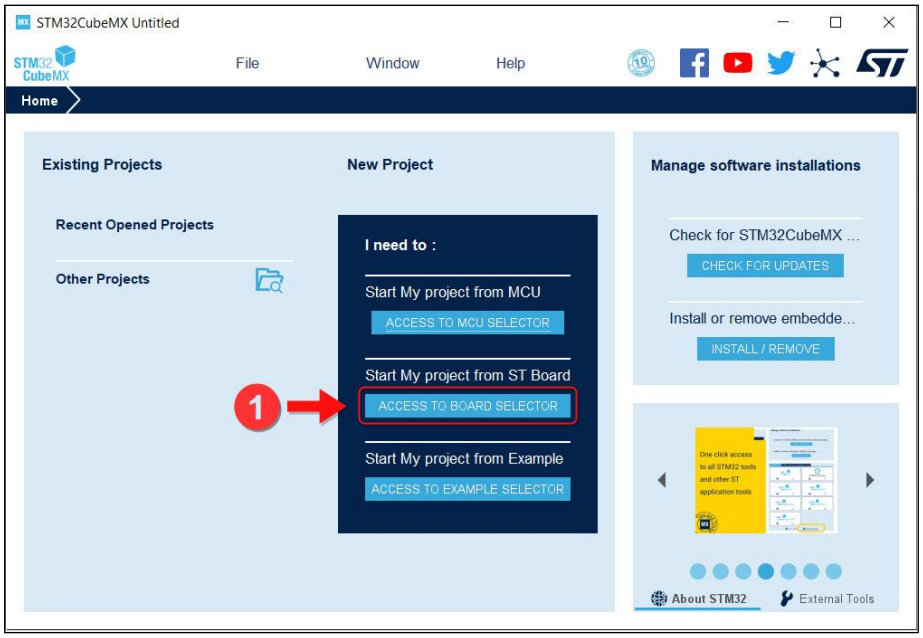
Program your target with "Run" or "Debug" command.

9 Create a new project from scratch

This section takes you step-by-step through the process of creating a project file from scratch.

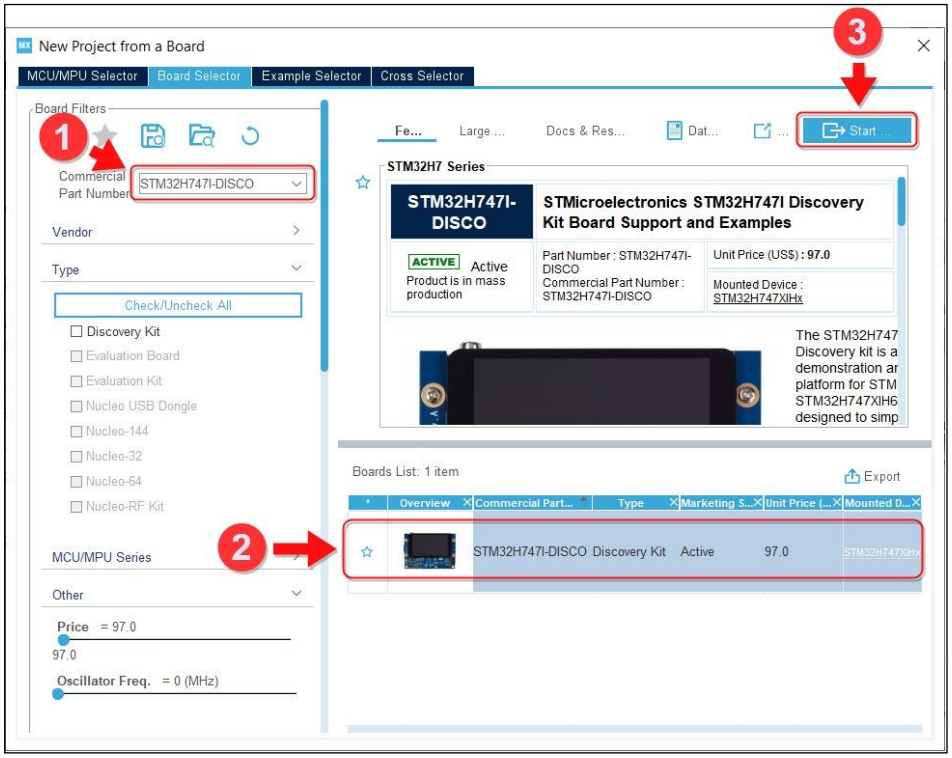
9.1 Create a project for specific board

1. Start creating a project via the **ACCESS TO BOARD SELECTOR** option.



2. Select a board.

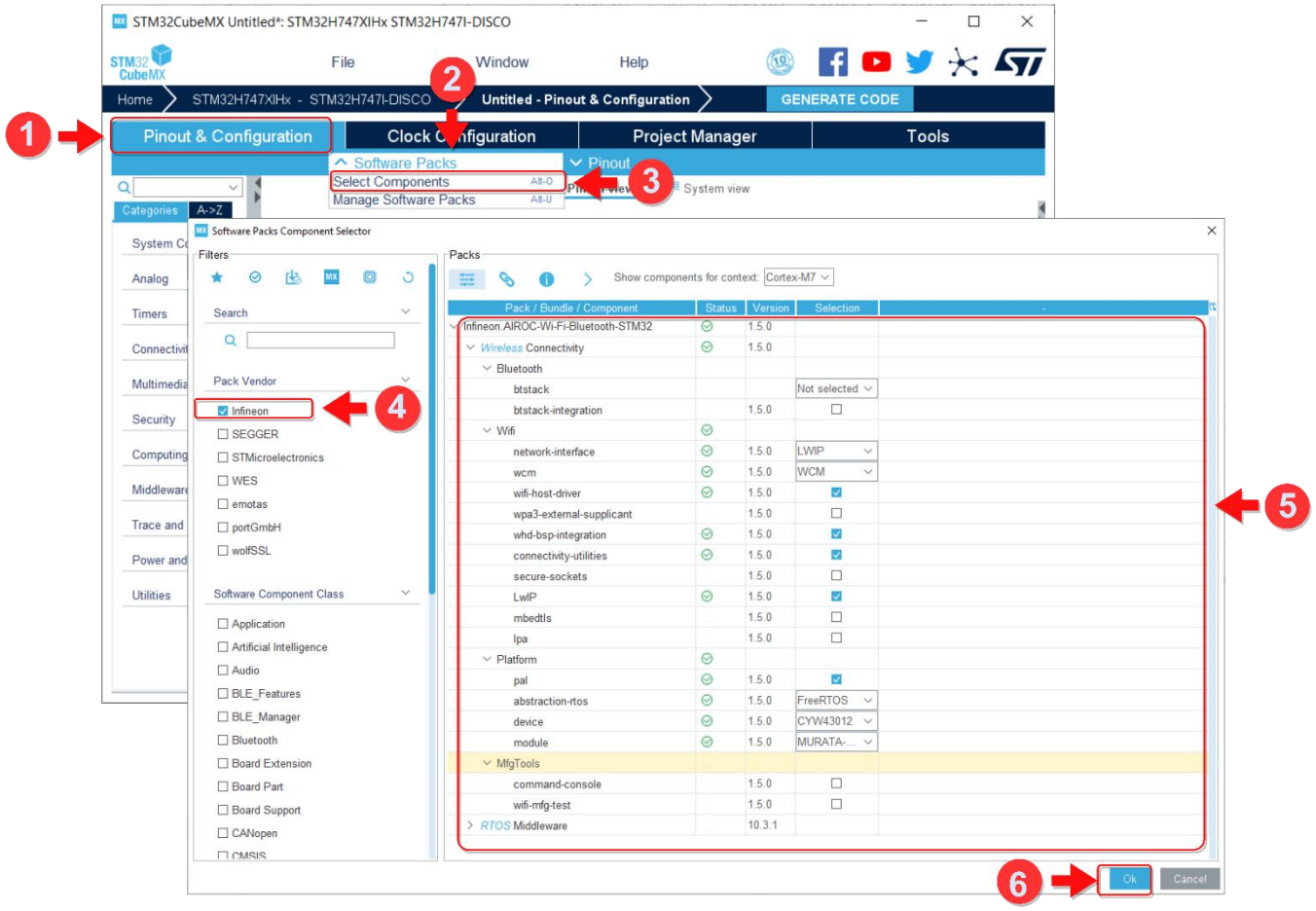
   - Enter/select the board number.
   - Click on your selected board.
   - Select Start Project.



9.2 Enable software components from AIROC™ Wi-Fi/Bluetooth® STM32 expansion pack

1. Select the **Pinout & Configuration** tab.
2. Select **Software Packs**.
3. Select **Select Components**

   This will open the Software Packs Component Selector dialog with a list of the installed packs and their contents.



4. Select Infineon under **Pack Vendor**.
5. Select the components you need for your project.

   All projects will use three ‘Platform’ components. If you are using Wi-Fi, select all the ‘Wifi’ components. If you are using Bluetooth® LE, select all ‘Bluetooth’ components.

Note: *Platform components are required for each type of application – either Wi-Fi-only, Bluetooth-only or combined.*

   a. For the ‘Platform / device’ component, select the appropriate connectivity device for your system (CYW43012, CYW4343W or CYW43438, etc.).

   b. For the ‘Wifi / network-interface’ component, select the appropriate network interface for your system (LwIP or NetxDuo).

Create a new project from scratch

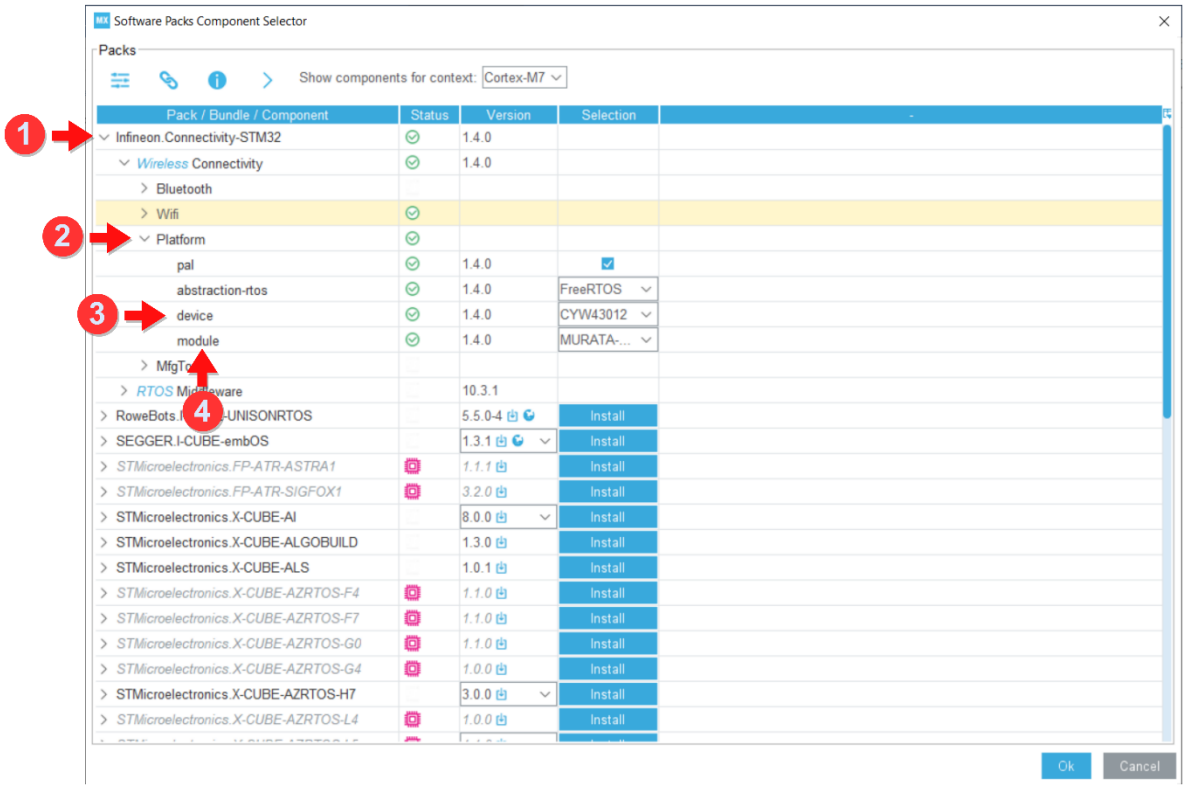
c. For the 'Wifi / wcm' component, select the appropriate variant (WCM or WCM /WPS/MBEDTLS).

- WCM Variant compiles only Wi-Fi connection manager files, which provide a set of APIs that can be used to establish and monitor Wi-Fi connections on Infineon platforms that support Wi-Fi connectivity.
- WCM /WPS/MBEDTLS Variant also includes APIs to connect to a Wi-Fi network using Wi-Fi Protected Setup (WPS) methods which uses MBED TLS security stack.

6. Click **OK**.

9.2.1 Module selection

The AIROC™ Wi-Fi/Bluetooth® STM32 expansion pack has a software component named module configuration (in the "Platform" section), which is used to select different modules for the Connectivity device. Also, you can select the USER_MODULE variant for custom configuration. To do this, provide own your NVRAM header, Firmware, CLM somewhere in your project (for example, in the "Core/Inc" folder).



Module	Device	Description
MURATA-1LV	CYW43012	Type 1LV is a small and high-performance module based on Infineon CYW43012 combo chipset which supports Wi-Fi® 802.11a/b/g/n + Bluetooth® 5.0 BR/EDR/LE up to 72.2Mbps PHY data rate on Wi-fi® and 3Mbps PHY data rate on Bluetooth®. 2Mbps LE PHY is also supported. The WLAN section supports SDIO v2.0 SDR25 interface and the Bluetooth® section supports high-speed 4-wire UART interface and PCM for audio data.
MURATA-1YN	CYW43439	Type 1YN is a small and high-performance module based on Infineon CYW43439 combo chipset which supports Wi-Fi® 802.11b/g/n + Bluetooth® 5.2 BR/EDR/LE up to 65Mbps PHY data rate on Wi-fi® and 3Mbps PHY data rate on Bluetooth®. The WLAN section supports SDIO v2.0 interface and the Bluetooth® section supports high-speed 4-wire UART interface and PCM for audio data.

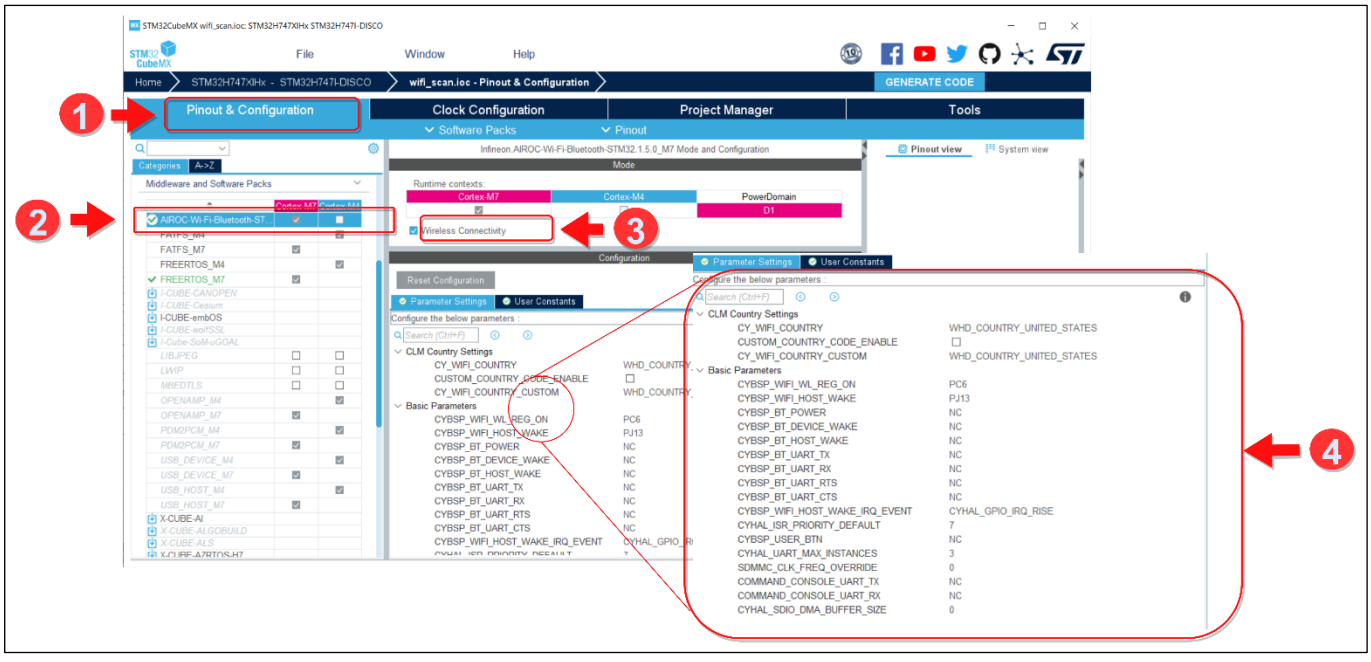
Create a new project from scratch

Module	Device	Description
MURATA-1DX	CYW4343W	Type 1DX is a small and high-performance module based on Infineon CYW4343W combo chipset which supports Wi-Fi® 802.11b/g/n + Bluetooth® 5.1 BR/EDR/LE up to 65Mbps PHY data rate on Wi-fi® and 3Mbps PHY data rate on Bluetooth®. The WLAN section supports SDIO v2.0 interface and the Bluetooth® section supports high-speed 4-wire UART interface and PCM for audio data.
QUECTEL-FC909A	CYW43439	FC909A is a high-performance Wi-Fi 4 and Bluetooth® LE (5.2) module in an LCC package. It can be used to establish WLAN and Bluetooth® connections. With an ultra-compact size of 12.0mm × 12.0mm × 1.95mm, FC909A optimizes the size and cost for end-products.
CYW9P62S2-M2BASE-4373	CYW4373	Infineon's AIROC™ CYW4373/CYW4373E/CYW43732 single-chip combo device features 1x1 dual-band 2.4 GHz and 5 GHz Wi-Fi 5 (802.11ac) and Bluetooth® 5.2.
STERLING-LWB5plus	CYW4373	Sterling LWB5+ development board.
MURATA-2AE	CYW4373	Type 2AE is a small and very high-performance module based on the Infineon CYW4373E combo chipset which supports Wi-Fi 802.11a/b/g/n/ac + Bluetooth® 5.2 BR/EDR/LE up to 433Mbps PHY data rate on Wi-Fi and 3Mbps PHY data rate on Bluetooth®.
MURATA-2EA	CYW55573	Type 2EA is a small and very high-performance module based on Infineon CYW55573 combo chipset which supports Wi-Fi® 802.11a/b/g/n/ac/ax 2×2 MIMO Bluetooth® 5.3 BR/EDR/LE up to 1.2Gbps PHY data rate on Wi-Fi® and 3Mbps PHY data rate on Legacy Bluetooth® (EDR), and 2Mbps PHY data rate on Bluetooth® LE. The WLAN section supports PCIe v3.0 Gen 2 and SDIO 3.0 interface and the Bluetooth® section supports high-speed 4-wire UART interface and PCM for audio data. The CYW55573 implements highly sophisticated enhanced collaborative coexistence hardware mechanisms and algorithms, which ensure that WLAN and Bluetooth® collaboration is optimized for maximum performance.
MURATA-2BC	CYW4373	Type 2BC is a small and very high-performance module based on the Infineon CYW4373 combo chipset which supports Wi-Fi® 802.11a/b/g/n/ac + Bluetooth® 5.2 BR/EDR/LE up to 433Mbps PHY data rate on Wi-Fi® and 3Mbps PHY data rate on Bluetooth®. The WLAN section supports SDIO v3.0 DDR50 interface and the Bluetooth® section supports high-speed 4-wire UART interface and PCM for audio data. Both WLAN and Bluetooth® section support USB2.0 interface too.
CYW43022	CYW43022	Infineon's AIROC™ CYW43022 an ultra-low power single-chip, combo device features 1x1 dual-band 2.4 GHz and 5 GHz Wi-Fi 5 (802.11ac) and Bluetooth® 5.4. With a low-power architecture, the CYW43012 is ideal for battery powered applications where best-in-class power consumption is critical. An embedded Bluetooth stack and Wi-Fi networking offloads allow the CYW43022 to maintain connectivity activity even while a host processor is in low-power sleep mode.
CYW955513WLPA	CYW55500	The CYW55500 is a low-power, single-chip device that supports single-stream, tri-band, Wi-Fi 6/6E, IEEE 802.11ax-compliant Wi-Fi MAC/baseband/radio and Bluetooth®/Bluetooth® Low Energy 5.3. In 802.11ax mode, the device supports rates up to 1024 QAM MCS11 in 20 MHz channels. All legacy rates in the 802.11a/b/g/n/ac are also supported. Included on-chip are 2.4 GHz, 5-7 GHz transmit power amplifiers (PA) and low-noise amplifiers (LNA). The device is also capable of operating with external power amplifiers and low-noise amplifiers, and antenna diversity, if improved range is required. An SDIO v3.0 interface or GSPI are available for interfacing with the host.

Module	Device	Description
CYW955572FCIPA-SM	CYW55572	The AIROC™ CYW55572 is part of the Wi-Fi 6 and Bluetooth® 5.3 SoC family. The highly integrated solution supports Wi-Fi 6 features, is Dual-band capable (2.4G, 5G). It offers an exceptional video/audio streaming and seamless gaming experience in congested network environments and significantly reduces latency, while also decreasing total Bill of Materials cost and board space.
CYW955573M2IPA1-SM	CYW55573	The AIROC™ CYW55573 is part of Infineon's Wi-Fi 6/6E and Bluetooth® 5.3 SoC family. The solution supports Wi-Fi 6/6E features, is tri-band capable (2.4G, 5G, 6G). Its features improve range, power efficiency, network robustness, and security, while reducing the total Bill of Materials cost and board space. The solution delivers an exceptional high-quality video/audio streaming and seamless connectivity experience in congested network environments and significantly reduces latency by operating in the 6G spectrum.
USER-MODULE	ALL	Custom USER-MODULE configuration, in this case User should provide own NVRAM header somewhere in project (e.g. in Core/Inc folder).

9.3 Enable Software pack

1. Select the **Pinout & Configuration** tab.
2. Expand **Middleware and Software Packs** and select **AIROC-Wi-Fi-Bluetooth-STM32**
3. Click the checkbox next to **Wireless Connectivity**.
4. Configure different Settings including these (this will generate cybsp.h):
   - Wi-Fi Country Code
   - WL_REG_On Pin
   - BT_REG_On Pin
   - SDMMC Clock Override
   - BT UART TX, RX, RTS, CTS



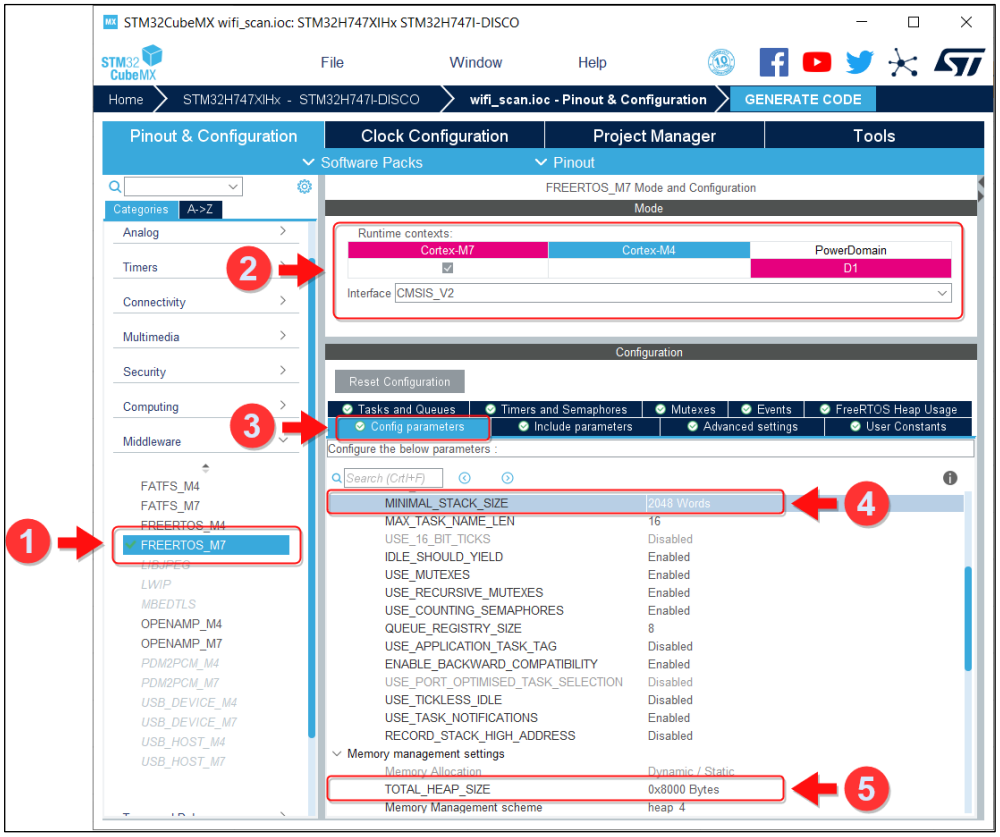
9.4 Country Code Configuration

WLAN Devices have support for different Country Code based on the CLM file, which is specific to Module vendors.

Some CLM files do not have support for all countries. So, you need to configure Country Code based on the WLAN TX/RX regulations and CLM blob, which is loaded.

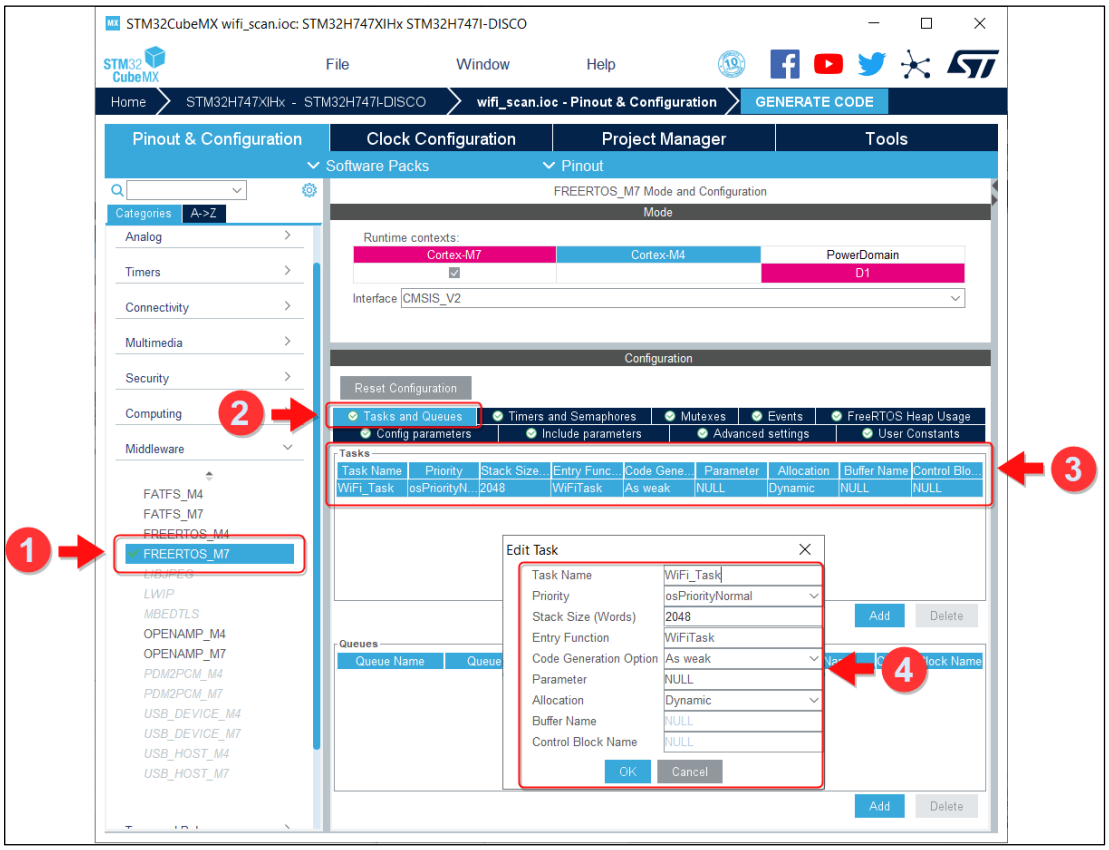
9.5 FreeRTOS configuration

Select FreeRTOS version and configure Stack Size and Heap size as required for the application.



Create a new project from scratch

Under **Tasks and Queues**, configure Default task and its stack size.



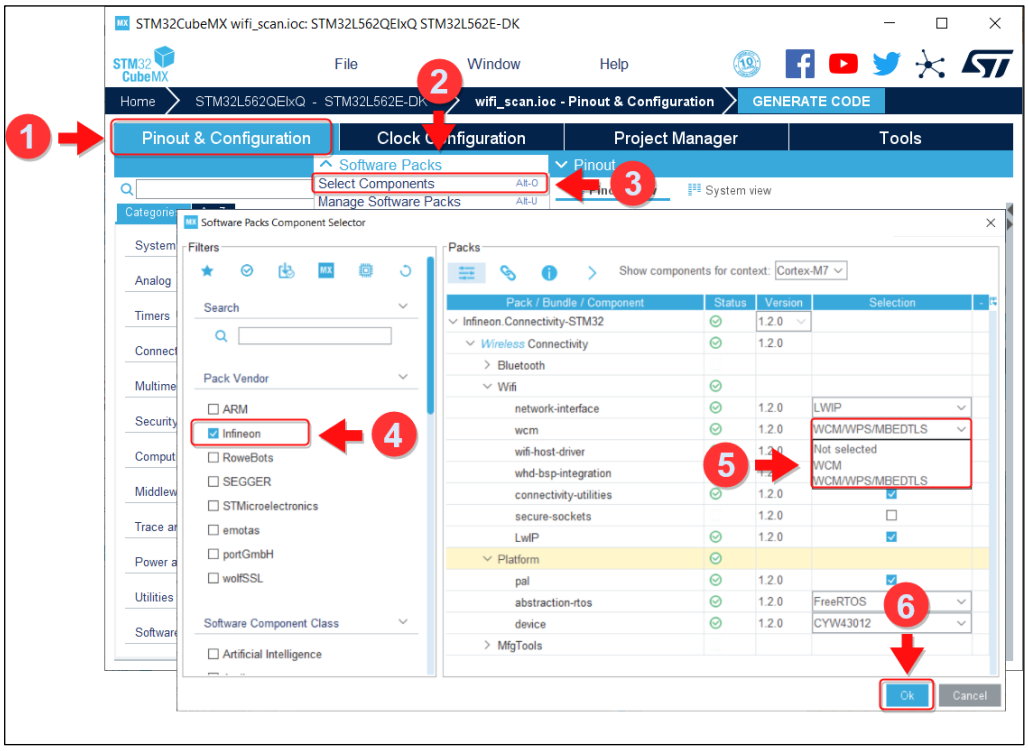
9.6 MbedTLS configuration

The mbedTLS is required by LwIP (Lightweight IP) WCM (Wi-Fi Connection Manager) Pack's components. To enable mbedTLS. The STM32L5 MCU is used as an example to demonstrate Crypto features (AES, HASH, etc.) HW acceleration configuration:

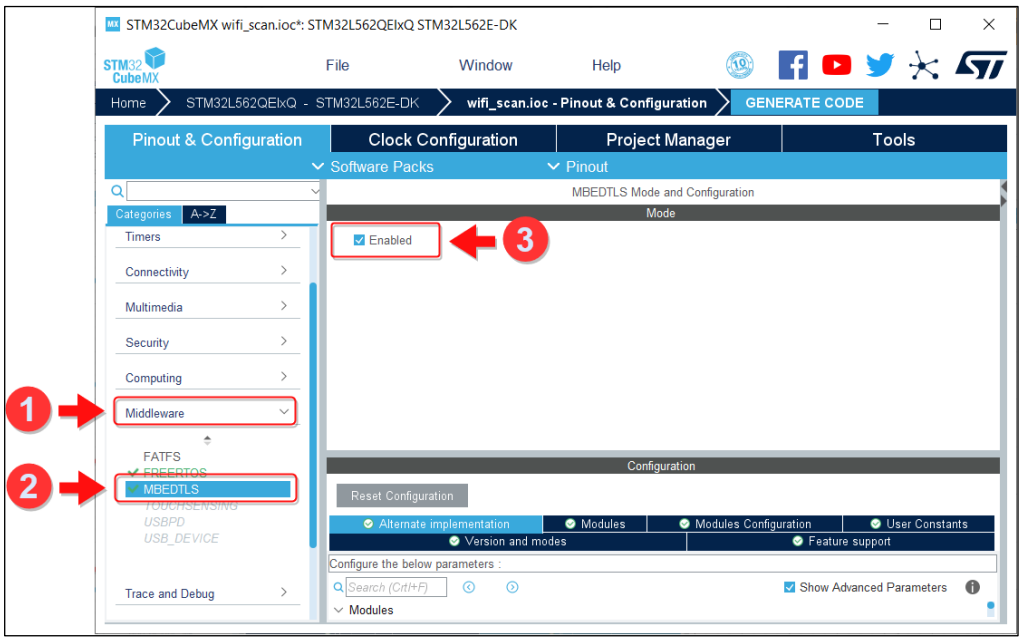
Open the project's *.IOC file w/ STM32CubeMx.

Navigate to Infineon Pack's components and switch WCM to WCM/WPS/mbedTLS.

Create a new project from scratch



Navigate to **Select Components**, select **Middleware** and then select **MBEDTLS** for target device and select the **Enabled** check box.



Ensure that the following features and modes are enabled by performing appropriate steps:

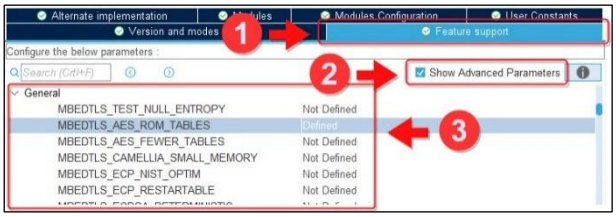
- mbedTLS sources are added to application
- mbedTLS config is applied to support Infineon's connectivity middleware

```
MBEDTLS_ENTROPY_HARDWARE_ALT
MBEDTLS_AES_ROM_TABLES
MBEDTLS_CIPHER_MODE_CBC
MBEDTLS_NO_PLATFORM_ENTROPY
MBEDTLS_ENTROPY_FORCE_SHA256
```

Create a new project from scratch

```
MBEDTLS_AES_C
MBEDTLS_SHA256_C
```

Note: *Set "Not defined" for unneeded modes to reduce memory consumption and eliminate unused code.*



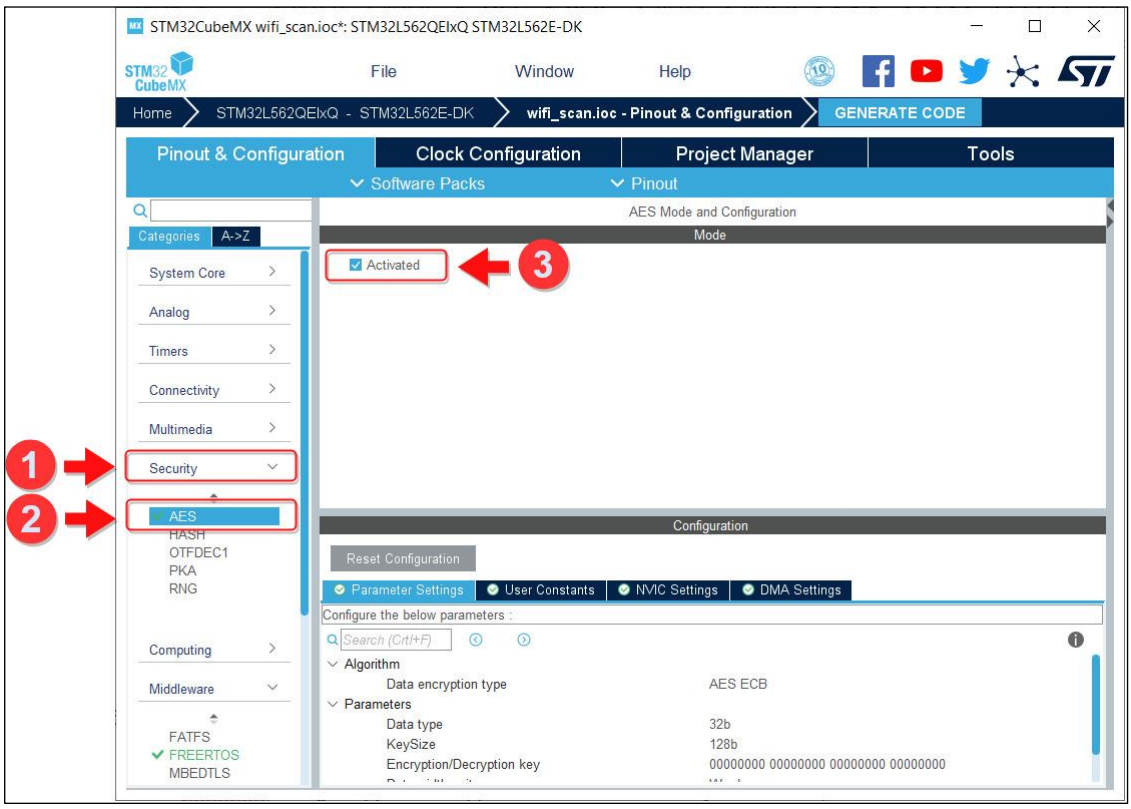
9.6.1 Crypto HW acceleration

STM32 offers HW acceleration for the following crypto-related functions:

	STM32H7	STM32L5	Notes
RNG	+	+	
AES		+	AES-128/256 (ECB, CBC, CTR, GCM GMAC, CCM)
HASH		+	SHA1, SHA224, SHA256, MD5 HMAC SHA1, HMAC SHA224, HMAC SHA256, HMAC MD5
PKA		+	Public Key Cryptography
OTFDEC1		+	On-the-fly decryption of Octo-SPI external memories (AES-128)

Create a new project from scratch

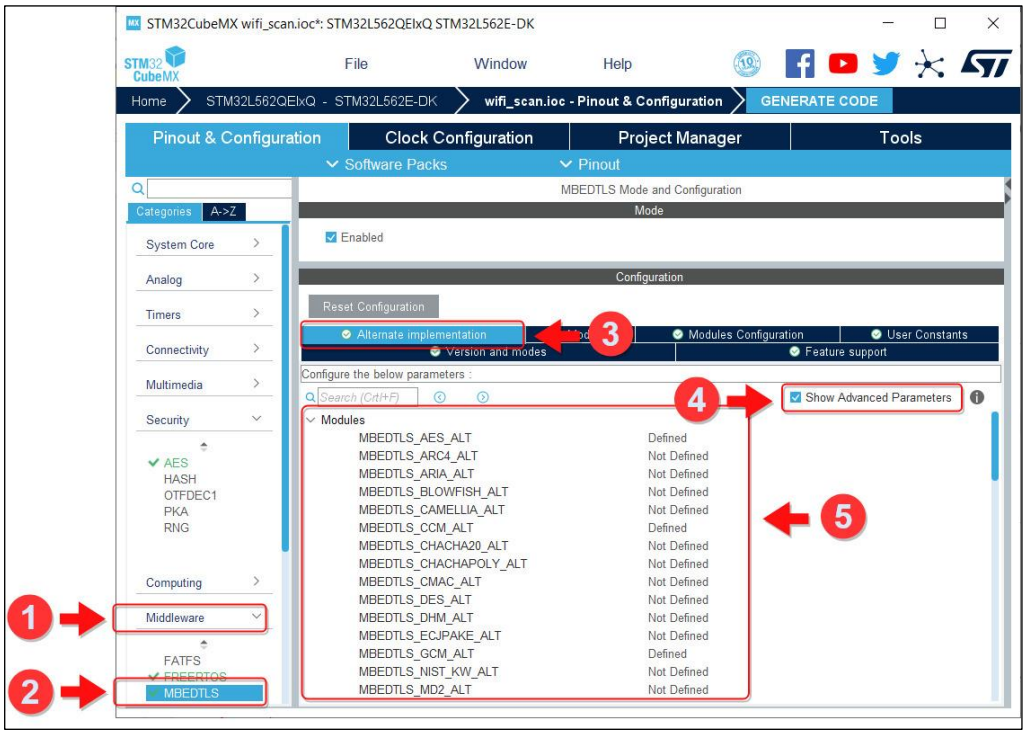
The IP modules listed above must be enabled (Activated) from the "Security" section of STM32CubeMX configurator.



To enable HW acceleration the following literals have to be defined for mbedTLS (should be done in STM32CUbeMX configurator):

```
MBEDTLS_AES_ALT
MBEDTLS_CCM_ALT
MBEDTLS_GCM_ALT
MBEDTLS_MD5_ALT
MBEDTLS_SHA1_ALT
MBEDTLS_SHA256_ALT
MBEDTLS_ENTROPY_HARDWARE_ALT
```

Create a new project from scratch

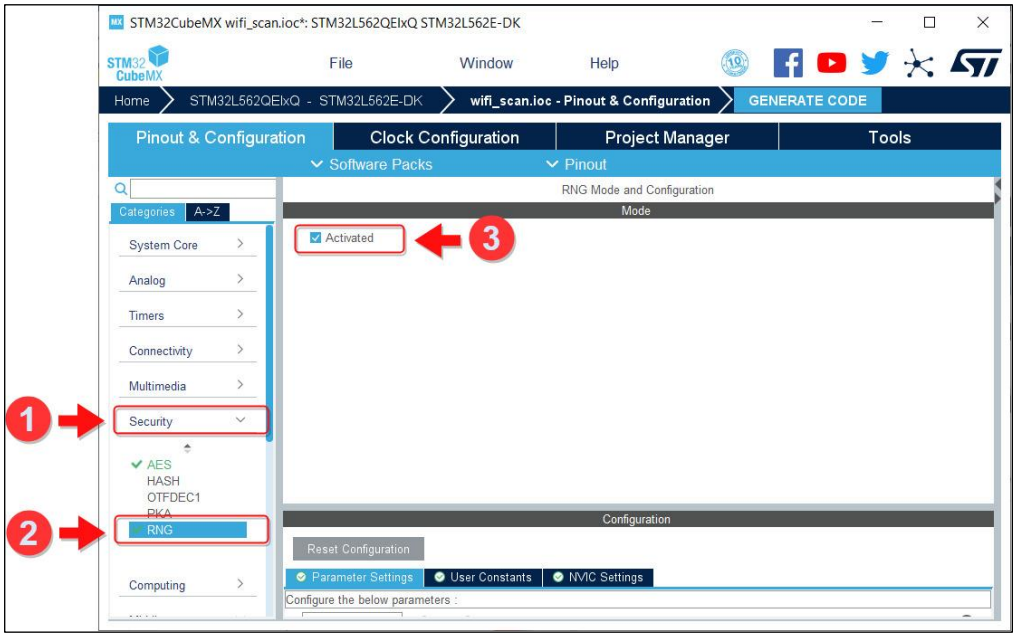


After these steps, source files marked with *_alt suffixes (meaning "alternative", not the original mbedTLS version) will be added into the user's project. They will provide an interface between the mbedTLS crypto functions and its HAL HW counterpart.

9.6.2 HW source of entropy example

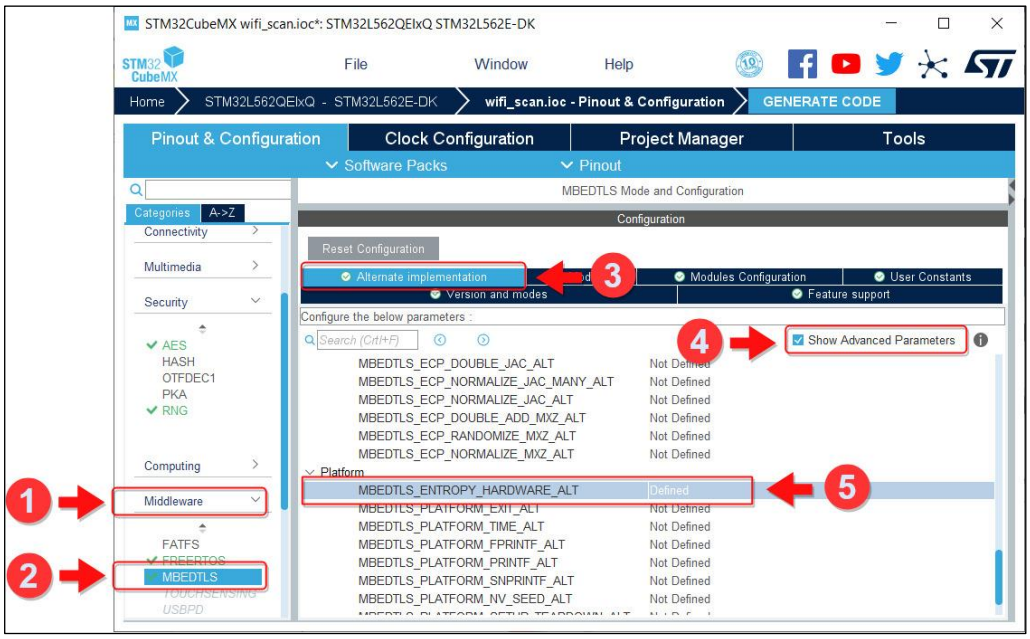
To obtain a good source of entropy used for a public/private key generation and other cryptographic functions:

Enable the RNG module in the STM32CubeMX configurator.



Create a new project from scratch

Set MBEDTLS_ENTROPY_HARDWARE_ALT to "Defined" in the STM32CubeMX configurator:



The tool will add hardware_rng.c source file to the user's project. This will provide the mbedtls_hardware_poll() implementation, which relies on the devices' HW RNG IP block.

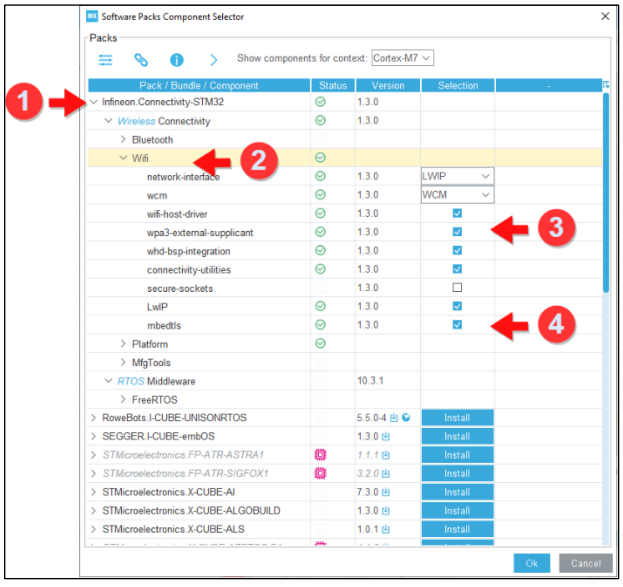
A call to the standard STM32 HAL RNG API (HAL_RNG_GenerateRandomNumber()) will be used by the system to fulfill the mbedTLS entropy pool.

9.7 Wpa3-external-suplicant

Library wpa3-external-suplicant supports WPA3 SAE authentication using HnP (Hunting and Pecking Method) using RFC https://datatracker.ietf.org/doc/html/rfc7664 and H2E (Hash to Element Method) using RFC https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-hash-to-curve-10 and following 802.11 spec 2016.

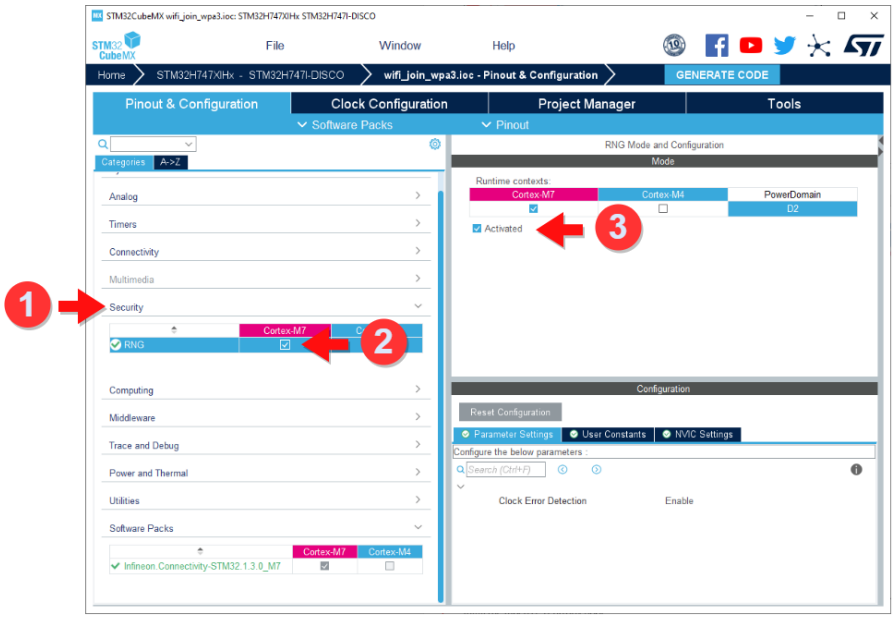
This library required mbedTLS version 2.25.0. To enable wpa3-external-suplicant supports:

Navigate to Infineon Pack's components, then **Wifi** and enable **wpa3-external-supplicant** and **mbedTLS**.



Create a new project from scratch

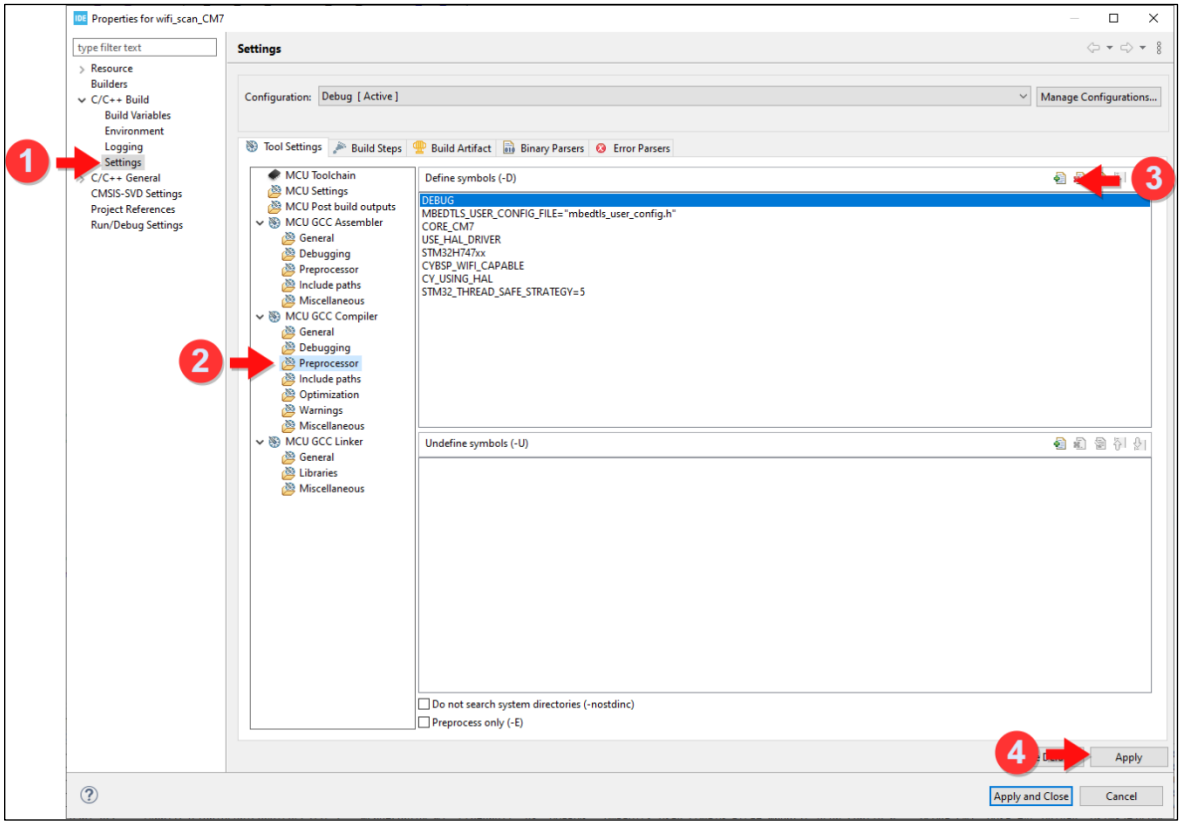
Navigate to **Select Components**, select **Security** and then select **RNG** for target device and select the **Enabled** check box, and enable **Activated** check box.



After generating your code copy **mbedtls_user_config.h** folder from Infineon pack to your project directory (e.g. CORE/Inc folder):

C:\Users\<USER>\STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Middlewares\Third_Party\configs\ mbedtls_user_config.h

Add Preprocessor macro name: *MBEDTLS_USER_CONFIG_FILE="mbedtls_user_config.h"*



Create a new project from scratch

Add implementation for Mbedtls entropy, as shown in the following example for *STM32H7* RNG:

```
#include "mbedtls_user_config.h"

#ifdef MBEDTLS_ENTROPY_HARDWARE_ALT

#include "main.h"
#include "string.h"
#include "stm32h7xx_hal.h"
#include "mbedtls/entropy_poll.h"

extern RNG_HandleTypeDef hrng;

int mbedtls_hardware_poll( void *Data, unsigned char *Output, size_t Len, size_t *oLen )
{
  uint32_t index;
  uint32_t randomValue;

  for (index = 0; index < Len/4; index++)
  {
    if (HAL_RNG_GenerateRandomNumber(&hrng, &randomValue) == HAL_OK)
    {
      *oLen += 4;
      memset(&(Output[index * 4]), (int)randomValue, 4);
    }
    else
    {
      Error_Handler();
    }
  }

  return 0;
}
#endif /*MBEDTLS_ENTROPY_HARDWARE_ALT*/
```

For more information, refer to the code example wifi_join_wpa3.

9.8 Configure resources for Wi-Fi connectivity

The following Peripherals and I/O lines are required for the host MCU to communicate to Infineon connectivity device(s) for Wi-Fi:

9.8.1 SDIO

SDIO is used as an interface with Infineon Connectivity devices. The SDMMC HAL component is required for STM32 host MCU to access/control Infineon connectivity device(s).

1. Add the API call at initialization with appropriate handle passed in:

```
SD_HandleTypeDef SDHandle = { .Instance = SDMMC1 };
cy_rslt_t result = stm32_cypal_wifi_sdio_init(&SDHandle);
```

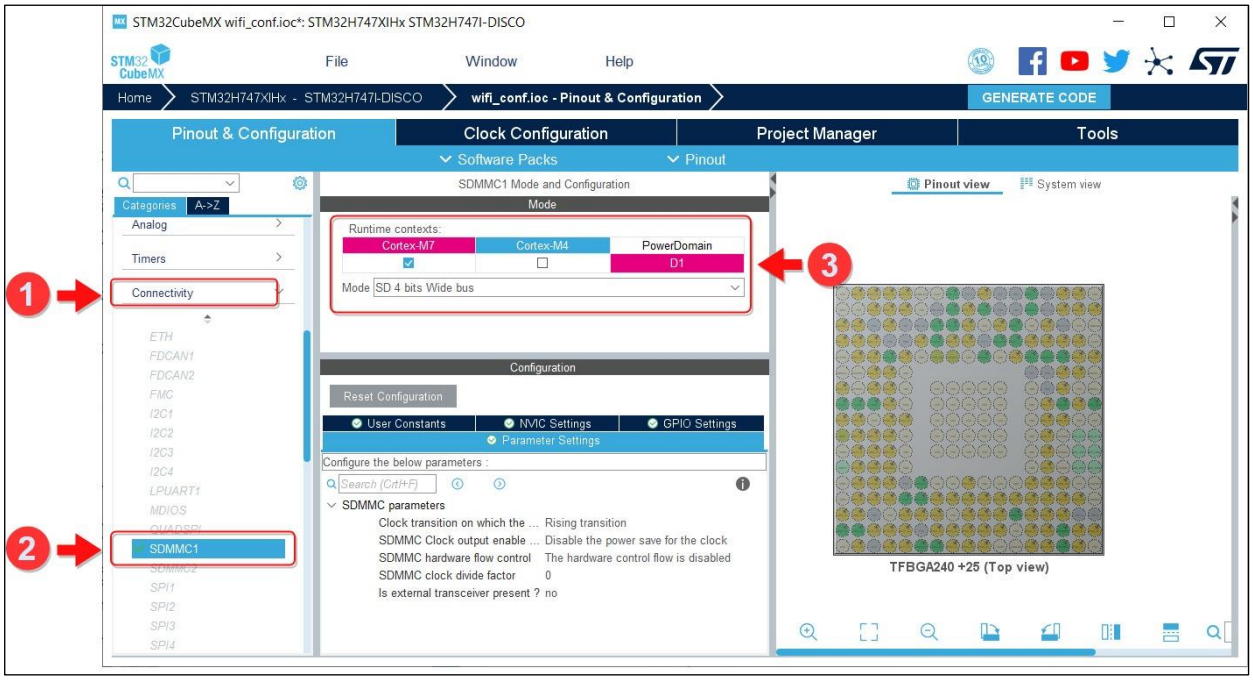
2. SDMMC Interrupt handler must be overwriting in application and call stm32_cyhal_sdio_irq_handler function:

```
void SDMMC1_IRQHandler(void)
{
    stm32_cyhal_sdio_irq_handler();
}
```

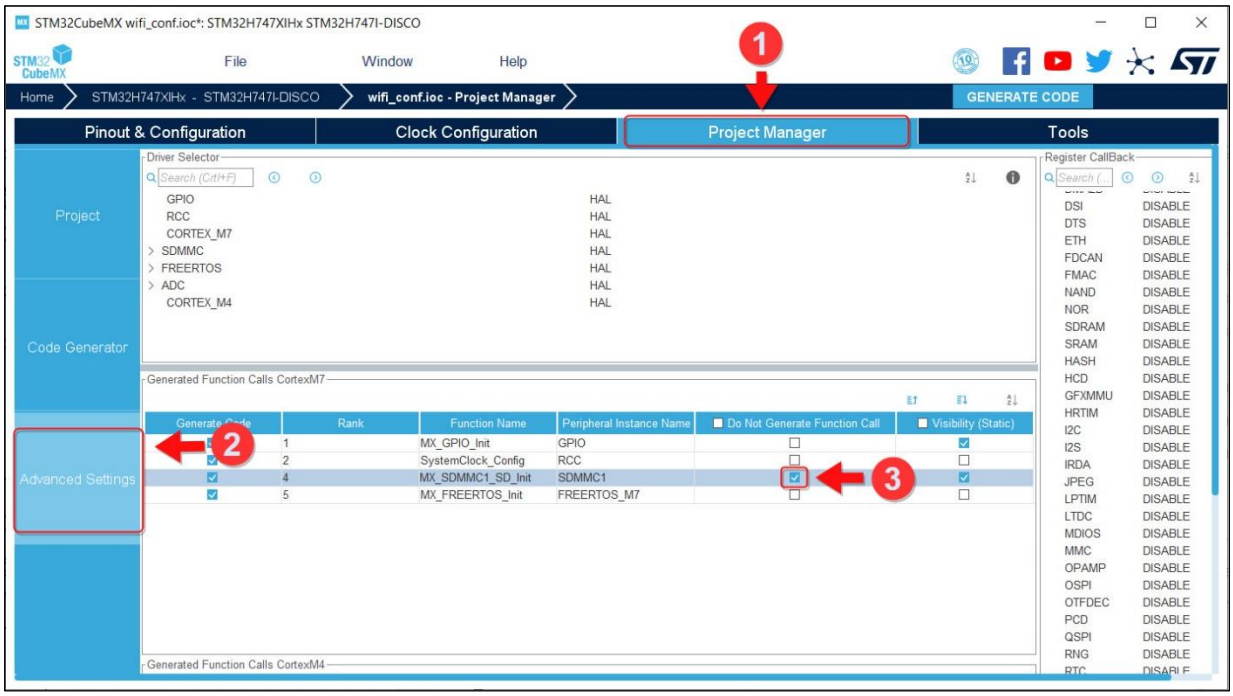
Create a new project from scratch

Make sure the SDMMC instance selected has its pins routed to the Infineon Connectivity device. Follow the steps listed to enable/configure SDIO in STM32CubeMX:

1. Enable SDMMC block in **STM32CubeMX > Pinout & Configuration > Connectivity**.



2. Disable generation function call of SDMMC initialization (MX_SDMMC_SD_Init).



9.8.2 Control pins

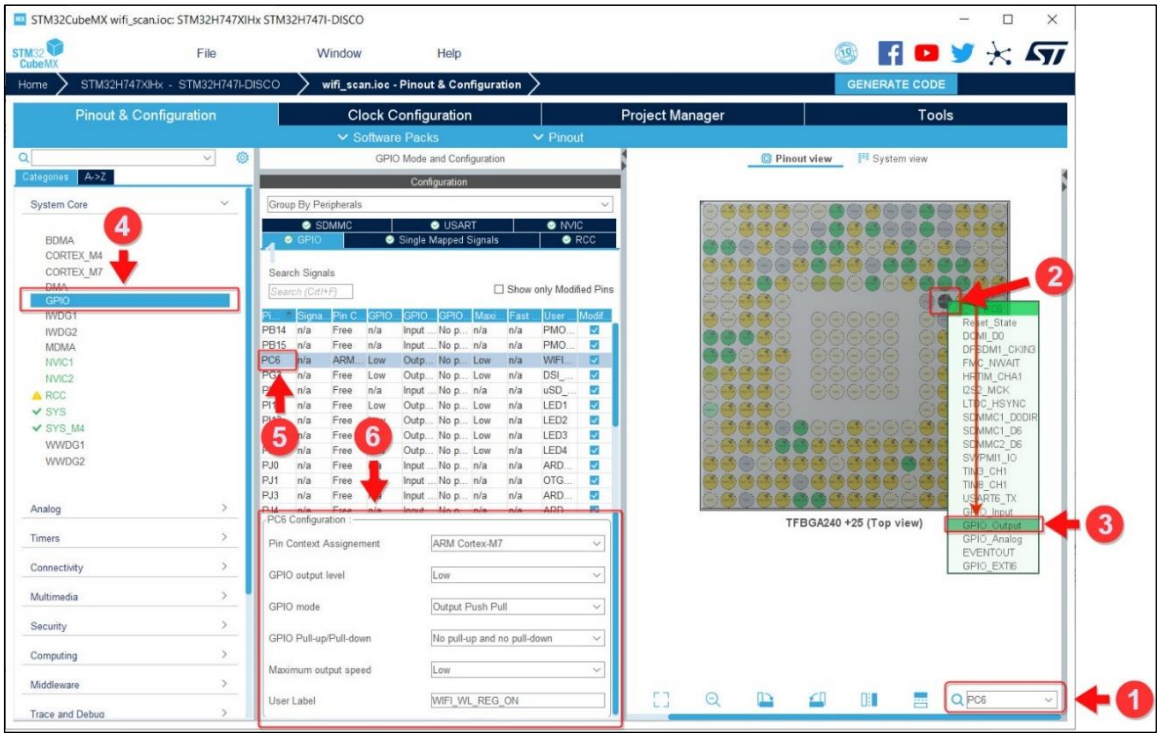
Infineon Connectivity devices require control lines to be connected to host MCU:

Line Name	FW Name	Description
WL_REG_ON	WIFI_WL_REG_ON	This is a power pin that shuts down the device WLAN section.
WL_HOST_WAKE	CYBSP_WIFI_HOST_WAKE	WLAN Host Wake: Active Low (OOB IRQ)
WL_DEV_WAKE		WLAN Device Wake *Note: WL_DEV_WAKE is not used in current version of PAL.*

9.8.2.1 WL_REG_ON

A power pin that shuts down the device WLAN section. WL_REG_ON must be configured as output with following parameters:

GPIO Parameter	Value	Note
Direction	GPIO_Output	
Pin Context Assignment	ARM Cortex-M7	Assign to core, where Connectivity run.
GPIO output level	Low	
GPIO mode	Output Push Pull (PP)	
GPIO Pull-up/Pull-down	No pull-up and no pull-down	
Maximum output speed	Low	
User label	WIFI_WL_REG_ON	



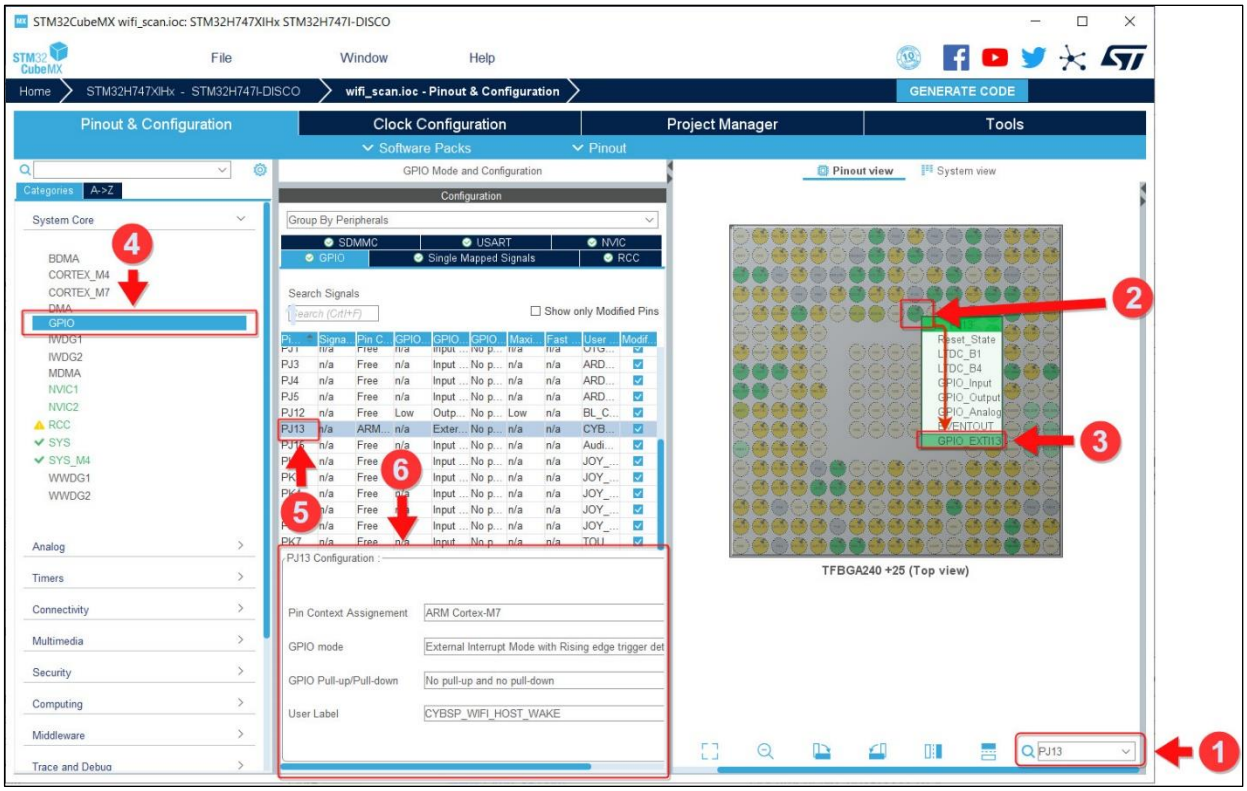
9.8.2.2 WL_HOST_WAKE

Host MCU Wake signal from WLAN section. This is required for SDIO out-of-band (OOB) interrupt support.

WL_HOST_WAKE must be configured in External Interrupt mode / EXTI with following parameters:

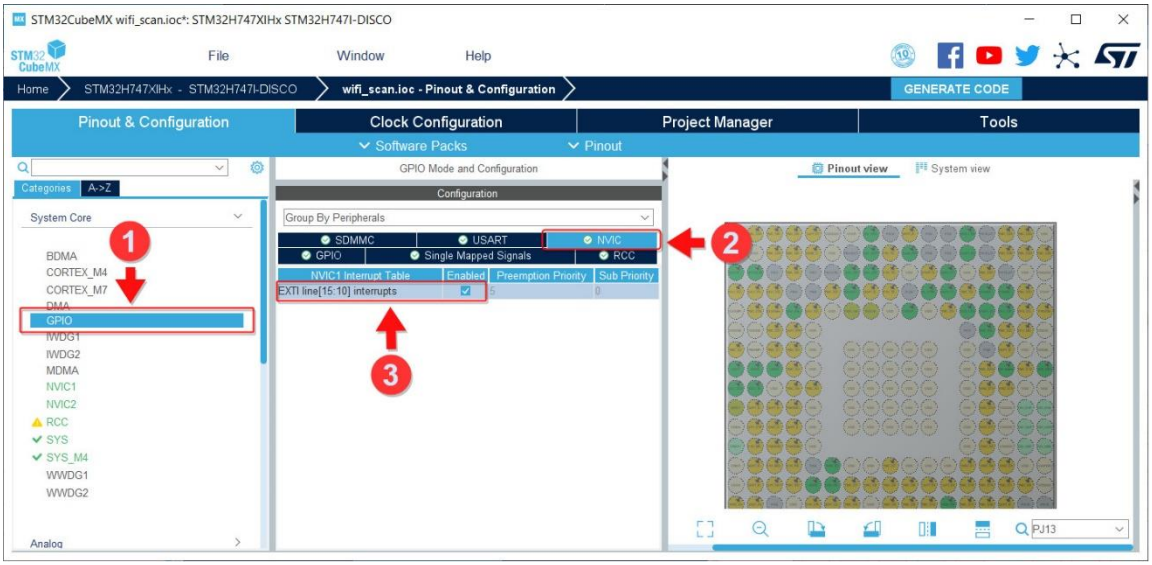
GPIO Parameter	Value	Note
Direction	GPIO_EXTIxx	
Pin Context Assignment	ARM Cortex-M7	Assign to core, where Connectivity runs.
GPIO mode	External Interrupt mode with Rising edge trigger detection	
GPIO Pull-up/Pull-down	No pull-up and no pull-down	
User label	CYBSP_WIFI_HOST_WAKE	
NVIC for EXTI	Enable	

1. Configure in STM32CubeMX:

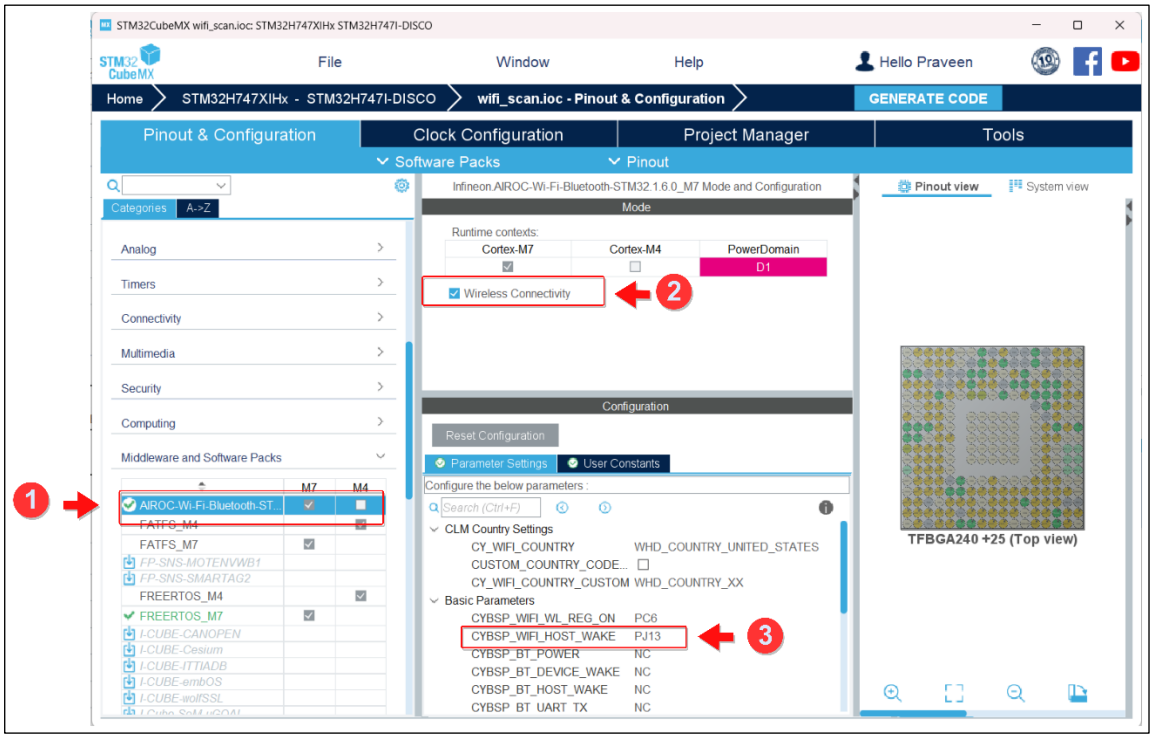


Create a new project from scratch

2. Enable NVIC interrupt for EXTI line:



3. Configure HOST WAKE GPIO pin in AIROC™ Wi-Fi Bluetooth® pack selector.



4. EXTI Callback handler must be overwriting in application and call stm32_cyhal_gpio_irq_handler function:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    stm32_cyhal_gpio_irq_handler(GPIO_Pin);
}
```

5. Configure "muxenab" in WLAN nvram file "wifi_nvram_image.h" depending on the module used.

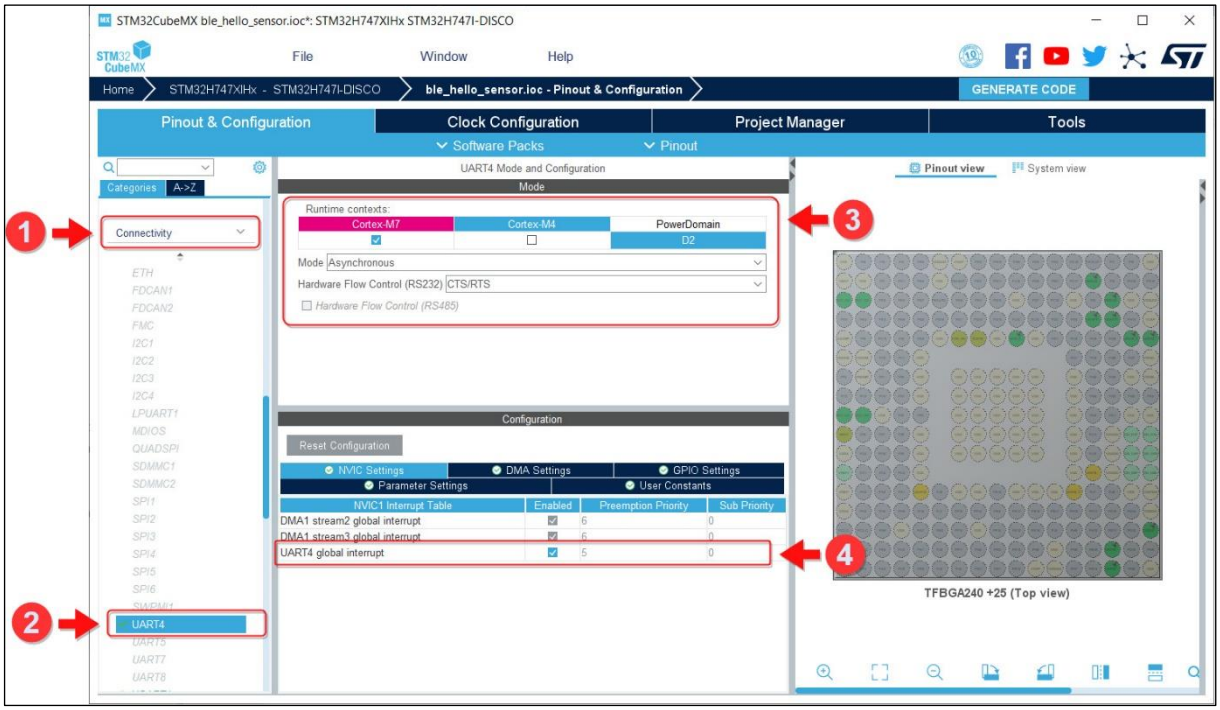
   For example, for Module 1LV, muxenab=0x1; for Module IDX & IYN, muxenab=0x10.

9.9 Configure resources for Bluetooth® connectivity

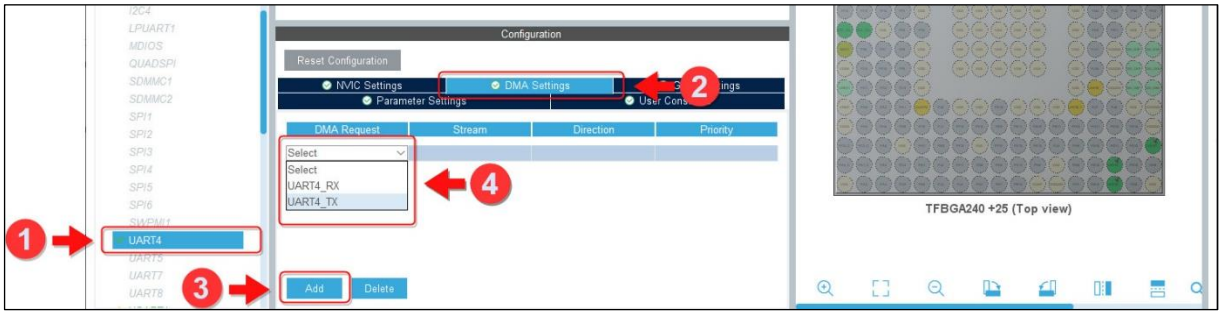
The following Peripherals and I/O lines required for the host MCU to communicate to Infineon connectivity device(s) for Bluetooth:

9.9.1 UART

1. Enable UART block in **STM32CubeMX > Pinout & Configuration > Connectivity.**
2. Configure Mode as **Asynchronous**.
3. Configure Hardware Flow Control (RS232) as **CTS/RTS.**
4. Enable UART interrupt in **NVIC Settings**.

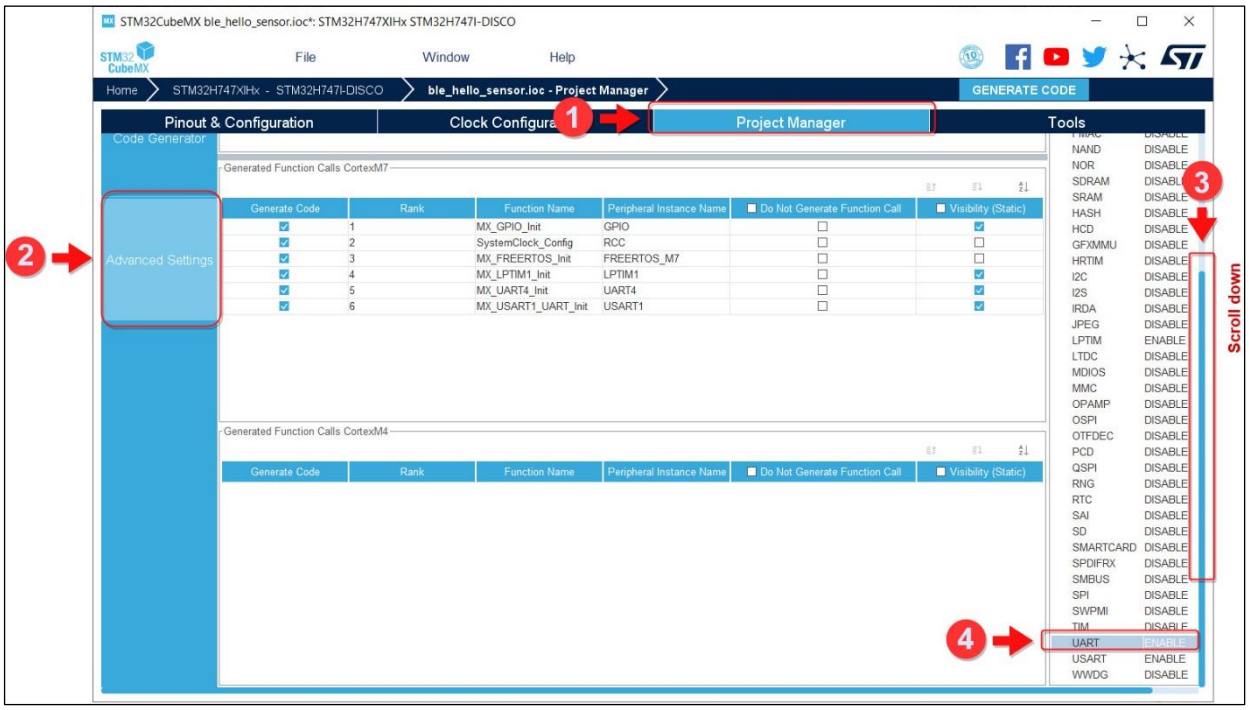


5. Add DMA for RX and TX in **DMA Settings**. Use default settings for RX/TX.



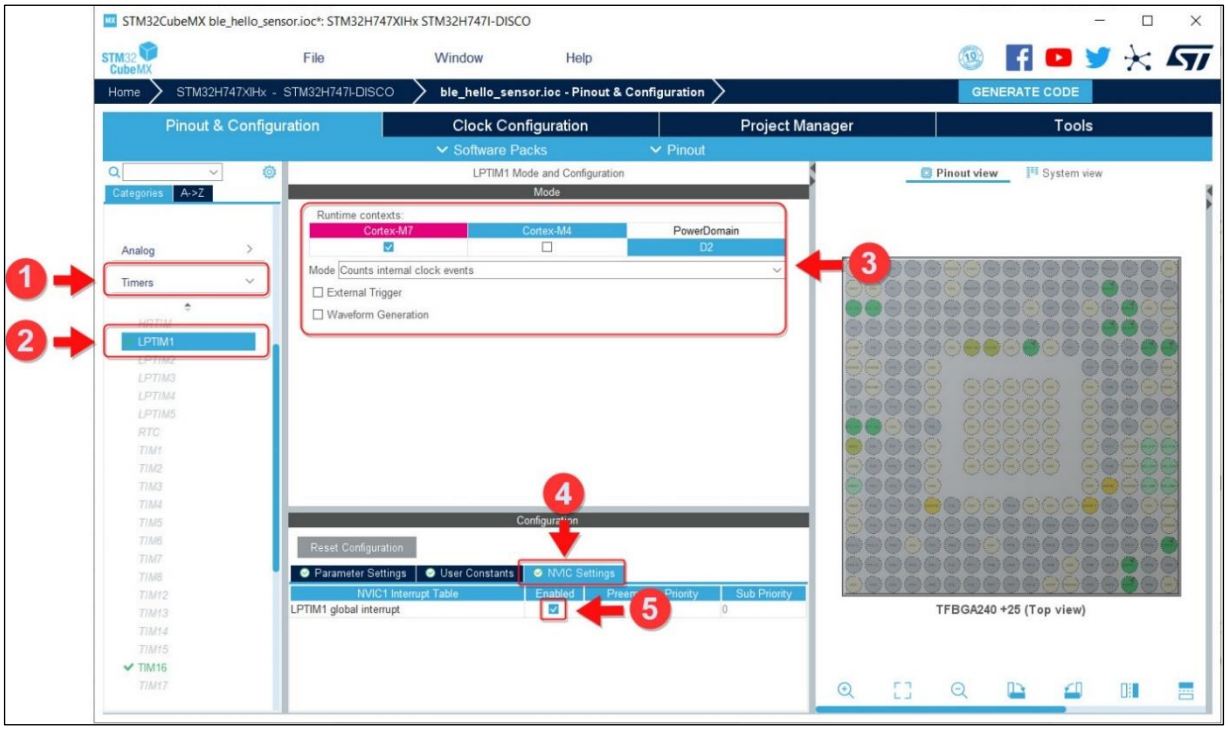
Create a new project from scratch

6. Enable UART Callback.



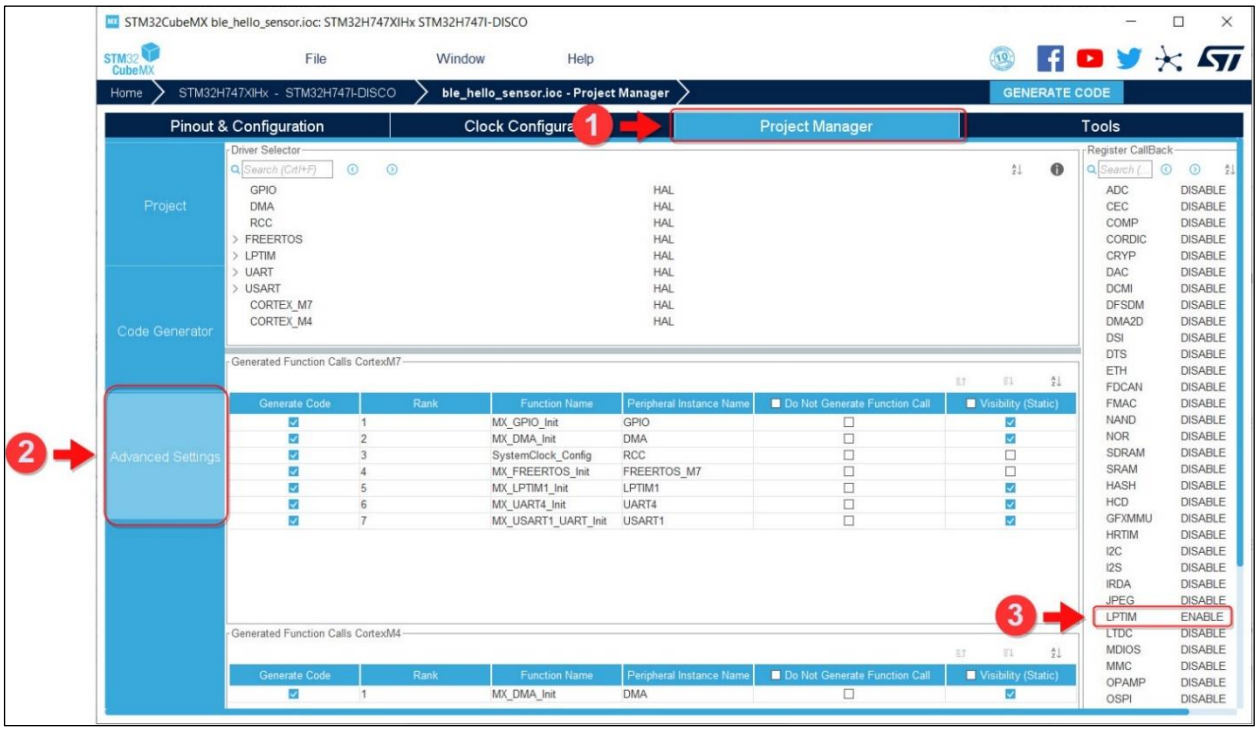
9.9.2 LPTIMER

1. Enable LPTIMER block in **STM32CubeMX > Pinout & Configuration > Timers**.
2. Configure Mode as **Counts internal clock events**.
3. Enable LPTIMER interrupt in **NVIC Settings**.



Create a new project from scratch

4. Enable LPTIM Callback.



9.9.3 Control pins

Infineon Connectivity devices require control lines to be connected to host MCU:

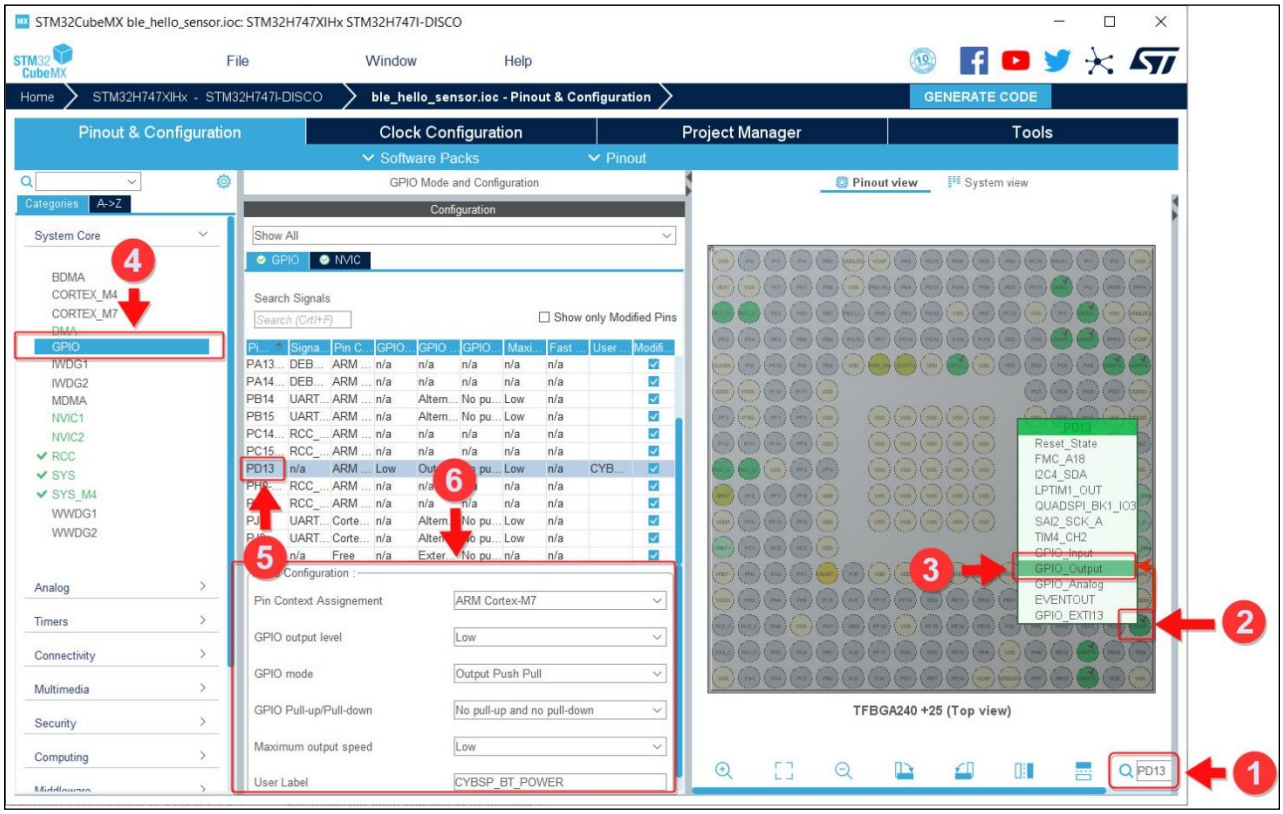
Line Name	FW Name	Description
BT_REG_ON	CYBSP_BT_POWER	Used by the PMU to power-up or power-down the internal regulators used by the Bluetooth® section.
BT_HOST_WAKE	CYBSP_BT_HOST_WAKE	Bluetooth® device wake-up: Signal from the host to the CYW43xx indicating that the host requires attention. • Asserted: The Bluetooth® device must wake-up or remain awake. • De-asserted: The Bluetooth® device may sleep when sleep criteria are met. The polarity of this signal is software configurable and can be asserted HIGH or LOW. *Note: BT_HOST_WAKE is not used in current version of PAL.*
BT_DEV_WAKE	CYBSP_BT_DEVICE_WAKE	Host wake-up. Signal from the CYW43xx to the host indicating that the CYW43xx requires attention. • Asserted: host device must wake-up or remain awake. • De-asserted: host device may sleep when sleep criteria are met. The polarity of this signal is software configurable and can be asserted HIGH or LOW *Note: BT_DEV_WAKE is not used in current version of PAL.*

9.9.3.1 BT_REG_ON

A power pin that shuts down the device Bluetooth® section. BT_REG_ON must be configured as output with the following parameters:

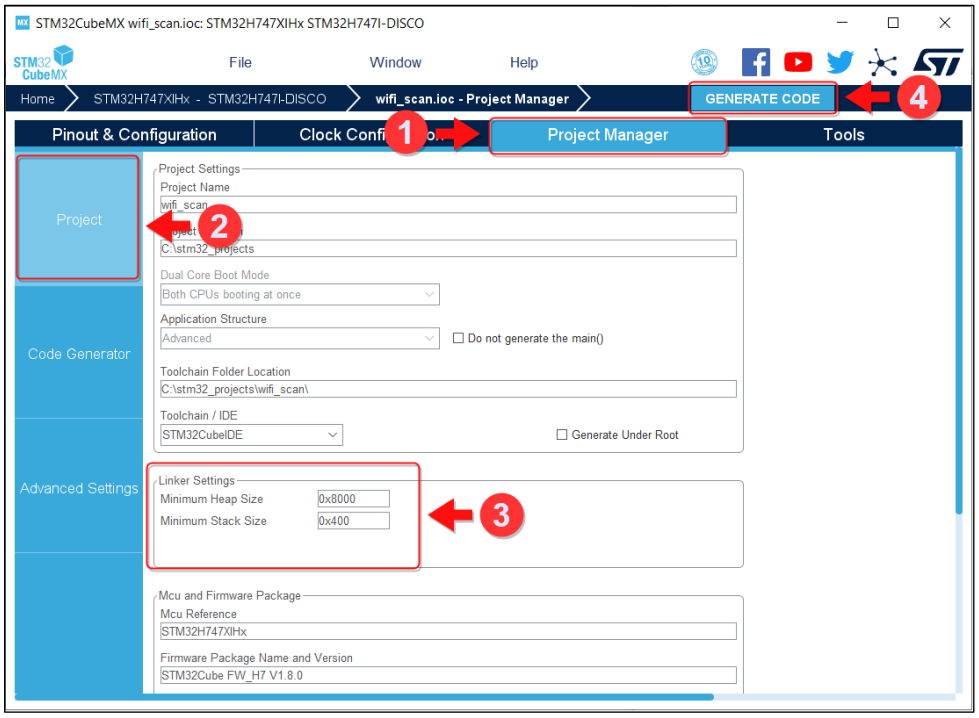
GPIO Parameter	Value	Note
Direction	GPIO_Output	
Pin Context Assignment	ARM Cortex-M7	Assign to core, where Connectivity run.
GPIO output level	Low	
GPIO mode	Output Push Pull (PP)	
GPIO Pull-up/Pull-down	No pull-up and no pull-down	
Maximum output speed	Low	
User label	CYBSP_BT_POWER	

Configuration in STM32CubeMX:



9.10 Heap and stack configuration

Configure Heap and Stack size required for the example app.



9.11 Generating code

1. After clicking **Generate Code**, copy the following files from existing examples provided along with the pack:

   - cybsp.h
   - lwipopts.h

   Location of these files in the pack:

   STM32Cube\Repository\Packs\Infineon\Connectivity-STM32\1.8.0\Projects\STM32H747I-DISCO\Applications\wifi_scan\Core\Inc

2. Add the following to the *FreeRTOSConfig.h* file:

   ```
   /* Enable using CY_HAL for rtos-abstraction */
   #define CY_USING_HAL
   ```

3. Update the following fields in the *cybsp.h* file to match the configurations done in the Configuring Control pins section

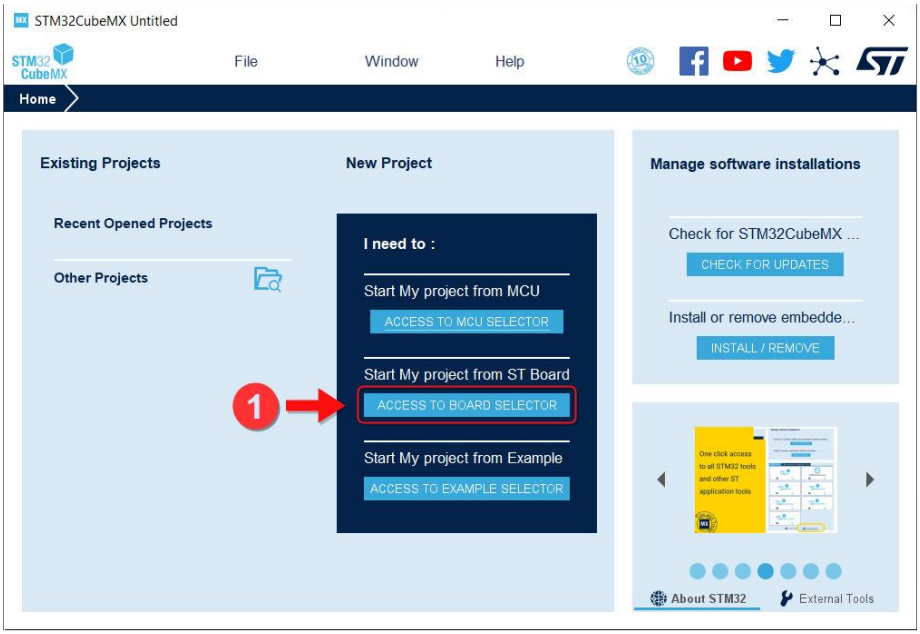
   ```
   /** These names are explicitly referenced in the support libraries */
   #define  CYBSP_WIFI_WL_REG_ON           ***
   #define  CYBSP_WIFI_HOST_WAKE           ***
   ```

10 Create a new project for non-H7 MCU boards

This section explains how to create new example project for any non-H7 MCU boards using the expansion pack.

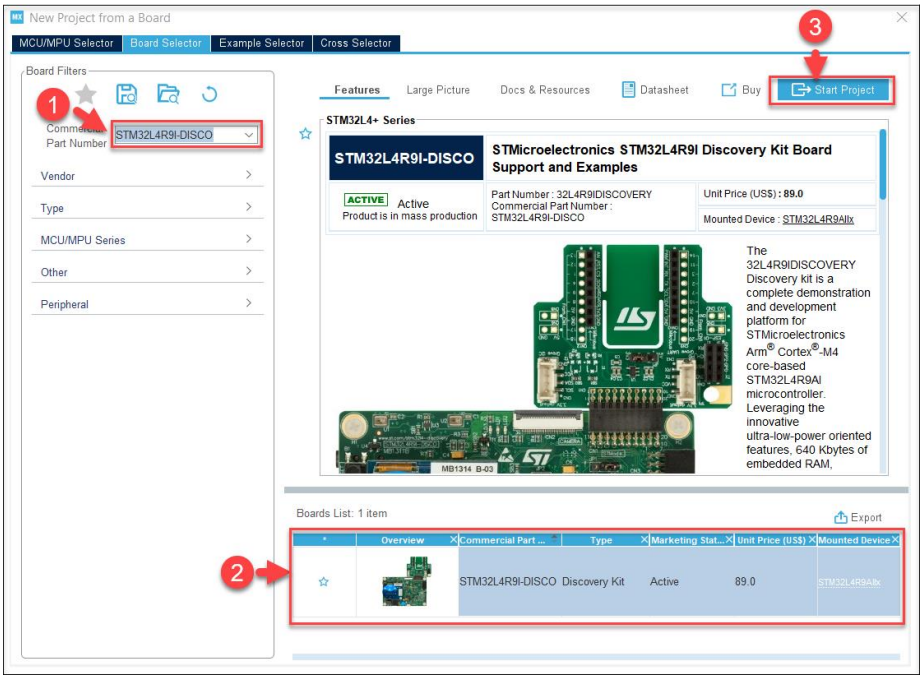
10.1 Creating a project

1. Start creating a project via the Access to Board Selector option.



2. Select a board like STM32L4R9I-DISCO

   - Enter/select the board number (STM32L4R9I-DISCO) and click on your selected board.
   - Select **Start Project**.



Create a new project for non-H7 MCU boards

3. Select Software Components from the AIROC™ Wi-Fi/Bluetooth® STM32 Expansion Pack

   - Select the Pinout & Configuration tab.
   - Select **Software Packs > Select Components**. This will show a list of the installed packs and their contents.
   - Platform/device is selected as CYW43438 for reference along with other components required for the Wi-Fi Example.
   - Enable Software components as required for the Wi-Fi Example.
   - Refer to Enable Software components from the AIROC™ Wi-Fi/Bluetooth® STM32 Expansion Pack.

10.2 FreeRTOS configuration

Follow same steps as mentioned in FreeRTOS Configuration.

10.3 Other configurations

1. Configure SDMMC (refer to SDIO).
2. Configure Control Pins (refer to Control Pins).
3. Configure Heap and Stack size (refer to Heap and Stack Configuration).

10.4 Changes required in PAL library

By default, Expansion pack supports only H7 MCU variant. The following changes are required to support other MCU variants.

1. stm32_cyhal_common.h
   (Middlewares\Third_Party\Infineon_Wireless_Infineon\pal\targets\TARGET_STM32\Inc) folder

```
#elif defined (STM32L4R9xx)
    #define TARGET_STM32L4xx
#elif defined (TARGET_STM32L4xx)
    #include "stm32l4xx.h"
    #include "stm32l4xx_hal.h"
    #include "stm32l4xx_hal_def.h"
```

2. stm32_cyhal_sdio_ex.h

   - Define STM32_RCC_PERIPHCLK_SDMMC based in the SDMMC* type supported by MCU variant.
   - For L4, it is RCC_PERIPHCLK_SDMMC1:

```
#elif defined (TARGET_STM32L4xx)
  /* RCC clock for SDMMC */
  #define STM32_RCC_PERIPHCLK_SDMMC RCC_PERIPHCLK_SDMMC1
```

3. stm32_cyhal_gpio.c

   Define "exti_table" based on the IRQn_Type defined in the stm32l4r9xx.h.

10.5 Changes required in main.c

To enable SDMMC to work with Wi-Fi connectivity device:

1. The API call has to be added at initialization with appropriate handle passed in:

```
SD_HandleTypeDef SDHandle = { .Instance = SDMMC1 };
cy_rslt_t result = stm32_cypal_wifi_sdio_init(&SDHandle);
```

2. SDMMC Interrupt handler must be overwriting in application and call stm32_cyhal_sdio_irq_handler function:

```
void SDMMC1_IRQHandler (void)
{
    stm32_cyhal_sdio_irq_handler();
}
```

3. GPIO Interrupt handler must be overwriting in application and call stm32_cyhal_gpio_irq_handler function:

```
void HAL_GPIO_EXTI_Callback (uint16_t GPIO_Pin)
{
    stm32_cyhal_gpio_irq_handler (GPIO_Pin);
}
```

10.6 DMA configuration

PAL Library is currently supporting SDIO CMD53 transfer using Internal DMA Registers in SDMMC. If the MCU variant does not support IDMABASE, Use DMA Channels and Modify below functions to handle SDIO Command 53.

- cyhal_sdio_bulk_transfer
- stm32_cyhal_sdio_irq_handler

10.7 OctoSPI configuration

STM32L4R9I-DISCO has external flash memory available and can be used for placing the Wi-Fi Firmware.

1. Linker script (*.ld) change to address external memory:

```
OSPI(rx)    : ORIGIN = 0x90000000,       LENGTH = 131072K
```

2. Add Linker script with section name defining where WiFi Firmware needs to be placed:

```
.whd_fw :
{
__whd_fw_start = .;
KEEP(*(.whd_fw))
__whd_fw_end = .;
} > OSPI
```

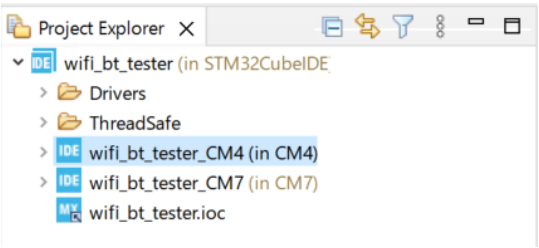
3. Add Preprocessor macro name:

```
CY_STORAGE_WIFI_DATA=".whd_fw"
```

11 Miscellaneous Information

11.1 Muli-Core MCU: STM32H747I-DISCO

Some MCUs (e.g. STM32H7) have multiples cores: Cortex-M7(CM7) and Cortex-M4(CM4). Although a Wi-Fi/Bluetooth® application uses only the CM7, please make sure to build and flash the CM4 application at the very beginning. ST Micro pre-installs demo applications on both CM7 and CM4. So, if you flash only the CM7 application, your CM7 Wi-Fi application and pre-installed CM4 demo application access to SDIO bus and Wi-Fi application will not work correctly.



12 Known issues, limitations, and workarounds.

This section lists the known issues/limitations of this release:

Problem	Component	Workaround
AP and STA Concurrent Mode does not work properly on different bands	wifi-host-driver	None. This will be addressed in a future release.
BLE is not supported on CYW43022, CYW55500, CYW55572 devices.	btstack-integration (CYW43022, CYW55500, CYW55572)	None. BT FW for CYW43022, CYW55500, CYW55572 devices will be added in a future release.
STM32U5 + CYW43012 is not able to join to WPA3 network.	wifi-host-driver, wpa3-external-supplicant	None. This will be addressed in a future release.
STM32L5 is not functional with CYW55500 device.	wifi-host-driver (CYW55500)	None. This will be addressed in a future release.
Sometimes, STM32 detects UART "Frame error" during the Bluetooth® LE communication (with CYW43012), which causes the Bluetooth® LE functionality to stop.	btstack-integration (CYW43012 BT FW)	Register a User UART Error Callback (by using HAL_UART_RegisterCallback function) with implementing the Bluetooth® LE or System reset.
STM32CubeIDE returns the linkage error "undefined reference to _nx_nd_cache***" when IPv6 is enabled in the NetxDuo configuration.	STM32CubeMx/ STM32CubeIDE	Manually add nx_nd_cache_***.c files from the MCU pack (e.g STM32Cube_FW_U5_V1.1.1\Middlewares\ ST\netxduo\common\src) to the project workspace.
STM32CubeMx does not remove sources/includes of the PDSC component from the project workspace (STM32CubeIde/EWARM), when another variant of this component is disabled or changed. It causes a build error when two versions of one component are added to the project (e.g. device CYW43012 and CYW4373)	STM32CubeMx/ STM32CubeIDE	Option 1: Manually remove files/includes of the previous component variant from the project workspace. Option 2: Remove the project workspace folder and generate a project from STM32CubeMx again. Be careful with the custom linker script – it may be missing after removing the project folder.
STM32CubeIDE does not include source files of modified device component for wifi_bt_tester project (i.e. if CYW4343W is selected instead of CYW43012 in device dropdown during Code Generate in STM32CubeMX)	STM32CubeMx/ STM32CubeIDE	This can be fixed by modifying the project to "C" instead of C++ in STM32CubeIDE before Generating the project.

Revision history

Date	Version	Description
2021-03-25	**	Initial release.
2022-11-14	*A	Updated from version 1.1.0 to version 1.2.0.
2022-12-22	*B	Updated from version 1.2.0 to version 1.3.0.
2023-07-14	*C	Updated to version 1.5.0.
2023-08-29	*D	Updated to version 1.5.1.
2024-03-18	*E	Updated to version 1.6.0.
2024-04-12	*F	Added reference to AIROC™ CYW5553x device.
2024-08-12	*G	Updated to version 1.6.1; updated a few component versions
2024-11-15	*H	Updated to version 1.7.0; added section 7.4.7.
2025-11-11	*I	Updated to version 1.8.0; updated a few component versions, and added sections 6 and 7.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.
The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

Edition 2025-11-11

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-32903 Rev. *H

Important notice
The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie")

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.