Getting started with X-CUBE-ST67W61

Introduction

This user manual briefly describes the X-CUBE-ST67W61 Expansion Package. The X-CUBE-ST67W61 is a set of software components implementing host applications driving a Wi-Fi® and Bluetooth® LE coprocessor (ST67W611M1).

1 General information

The ST67W611M1 Expansion Package runs on STM32 32-bit microcontrollers based on Arm®Cortex® processors.

Note: Arm and TrustZone are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

1.1 Acronyms and definitions

Table 1 presents the definitions of the acronyms that are relevant for a better understanding of this document.

Table 1. List of acronyms

Acronym	Definition
ACK	Acknowledgment
AP	Access point
API	Application programming interface
AT	Attention (command set used in modems)
BLE	Bluetooth® LE
BSD socket	Berkeley socket
BSP	Board support package
BSSID	Basic service set identifier
CLI	Command-line interface
DCM	Dual carrier modulation
DHCP	Dynamic host configuration protocol
DL/UL	Downlink/Uplink
DNS	Domain name system
DTIM	Delivery traffic indication message
ER	Extended range
FOTA	Firmware (update) over-the-air
FSM	Finite state machine
FW	Firmware
GATT	Generic attribute profile
HAL	Hardware abstraction layer
HTML	Hypertext Markup Language
HTTP	Hypertext transfer protocol
HTTPS	Hypertext transfer protocol secure
IDE	Integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet protocol
ITM	Instrumentation trace macrocell
LAN	Local area network
LDPC	Low-density parity-check
LED	Light-emitting diode

Acronym	Definition
LwIP	Lightweight IP
MAC	Media access control
MCU	Microcontroller unit
MEMS	Micro-electro-mechanical systems
MFG	Manufacturing
MPDU	MAC protocol data unit
MQTT	Message queuing telemetry transport
MSDU	MAC service data unit
MTU	Maximum transmission unit
MU-MIMO	Multi-user multiple-input multiple-output
MW	Middleware
NCP	Wi-Fi® and Bluetooth® LE network coprocessor. Also refers to ST67W611M1.
OFDMA	Orthogonal frequency-division multiple access
OS	Operating system
OTA	Over-the-air
PA/LNA	Power amplifier / low-noise amplifier
P2P	Peer-to-peer
QoS	Quality of service
RAM	Random-access memory
RF	Radio frequency
RSSI	Received signal strength indicator
SCTP	Stream control transmission protocol
SDK	Software development kit
SNTP	Simple network time protocol
SoftAP	Software access point
SPI	Serial peripheral interface
SPISYNC	SPI synchronous driver
SR	Spatial reuse
SSID	Service set identifier
SSL	Secure sockets layer
STA	Station
STBC	Space-time block coding
SW	Software
SWO	Serial wire output
SWD	Serial wire debug
TCP/IP	Transmission control protocol/internet protocol
TLS	Transport layer security
TOS	Type of service
TWT	Target wake time
UART	Universal asynchronous receiver transmitter
UDP	User datagram protocol

Acronym	Definition
URL	Uniform resource locator
USB	Universal serial bus
UTC	Coordinated universal time
UUID	Universally unique identifier
WEP	Wired equivalent privacy
Wi-Fi®	Wireless fidelity
WPA	Wi-Fi® protected access
WPA-EAP	Wi-Fi® protected access - extensible authentication protocol
WPS	Wi-Fi® protected setup
XTAL	Crystal oscillator

2 X-CUBE-ST67W61 architecture overview

The X-CUBE-ST67W61 is a set of software components implementing host applications driving a Wi-Fi® and Bluetooth® LE coprocessor (ST67W611M1). The coprocessor is controlled via AT command over SPI interface.

Two system architectures are proposed:

- The legacy LwIP off-load architecture (also known as architecture T01), where LWwIP is embedded in the ST67W611M1 module. This is implemented in the ST67W611M1 within the `st67w611m_mission_t01_v2.x.y.bin` binary.
- The LwIP on-host architecture (also known as architecture T02) where LwIP is embedded in the host STM32. This is implemented in the ST67W611M1 within the `st67w611m_mission_t02_v2.x.y.bin` binary.

The figures below depict both system architecture overviews:

Figure 1. LwIP off-load architecture (T01)

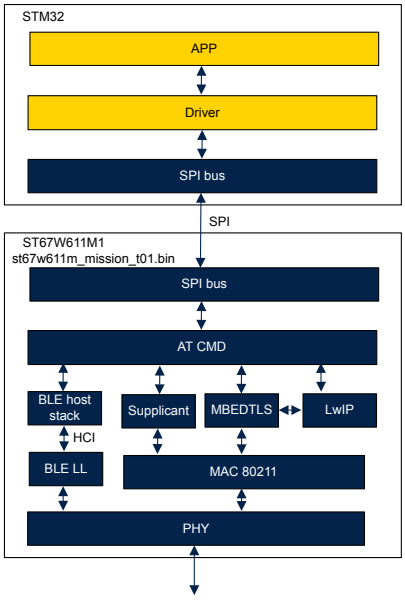
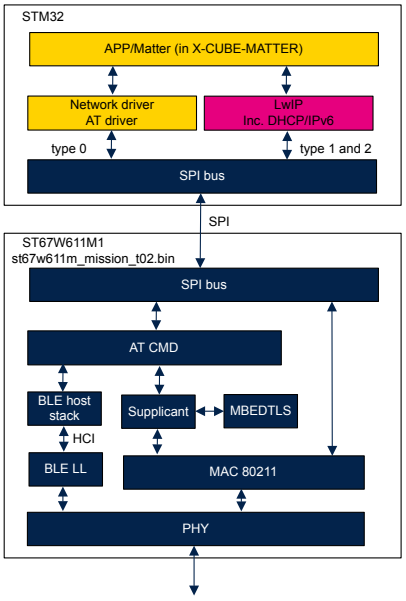


Figure 2. LwIP on-host architecture (T02)



3 ST67W611M1 features

3.1 Module content

- All-in-one Wi-Fi® / Bluetooth® LE wireless microcontroller
- Embedded 40 MHz high-precision crystal
- All RF components for the transmission and reception matching network, including antenna filter
- Three versions:
  - Embedded antenna (-B version)
  - RF connector (-U version)
  - RF pin (-P version)

3.2 Supported wireless connectivity standards

- IEEE 802.11b/g/n/ax™
- Bluetooth® LE 5.4

3.3 Regulation certifications

The ST67W611M1 device is certified for multiple countries and regions worldwide.

As a result, certifying the final product is more efficient and less complex.

3.4 Wi-Fi® features

- Wi-Fi® 6, 2.4 GHz RF transceiver
- Wi-Fi® 20/40 MHz bandwidth, 1T1R
- Wi-Fi® security: WPS, WEP, WPA, WPA2, WPA3
- Maximum Tx power (11b 1 Mbps): 21 dBm
- Tx power (HE40 and MCS9): 16 dBm
- Rx sensitivity (HE40 and MCS9): - 67 dBm
- LDPC, STBC, beamforming, DL/UL OFDMA, MU-MIMO, spatial reuse (SR), dual carrier modulation (DCM), extended range (ER)(1)
- A-MPDU, A-MSDU, immediate block ACK, fragmentation and defragmentation
- STA, SoftAP, concurrent STA + SoftAP
- Application throughput more than 20 Mbps with LwIP on host
- Target wake time (TWT): refer to the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi6_Target_Wake_Time_(TWT)

1. *Some of these features are linked to the support of multiple antennas. Supporting these features means supporting access pointing (AP) to use them. It also means that the ST67W611M1 can provide feedback to the AP on channel condition, which the AP can use to optimize the link. For beamforming, the device supports the 'SU Beamformee' mode.*

3.5 Bluetooth® LE features

- Maximum Tx power: + 10 dBm
- Rx sensitivity:
  - Bluetooth® LE (2 Mbps): - 96.5 dBm
  - Bluetooth® LE (1 Mbps): - 99 dBm

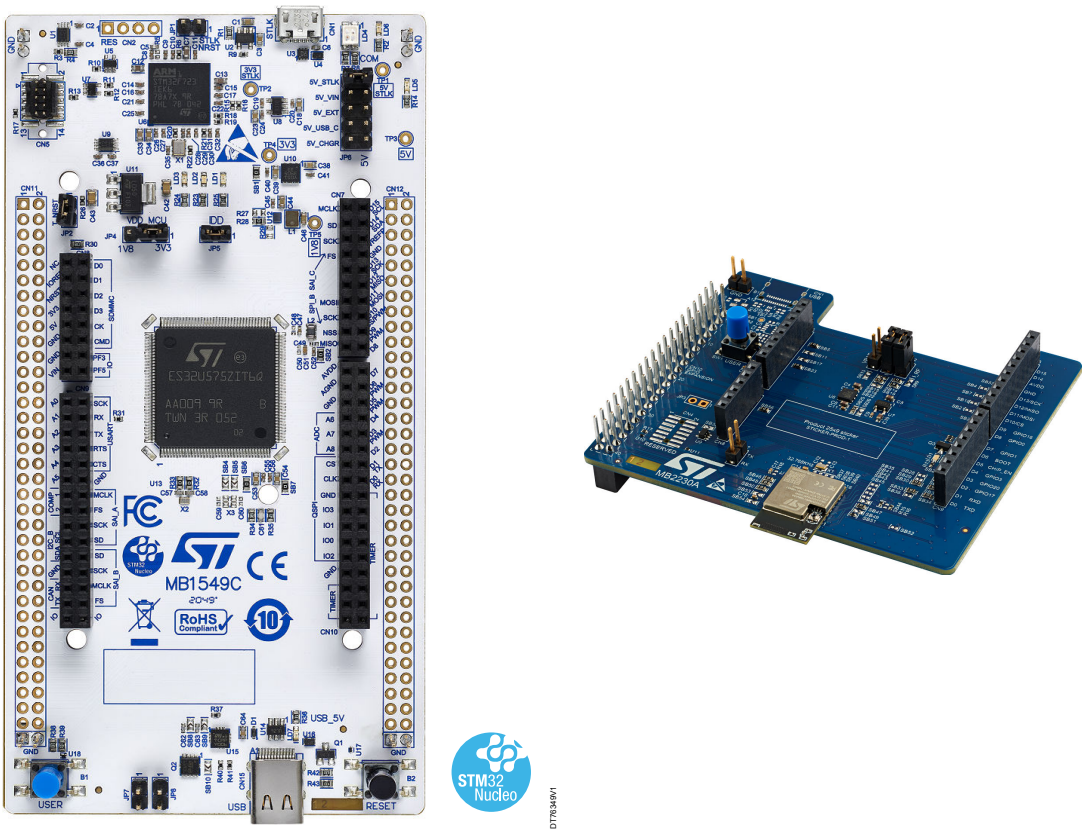
4 Hardware overview

For evaluation purposes, the ST67W611M1 is integrated into the X-NUCLEO-67W61M1 board. It is compatible with any STM32 Nucleo board via the ARDUINO® connector.

The system is composed of a host and the ST67W611M1 Wi-Fi® coprocessor.

Note: Any STM32 Nucleo board can be used as a host, NUCLEO-U575ZI-Q is shown as an example.

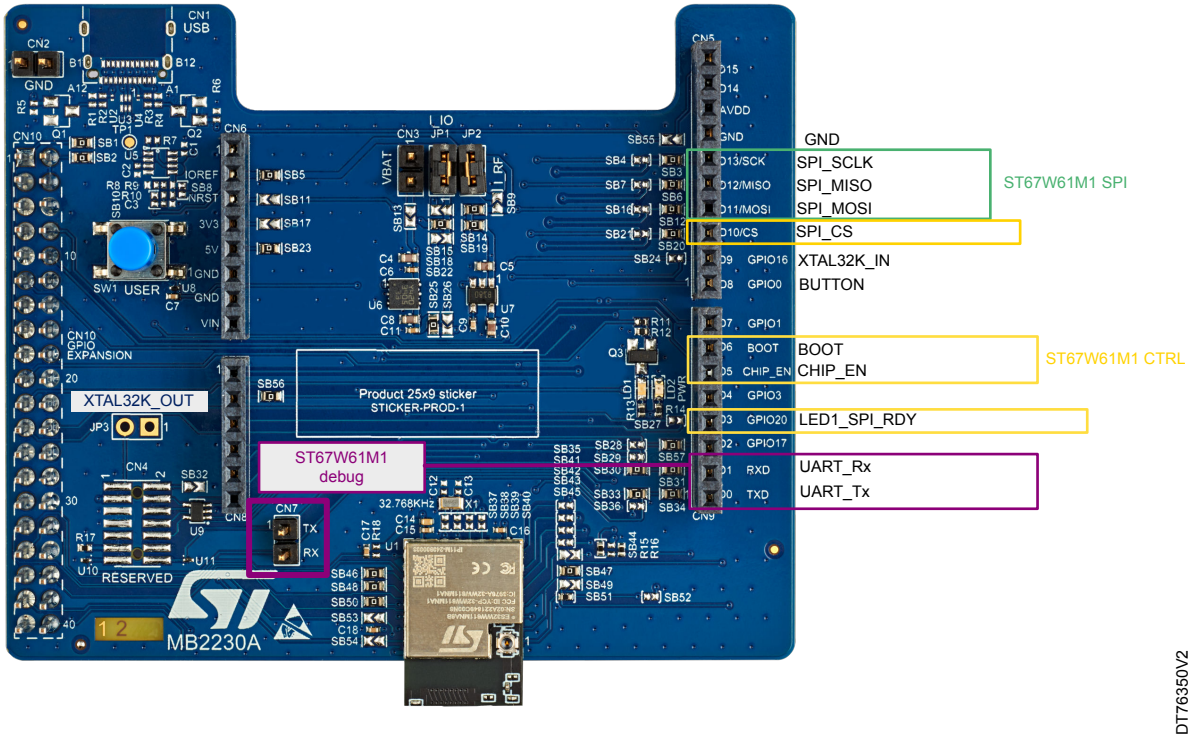
Figure 3. NUCLEO-U575ZI-Q (left) and X-NUCLEO-67W61M1 (right)



4.1 X-NUCLEO-67W61M1 mezzanine board overview

The image below shows the X-NUCLEO-67W61M1 and its pin mapping with the NUCLEO-U575ZI-Q board.

Figure 4. **Pin mapping**



The user must ensure that JP1 and JP2 are closed.

The pin description is listed below:

- CHIP_EN pin is asserted to start the ST67W611M1 device.
- BOOT pin is asserted when a new binary must be loaded in the ST67W611M1 device.
- SPI_CLK/SPI_MISO/SPI_MOSI pins are used for data communication.
- SPI_RDY pin is asserted by the ST67W611M1 to request SPI clock from the host.
- SPI_CS pin is used by the host to wake up the ST67W611M1 device.
- UART_Tx and UART_Rx are used for both loading the ST67W611M1 and for RF testing (manufacturing mode).

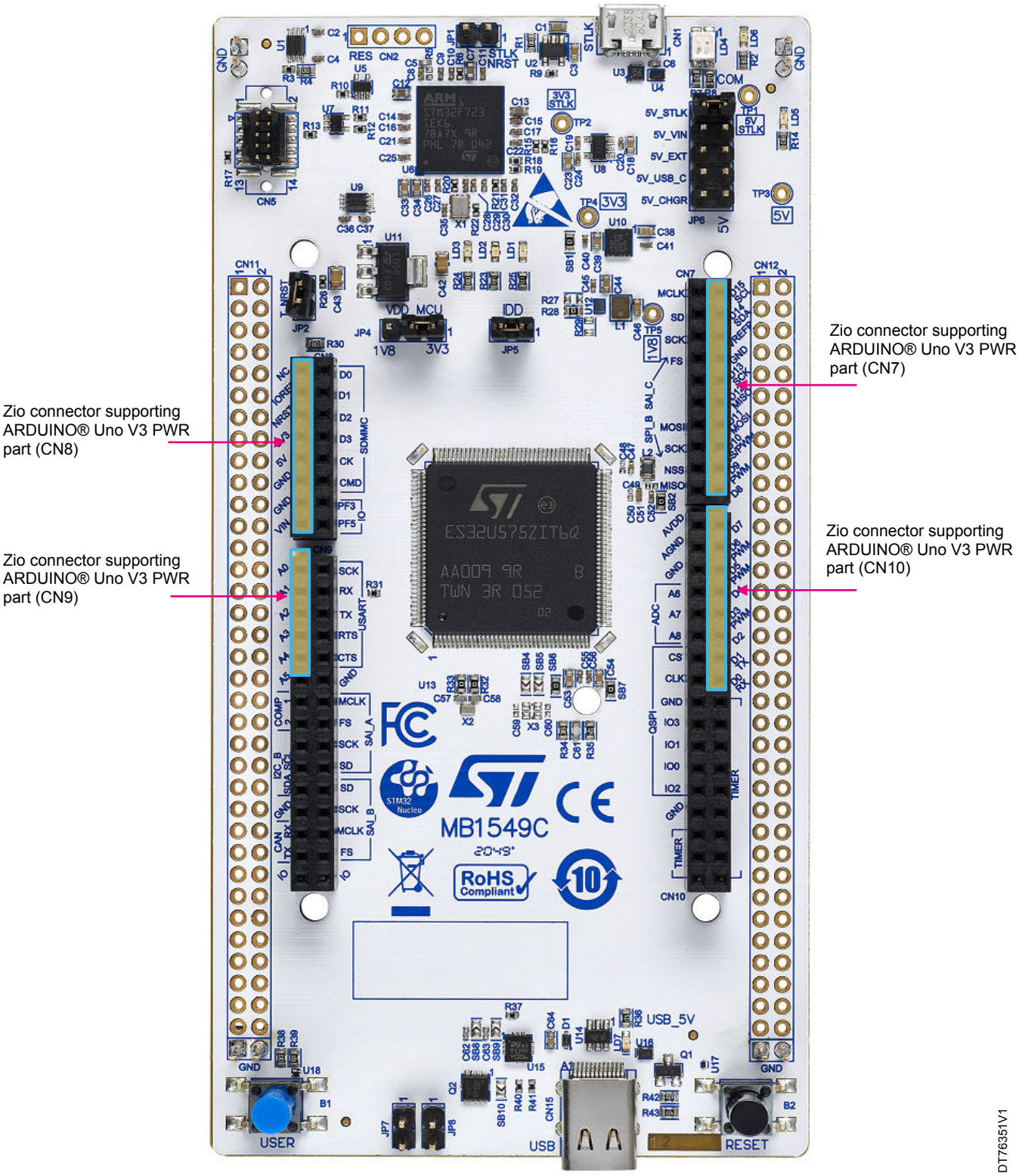
Note: UART_Tx and UART_Rx are only to:

- *Flash the ST67W611M1 binaries using the QConn_Flash tool.*
- *Test RF performances with the ST67W6X-RCT-TOOL*

4.2 Nucleo-144 main board overview

The X-NUCLEO-67W61M1 must be plugged on the ARDUINO® Uno V3 connector as illustrated below:

Figure 5. **NUCLEO-U575ZI-Q connections**



4.3 How to flash the ST67W611M1 device

The ST67W611M1 network coprocessor requires to be loaded with either a mission binary (see Section 4.5: Mission mode) or a manufacturing binary (Section 4.4: Manufacturing mode).

Warning: *Make sure that the module firmware version is up to date.*

The hardware setup is composed of:

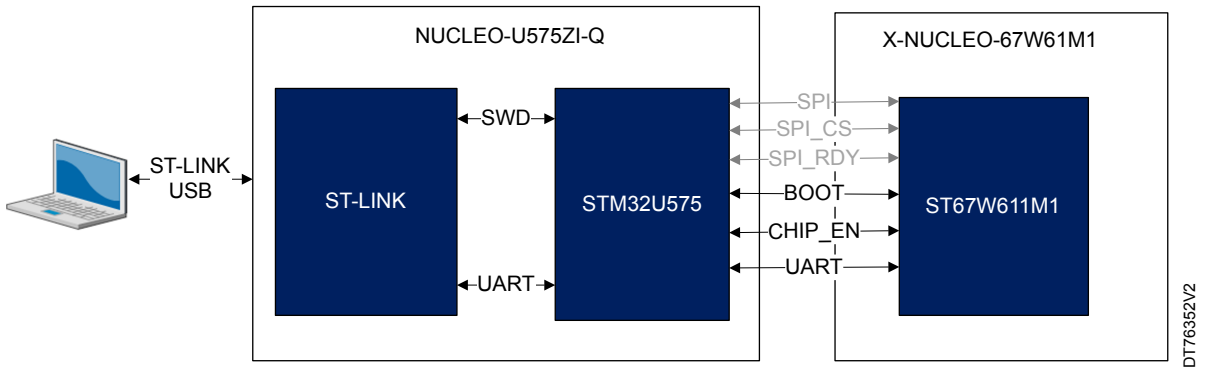
1. A PC running QConn_Flash tool
2. Any host STM32 board.
   X-CUBE-ST67W61 package contains `Bootloader.bin` binaries of the boards that are included in the package. Associated source code and project are located in *\Projects\$BOARD_NAME\Utilities\NCP\NCP_Loader*. If the host board is not included in the X-CUBE-ST67W61, the *NCP_Loader* project must be migrated.

Note: NUCLEO-U575ZI-Q is used as a host example.

3. A X-NUCLEO-67W61M1 board

The interface between the boards is illustrated in the figure below:

Figure 6. ST67W611M1 programming hardware setup



The programming process is automated using a batch file and is completed in three steps:

Table 2. 3-step ST67W611M1 programming process

Steps	Manufacturing mode	Mission 1 mode (LwIP running on ST67W611M1)	Mission 2 mode (LwIP running on host)
Step 1: Programs the bootloader in STM32	Programs `Bootloader.bin` in STM32	Programs `Bootloader.bin` in STM32	Programs `Bootloader.bin` in STM32
Step 2: Programs the ST67W611M1 device	Programs `st67w611m_mfg_v2.x.y.bin` in ST67W611M1 device	Programs `st67w611m_mission_t01_v2.x.y.bin` in ST67W611M1 device	Programs `st67w611m_mission_t02_v2.x.y.bin` in ST67W611M1 device
Step 3: Programs the host STM32 application software	Programs `UART_bypass.bin` in STM32	Programs `ST67W6X_CLI.bin` in STM32	Programs `ST67W6X_CLI_LWIP.bin` in STM32
Batch script(1)	`NCP_update_mfg.bat`	`NCP_update_mission_profile_t01.bat`	`NCP_update_mission_profile_t02.bat`

1. *Scripts ".bat" are defined for Windows®. Similar scripts ".sh" are available for Linux® only.*

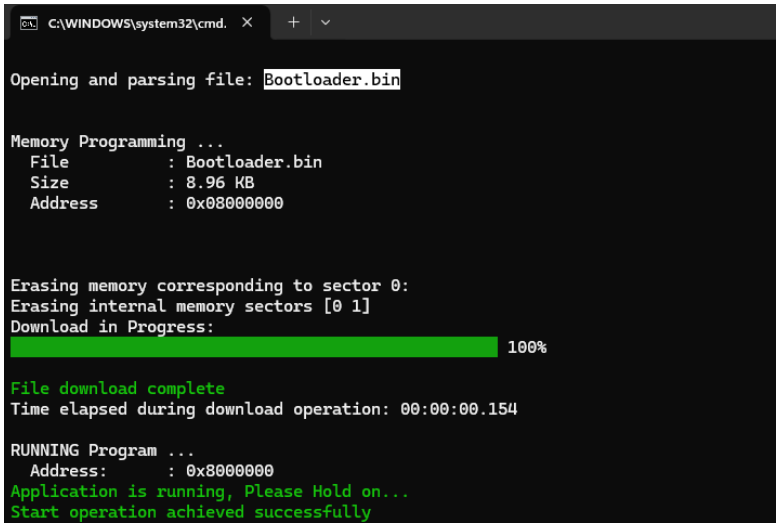
These modes are described in Section 4.4: Manufacturing mode and Section 4.5: Mission mode.

Prerequisite 1: Verify that the jumper JP2 is ON on the NUCLEO-U575ZI-Q.

Prerequisite 2: Disconnect any terminal connected to the STM32 Nucleo board as UART is used during the process.

Prerequisite 3: Verify the jumper position, as specified in the *getting started* section of UM3449.
Some command prompt screenshots of the steps are shown below. All the steps are done in one shot.

Figure 7. Step 1 - ST67W611M1 programming process



DT76353V2

Note: After step 1, check that the red led has blinked on the Nucleo board indicating that the `Bootloader.bin` is properly running.

Figure 8. **Step 2 - ST67W611M1 programming process**

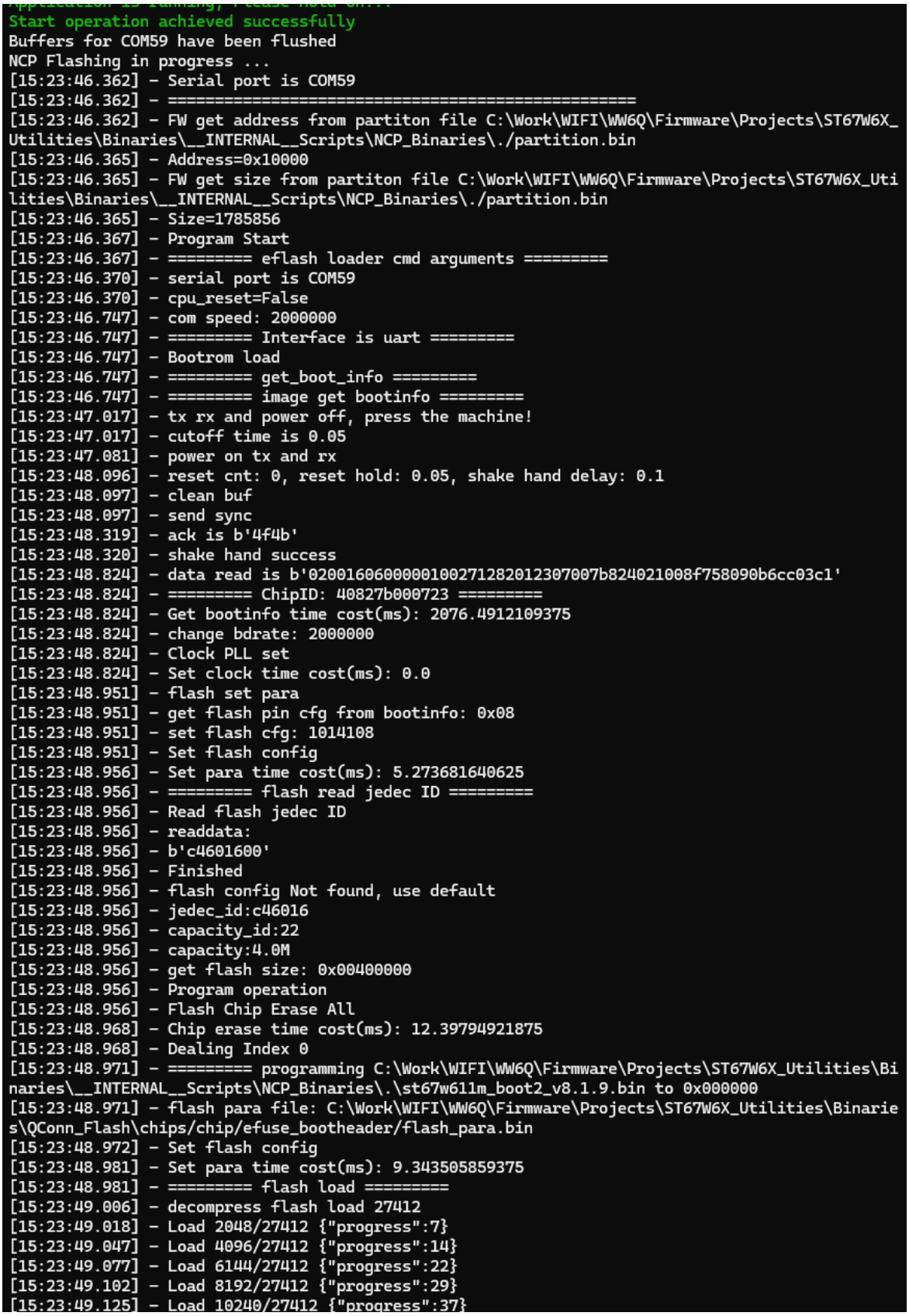
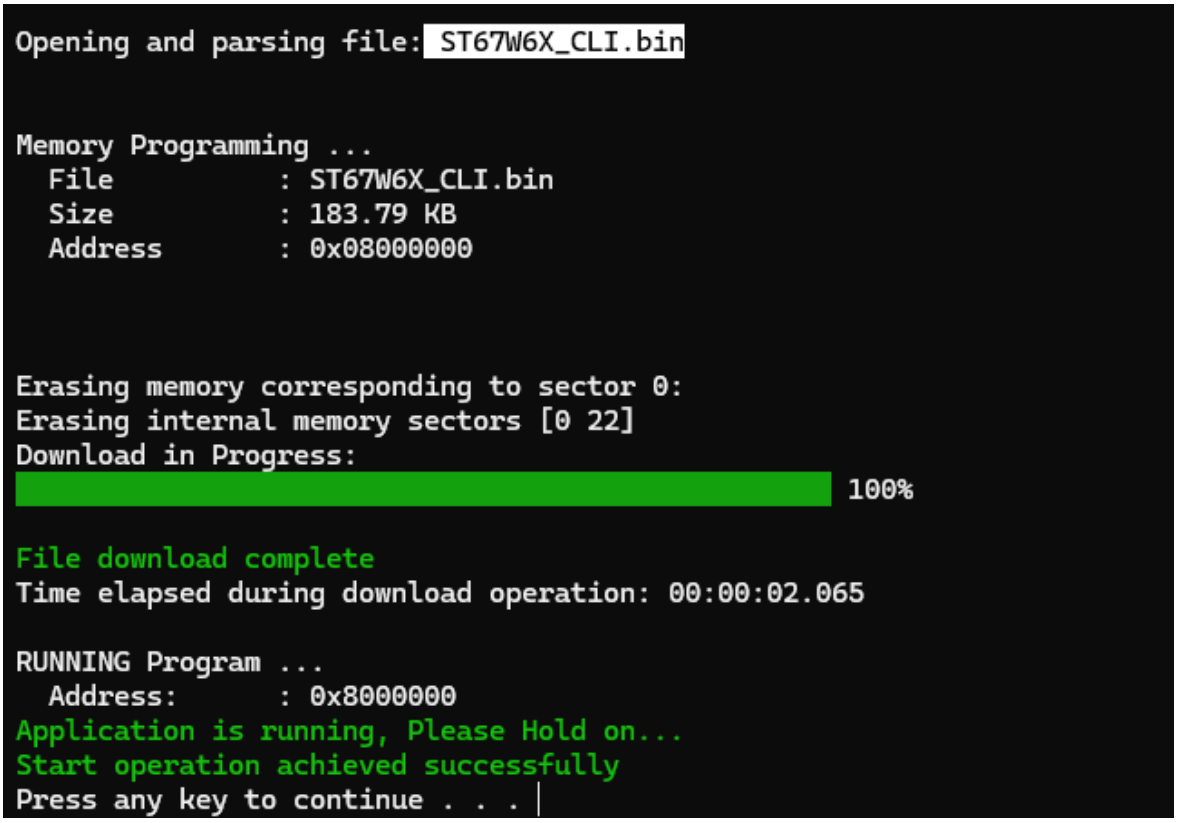


Figure 9. Step 2 (continued) - ST67W611M1 programming process

```
[15:23:49.389] - ========= programming C:\Work\WIFI\WW6Q\Firmware\Projects\ST67W6X_Utilities\Bi
naries\__INTERNAL__Scripts\NCP_Binaries\.\st67w611m_mission_t01_v2.0.75.bin to 0x10000
[15:23:49.391] - flash para file: C:\Work\WIFI\WW6Q\Firmware\Projects\ST67W6X_Utilities\Binarie
s\QConn_Flash\chips/chip/efuse_bootheader/flash_para.bin
[15:23:49.391] - Set flash config
[15:23:49.398] - Set para time cost(ms): 7.274658203125
[15:23:49.398] - ========= flash load =========
[15:23:49.961] - decompress flash load 795772
[15:23:49.973] - Load 2048/795772 {"progress":0}
[15:23:50.011] - Load 4096/795772 {"progress":0}
[15:23:50.034] - Load 6144/795772 {"progress":0}
[15:23:50.057] - Load 8192/795772 {"progress":1}
[15:23:50.081] - Load 10240/795772 {"progress":1}
[15:23:50.092] - Load 12288/795772 {"progress":1}
[15:23:50.114] - Load 14336/795772 {"progress":1}
[15:23:50.137] - Load 16384/795772 {"progress":2}
[15:23:50.148] - Load 18432/795772 {"progress":2}
[15:23:58.297] - Load 795772/795772 {"progress":100}
[15:23:58.297] - Load 795772/795772 {"progress":100}
[15:23:58.297] - Write check
[15:23:58.317] - Flash load time cost(ms): 8919.31005859375
[15:23:58.318] - Finished
[15:23:58.327] - Sha caled by host: ba739e7338ead70bf707c8a771837562b2851fb392cf7337cff111b01d4
88a2c
[15:23:58.327] - xip mode Verify
[15:23:59.480] - Read Sha256/1380608
[15:23:59.480] - Flash xip readsha time cost(ms): 1153.06640625
[15:23:59.480] - Finished
[15:23:59.480] - Sha caled by dev: ba739e7338ead70bf707c8a771837562b2851fb392cf7337cff111b01d48
8a2c
[15:23:59.480] - Verify success
[15:23:59.480] - Dealing Index 3
[15:23:59.480] - ========= programming C:\Work\WIFI\WW6Q\Firmware\Projects\ST67W6X_Utilities\Bi
naries\__INTERNAL__Scripts\NCP_Binaries\.\..\..\..\LittleFS\littlefs\littlefs.bin to 0x378000
[15:23:59.480] - flash para file: C:\Work\WIFI\WW6Q\Firmware\Projects\ST67W6X_Utilities\Binarie
s\QConn_Flash\chips/chip/efuse_bootheader/flash_para.bin
[15:23:59.495] - Set flash config
[15:23:59.500] - Set para time cost(ms): 4.853515625
[15:23:59.500] - ========= flash load =========
[15:23:59.576] - decompress flash load 15444
[15:23:59.590] - Load 2048/15444 {"progress":13}
[15:23:59.908] - Load 4096/15444 {"progress":26}
[15:23:59.955] - Load 6144/15444 {"progress":39}
[15:23:59.990] - Load 8192/15444 {"progress":53}
[15:24:00.034] - Load 10240/15444 {"progress":66}
[15:24:00.068] - Load 12288/15444 {"progress":79}
[15:24:00.101] - Load 14336/15444 {"progress":92}
[15:24:00.164] - Load 15444/15444 {"progress":100}
[15:24:00.164] - Load 15444/15444 {"progress":100}
[15:24:00.164] - Write check
[15:24:01.354] - Flash load time cost(ms): 1843.731689453125
[15:24:01.354] - Finished
[15:24:01.370] - Sha caled by host: 35f47ae598b0371b253edb87a098dbc441ab2d7be0b2a70a802fc682abf
c8f89
[15:24:01.370] - xip mode Verify
[15:24:01.767] - Read Sha256/462848
[15:24:01.767] - Flash xip readsha time cost(ms): 397.70849609375
[15:24:01.767] - Finished
[15:24:01.767] - Sha caled by dev: 35f47ae598b0371b253edb87a098dbc441ab2d7be0b2a70a802fc682abfc
8f89
[15:24:01.772] - Verify success
[15:24:01.772] - Program Finished
[15:24:01.772] - All time cost(ms): 15404.87939453125
[15:24:01.878] - close interface
[15:24:01.878] - [All Success]
```

DT76392V2

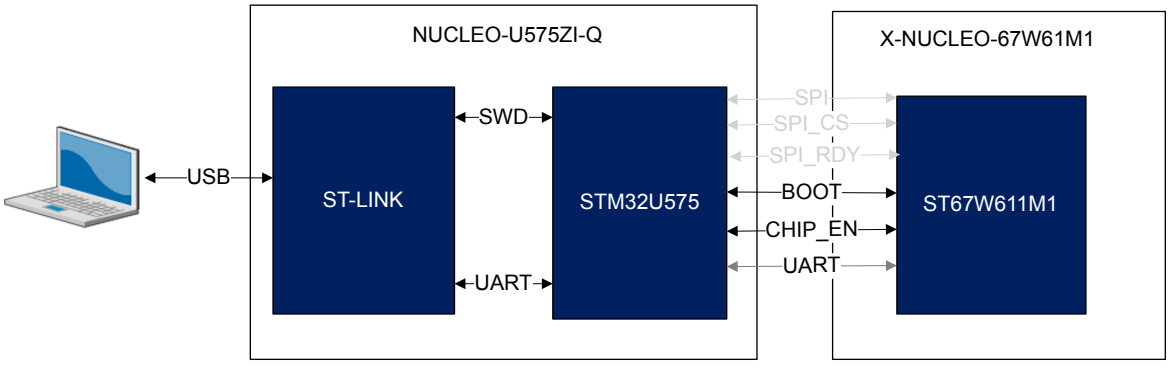
Figure 10. **Step 3 - ST67W611M1 programming process**



4.4 Manufacturing mode

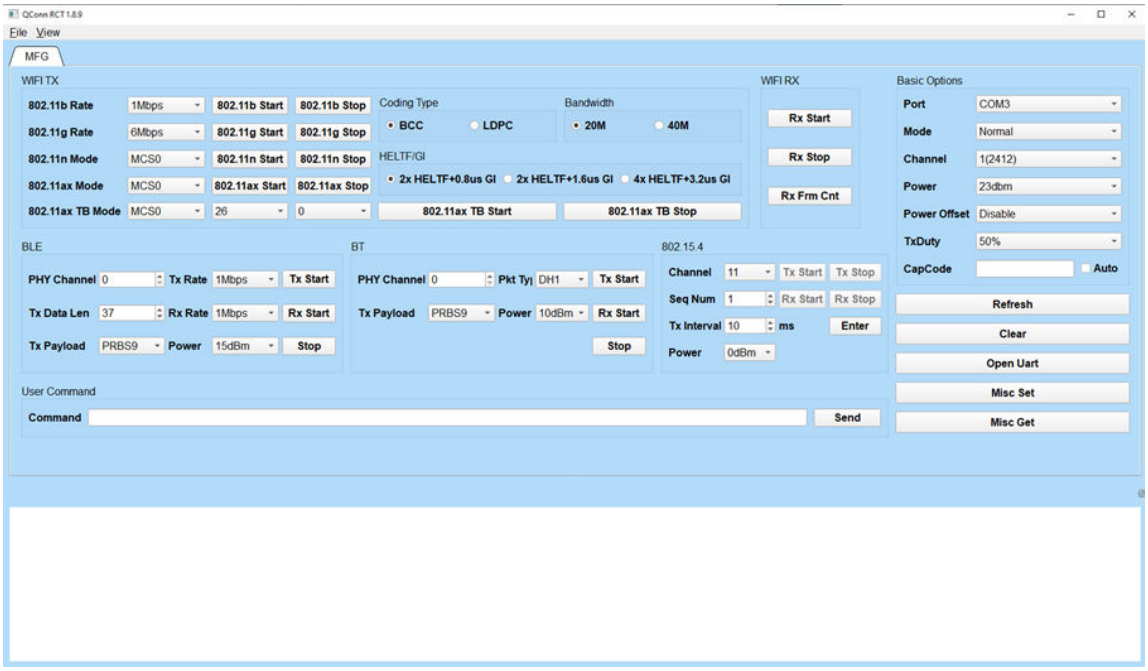
The manufacturing mode is used to measure RF performance of the ST67W611M1.

Figure 11. **Manufacturing mode hardware setup**



The ST67W6X-RCT-TOOL (https://www.st.com/en/embedded-software/x-cube-st67w61.html) tool can be used to perform RF testing. A screenshot of the tool is provided below:

Figure 12. **ST67W6X-RCT-TOOL manufacturing tool**



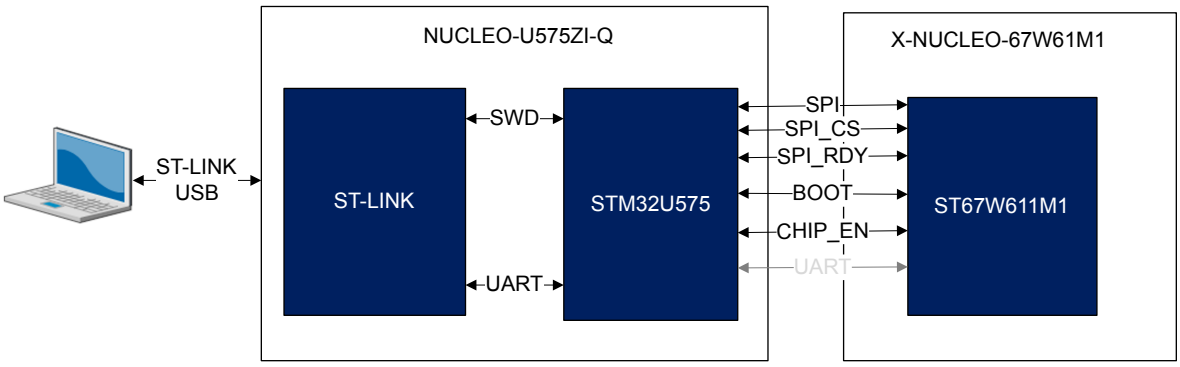
DT76357V1

Note: The ST67W611M1 manufacturing mode can be run in standalone mode.

4.5 Mission mode

The mission mode, also known as regular mode, features both Wi-Fi® and Bluetooth® LE capabilities.
The setup is depicted below:

Figure 13. **Mission mode hardware setup**



DT76358V2

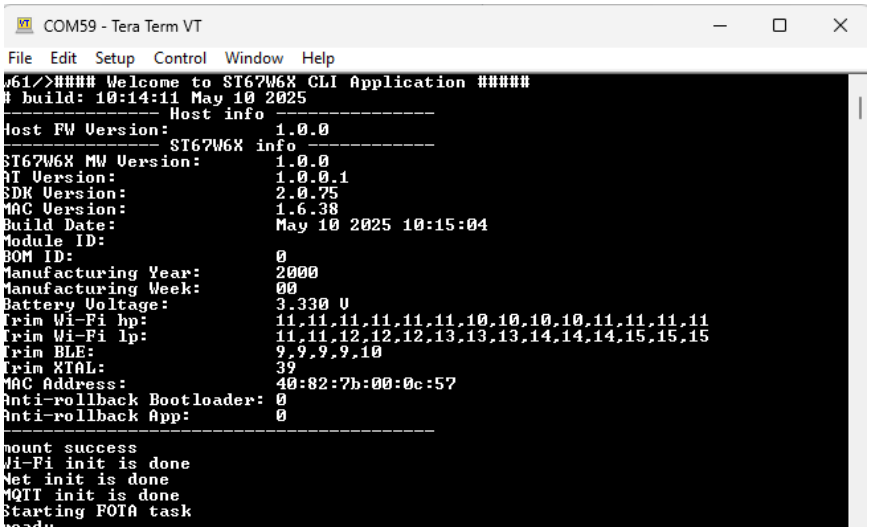
Note: Unlike the previous setup, the UART is no longer used. Instead, the communication bus has been switched to SPI bus.

There are two types of architecture available (refer to Section 2: X-CUBE-ST67W61 architecture overview):

- Legacy LwIP offload where LwIP runs on the ST67W611M1 device. This architecture requires `st67w611m_mission_t01_v2.x.y.bin` binary
- LwIP on host, where LwIP runs on the host. This architecture requires `st67w611m_mission_t02_v2.x.y.bin` binary

For example, the ST67W6X_CLI project (described in Section 6.5: CLI application) is loaded by default by the script. It appears as shown below (refer to Section 6.2.3: Terminal for terminal configuration):

Figure 14. ST67W6X_CLI Tera Term screenshot



DT76359V2

5 X-CUBE-ST67W61 architecture

5.1 Overview

5.1.1 Software top view

Figure 15. LwIP off-load architecture

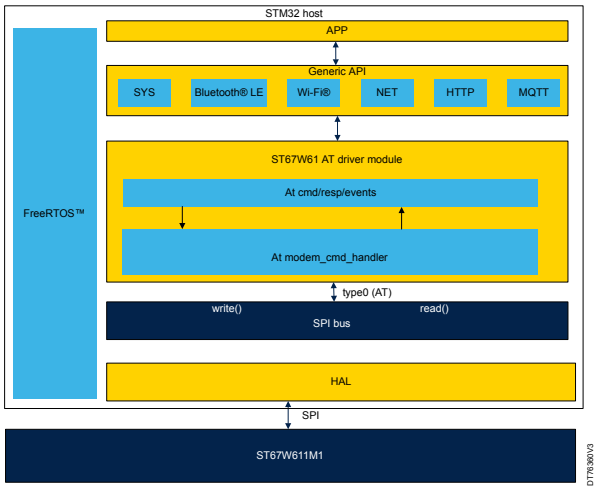
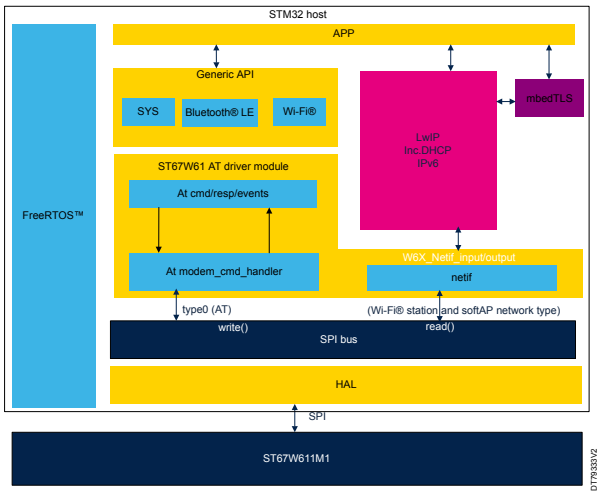


Figure 16. LwIP on-host architecture



The X-CUBE-ST67W61 is composed of a set of applications, ST67W6X_Network_Driver middleware, and ST67W61 drivers. In particular, it:

- Exposes the generic API to applications offering:
  - System and OTA services
  - Bluetooth® LE services
  - Wi-Fi® API services
  - Net module is aligned on BSD socket (blocking socket). Note that it is disabled when LwIP is on-host.
  - MQTT and HTTP services. Note that it is disabled when LwIP is on-host.
- Encompasses ST67W611M1 AT driver which:
  - Exposes basic AT command API
  - Parses AT responses and AT event with `modem_cmd_handler`.
- Requires SPI sync module:
  - For sending and receiving messages to/from SPI link
  - Does not have a callback mechanism to be informed that some data are ready to be read; the dedicated Rx thread at higher level handles the received data and messages
- When LwIP is on the host the netif module is enabled, in particular it:
  - Notifies station and softAP link status (up or down)
  - Sends network data to `spi_iface`
  - Forwards network data from `spi_iface` to the station and the softAP interface

Eight applications are delivered as part of the X-CUBE-ST67W61:

- ST67W6X_Echo (described in Section 6.4: Echo application)
- ST67W6X_CLI and ST67W6X_CLI_LWIP (described in Section 6.5: CLI application)
- ST67W6X_BLE_Commissioning (described in Section 6.6: Bluetooth® LE commissioning application)
- ST67W6X_BLE_p2pClient and ST67W6X_BLE_p2pServer (described in Section 6.7: Bluetooth® LE P2P application)
- ST67W6X_HTTP_Server (described in Section 6.8: HTTP server application)

- ST67W6X_HTTPS_Client (described in Section 6.9: HTTPS client application)
- ST67W6X FOTA (described in Section 5.2.5: Firmware over-the-air support (FOTA))
- ST67W6X_WiFi_Commissioning application (described in Section 6.12: Wi-Fi® Commissioning application)

Additionally, one demonstration is provided:

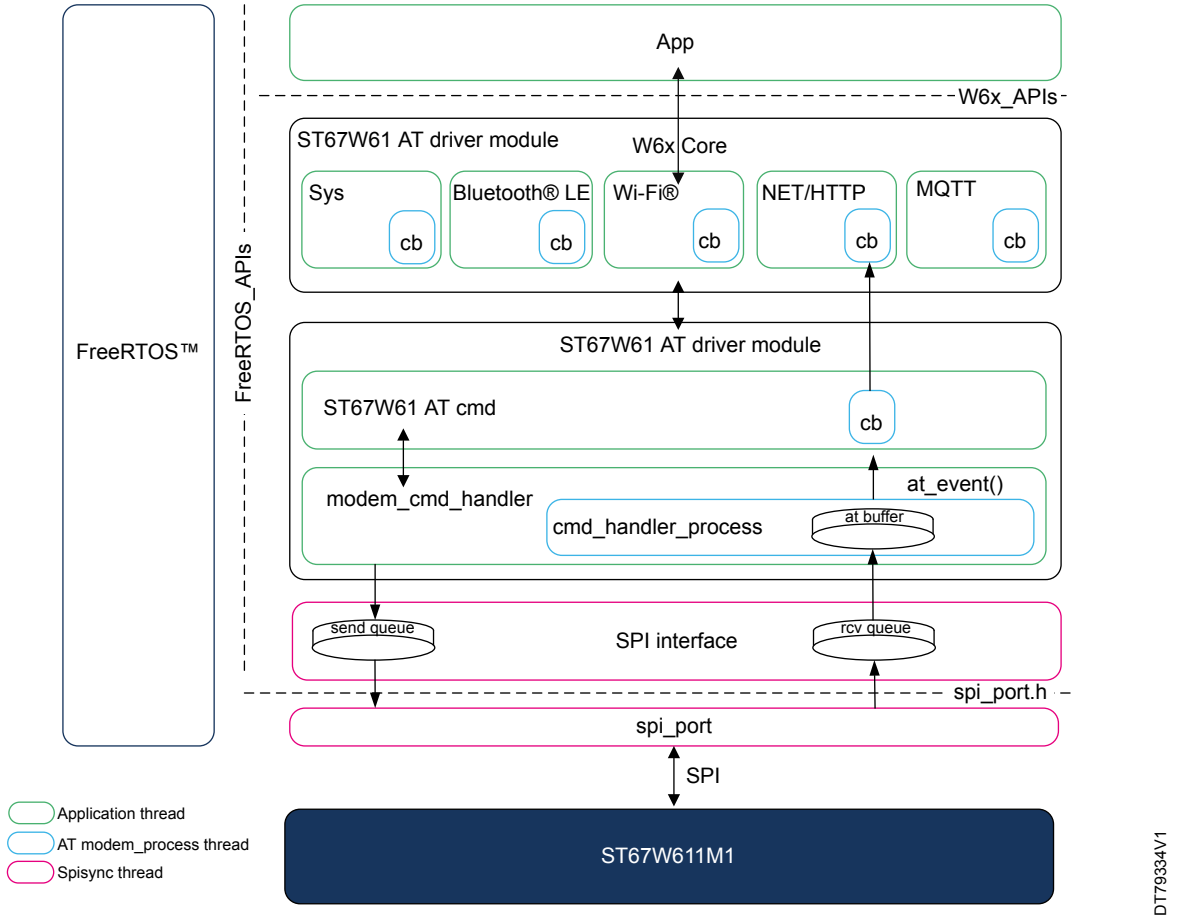
- ST67W6X_MQTT and ST67W6X_MQTT_LWIP (on STM32H5 only) (described in Section 6.11: MQTT demonstration)

5.1.1.1 *FreeRTOS™ implementation view*

5.1.1.1.1 FreeRTOS™ LwIP offload implementation view

The figure below shows the legacy LwIP off-load architecture, where LwIP is embedded in the module.

Figure 17. ST67W61 middleware driver: FreeRTOS™ view



The `spi_iface` runs within a FreeRTOS™ thread, which wakes up whenever a DMA transfer from SPI Rx or SPI Tx is triggered. The `spi_iface` embeds maximum one buffer queue for the data transmission and maximum two buffer queues in reception per active type (one for the AT type, one for the Wi-Fi® network type and one for the Wi-Fi® softAP network type) from the ST67W611M1 NCP. The `spi_iface` module implements the frame protocol between the host and the ST67W611M1 NCP. This frame protocol features a synchronization word, sequence number, length, and data buffer.

The scope of the AT driver is to format the AT messages on the Tx path (host to ST67W611M1 coprocessor) and parse the AT response and events on the Rx path (ST67W611M1 coprocessor to host). Its sits on top of the `spi_iface`.

The AT driver is composed of two units:

- The W61 AT commands which abstract any AT commands: `Execute`, `Set` or `Query`.
  - The AT `Execute` command sends the AT command and waits for a command response (OK or ERROR)
  - The AT `Set` command sends the AT command with the parameters and waits for a command response (OK or ERROR)
  - The AT `Query` command sends an AT command, waits for queried responses, and then waits for a command response (OK or ERROR)
  - Unsolicited events, which may be propagated to the upper layers
- The modem command handler is designed to parse and process AT commands and responses from a modem interface in an embedded system. It manages communication between the system and a modem by parsing, matching, and processing different types of AT commands and responses. It runs in its own FreeRTOS™ thread. The key functionalities are:
  - Command sending which provides functions to send commands to the modem, optionally waiting for responses and handling timeouts.
  - Command parsing, which reads data from a modem interface, identifies complete command lines (terminated by CR/LF), and matches them against a set of known command definitions.
  - Command matching, which supports both direct command matching (for commands that must be handled immediately) and regular matching to be parsed (for standard responses and unsolicited messages).
  - Parameter extraction, which parses its parameters when a command is matched, handling quoted strings and delimiters, and passes them to the appropriate handler function.
  - Command execution, which calls the handler function associated with each command, passing parsed parameters.
  - Synchronization, which uses FreeRTOS™ semaphores to protect shared resources and synchronize command processing and transmission.
  - Error handling, which tracks and reports errors encountered during command processing.

Services APIs are the application APIs provided to the customers to build their application. The services APIs feature:

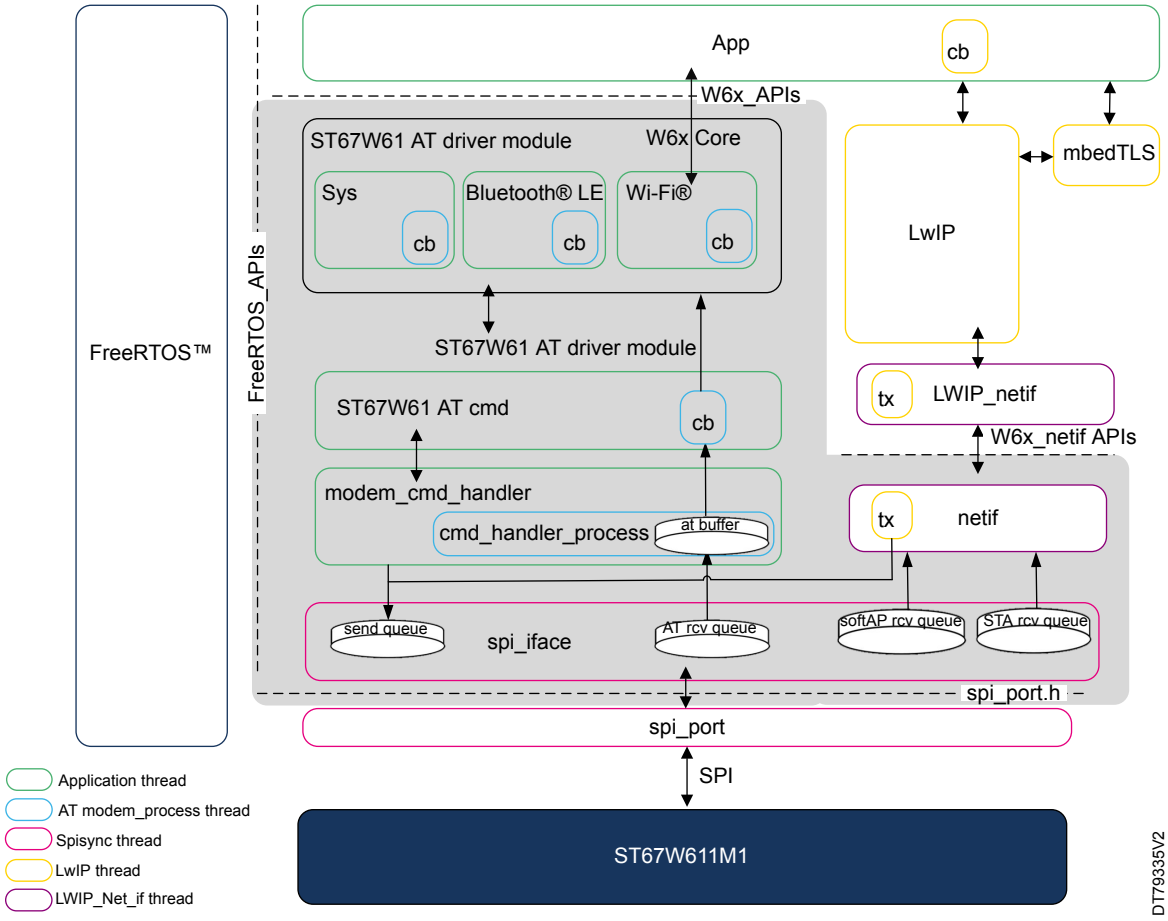
- System services, managing the central system functions of the network control processor (NCP). Key functionalities include:
  - Power management: controls power modes, sleep states, and energy-saving features
  - System information: provides access to system status, diagnostics, and hardware information
- ST67W611M1 firmware updates for OTA: manages firmware updates and system upgrades.
- Bluetooth® LE: manages Bluetooth® connections and data exchange. Key functionalities include:
  - Connection management: establishes and maintains Bluetooth® LE connections with other devices
  - Data transmission: sends data packets over Bluetooth® LE connections
  - Notifications: receives and processes Bluetooth® LE notifications from connected devices
  - Security: establishes secure Bluetooth® LE connections
- Wi-Fi® services: handle Wi-Fi® network control and management. Key functionalities include:
  - Network scanning: searches for available Wi-Fi® networks
  - Connection management: connects to and disconnects from Wi-Fi® networks
  - Signal strength monitoring: monitors and reports the strength of Wi-Fi® signals
- Net services manage the TCP/IP stack and route data across the network. Key functionalities include:
  - TCP/IP stack management: handles the core TCP/IP protocols for data communication
  - Data routing: routes TCP/IP data packets to and from the network
  - Network configuration: manages IP addresses, subnets, and other network settings
- MQTT (message queuing telemetry transport) services:
  - Publish/subscribe: manages MQTT topics and message exchanges
  - Broker communication: connects to and communicates with MQTT brokers

- HTTP (HyperText transfer protocol) services:
  - Request handling: manages HTTP HEAD, GET, POST, and PUT requests
  - Response processing: processes and interprets HTTP responses from servers

5.1.1.1.2 FreeRTOS™ LwIP on-host implementation view

The figure below shows the LwIP on-host architecture where LwIP is embedded in the host STM32.

Figure 18. ST67W61 middleware driver with LwIP aside: FreeRTOS™ view



- The `spi_iface` keeps the AT interface as above. The NET_IF interface binds to a new "receive queue" for the LwIP station and softAP network interfaces. The "send queue"'is used for both AT command and network data output.
- Refer to Section 5.1.1.1.1 for more information on the AT driver description

- Services APIs are adjusted as follows:
  - Refer to Section 5.1.1.1.1 for more information on the system services description
  - Refer to Section 5.1.1.1.1 for more information on the ST67W611M1 firmware updates for OTA.
  - Refer to Section 5.1.1.1.1 for more information on Bluetooth® LE services.
  - Refer to Section 5.1.1.1.1 for more information on Wi-Fi® services.
  - Net services are disabled. LwIP manages all TCP/IP services and other network services
  - MQTT services are disabled. MQTT has to be implemented on top of LwIP.
  - HTTP is disabled. HTTP has to be implemented on top of LwIP.
  - `NET_IF` is enabled. It can:
    - Notify the station and softAP link status (up or down)
    - Send network data to `spi_iface`
    - Receive network data from `spi_iface` dedicated queue (one queue for station interface and one queue for softAP interface)

Note: LwIP network data are sent/received directly to/from the `spi_iface` and are not encapsulated into AT command.

Further details and technical specifications are provided in Section 5.4: System services.

5.1.1.1.3 General requirements

- All applications are based on the embedded system RTOS (refer to Section 5.2.1: FreeRTOS™).
- Each protocol module API has a CLI command (refer to Section 5.2.2: CLI from shell).
- Each module uses the logging mechanism from FreeRTOS (refer to Section 5.2.3: Logging features).
- The application is easily portable across different STM32 devices.
- No secure channel (for example, safelink) is implemented.

5.2 Framework

5.2.1 FreeRTOS™

The projects and the middleware are based on FreeRTOS™.

Note: FreeRTOS is a trademark of Amazon in the United States and/or other countries.

For ThreadX porting, ThreadX to FreeRTOS™ port is available through the following link: threadx/utility/rtos_compatibility_layers/FreeRTOS.

Note: The stack size in ThreadX is defined in bytes while the stack size in FreeRTOS™ is defined in words.

5.2.2 CLI from shell

The shell includes the following features:

- Auto-completion
- Command history (down and up key)
- Right and left key navigation
- Backspace functionality
- Help and command-specific help
- Decentralized command implementation
- Colorized CLI

5.2.3 Logging features

The logging component provides four logging macros listed in increasing order of verbosity:

- LogError()
- LogWarn()
- LogInfo()
- LogDebug()

The logging component features:

- Differed logging. Message is built within the calling function and sent to a log queue. The log queue is consumed by an output task. This limit the real-time impact of the logging.
- Standard log levels (refer to FreeRTOS™ website)
- Add metadata to the log message such as timestamp, filename and line number, task name. Metadata inclusion is configurable.
- Memory consumption scales with needs (dynamic allocation of log buffer). The maximum size is configurable.
- Thread safe
- Output hardware choice left to application writer.

For example, LogError() is called only when an error occurs, making it the least verbose, whereas LogDebug() is called more frequently to provide debug-level information.

The application can output the logs on:

- UART hardware (same or other UART than CLI)
- ITM (dedicated Arm® debug port on SWO)

5.2.4 FreeRTOS™ low-power application

The low-power host supports FreeRTOS™ tickless idle mode to allow the STM32 to enter low-power mode anytime FreeRTOS™. LPTIM1 is used as a low-power timer while the system is in low-power mode to maintain the time. Minimum low power mode is configurable in *Appli/App/app_config.h*.

- The ST67W611M1 power save depends on the connection of the ST67W611M1:
  - In unconnected mode, the power save mode is activated by setting `W6X_SetPowerMode` to 1. This sets the ST67W611M1 in standby lowpower mode after each command sent to the ST67W611M1.
  - If the ST67W611M1 device is connected to an access point, the device enters in DTIM standby power-save mode by default (if `W6X_SetPowerMode` is set to 1).
  - If the ST67W611M1 device is connected to a Wi-Fi®-6 access point, the individual TWT power save can be entered. `W6X_SetPowerMode` must be set to 1 before configuring and starting TWT `W6X_WiFi_TWT_Setup`.

5.2.5 Firmware over-the-air support (FOTA)

There are two types of FOTA available:

- The FOTA application that updates the NUCLEO-U575ZI-Q and the ST67W611M1 using a FOTA header descriptor in JSON format
- The FOTA application that only updates the ST67W611M1 without using a FOTA header descriptor

When the FOTA option is enabled, a low-priority task is initiated and waits for an event to start the FOTA update process (timer, button push, or CLI command for example). Then, it fetches all the necessary resources to proceed with the update.

For the NUCLEO-U575ZI-Q FOTA variant, the header descriptor contains information such as the versions of the binaries and values for a binary integrity check. The NUCLEO-U575ZI-Q binary is stored in flash and if the integrity verification passes, it boots on the binary stored in flash on next reboot.

For the ST67W611M1 update, during download of the binary, the data is transferred to the ST67W611M1. When the transfer is done, the ST67W611M1 performs the necessary check and reboots on the new binary.

Pythons scripts are provided to help with FOTA header descriptor generation and run server hosting files needed by the FOTA in a local environment.

5.3 ST67W6X_Network_Driver configuration

The ST67W6X_Network_Driver is configured with default values. Each default values can be overridden in `w6x_config.h` and `w61_driver_config.h` .

The following sections list the configurations that can be overridden:

5.3.1 System

w6x_config_template.h

- W6X_POWER_SAVE_AUTO enables automatic low-power mode when ST67W611M1 is idle.
- W6X_CLOCK_MODE selects NCP clock source.
- W6X_ASSERT_ENABLE enables/disables NULL pointer checks in API functions.

w61_driver_config_template.h

- W61_ASSERT_ENABLE enables/disables NULL pointer checks in AT functions.
- SYS_LOG_ENABLE enables/disables system module logging.

5.3.2 Wi-Fi®

w6x_config_template.h

- W6X_WIFI_AUTOCONNECT enables/disables automatic Wi-Fi connection.
- W6X_WIFI_SAP_MAX_CONNECTED_STATIONS is the maximum number of stations for SoftAP.
- W6X_WIFI_COUNTRY_CODE is the Wi-Fi® region code.
- W6X_WIFI_ADAPTIVE_COUNTRY_CODE matches the AP country code or uses static code.

w61_driver_config_template.h

- W61_WIFI_MAX_DETECTED_AP is the maximum number of Wi-Fi® APs detected during scan.
- WIFI_LOG_ENABLE enables/disables Wi-Fi® module logging.
- W61_WIFI_TIMEOUT is the timeout value for remote Wi-Fi® device operations.

5.3.3 Bluetooth® LE

w6x_config_template.h

- W6X_BLE_HOSTNAME is the Bluetooth® LE device hostname.

w61_driver_config_template.h

- W61_BLE_MAX_DETECTED_PERIPHERAL is the maxumu number of Bluetooth® LE peripherals detected during scan.
- BLE_LOG_ENABLE enables/disables Bluetooth® LE module logging.
- W61_BLE_TIMEOUT is the timeout value for remote Bluetooth® LE device operations.
- W61_BLE_MAX_CONN_NBR is the maximum number of Bluetooth® LE connections supported by the application.

5.3.4 Net

w6x_config_template.h

- W6X_NET_DHCP configures is the DHCP configuration.
- W6X_NET_SAP_IP_SUBNET is SoftAP subnet definition.
- W6X_NET_HOSTNAME is the Wi-Fi® hostname.
- W6X_NET_RECV_TIMEOUT is the socket receive timeout.
- W6X_NET_SEND_TIMEOUT is the socket send timeout.
- W6X_NET_RECV_BUFFER_SIZE is the network socket receive buffer size.

w61_driver_config_template.h

- NET_LOG_ENABLE enables/disables network module logging.

- W61_NET_TIMEOUT is the timeout value for remote network operations.
- W61_NET_IPV6_ENABLE enables IPv6 support.

5.3.5 HTTP

w6x_config_template.h

- W6X_HTTP_CLIENT_THREAD_STACK_SIZE is the HTTP client thread stack size.
- W6X_HTTP_CLIENT_THREAD_PRIO is the HTTP client thread priority.
- W6X_HTTP_CLIENT_DATA_RECV_SIZE is the HTTP client data receive buffer size.
- W6X_HTTP_CLIENT_TCP_SOCK_RECV_TIMEOUT si the TCP socket receive timeout for HTTP client.
- W6X_HTTP_CLIENT_TCP_SOCKET_SIZE is the TCP socket size for HTTP client.

5.3.6 MQTT

w61_driver_config_template.h

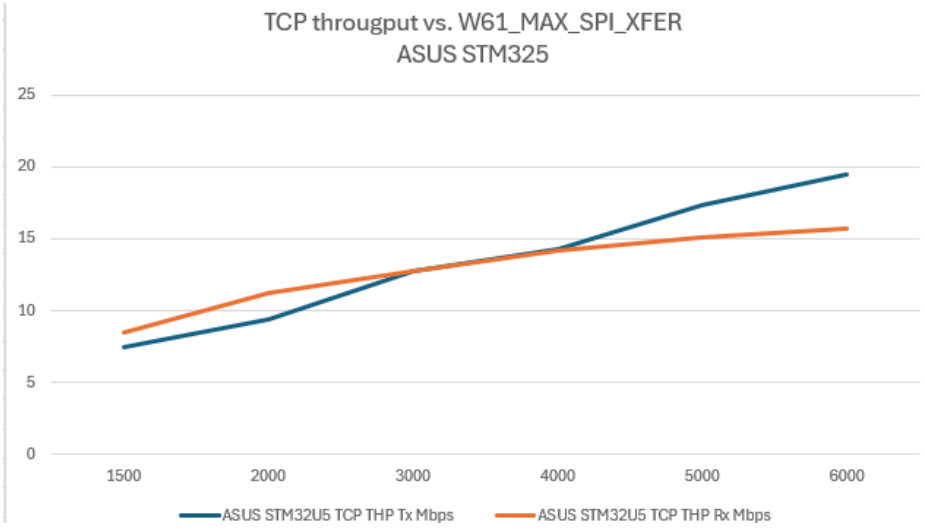
- MQTT_LOG_ENABLE enables/disables MQTT module logging.

5.3.7 AT common

w61_driver_config_template.h

- W61_MAX_SPI_XFER is the maximum SPI buffer size, which is a key parameter (between 1520 and 6000 bytes). W61_MAX_SPI_XFER can be used to tune the maximum bytes that can be sent in one AT command.
  - In case of LwIP on-host, the value must be set to 1520 bytes (just greater than MTU size). Setting this to a greater value would be a waste of dynamic memory.
  - In case of LwIP off load, the W61_MAX_SPI_XFER is almost proportional to performance: The greater the value, the better the performance. However, it is observed (at worst case) that dynamic allocation is five times the W61_MAX_SPI_XFER value (one application buffer + one AT buffer + one Tx buffer + two Rx buffers).

Figure 19. TCP throughput versus W61 MAX SPI



- SPI_THREAD_STACK_SIZE defines the stack size for SPI thread.
- SPI_THREAD_PRIO specifies the priority for SPI thread.
- W61_MAX_AT_LOG_LENGTH sets the maximum size of AT log.
- W61_AT_LOG_ENABLE enables/disables AT command logging.

- W61_NCP_TIMEOUT defines the timeout for NCP reply/execute.
- W61_SYS_TIMEOUT defines the timeout for special cases (flash write, OTA, and more).
- W61_MDM_RX_TASK_STACK_SIZE_BYTES defines the stack size for modem Rx task.
- W61_MDM_RX_TASK_PRIO specifies the priority for modem Rx task

5.3.8 Utility performance

w6x_config_template.h

- NET_RECVFROM, NET_RECV, NET_SENDTO, NET_SEND (etc...) maps generic network operations to W6X API functions.
- IPERF_ENABLE enables the Iperf feature and options.
- IPERF_TRAFFIC_TASK_PRIORITY, IPERF_TRAFFIC_TASK_STACK, IPERF_REPORT_TASK_STACK: configure iperf task settings.
- IPERF_MALLOC, IPERF_FREE provide memory allocation functions for iperf.
- MEM_PERF_ENABLE enables memory performance measurement.
- LEAKAGE_ARRAY specifies the number of memory allocations tracked.
- ALLOC_BREAK defines the maximum of iterations for the allocator.
- TASK_PERF_ENABLE enables task performance measurement.
- PERF_MAXTHREAD defines the maximum number of threads monitored.

5.3.9 WFA traffic generator

w6x_config_template.h

- WFA_TG_ENABLE enables Wi-Fi® Alliance traffic generator.

5.3.10 External service

w6x_config_template.h

- LFS_ENABLE enables LittleFS file system.

5.4 System services

System services manage the central system functions of the network control processor. Key functionalities include:

- Power management: Controls power modes, sleep states, and energy-saving features
- System information: Provides access to system status, diagnostics, and hardware information

5.4.1 System APIs functions

The system module proposes an API set of functions needed to control the chip. The main functions available are as follows:

Table 3. System APIs functions

Function	Description
W6X_Init	This function initializes the low-layer part of the W6X core.
W6X_DeInit	This function de-initializes the low-layer part of the W6X core.
W6X_RegisterAppCb	This function registers the upper-layer callbacks.
W6X_GetCbHandler	This function returns the W6X callback handler.
W6X_GetModuleInfo	This function gets the W6X module information.
W6X_ModuleInfoDisplay	This function diplays the W6X module information.

Function	Description
W6X_SetPowerMode	This function sets the power mode.
W6X_GetPowerMode	This function gets the power mode.
W6X_FS_WriteFileByName	This function writes a file from the host file system to the NCP file system.
W6X_FS_WriteFileByContent	This function writes a file from the local memory to the NCP file system.
W6X_FS_ReadFile	This function reads a file content from the NCP file system.
W6X_FS_DeleteFile	This function deletes a file content from the NCP file system.
W6X_FS_GetSizeFile	This function gets the size of a file available in the NCP file system.
W6X_FS_ListFiles	This function returns the list of all available files in file system (NCP and host if LFS is enabled).
W6X_Reset	This function resets the module.
W6X_ExeATCommand	This function executes AT command.
W6X_StatusToStr	This function converts the W6X status to a string.
W6X_ModelToStr	This function converts the W6X module ID to a string.

Note: The ST67W611M1 must exit low power before data traffic.

5.5 Firmware update services

The firmware update module manages ST67W611M1 firmware updates from the host. The host can get the firmware via the TCP/IP sockets, Bluetooth® LE, or any other means.

5.5.1 ST67W611M1 firmware update APIs functions

The firmware update module proposes an API set of functions needed to update the ST67W611M1. The main functions available are as follows:

Table 4. Firmware update APIs functions

Function	Description
W6X_FWU_Starts	This function starts the NCP firmware update.
W6X_FWU_Finish	This function finalizes the NCP firmware update, which, reboots the module to apply the new firmware, if the verification is successful.
W6X_FWU_Send	This function sends the firmware binary to the module.

5.6 Wi-Fi® services

5.6.1 Wi-Fi® APIs functions

The Wi-Fi® module offers an API set of functions necessary for controlling the chip. The main functions available are as follows:

Table 5. Wi-Fi® APIs function

Function	Description
W6X_WiFi_Init	This function initializes the Wi-Fi® module.
W6X_WiFi_DeInit	This function de-initializes the Wi-Fi® module.
W6X_WiFi_Scan	This function lists a defined number of available access points.
W6X_WiFi_PrintScan	This function prints the scan results.

Function	Description
W6X_WiFi_Connect	This function joins an access point.
W6X_WiFi_Disconnect	This function disconnects the device from a Wi-Fi® network.
W6X_WiFi_GetAutoConnect	This function retrieves auto-connect state.
W6X_WiFi_GetCountryCode	This function retrieves the country code configuration.
W6X_WiFi_SetCountryCode	This function sets the country code configuration.
W6X_WiFi_Station_Start	This function sets the module in station mode.
W6X_WiFi_Station_GetState	This function retrieves the Wi-Fi® station state.
W6X_WiFi_Station_GetMACAddress	This function retrieves the Wi-Fi® MAC address.
W6X_WiFi_AP_Start	This function configures a SoftAP.
W6X_WiFi_AP_Stop	This function stops a SoftAP.
W6X_WiFi_AP_GetConfig	This function gets the SoftAp configuration.
W6X_WiFi_AP_ListConnectedStations	This function lists the connected stations.
W6X_WiFi_AP_DisconnectStation	This function disconnects the station from the SoftAP.
W6X_WiFi_AP_GetMACAddress	This function retrieves the Wi-Fi® SoftAP MAC address.
W6X_WiFi_SetDTIM	This function sets the low-power Wi-Fi® DTIM (delivery traffic indication message).
W6X_WiFi_GetDTIM	This function gets the low-power Wi-Fi® DTIM.
W6X_WiFi_GetDTIM_AP	This function gets the low-power Wi-Fi® DTIM for the access point.
W6X_WiFi_TWT_Setup	This function sets up the target wake time (TWT) for the Wi-Fi® station.
W6X_WiFi_TWT_GetStatus	This function gets the TWT for the Wi-Fi® station.
W6X_WiFi_TWT_Teardown	This function teardowns the TWT for the Wi-Fi® station.
W6X_WiFi_GetAntennaDiversity	This function gets the antenna diversity information.
W6X_WiFi_SetAntennaDiversity	This function sets the antenna diversity information.
W6X_WiFi_StateToStr	This function converts the Wi-Fi® state to a string.
W6X_WiFi_SecurityToStr	This function converts the Wi-Fi® security type to a string.
W6X_WiFi_ReasonToStr	This function converts the Wi-Fi® reason code to a string.
W6X_WiFi_ProtocolToStr	This function converts the Wi-Fi® protocol to a string.
W6X_WiFi_AntDivToStr	This function converts the Wi-Fi® antenna mode to a string.

Note: The ST67W611M1 must exit low power before data traffic.

5.6.2 Wi-Fi® events

The following events are sent back to the application layer:

Table 6. Wi-Fi® events

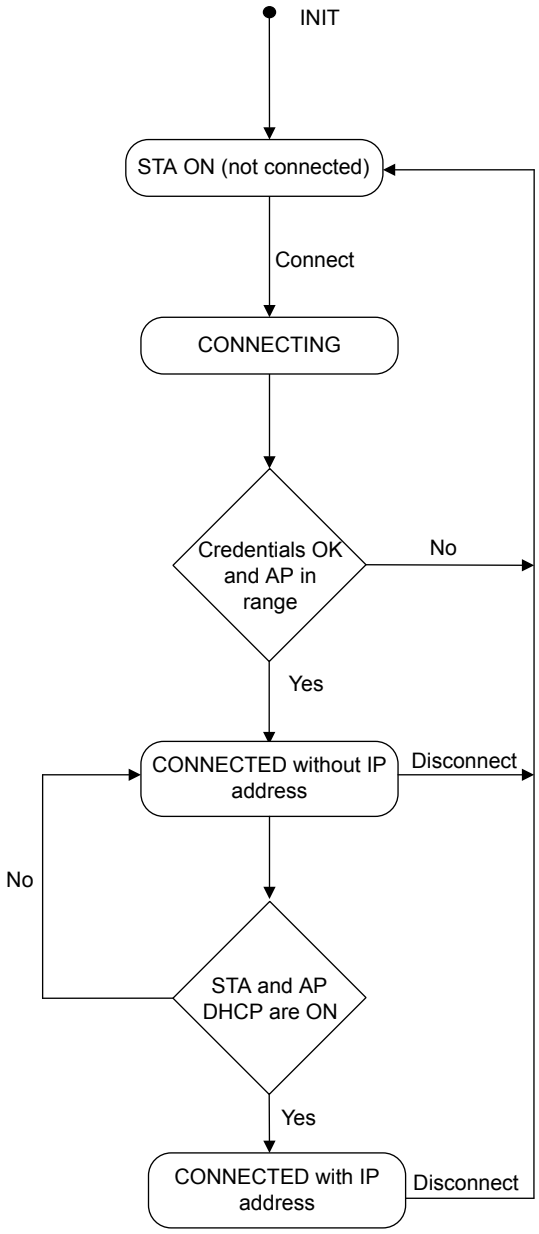
Event	Description
STA events	
W6X_WIFI_EVT_CONNECTED_ID	Event received when the station is connected to an access point.
W6X_WIFI_EVT_DISCONNECTED_ID	Event received when the station is disconnected to an access point.

Event	Description
W6X_WIFI_EVT_GOT_IP_ID	Event received when the station receives an IP address after a successful connection to an access point.
W6X_WIFI_EVT_CONNECTING_ID	Event received when the station is in the connection process to an access point.
W6X_WIFI_EVT_REASON_ID	Event received when the device encounters an issue in the connection process or at disconnection.
SoftAP events	
W6X_WIFI_EVT_STA_CONNECTED_ID	Event received when a station is connected to the SoftAP.
W6X_WIFI_EVT_STA_DISCONNECTED_ID	Event received when a station is disconnected from the SoftAP.
W6X_WIFI_EVT_DIST_STA_IP_ID	Event received when a station, connected to the SoftAP, receives an IP address.

5.6.3 Wi-Fi® finite state machine (FSM)

Below is the Wi-Fi® FSM for station mode only that describes the different states of the system:

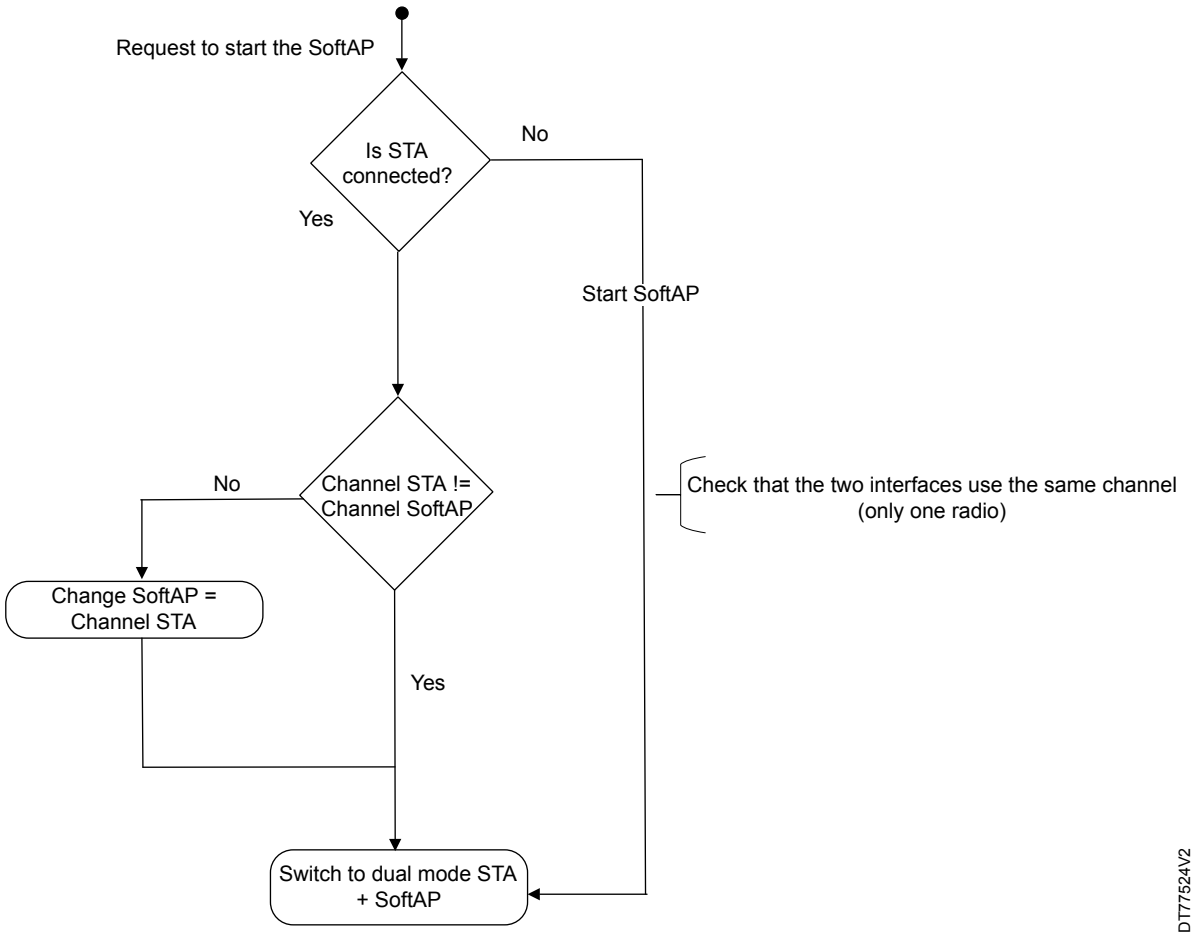
Figure 20. **Wi-Fi® station (STA) FSM**



The second chart below shows the behavior of the system when the station and the SoftAP modes are enabled (the station mode is active by default and cannot be deactivated).

When both interfaces are enabled, some verifications must be done by the user to ensure STA and SoftAP do not use the same subnet IP address to avoid unexpected behaviors. The channel they are operating on is automatically aligned between the STA and the SoftAP.

Figure 21. **Wi-Fi® STA + SoftAP behavior chart**



When the SoftAP is stopped, all the connected stations are disconnected from the SoftAP.

5.7 Net services

The Net module implements all the API related to the internet protocol. It supports the BSD socket API and some specific services that are commonly used, such as ping or SNTP services.

Up to five sockets can be managed in parallel.

Note: When LwIP runs on-host, the Net module is disabled, and socket API services are rendered by LwIP.

5.7.1 Net APIs functions

The Net module offers an API set of functions needed to control TCP/IP stack features. The main functions available are as follows:

Table 7. **Net APIs functions**

Function	Description
`W6X_Net_Init`	This function initializes the Net context with its callbacks and launches a sequence of commands to properly initialize the device.
`W6X_Net_DeInit`	This function de-initializes the Net module.
`W6X_Net_SetHostname`	This function sets the IP address from URL using DNS.
`W6X_Net_GetHostAddress`	This function gets the IP address from URL using DNS.
`W6X_Net_Station_GetIPAddress`	This function gets the Wi-Fi® station interface IP address

Function	Description
W6X_Net_Station_SetIPAddress	This function sets the Wi-Fi® station interface IP address
W6X_Net_AP_GetIPAddress	This function gets the SoftAP IP addresses.
W6X_Net_AP_SetIPAddress	This function sets the SoftAP IP addresses.
W6X_Net_GetDhcp	This function gets the DHCP configuration.
W6X_Net_SetDhcp	This function sets the DHCP configuration.
W6X_Net_GetDnsAddress	This function gets the Wi-Fi® DNP addresses.
W6X_Net_SetDnsAddress	This function sets the Wi-Fi® DNP addresses.
W6X_Net_Ping	This function pings an IP address in the network.
W6X_Net_ResolveHostAddress	This function gets the IP address from URL using DNS.
W6X_Net_Station_GetIPv6Address	This function gets the Wi-Fi® station interface IPv6 addresses (link-local and global).
W6X_Net_ResolveHostAddressByType	This function resolves a hostname to an IP (IPv4 or IPv6).
W6X_Net_Inet6_aton	This function converts an IPv6 text representation to binary (struct in6_addr) via driver aton.
W6X_Net_SNTP_GetConfiguration	This function gets the current SNTP status, time zone, and servers.
W6X_Net_SNTP_SetConfiguration	This function sets the current SNTP status, time zone, and servers.
W6X_Net_SNTP_GetInterval	This function gets the SNTP synchronization interval.
W6X_Net_SNTP_SetInterval	This function sets the SNTP synchronization interval.
W6X_Net_SNTP_GetTime	This function retrieves the query date string from the SNTP server, using the asctime-style time format.
W6X_Net_GetConnectionStatus	This function gets information for an opened socket.
W6X_Net_Socket	This function checks if an instance socket is available.
W6X_Net_Close	This function can be used to close a socket instance and release the associated resources.
W6X_Net_Shutdown	This function shutdowns a socket instance.
W6X_Net_Bind	This function binds a socket instance to a specific address.
W6X_Net_Connect	This function connects a socket instance to a specific address.
W6X_Net_Listen	This function listens for incoming connections on a socket.
W6X_Net_Accept	This function accepts an incoming connection on a socket.
W6X_Net_Send	This function sends data on a socket.
W6X_Net_Recv	This function receives data from a socket.
W6X_Net_Sendto	This function sends data on a socket to a specific address.
W6X_Net_Recvfrom	This function receives data on a socket to a specific address.
W6X_Net_Getsockopt	This function gets a socket option.
W6X_Net_Setsockopt	This function sets a socket option.
W6X_Net_TLS_Credential_AddByContent	This function adds the credential by local file content to the TLS context.
W6X_Net_TLS_Credential_AddByName	This function adds the credential by name from host LFS to the TLS context.
W6X_Net_TLS_Credential_Delete	This function deletes the credential from the TLS context.
W6X_Net_Inet_ntop	This function converts an IP address from uint32_t format to text.

Function	Description
W6X_Net_Inet_pton	This function converts an IP address from text format to uint32_t.

5.7.2 NET events

The following NET events are sent back to the application layer:

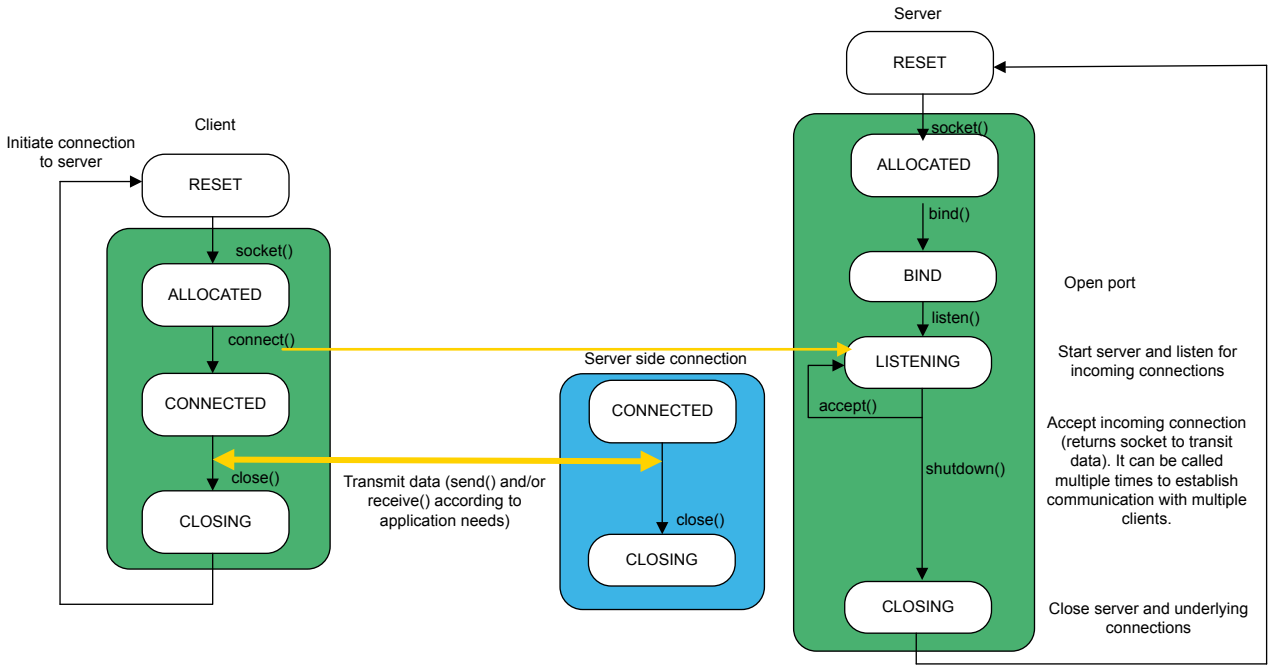
Table 8. NET events

Event	Description
W6X_NET_EVT_SOCK_DATA_ID	Event sent when data is received on a socket.

5.7.3 NET FSM

The following figure shows the FSM for both TCP client and server socket.

Figure 22. TCP sockets FSM

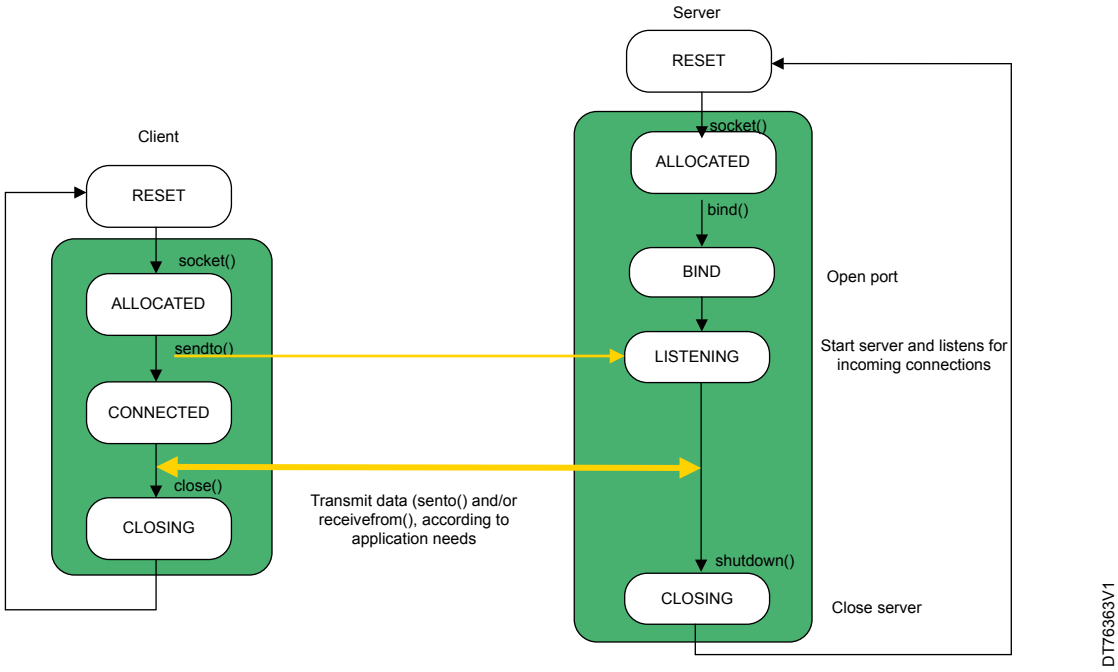


There are six possible states:

- RESET: Default state
- ALLOCATED: Available socket has been reserved
- CONNECTED: Connection is established, the socket is ready to exchange data. It can be as a client, or as a server (for incoming connections)
- BIND: Socket is configured for server mode with the port and protocol set, but the server is not yet running.
- LISTENING: Server is running, ready to catch incoming connections
- CLOSING: Transitory state during close or shutdown operations. Once these operations are completed, the state is reset to RESET

Since UDP is a connectionless protocol, UDP sockets use a slightly different state machine. The state CONNECTED is used after the first call to `sendto()` to indicate that incoming data can be expected, as communication with a server has been initiated:

Figure 23. UDP sockets FSM



5.8 Bluetooth® LE services

5.8.1 Bluetooth® LE APIs functions

The Bluetooth® LE module offers an API set of functions needed to control the chip. The main functions available are as follows:

Table 9. Bluetooth® LE APIs functions

Function	Description
`W6X_Ble_GetInitMode`	This function gets the Bluetooth® LE initialization mode.
`W6X_Ble_Init`	This function initializes the Bluetooth® LE server/client/dual mode.
`W6X_Ble_DeInit`	This function de-initializes Bluetooth® server/client/dual mode.
`W6X_Ble_SetRecvDataPtr`	This function sets/changes the pointer where to copy the received data.
`W6X_Ble_SetTxPower`	This function configures the Bluetooth® LE Tx power.
`W6X_Ble_GetTxPower`	This function gets the Bluetooth® LE Tx power.
`W6X_Ble_AdvStart`	This function starts the advertising.
`W6X_Ble_AdvStop`	This function stops the advertising.
`W6X_Ble_GetBDAddress`	This function gets the Bluetooth® LE device address.
`W6X_Ble_Disconnect`	This function disconnects from a Bluetooth® LE remote device.
`W6X_Ble_ExchangeMTU`	This function exchanges Bluetooth® LE maximum transmission unit (MTU) length.

Function	Description
W6X_Ble_SetBdAddress	This function sets the Bluetooth® LE BD address.
W6X_Ble_SetDeviceName	This function sets the Bluetooth® LE device name.
W6X_Ble_GetDeviceName	This function gets the Bluetooth® LE device name.
W6X_Ble_SetAdvData	This function sets the Bluetooth® LE advertising data.
W6X_Ble_SetScanRespData	This function sets the scan response data.
W6X_Ble_SetAdvParam	This function sets the advertising parameters.
W6X_Ble_StartScan	This function starts the Bluetooth® LE scan.
W6X_Ble_StopScan	This function stops the Bluetooth® LE scan.
W6X_Ble_SetScanParam	This function sets the scan parameters.
W6X_Ble_GetScanParam	This function gets the scan parameters.
W6X_Ble_Print_Scan	This function prints the Bluetooth® LE scan results.
W6X_Ble_GetAdvParam	This function gets the Bluetooth® LE advertising parameters.
W6X_Ble_SetConnParam	This function sets the connection parameters.
W6X_Ble_GetConnParam	This function gets the connection parameters.
W6X_Ble_GetConn	This function gets Bluetooth® LE connection information.
W6X_Ble_Connect	This function initiates a connection to a Bluetooth® LE remote device.
W6X_Ble_SetDataLength	This function sets the Bluetooth® LE data length.
W6X_Ble_CreateService	This function allows the creation of a new Bluetooth® LE service.
W6X_Ble_DeleteService	This function allows to delete a new Bluetooth® LE.
W6X_Ble_CreateCharacteristic	This function allows to create a new Bluetooth® LE characteristic.
W6X_Ble_GetServicesAndCharacteristics	This function gets the created services and characteristics.
W6X_Ble_RegisterCharacteristics	This function allows to register Bluetooth® LE services.
W6X_Ble_RemoteServiceDiscovery	This function allows to discover Bluetooth® LE services of the remote connected device.
W6X_Ble_RemoteCharDiscovery	This function allows to discover Bluetooth® LE characteristics of a remote connected device service.
W6X_Ble_ServerNotify	This function notifies a characteristic value from the server to the remote client.(1)
W6X_Ble_ServerIndicate	This function indicates a characteristic value from the server to the remote client.(2)
W6X_Ble_ServerSetReadData	This function sets data that can be read by the remote client.
W6X_Ble_ClientWriteData	This function allows to write data to a server characteristic.
W6X_Ble_ClientReadData	This function allows to read data from a server characteristic.
W6X_Ble_ClientSubscribeChar	This function allows to subscribe to notifications or indications from a server characteristic.
W6X_Ble_ClientUnsubscribeChar	This function allows to unsubscribe to notifications or indications from a server characteristic.
W6X_Ble_SetSecurityParam	This function sets the Bluetooth® LE security parameters
W6X_Ble_GetSecurityParam	This function gets the Bluetooth® LE security parameters
W6X_Ble_SecurityStart	This function starts the Bluetooth® LE security.

Function	Description
W6X_Ble_SecurityPassKeyConfirm	This function allows to confirm the Bluetooth® LE security pass key.
W6X_Ble_SecurityPairingConfirm	This function allows to confirm the Bluetooth® LE pairing.
W6X_Ble_SecuritySetPassKey	This function allows to set a Bluetooth® LE security pass key.
W6X_Ble_SecurityPairingCancel	This function allows to cancel the Bluetooth® LE pairing process.
W6X_Ble_SecurityUnpair	This function allows to unpair to from a Bluetooth® LE device.
W6X_Ble_SecurityGetBondedDeviceList	This function returns a list of Bluetooth® LE paired devices.

1. *W6X_Ble_ServerSendNotification is deprecated. It is still maintained in w6x_legacy.h but is planned for removal in future releases.*
2. *W6X_Ble_ServerSendIndication is deprecated. It is still maintained in w6x_legacy.h but is planned for removal in future releases.*

5.8.2 Bluetooth® LE events

The following Bluetooth® LE events are sent back to the application layer:

Table 10. Bluetooth® LE events

Event	Description
W6X_BLE_EVT_CONNECTED_ID	Bluetooth® LE connection established.
W6X_BLE_EVT_DISCONNECTED_ID	Bluetooth® LE connection ends.
W6X_BLE_EVT_CONNECTION_PARAM_ID	Bluetooth® LE connection parameters update.
W6X_BLE_EVT_READ_ID	Read operation from Bluetooth® LE remote device.
W6X_BLE_EVT_WRITE_ID	Write operation from Bluetooth® LE remote device.
W6X_BLE_EVT_SERVICE_FOUND_ID	Bluetooth® LE service discovered on remote device.
W6X_BLE_EVT_CHAR_FOUND_ID	Bluetooth® LE characteristic discovered on remote device.
W6X_BLE_EVT_INDICATION_STATUS_ENABLED_ID	Bluetooth® LE indication enabled.
W6X_BLE_EVT_INDICATION_STATUS_DISABLED_ID	Bluetooth® LE indication disabled.
W6X_BLE_EVT_NOTIFICATION_STATUS_ENABLED_ID	Bluetooth® LE notification enabled.
W6X_BLE_EVT_NOTIFICATION_STATUS_DISABLED_ID	Bluetooth® LE notification disabled.
W6X_BLE_EVT_INDICATION_ACK_ID	Bluetooth® LE indication acknowledged.
W6X_BLE_EVT_INDICATION_NACK_ID	Bluetooth® LE indication unacknowledged.
W6X_BLE_EVT_NOTIFICATION_DATA_ID	Bluetooth® LE service receives notification/indication data.
W6X_BLE_EVT_MTU_SIZE_ID	Bluetooth® LE MTU size update.
W6X_BLE_EVT_PAIRING_FAILED_ID	Bluetooth® LE pairing failed.
W6X_BLE_EVT_PAIRING_COMPLETED_ID	Bluetooth® LE pairing completed.
W6X_BLE_EVT_PAIRING_CONFIRM_ID	Bluetooth® LE pairing confirmation notification.
W6X_BLE_EVT_PASSKEY_ENTRY_ID	Bluetooth® LE pairing passkey entry notification.
W6X_BLE_EVT_PASSKEY_DISPLAY_ID	Bluetooth® LE pairing passkey display notification.
W6X_BLE_EVT_PASSKEY_CONFIRM_ID	Bluetooth® LE pairing passkey confirmation notification.

5.9 HTTP services

The HTTP module offers HTTP requests and processes and interprets HTTP responses from servers.

It supports both HTTP and HTTPS with HEAD, GET, PUT and POST methods.
For POST and PUT requests, the following content types can be used:

- Text/plain: Regular text without any formatting
- Application/x-www-form-urlencoded: Data is given as key value pairs (key1 = value1 and key2 = value2...)
- Application/json: JSON format
- Application/xml: XML format
- Application/octet-stream: Binary data

Note: When LwIP is on-host, the HTTP module is disabled, and HTTP services should be built using LwIP.

5.9.1 HTTP APIs functions

The HTTP module proposes an API function to create HTTP client requests.

Table 11. HTTP APIs functions

Function	Description
W6X_HTTP_Client_Request	HTTP client request based on BSD socket.

5.10 MQTT services

Note: When LwIP is on-host, the MQTT module is disabled, and MQTT services should be built using LwIP.

5.10.1 MQTT APIs functions

The MQTT module offers an API set of functions needed to control the chip. The main functions available are as follows:

Table 12. MQTT APIs functions

Function	Description
W6X_MQTT_Init	This function initializes the MQTT module.
W6X_MQTT_DeInit	This function de-initializes the MQTT module.
W6X_MQTT_SetRecvDataPt	This function sets/changes the pointer where to copy the received data.
W6X_MQTT_Configure	This function configures the user configuration for MQTT broker requirements.
W6X_MQTT_Connect	This function connects to the MQTT broker.
W6X_MQTT_GetConnectionStatus	This function gets connection status of the MQTT broker.
W6X_MQTT_Disconnect	This function disconnects from the MQTT broker.
W6X_MQTT_Subscribe	This function subscribes to an MQTT topic.
W6X_MQTT_GetSubscribedTopics	This function gets the list of subscribed MQTT topics.
W6X_MQTT_Unsubscribe	This function unsubscribes the client from defined topic.
W6X_MQTT_Publish	This function publishes the MQTT message into a topic.
W6X_MQTT_StateToStr	This function converts the MQTT connection status into a string.

5.10.2 MQTT events

The following MQTT events are sent back to the application layer:

Table 13. MQTT events

Event	Description
W6X_MQTT_EVT_CONNECTED_ID	Event received when the MQTT client is connected to the broker.

Event	Description
W6X_MQTT_EVT_DISCONNECTED_ID	Event received when the MQTT client is disconnected from the broker.
W6X_MQTT_EVT_SUBSCRIPTION_RECEIVED_ID	Event received when the MQTT client receives data from a topic it has subscribed to.

5.11 Netif service

5.11.1 Netif APIs functions

The network interface (Netif) module implements all the API functions related to the direct network link between LwIP on-host and Wi-Fi® stack in the ST67W611M1 device. There is a link for the station mode only (SoftAP is not available yet). These services are only available with the ST67 T02 architecture. The main functions available are as follows:

Table 14. Netif APIs functions

Function	Description
W6X_Netif_Init	This function initializes the network interface.
W6X_Netif_DeInit	This function de-initializes the network interface.
W6X_Netif_output	This function sends data on the network interface.
W6X_Netif_input	This function reads data from the network interface.
W6X_Netif_free	This function frees the internal buffer containing the data.

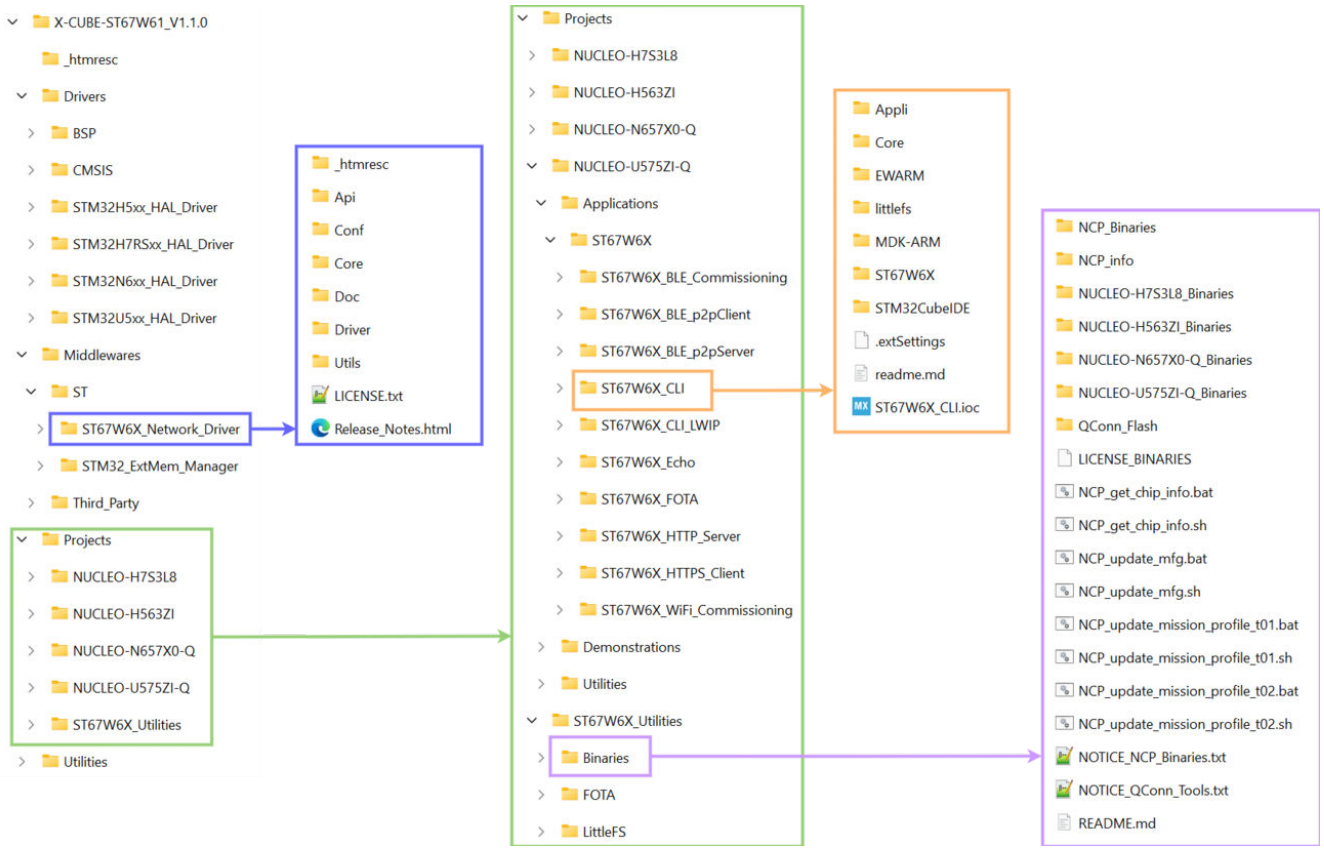
6 Package description

This section details the content of the X-CUBE-ST67W61 Expansion Package and how to use it.

6.1 Project directory

After downloading and decompressing X-CUBE-ST67W61_Vx.y.z.zip (decompress to C:/), the directory should appear as shown below.

Figure 24. X-CUBE-ST67W61 Expansion Package



The X-CUBE-ST67W61 Expansion Package contains all necessary source code and projects to generate the binaries running on the STM32 host (NUCLEO-U575ZI-Q). The ST67W6X_CLI and ST67W6X_Echo projects are also available for the NUCLEO-H7S3L8, NUCLEO-H563ZI and NUCLEO-N657X0-Q STM32 hosts. It also contains the binaries to be loaded into the ST67W611M1 module.

6.2 Requirements

6.2.1 Integrated development environment (IDE) requirements

The software requirements are the following (minimum IDEs version):

- IAR Embedded Workbench for Arm® (EWARM) toolchain V9.30.1
- STM32CubeIDE toolchain V1.18.1
- RealView Microcontroller Development Kit (MDK-ARM) toolchain V5.39

6.2.2 STM32CubeProgrammer

The STM32CubeProgrammer (STM32CubeProg) is utilized to upload the pre-generated binary file onto the board.

6.2.3 Terminal

The applications console and logs interface communicate through the serial link. To access this interface, open a terminal client connected to the host ST-LINK COM port.

The serial COM port must be configured as below:

Table 15. Serial COM port configuration

Serial configuration parameter	Value
Baudrate	921600
Data	8b
Stopbit	1b
Parity	None
Flow control	None
Rx	LF
Tx	LF
Local Echo	Off

6.2.4 Hardware requirements

This example runs on the NUCLEO-U575ZI-Q board combined with the X-NUCLEO-67W61M1 board.

The X-NUCLEO-67W61M1 board is plugged to the NUCLEO-U575ZI-Q board via the ARDUINO® connectors:

- The 5V, 3V3, GND through the CN6
- The SPI (SCLK, MOSI, MISO) and WAKE_UP signals through the CN5
- The BOOT, CHIP_EN, DATA_RDY, and Tx/Rx signals through the CN9

Specific actions must be taken before programming the STM32H7Rx/7Sx and the STM32N6 MCUs:

- Read the following STM32H7Rx/7Sx wiki article and specifically the "**Option bytes programming**" paragraph: https://wiki.st.com/stm32mcu/wiki/Security:How_to_start_with_DA_access_on_STM32H7RS
- Read the following wiki paragraphs before programming the STM32N6:
  - https://wiki.st.com/stm32mcu/wiki/Security:How_to_install_a_trusted_application_on_STM32N6_MCUs#Encrypt--sign_and_load_the_FSBL
  - https://wiki.st.com/stm32mcu/wiki/Security:How_to_install_a_trusted_application_on_STM32N6_MCUs#Execute_the_FSBL_2

6.3 Build and load a project

Applications are delivered with both STM32CubeIDE and EWARM project files.

6.3.1 STM32CubeIDE

- Open the STM32CubeIDE project file.
- Right click on the project, and *Rebuild all*

6.3.2 EWARM

- Open the EWARM project file.
- Right click on the project, and *Rebuild all*

6.3.3 MDK-ARM

- Open the MDK-ARM project file.
- Right click on the project, and *Rebuild all*

6.4 Echo application

6.4.1 Project overview

This application aims to demonstrate the TCP echo feature over Wi-Fi®.

The project has been used as starting point for developing most of other projects.

The application starts in STA mode and connects to a Local_AP (gateway/hotspot/and more) available in the range. The credential (SSID and password) of the Local_AP have to be configured by the user in the file *Appli/App/app_config.h*.

Once the device is connected, it pings http://www.google.com by default. Then, it opens a TCP socket and it echoes the http://tcpbin.com server by sending a message and getting it back. The results of the echo request are displayed via UART on the terminal.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_Echo_Application

6.5 CLI application

6.5.1 Project overview

This application is designed to evaluate and test the ST67W611M1 Wi-Fi® solution via a command-line interface (CLI).

As mentioned in the X-CUBE-ST67W61 architecture, two different of architectures are delivered. Two versions of the CLI are delivered:

- The legacy ST67W611_CLI requiring the `st67w611m_mission_t01_v2.x.y.bin` to be loaded in the ST67W611M1 device and the new ST67W611_CLI_LWIP (with LwIP on host) requiring the `st67w611m_mission_t02_v2.x.y.bin` to be loaded in the ST67W611M1 device.

The CLI application allows users to perform various basic Wi-Fi® operations such as scanning for available local access points (AP) and connecting to an AP. Additionally, it enables testing of LwIP functionalities, including DHCP, pinging a URL or a specific IP address, MQTT, and iPerf.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_CLI_Application

6.6 Bluetooth® LE commissioning application

6.6.1 Project overview

This application aims to demonstrate how to provision Wi-Fi® credentials via Bluetooth® LE to establish a Wi-Fi® connection to an access point.

The solution is using a vendor-specific, a dedicated Bluetooth® LE profile, and a remote client interface (ST web Bluetooth®, or Android™/iOS™ ST BLE Toolbox Application).

It also provides an example on how to update over the air the ST67W611M1 firmware via Bluetooth® LE.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:ST67W6X_BLE_Commissioning_Application

6.7 Bluetooth® LE P2P application

6.7.1 Project overview

Peer-to-peer server and client applications aim to demonstrate Bluetooth® LE direct connection between two devices, based on STMicroelectronics proprietary service and characteristics. These two projects are provided within the ST67W611M1 Firmware Package.

The client device interfacing with the peer-to-peer server can be a ST67W611M1 platform with peer-to-peer client application, a smartphone using the ST BLE Toolbox Android™/iOS™ application, or a laptop with a Bluetooth® interface webpage.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:ST67W6X_BLE_Peer_To_Peer

6.8 HTTP server application

6.8.1 Project overview

This application gives an example of an HTTP server over Wi-Fi®, hosted by a device configured as a soft access point (SoftAP).

It gives a starting point for an HTTP server solution with basic GET requests to allow the client to interact with the server.

The application starts in SoftAP mode, with its configuration SSID (and password) in the file *Appli/App/app_config.h* to which a connection must be made (WPA2) and displays its IP address. Once the device is connected, open a preferred web browser, and enter the IP address of the server/SoftAP. The client sends a request to the server, which returns the main HTML page. This page includes the different buttons to interact with the LEDs on the board and a status of the USER button on the host board. HTML pages are hosted on the STM32 host device

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_HTTP_Server_Application

6.9 HTTPS client application

6.9.1 Project overview

This example illustrates an HTTPS client over Wi-Fi, with the client being a station (STA) connected to the internet. It provides a starting point for an HTTP or HTTPS client that downloads a file and processes the received data.

The application starts in station mode and connects to an AP using predefined credentials. Once connected, it performs a DNS resolution of the URL used for the HTTPS GET request, which is a weather API.

The client then performs the weather request using a hardcoded latitude and longitude (around Montreal) and extracts the temperature from the response. Finally, the CLI mode (see Section 6.5: CLI application) starts with an additional command: "weather" which allows retrieving the current temperature in one of the registered cities.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_HTTPS_Client_Application

6.10 FOTA application

6.10.1 Project overview

This application aims to demonstrate the FOTA feature over Wi-Fi®.

The application starts in STA mode and connects to an AP (gateway/hotspot/and more) available in the range. The credentials (SSID and password) of the AP must be configured by the user in the file *Appli/App/app_config.h*.

Once the device is connected, it creates and runs the FOTA task, awaiting an event trigger to start the FOTA process. FOTA options can be configured in the file *Appli/App/fota.h*.

By default, a timer is used to trigger the FOTA process. Additionally, an user button can also be used in this example to trigger the FOTA process.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_FOTA_Application

6.11 MQTT demonstration

These applications demonstrate an example of MQTT client over Wi-Fi®, connecting to an MQTT broker to publish telemetry data and receive parameter updates or commands from the cloud.

Two MQTT demonstrations are available in the project directory:

Table 16. **MQTT demonstrations**

Project name	Description
ST67W6X_MQTT	It exercises the network coprocessor (ST67W611M1) capabilities embedding the MQTT client, LwIP, and Wi-Fi® stack. This demonstration is developed for the NUCLEO-U575ZI-Q host board.
ST67W6X_MQTT_LWIP	It exercises the network coprocessor (ST67W611M1) capabilities embedding the Wi-Fi® stack. The MQTT client and LwIP are offloaded and present in the host code. This demonstration is developed for the NUCLEO-H563ZI host board.

These applications periodically publish data to an MQTT broker (this example uses EMQX, but it can be changed to a preferred MQTT broker). A subscription to the same topic is also done to monitor the published data.

The MQTT uses the TCP protocol for data transmission and can be encrypted using TLS.

In the applications, the subscribed topic can be used to receive additional control fields. Two actions are predefined:

- Change the LED state on the host board
- Reboot the host

To monitor the data sent and received to and from the topics, an MQTT client can be installed on a PC, smartphone, or tablet. By subscribing to the configured topics, the client receives notifications each time the device publishes data to those topics.

6.12 Wi-Fi® Commissioning application

6.12.1 Project overview

This application gives an example of a Wi-Fi® Commissioning project, using an HTTP server over Wi-Fi®, hosted by a device configured as a soft access point (SoftAP) and station (STA).

It provides a starting point for an HTTP server solution with basic requests to allow the client to interact with the server.

The application starts in SoftAP mode, with a set SSID and password for WPA2 connection, and then shows its IP address. Once the device is connected, open a preferred web browser, and enter the IP address of the server/ SoftAP. The client sends a request to the server which returns the main HTML page. This page features input fields to enter the SSID and the password of the AP for connecting the STA.

If the target AP operates on a different channel than the SoftAP, the SoftAP adjusts to match that channel. To achieve this, connected stations are disconnected, and the SoftAP is stopped and then restarted after establishing the STA connection.

Once the STA is connected, a ping button becomes clickable to send a ping to the default gateway of the STA.

The description of the application is available on the following wiki page: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_ST67W6X_Wi-Fi_Commissioning_Application

7 How to measure Wi-Fi® throughput

The CLI application described in Section 6.5: CLI application must be loaded onto the STM32 host.

7.1 Overview

iPerf is a widely used network testing tool that can create TCP and UDP data streams and measure the throughput of a network. It is particularly useful for diagnosing network issues and testing the performance (throughput) of a network.

7.2 iPerf installation

7.2.1 Linux®

On Linux®, iPerf should be available in the native package management system.

Note: Linux® is a registered trademark of Linus Torvalds.

7.2.2 Debian®

apt install iperf

Note: Debian is a registered trademark of Software in the Public Interest, Inc.

7.2.3 Red Hat®

yum install iperf

Note: Red Hat® is a registered trademark of Red Hat, Inc.

7.2.4 Windows®

To obtain binaries of iPerf for Windows®, it is recommended to visit one of the following websites. No installation is needed, once downloaded and unzipped, user just has to navigate to the extracted folder, open command prompt and launch iPerf commands.

- https://files.budman.pw/
- https://github.com/ar51an/iperf3-win-builds

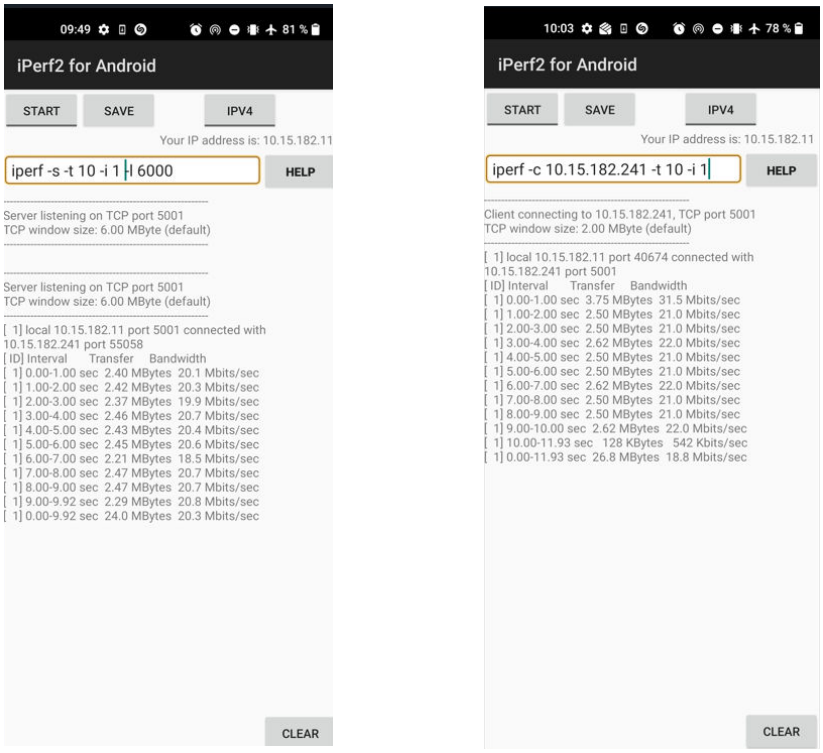
Note: Windows is a trademark of the Microsoft group of companies.

7.2.5 iPerf on Android™ smartphone

Testing the performance of our devices against smartphones is another possibility. Using the iPerf2 for Android application is recommended. Although it is no longer available on the Play Store, an unofficial version can be found at this link: https://apkpure.com/iperf2-for-android/iperf.project?_sm_nck=1.

To connect the two devices, either set up a Wi-Fi® hotspot on the smartphone or configure SoftAP on our device. Once connected, iPerf can be used on both devices as illustrated below:

Figure 25. *Iperf2 for Android* screenshot



DT76374V3

Note: Android is a trademark of Google LLC.

7.3 iPerf usage

7.3.1 Features selection

Key features include:

- TCP and UDP testing: iPerf can test both TCP and UDP protocols (and SCTP since iPerf 3.1)
- Server (Rx) and client (Tx) mode: iPerf operates in a client-server mode, where one device acts as the server and another as the client
- Multiple connections: Supports multiple simultaneous connections to test the network under load
- Bandwidth measurement
- Bidirectional testing
- IPv4 and IPv6 support

For detailed documentation, visit the iPerf 2.0, iPerf 3.0 and iPerf 3.1 documentation.

7.3.2 Main iperf command

For the test scenarios outlined in the table below, an access point with a Linux®-based operating system or any Windows®/Linux® machine with iPerf installed might be used. To view the details and limitations of our implementation of iPerf, use the command `iperf -h` on the STM32 host CLI.

Note: The iPerf implementation in the CLI has additional limitations:

- *It does not support simultaneous connections.*
- *It is compatible with iPerf 2.0 and 2.1 (other versions may cause erroneous reports).*
- *There is no IPv6 support yet.*

Table 17. Main iperf commands

Test	AP command	ST67W6X CLI command
UDP Tx	iperf -s -u -t 10 -i 1 -f m	iperf -c <AP_IP> -u -b 50M -t 10
UDP Rx	iperf -c <DUT_IP> -u -b 50M -t 10 -i 1 -f m	iperf -s -u -t 10
TCP Tx	iperf -s -t 10 -i 1 -f m	iperf -c <AP_IP> -t 10
TCP Rx	iperf -c <DUT_IP> -t 10 -i 1 -f m	iperf -s -t 10

Legend:

```
-s server
-c client
-u UDP mode (default TCP)
-t time of the test in seconds
-i reporting intervals in seconds
-f reporting format ('k'=Kbits/sec; 'K' = KBytes/sec; 'm' = Mbits/sec; 'M' = MBytes/sec)
-b bandwidth (desired client bandwidth - UDP only! TCP manages this automatically)
```

8 X-CUBE-ST67W61 STM32CubeMX pack

This section briefly introduces the X-CUBE-ST67W61 pack using STM32CubeMX.

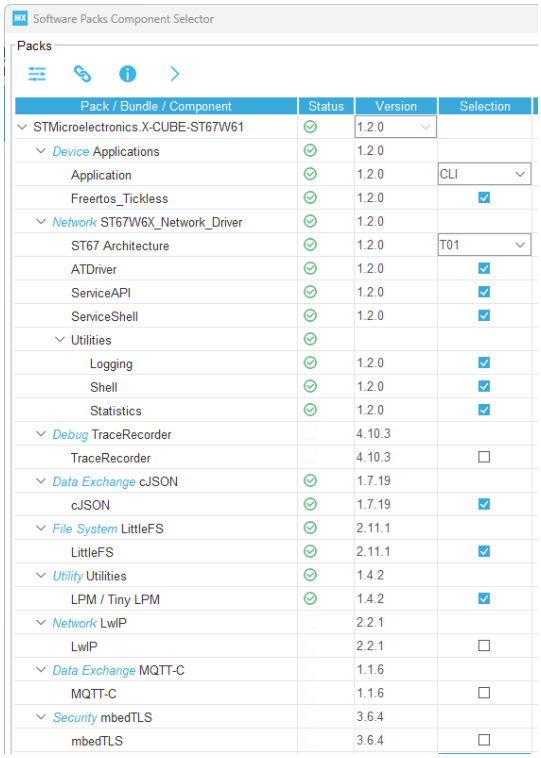
Prerequisite is:

- It is recommended to choose an STM32 with 128 Kbytes of flash and 36 Kbytes of RAM minimum. For CLI and LwIP projects, more ROM/RAM are needed

Note: To use the X-CUBE-ST67W61 v1.2.0, it is required to use at least STM32CubeMX version 6.16.0.

The picture below shows the pack content. Application and modules can be selected.

Figure 26. X-CUBE-ST67W61 pack content



DT77578V3

To learn more about ST67W611M1 project generation, visit the corresponding wiki page at: https://wiki.st.com/stm32mcu/wiki/Connectivity:Wi-Fi_How_to_use_X_CUBE_ST67W61_STM32CubeMx_pack

Revision history

Table 18. **Document revision history**

Date	Version	Changes
13-Mar-2025	1	Initial release.
02-Jun-2025	2	First public release. Updated: • Section 1: General information • Section 1.1: Acronyms and definitions • Section 3: ST67W611M1 features • Section 4: Hardware overview • Section 4.1: X-NUCLEO-67W61M1 mezzanine board overview • Section 4.2: Nucleo-144 main board overview • Section 4.3: How to flash the ST67W611M1 device • Section 4.5: Mission mode • Section 5.1: Overview • Section 5.2.3: Logging features • Section 5.2.4: FreeRTOS™ low-power application • Section 5.2.5: Firmware over-the-air support (FOTA) • Section 5.4: System services • Section 5.5: Firmware update services • Section 5.5.1: ST67W611M1 firmware update APIs functions • Section 5.6: Wi-Fi® services • Section 5.6.1: Wi-Fi® APIs functions • Section 5.7.1: Net APIs functions • Section 5.8.1: Bluetooth® LE APIs functions • Section 5.9.1: HTTP APIs functions • Section 5.10.1: MQTT APIs functions • Section 6.1: Project directory • Section 6.2.1: Integrated development environment (IDE) requirements • Section 6.2.4: Hardware requirements • Section 6.6: Bluetooth® LE commissioning application • Section 6.5.1: Project overview • Section 6.7.1: Project overview • Section 6.10.1: Project overview • Section 7.2.5: iPerf on Android™ smartphone • Section 7.3.2: Main iperf command Added: • Section 5.6.2: Wi-Fi® events • Section 5.7.2: NET events • Section 5.8.2: Bluetooth® LE events • Section 5.10.2: MQTT events • Section 6.3.3: MDK-ARM • Section 6.12: Wi-Fi® Commissioning application and Section 6.12.1: Project overview Deleted all the sections describing the applications and replaced them with links to the corresponding wiki pages.
05-Jun-2025	3	Updated wiki links in: • All the project overview sections • Section 8: X-CUBE-ST67W61 STM32CubeMX pack
08-Oct-2025	4	Updated: • Section 2: X-CUBE-ST67W61 architecture overview • Section 4.1: X-NUCLEO-67W61M1 mezzanine board overview • Section 4.3: How to flash the ST67W611M1 device • Section 4.5: Mission mode • Section 5: X-CUBE-ST67W61 architecture and its subsections • Section 6.1: Project directory

Date	Version	Changes
		• Section 6.6: Bluetooth® LE commissioning application and its subsection • Section 6.6.1: Project overview • Section 6.11: MQTT demonstration • Section 7.2.5: iPerf on Android™ smartphone • Section 8: X-CUBE-ST67W61 STM32CubeMX pack
18-Dec-2025	5	Updated: • Section 2: X-CUBE-ST67W61 architecture overview • Section 3: ST67W611M1 features • Section 4.3: How to flash the ST67W611M1 device • Section 4.4: Manufacturing mode • Section 4.5: Mission mode • Section 5.1.1: Software top view • Section 5.1.1.1.1: FreeRTOS™ LwIP offload implementation view • Section 5.1.1.1.2: FreeRTOS™ LwIP on-host implementation view • Section 5.3.3: Bluetooth® LE • Section 5.3.4: Net • Section 5.5.1: ST67W611M1 firmware update APIs functions • Section 5.7.1: Net APIs functions • Section 5.8.1: Bluetooth® LE APIs functions • Section 6.2.3: Terminal

Contents

1 **General information** 2
1.1 Acronyms and definitions 2
2 **X-CUBE-ST67W61 architecture overview** 5
3 **ST67W611M1 features** 6
3.1 Module content 6
3.2 Supported wireless connectivity standards 6
3.3 Regulation certifications 6
3.4 Wi-Fi® features 6
3.5 Bluetooth® LE features 6
4 **Hardware overview** 7
4.1 X-NUCLEO-67W61M1 mezzanine board overview 7
4.2 Nucleo-144 main board overview 8
4.3 How to flash the ST67W611M1 device 9
4.4 Manufacturing mode 14
4.5 Mission mode 15
5 **X-CUBE-ST67W61 architecture** 17
5.1 Overview 17
5.1.1 Software top view 17
5.2 Framework 21
5.2.1 FreeRTOS™ 21
5.2.2 CLI from shell 21
5.2.3 Logging features 21
5.2.4 FreeRTOS™ low-power application 22
5.2.5 Firmware over-the-air support (FOTA) 22
5.3 ST67W6X_Network_Driver configuration 22
5.3.1 System 23
5.3.2 Wi-Fi® 23
5.3.3 Bluetooth® LE 23
5.3.4 Net 23
5.3.5 HTTP 24
5.3.6 MQTT 24
5.3.7 AT common 24
5.3.8 Utility performance 25
5.3.9 WFA traffic generator 25

5.3.10 External service . . . 25
5.4 System services . . . 25
5.4.1 System APIs functions . . . 25
5.5 Firmware update services . . . 26
5.5.1 ST67W611M1 firmware update APIs functions . . . 26
5.6 Wi-Fi® services . . . 26
5.6.1 Wi-Fi® APIs functions . . . 26
5.6.2 Wi-Fi® events . . . 27
5.6.3 Wi-Fi® finite state machine (FSM) . . . 28
5.7 Net services . . . 30
5.7.1 Net APIs functions . . . 30
5.7.2 NET events . . . 32
5.7.3 NET FSM . . . 32
5.8 Bluetooth® LE services . . . 33
5.8.1 Bluetooth® LE APIs functions . . . 33
5.8.2 Bluetooth® LE events . . . 35
5.9 HTTP services . . . 35
5.9.1 HTTP APIs functions . . . 36
5.10 MQTT services . . . 36
5.10.1 MQTT APIs functions . . . 36
5.10.2 MQTT events . . . 36
5.11 Netif service . . . 37
5.11.1 Netif APIs functions . . . 37
6 Package description . . . 38
6.1 Project directory . . . 38
6.2 Requirements . . . 38
6.2.1 Integrated development environment (IDE) requirements . . . 38
6.2.2 STM32CubeProgrammer . . . 38
6.2.3 Terminal . . . 39
6.2.4 Hardware requirements . . . 39
6.3 Build and load a project . . . 39
6.3.1 STM32CubeIDE . . . 39
6.3.2 EWARM . . . 39
6.3.3 MDK-ARM . . . 39
6.4 Echo application . . . 40
6.4.1 Project overview . . . 40
6.5 CLI application . . . 40

6.5.1 Project overview . . . 40
6.6 Bluetooth® LE commissioning application . . . 40
6.6.1 Project overview . . . 40
6.7 Bluetooth® LE P2P application . . . 40
6.7.1 Project overview . . . 40
6.8 HTTP server application . . . 41
6.8.1 Project overview . . . 41
6.9 HTTPS client application . . . 41
6.9.1 Project overview . . . 41
6.10 FOTA application . . . 41
6.10.1 Project overview . . . 41
6.11 MQTT demonstration . . . 41
6.12 Wi-Fi® Commissioning application . . . 42
6.12.1 Project overview . . . 42
7 How to measure Wi-Fi® throughput . . . 43
7.1 Overview . . . 43
7.2 iPerf installation . . . 43
7.2.1 Linux® . . . 43
7.2.2 Debian® . . . 43
7.2.3 Red Hat® . . . 43
7.2.4 Windows® . . . 43
7.2.5 iPerf on Android™ smartphone . . . 43
7.3 iPerf usage . . . 44
7.3.1 Features selection . . . 44
7.3.2 Main iperf command . . . 44
8 X-CUBE-ST67W61 STM32CubeMX pack . . . 46
Revision history . . . 47
List of tables . . . 52
List of figures . . . 53

List of tables

Table 1. List of acronyms . . . 2
Table 2. 3-step ST67W611M1 programming process . . . 10
Table 3. System APIs functions . . . 25
Table 4. Firmware update APIs functions . . . 26
Table 5. Wi-Fi® APIs function . . . 26
Table 6. Wi-Fi® events . . . 27
Table 7. Net APIs functions . . . 30
Table 8. NET events . . . 32
Table 9. Bluetooth® LE APIs functions . . . 33
Table 10. Bluetooth® LE events . . . 35
Table 11. HTTP APIs functions . . . 36
Table 12. MQTT APIs functions . . . 36
Table 13. MQTT events . . . 36
Table 14. Netif APIs functions . . . 37
Table 15. Serial COM port configuration . . . 39
Table 16. MQTT demonstrations . . . 42
Table 17. Main iperf commands . . . 45
Table 18. Document revision history . . . 47

List of figures

Figure 1. LwIP off-load architecture (T01) . . . 5
Figure 2. LwIP on-host architecture (T02) . . . 5
Figure 3. NUCLEO-U575ZI-Q (left) and X-NUCLEO-67W61M1 (right) . . . 7
Figure 4. Pin mapping . . . 8
Figure 5. NUCLEO-U575ZI-Q connections . . . 9
Figure 6. ST67W611M1 programming hardware setup . . . 10
Figure 7. Step 1 - ST67W611M1 programming process . . . 11
Figure 8. Step 2 - ST67W611M1 programming process . . . 12
Figure 9. Step 2 (continued) - ST67W611M1 programming process . . . 13
Figure 10. Step 3 - ST67W611M1 programming process . . . 14
Figure 11. Manufacturing mode hardware setup . . . 14
Figure 12. ST67W6X-RCT-TOOL manufacturing tool . . . 15
Figure 13. Mission mode hardware setup . . . 15
Figure 14. ST67W6X_CLI Tera Term screenshot . . . 16
Figure 15. LwIP off-load architecture . . . 17
Figure 16. LwIP on-host architecture . . . 17
Figure 17. ST67W61 middleware driver: FreeRTOS™ view . . . 18
Figure 18. ST67W61 middleware driver with LwIP aside: FreeRTOS™ view . . . 20
Figure 19. TCP throughput versus W61 MAX SPI . . . 24
Figure 20. Wi-Fi® station (STA) FSM . . . 29
Figure 21. Wi-Fi® STA + SoftAP behavior chart . . . 30
Figure 22. TCP sockets FSM . . . 32
Figure 23. UDP sockets FSM . . . 33
Figure 24. X-CUBE-ST67W61 Expansion Package . . . 38
Figure 25. *Iperf2 for Android* screenshot . . . 44
Figure 26. X-CUBE-ST67W61 pack content . . . 46