Getting started with the X-CUBE-WB05N Bluetooth® Low Energy software expansion for STM32Cube

Introduction

The X-CUBE-WB05N expansion software package for the STM32Cube runs on the STM32 and includes drivers for the STM32WB05xN Bluetooth® Low Energy network coprocessor device.

The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with sample implementations of the drivers running on the X-NUCLEO-WB05KN1 when connected to a NUCLEO-U575ZI-Q board.

Related links

Visit the STM32Cube ecosystem web page on www.st.com for further information

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
ACI	Application controller interface
ATT	Attribute protocol
Bluetooth LE	Bluetooth low energy
BSP	Board support package
BT	Bluetooth
GAP	Generic access profile
GATT	Generic attribute profile
GUI	Graphical user interface
HAL	Hardware abstraction layer
HCI	Host controller interface
IDE	Integrated development environment
L2CAP	Logical link control and adaptation protocol
LED	Light emitting diode
LL	Link layer
LPM	Low power manager
MCU	Microcontroller unit
PCI	Profile command interface
PHY	Physical layer
SIG	Special interest group
SM	Security manager
SPI	Serial peripheral interface
UUID	Universally unique identifier

2 X-CUBE-WB05N software expansion for STM32Cube

2.1 Overview

The X-CUBE-WB05N software package expands STM32Cube functionality and provides the Bluetooth® Low Energy connectivity.

The key features are:

- Complete middleware to build Bluetooth® Low Energy applications using STM32WB05xN devices.
- Easy portability across different MCU families, thanks to STM32Cube.
- Numerous examples to aid comprehension of Bluetooth® connectivity applications.
- Package compatible with STM32CubeMX, can be downloaded from and installed directly into STM32CubeMX.
- Free, user-friendly license terms.

2.1.1 Bluetooth® Low Energy

Bluetooth® Low Energy is a wireless personal area network technology designed and marketed by Bluetooth® SIG. It can be used to develop innovative applications in various fields such as fitness, security, and healthcare. These applications can run on devices powered by coin-cell batteries, which can remain operational indefinitely without draining the battery.

2.1.1.1 Operating modes

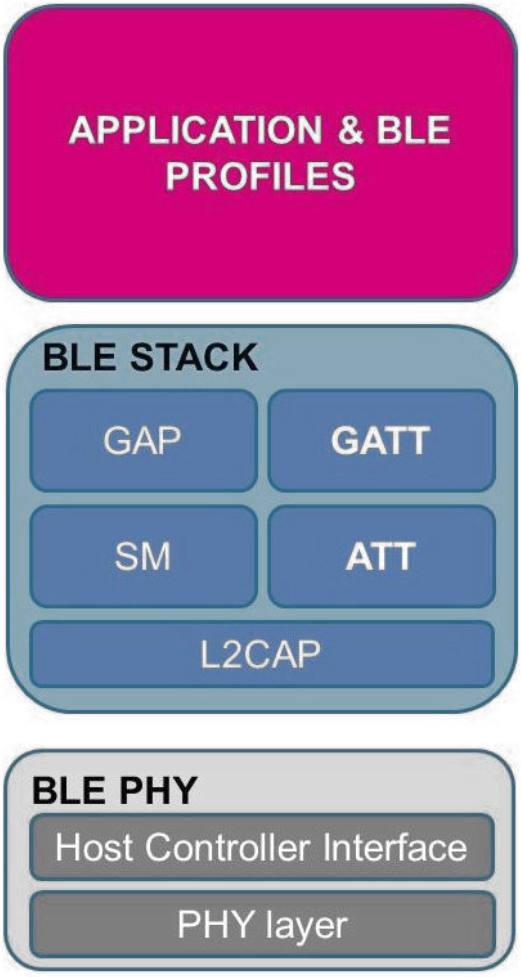
According to the Bluetooth® standard specification (version 4.0 and above), Bluetooth® classic and Bluetooth® Low Energy can be supported on the same device (dual-mode, also called Bluetooth® smart ready). Instead, a single-mode (Bluetooth® smart) device supports the Bluetooth® Low Energy protocol only.

2.1.1.2 *Software partitioning*

A typical Bluetooth LE system consists of:

- An LE controller, containing a physical layer (PHY) including the radio, a link layer (LL), and a standard host controller interface (HCI).
- A host, containing a HCI and other higher protocol layers (for example, L2CAP, SM, ATT/GATT, and GAP).

Figure 1. Bluetooth LE protocol stack



The host can send HCI commands to control the LE controller. The HCI interface and the HCI commands are standardized by the Bluetooth® core specification (refer to the Bluetooth® standard for further information).

The PHY layer ensures communication with the stack and data (bits) transmission over-the-air. Bluetooth LE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 radio frequency (RF) channels with 2 MHz channel spacing.

In Bluetooth LE, when a device only needs to broadcast data, it transmits the data in advertising packets through the advertising channels. Any device that transmits advertising packets is called an advertiser. Devices that aim only at receiving data through the advertising channels are called scanners. Bidirectional data communication between two devices requires them to connect to each other.

Bluetooth LE defines two device roles at the link layer (LL) for a created connection: the master and the slave. These are the devices that act as the initiator and advertiser during the connection creation, respectively.

The host controller interface (HCI) layer provides a standardized interface to enable communication between the host and controller. In the STM32WB05xN, this layer is implemented through the SPI and USART hardware interface.

In Bluetooth LE, the main goal of the L2CAP is to multiplex the data of three higher layer protocols, ATT, SMP, and link layer control signaling, on top of a link layer connection.

The SM layer is responsible for pairing and key distribution and enables secure connection and data exchange with another device.

At the highest level of the core Bluetooth LE stack, the GAP specifies device roles, modes, and procedures for the discovery of devices and services, the management of connection establishment, and security. In addition, the GAP handles the initiation of security features. The Bluetooth LE GAP defines four roles with specific requirements on the underlying controller: Broadcaster, Observer, Peripheral, and Central.

The ATT protocol allows a device to expose certain pieces of data, known as attributes, to another device. The ATT defines the communication between two devices playing the roles of server and client, respectively, on top of a dedicated L2CAP channel. The server maintains a set of attributes. An attribute is a data structure that stores the information managed by the GATT, the protocol that operates on top of the ATT. The client or server role is determined by the GATT and is independent of the slave or master role.

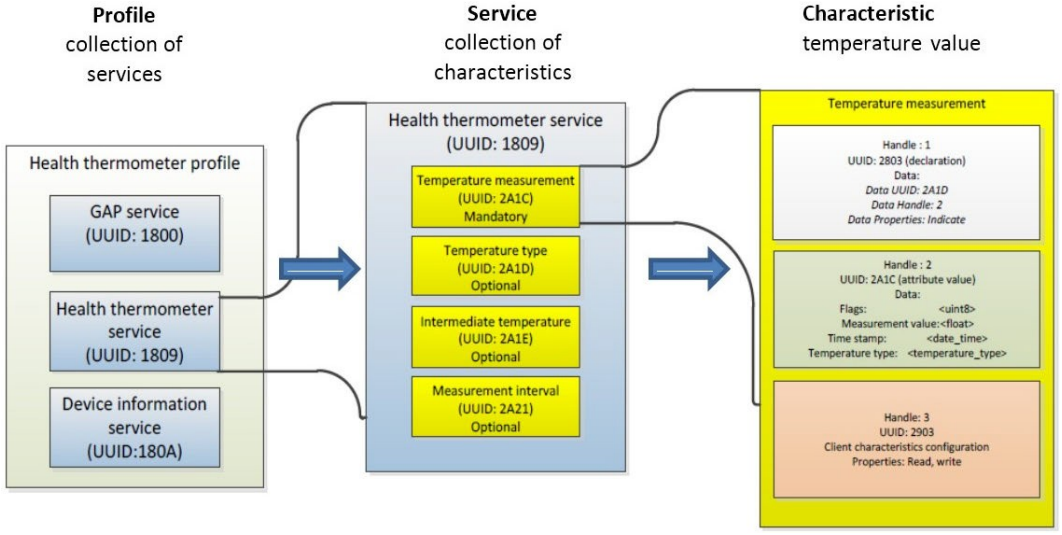
The GATT defines a framework that uses the ATT for the discovery of services, and the exchange of characteristics from one device to another. GATT specifies the structure of profiles. In Bluetooth LE, all pieces of data that are being used by a profile or service are called characteristics. A characteristic is a set of data which includes a value and properties.

2.1.1.3 *Profiles and services*

The Bluetooth LE protocol stack is used by the applications through its GAP and GATT profiles. The GAP profile is used to initialize the stack and set up the connection with other devices. The GATT profile is a way of specifying the transmission - sending and receiving - of short pieces of data known as attributes over a Bluetooth® smart link.

All current Low Energy application profiles are based on GATT. The GATT profile allows the creation of profiles and services within these application profiles.

Figure 2. Structure of a GATT-based profile



In this example, the profile is created with the following services:

- GAP service, which has to be always set up.
- Health thermometer service.
- Device information service.

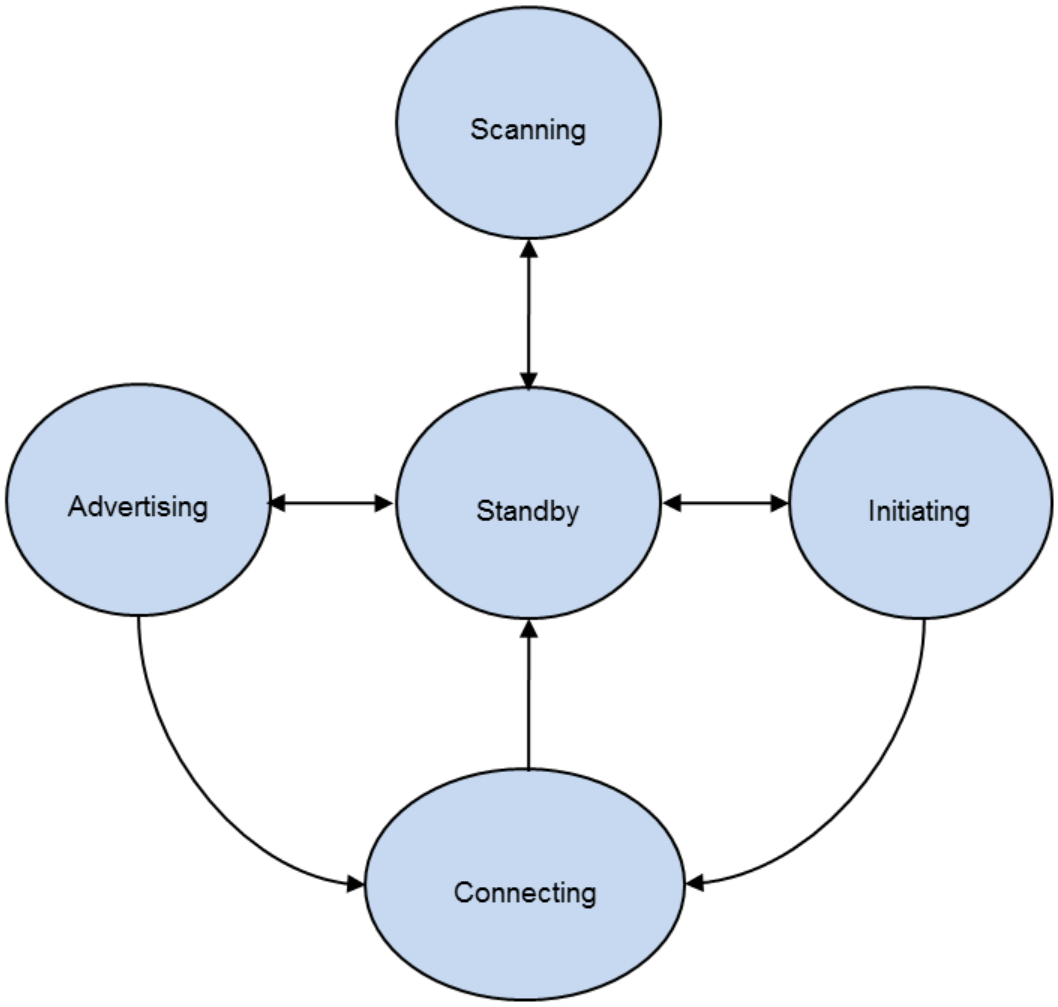
Each service consists of a set of characteristics defining the service and the type of data it provides as part of the service. In the above example, the health thermometer service contains the following characteristics:

- Temperature measurement.
- Temperature type.
- Intermediate temperature.
- Measurement interval.

Each characteristic details the data type and value, and is defined by attributes (at least two for characteristic): the main attribute (0x2803) and a value attribute that actually contains the data. The main attribute defines the value attribute handle and UUID.

2.1.1.4 *State machine*

Figure 3. State machine during Bluetooth LE operations



During normal operation, a Bluetooth LE device can be in the following states:

- Standby: does not transmit or receive packets.
- Advertising: broadcasts advertisements in advertising channels. The device transmits advertising channel packets and possibly listens and answers too, triggered by the advertising channel packets.
- Scanning: looks for advertisers. The device is listening for advertising channel packets from advertising devices.
- Initiating: the device initiates a connection to the advertiser and listens to advertising channel packets from specific device(s) and responds to these packets to initiate a connection with another device.
- Connection: connection has been established and the device is transmitting or receiving:
  - The initiator device plays the master role, that is, it communicates with the device in the slave role and defines timings of transmission.
  - The advertiser device plays the slave role, that is, it communicates with a single device in the master role.

2.2 Architecture

This software is a fully compliant expansion for STM32Cube enabling development of applications using the Bluetooth® Low Energy connectivity.

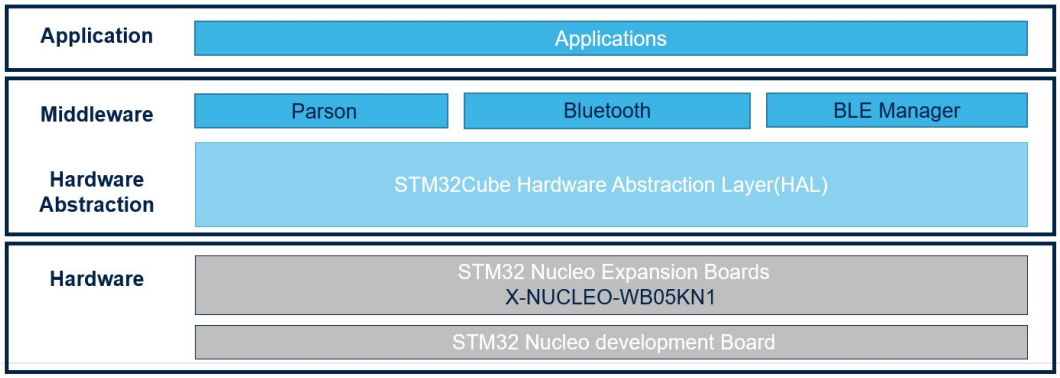
The software is based on the hardware abstraction layer for the STM32 microcontroller, STM32CubeHAL. The package extends STM32Cube by providing complete middleware for the Bluetooth® Low Energy expansion board and several sample applications.

The software layers used by the application software to access the STM32WB05xN module expansion board are:

• The STM32Cube HAL driver layer provides a simple, generic, and multi-instance set of APIs (application programming interfaces) to interact with the upper layers (application, libraries, and stacks). It includes generic and extension APIs and is based on a generic architecture which allows the layers built on it (such as the middleware layer) to implement their functionalities without dependence on the specific hardware configuration of a given Microcontroller Unit (MCU). This structure improves library code reusability and ensures high portability across other devices.

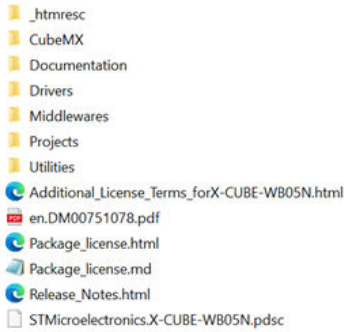
• The Board Support Package (BSP) layer provides supporting software for the peripherals on the STM32 Nucleo board, except for the MCU. It has a set of APIs to provide a programming interface for certain board-specific peripherals (for example, the LED, the user button, etc.) and allows identification of the specific board version.

Figure 4. X-CUBE-WB05N software architecture



2.3 Folders

Figure 5. X-CUBE-WB05N package folder structure



The following folders are included in the software package:

- **Documentation**: contains a compiled HTML file generated from the source code and detailed documentation regarding the software components and APIs.
- **Drivers**: contains the HAL drivers, the board-specific drivers for each supported board or hardware platform, including those for the onboard components, and the CMSIS layer, which is a vendor-independent hardware abstraction layer for the Cortex®-M processor series.
- **Middleware**: libraries and protocols related to host software and applications to interface the STM32WB05xN controller.
- **Projects**: contains some sample applications, showing how to use the STM32WB05xN devices, for the NUCLEO-U575ZI-Q platform with three development environments, IAR Embedded Workbench for Arm, RealView Microcontroller Development Kit (MDK-ARM), and STM32CubeIDE.

2.4 APIs

Detailed technical information about the APIs available to the user can be found in the compiled HTML file "X_CUBE_WB05N.chm" in the "Documentation" folder of the software package, where all the functions and parameters are fully described.

2.5 Sample application description

Several sample applications using the X-NUCLEO-WB05KN1 expansion board with a NUCLEO-U575ZI-Q board are provided in the "Projects" directory. Ready to be built projects are available for multiple IDEs.

These applications are included in the package to provide the users with examples showing how to use the STM32WB05xN library APIs. Each sample application folder contains a readme.txt file describing the application and how to use it.

The functions to be called when creating an application using the STM32WB05xN are listed in the X-CUBE-WB05N/Documentation/X-CUBE-WB05N.chm file.

3 X-CUBE-WB05N software package for STM32WB05xN

3.1 STM32WB05xN overview

The X-CUBE-WB05N software package contains drivers for the STM32WB05xN ultra low power (ULP) network coprocessor device for Bluetooth® Low Energy applications.

The STM32WB05xN comes preprogrammed with a production-ready and certified Bluetooth LE stack image. The STMicroelectronics Bluetooth LE stack is stored into the on-chip non-volatile flash memory and can be easily upgraded via SPI, UART interface or through the dedicated STM32Cube software tools.

The STM32WB05xN is an ultra-low-power Bluetooth® Low Energy wireless network coprocessor addressing the standard Bluetooth LE protocol. Interfacing with an external microcontroller can be achieved through SPI and USART. It embeds STMicroelectronics' state-of-the-art 2.4 GHz radio IPs, optimized for ultra-low-power consumption and excellent radio performance, for unparalleled battery lifetime. It is compliant with Bluetooth® Low Energy SIG core specification version 5.4.

The STM32WB05xN embeds an Arm Cortex®-M0+ microcontroller that can operate up to 64 MHz.

The STM32WB05xN features standard and advanced communication interfaces:

1x SPI

1x USART

The STM32WB05xN operates in the -40 to +105 °C temperature range from a 1.7 V to 3.6 V power supply. A comprehensive set of power-saving modes enables the design of low-power applications.

The STM32WB05xN integrates a high efficiency SMPS step-down converter and an integrated PDR circuitry with a fixed threshold that generates a device reset when the VDD drops under 1.65 V.

The device can be emulated using the STM32WB05xN mounted on the X-NUCLEO-WB05KN1 expansion board.

3.2 STM32WB05xN binary images

Two types of binary images are available in the X-CUBE-WB05N/Utilities folder for configuring the STM32WB05xN as a Bluetooth LE network coprocessor (UART and SPI mode):

1. Location + Essential configuration.
2. HCI Controller only configuration.

The following table describes the main supported Bluetooth LE features.

Table 2. Bluetooth LE stack supported features

	Location + Essential configuration	HCI Controller only configuration
Central & Peripheral role	x	x
Controller Privacy	-	x
Secure Connections	x	-
Data length extension	x	x
LE Coded Phy and LE 2M PHY	-	x
Extended Advertising	x	x
Direction Finding	x	x
Periodic Advertising	x	x
Power Control	-	x
Channel Classification	-	x
Connection subrating	-	x

Note:
- *x: feature is supported.*
- *-: feature is not supported.*

The maximum number of simultaneous radio tasks supported on the two binary images is described in the following table:

Table 3. Bluetooth LE maximum number of simultaneous radio tasks

	Value	
Maximum number of simultaneous radio tasks	Location + Essential configuration	HCI Controller only configuration
	3	6

Appendix

The STM32WB05xN provides a hardware interface to an external microcontroller based on two very common protocols:

- SPI slave protocol with interrupt signal
- UART

The pins dedicated to the UART interface are:

- PB0 (UART RX)
- PA1 (UART TX)

The pins dedicated to the SPI interface are:

- PB3 (SPI CLOCK)
- PA11 (SPI MOSI)
- PA8 (SPI MISO)
- PA9 (SPI CS)
- PA10 (SPI IRQ)

3.3 How to emulate the STM32WB05xN device with X-NUCLEO-WB05KN1

To emulate the STM32WB05xN device, the STM32WB05xN DTM firmware image contained in the X-CUBE-WB05N software package must be loaded onto the STM32WB05xN device mounted on the X-NUCLEO-WB05KN1 expansion board. This operation can be performed using a standard ST-LINK/V3 debugger and following the procedure below.

Step 1. Connect the X-NUCLEO-WB05KN1 CN8 pins and the ST-LINK/V3 pins as per the following table and figure.

Table 4. Jumper connection between X-NUCLEO-WB05KN1 and ST-LINK/V3 MB1440B

Pin name	CN8 pin N.	ST-LINK/V3 MB1440B CN6 pin N.
VDD	1	1
SWTCK	2	2
GND	3	3
SWDIO	4	4
RST	5	5

Figure 6. X-NUCLEO-WB05KN1 connected to ST-LINK/V3

Step 2. Download and unpack the X-CUBE-WB05N containing the STM32WB05xN firmware image.

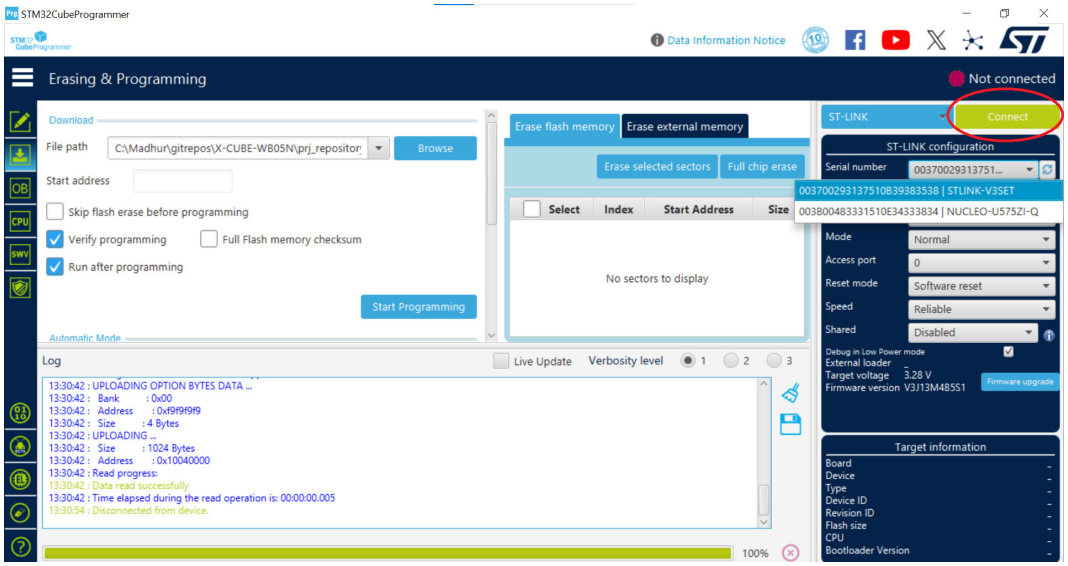
Step 3. Download and install the STM32CubeProg (v2.17.0 or later only).

Step 4. Connect the ST-LINK/V3 debugger to your PC.

Step 5. Open the STM32CubeProgrammer (STM32CubeProg) and:

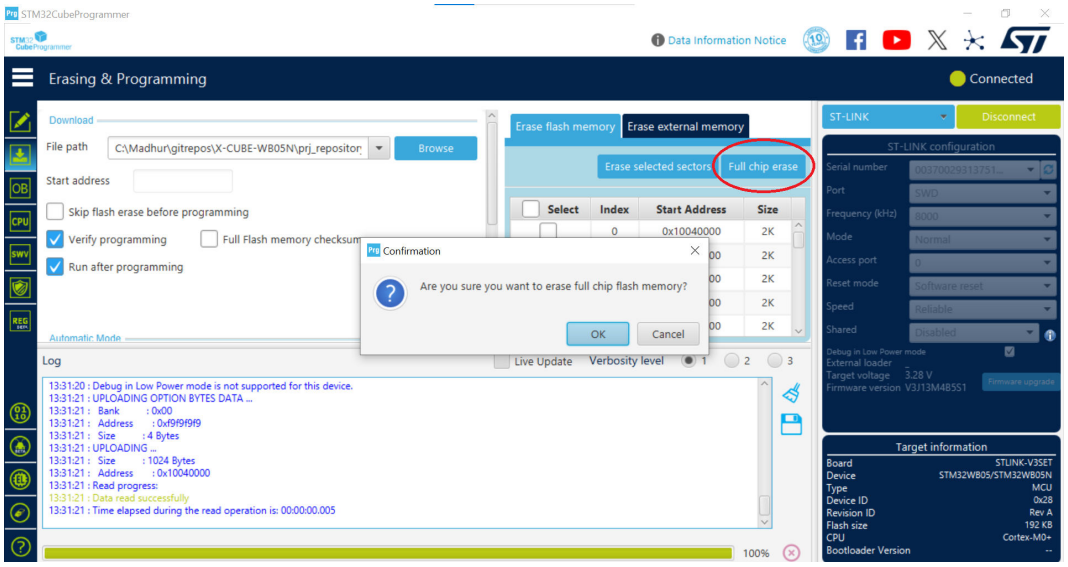
Step 5a. Select the ST-LINK-V3SET SWD probe and select the Connect button.

Figure 7. STM32CubeProgrammer - Connect



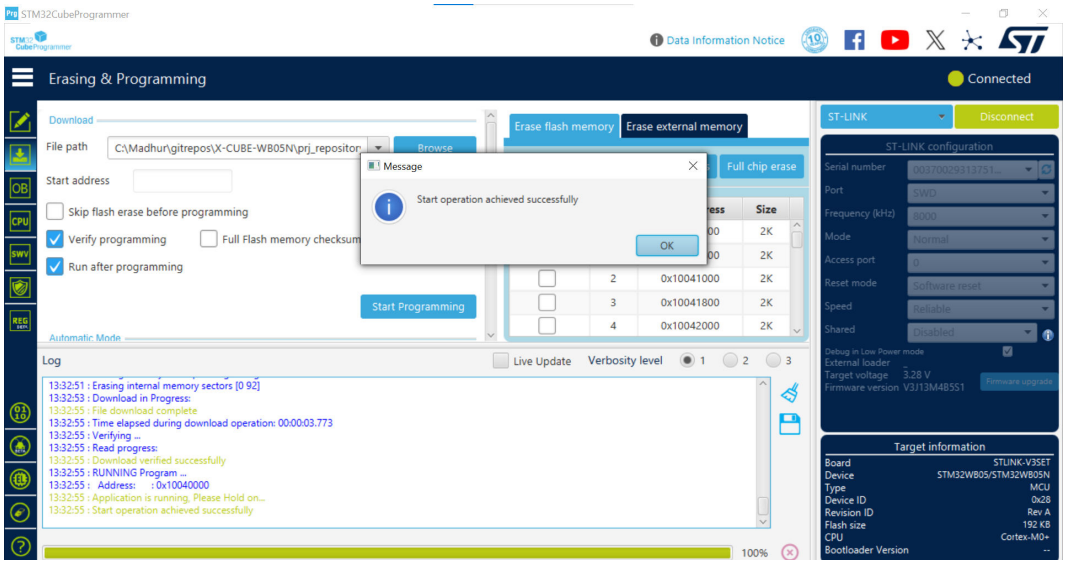
Step 5b. Erase the STM32WB05xN flash memory from [**Erase flash memory**]>[**Full chip erase**].

Figure 8. STM32CubeProgrammer - Mass Erase



Step 6. Load the DTM firmware contained in the X-CUBE-WB05N/Utilities/ BLE_Transparent_Mode_STM32WB05_location_essential folder and press the [**Start Programming**] button.

Figure 9. STM32CubeProgrammer - Program



The image flashed on the STM32WB05xN device can be used with the X-CUBE-WB05N software package.

3.4 How to upgrade the STM32WB05xN firmware image

To upgrade the STM32WB05xN firmware image on the STM32WB05xN device:

Step 1. Download and install X-CUBE-WB05N and STM32CubeProg.

Step 2. Open the STM32CubeProgrammer (STM32CubeProg).

Step 3. Erase the STM32WB05xN flash memory. Refer to Section 3.2 of this document for steps.

Step 4. Load the BLE_TransparentMode_UART_with_Updater.hex or BLE_TransparentMode_SPI_with_Updater.hex (depending on use case) firmware contained in the X-CUBE-WB05N installation folder:

– X-CUBE-WB05N\Utilities\BLE_Transparent_Mode_STM32WB05_location_essential).

Note: User needs to upgrade the firmware of X-NUCLEO-WB05KN1 with latest BLE_Transparent_Mode_STM32WB05_location_essential binary images present in X-CUBE-WB05N_V2.0.0/ Utilities/BLE_Transparent_Mode_STM32WB05_location_essential folder.

3.5 SPI protocol for network coprocessor

This document outlines the main specification for the SPI protocol to be used in the STM32WB05xN network coprocessor configuration. The document content is valid for the STM32WB05xN device.

The specification aims at achieving the following targets:

- Power efficient
- Code efficient
- Fast data transfer

3.5.1 SPI protocol hardware details

The SPI port requires five pins:

- SPI CLK
- SPI MOSI

- SPI MISO
- SPI CS
- SPI IRQ

The maximum SPI baud rate supported is 6 MHz. The timing diagram adopted is CPOL 1 and CPHA 1, which means data are captured on the SPI clock's rising edge and data is output on a rising edge. The SPI CS also acts as a wake-up pin for the STM32WB05xN, so that if the SPI CS pin is low (the external microcontroller selects the STM32WB05xN for communication) the STM32WB05xN is woken up if it was asleep. The STM32WB05xN notifies the external microcontroller of an event pending through the SPI IRQ pin. If the SPI IRQ pin is high, the STM32WB05xN has at least an event for the external microcontroller.

Table 5. STM32WB05xN SPI lines

Pin function	Pin name	Pin number	Information
SPI CLK	PB3	5	SPI clock signal
SPI MOSI	PA11	12	SPI controller output completer input signal
SPI MISO	PA8	9	SPI controller input completer output signal
SPI CS	PA9	10	SPI chip select signal/wake-up signal
SPI IRQ	PA10	11	SPI IRQ request for event pending signal

3.5.2 SPI communication protocol

To communicate with the STM32WB05xN, the data on the SPI bus must be formatted as described in this section. An SPI transaction is defined from a rising edge of the SPI CS signal to the next rising edge of the SPI CS signal. Each SPI transaction must contain only one data frame. Each data frame should contain at least 5 bytes of header, and may have from 0 to N bytes of data.

Figure 10. Generic SPI transaction

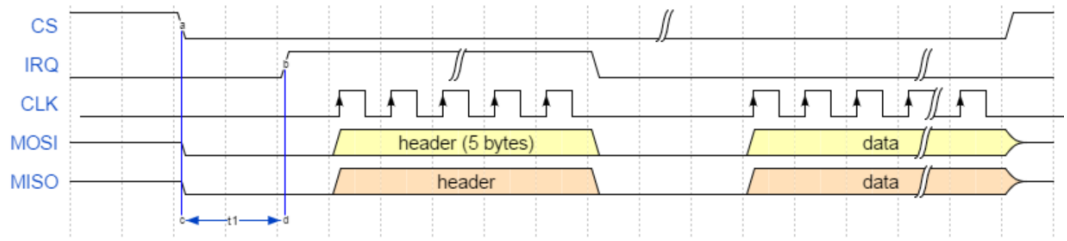


Figure 10 shows a generic SPI transaction; the list of steps is as follows:

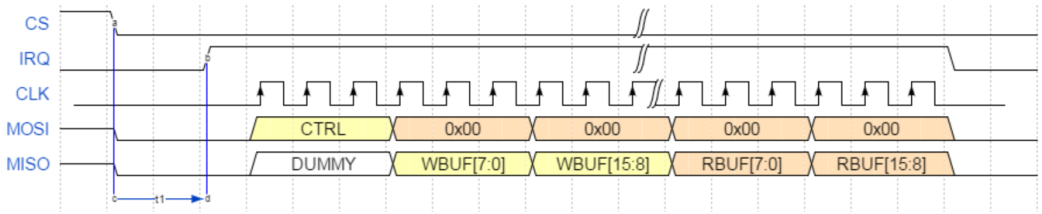
1. The external microcontroller lowers the SPI CS signal to start the communication.
2. The STM32WB05xN raises the SPI IRQ signal to indicate that it is ready for communication. The time t1 changes according to the state of the STM32WB05xN. This time t1 can include wakeup of the STM32WB05xN and preparation of the header part of the frame.
3. The external microcontroller must wait for the SPI IRQ signal to become high and then start to transfer the five bytes of the header that include the control field with the intended operation. In addition, the external microcontroller reads five bytes from the STM32WB05xN, which includes information about the actual size of the read buffer and of the write buffer.
4. The external microcontroller, after checking the 5 bytes of the header, performs the data transaction.
5. The STM32WB05xN lowers the SPI IRQ signal after the five bytes of the header are transferred.
6. The external microcontroller waits until the SPI IRQ is low before raising the SPI CS signal to mark the end of the communication.

Some important notes are:

- Setting the SPI CS signal low wakes up STM32WB05xN if the device is asleep.
- If the SPI IRQ signal is low before setting the SPI CS signal low, this means the STM32WB05xN has no data events for the external uc, so the read buffer size is zero (RBUF=0).

- The time t1 is the time between wakeup (point a in Figure 10) and the STM32WB05xN being ready to perform the SPI transaction (point b in Figure 10). The time t1 ranges from a minimal value (the STM32WB05xN already awake when the SPI CS is asserted), to a maximum value that involves a wake-up sequence and software boot.
- Even if there are events pending after the completion of the transaction, the SPI IRQ signal goes low to allow the STM32WB05xN to update the five bytes of the header and rearm the SPI for the next transaction (after this delay the SPI IRQ signal goes high again if events are pending).
- The SPI CS signal marks the beginning and end of the transaction.
- The SPI CS high marks the end of the transaction and must be set to high only when the IRQ line is low.
- The gap between the header and the data is not mandatory, but it is normally required by the external microcontroller to process the header and check if there is enough space in the buffers to perform the wanted transaction.
- When the SPI IRQ signal is high, the five bytes of the header are locked and cannot be modified by STM32WB05xN firmware.

Figure 11. SPI header format

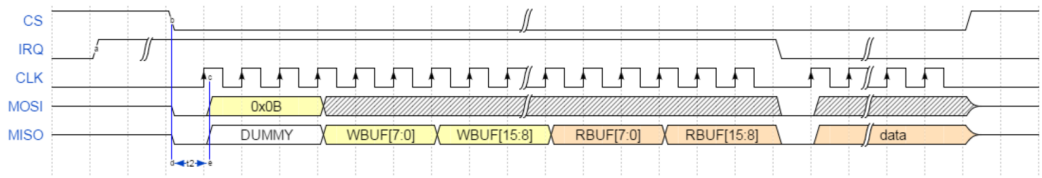


- The header of the external microcontroller (the SPI master) is on the MOSI line, which is composed by one control byte (CTRL) and four bytes 0x00. CTRL field can only have the value of 0x0A (SPI write) or 0x0B (SPI read). The STM32WB05xN returns the header on the MISO line at the same time. When the STM32WB05xN asserts the SPI IRQ signal, it is ready. Otherwise, the STM32WB05xN is still not initialized. The external microcontroller must wait for the IRQ line to become high and perform a five bytes transaction. The 5 bytes in the MISO line gives one byte of starting frame, two bytes with the size of the write buffer (WBUF), and two bytes with the size of the read buffer (RBUF). The endianness for WBUF and RBUF is LSB first. The value in WBUF means how many bytes the master can write to the STM32WB05xN. The value in RBUF means how many bytes in the STM32WB05xN are waiting to be read by the external uc.

Read transaction

- A read transaction is performed when the STM32WB05xN raises the SPI IRQ line before the SPI CS signal is lowered by the external uc.

Figure 12. SPI read transaction



- In this case, the SPI IRQ signal is high indicating the STM32WB05xN is awake and ready to perform the SPI transaction, after a hardware-dependent set-up time t2, typical value is 1.5 us. The transaction is performed as follows:

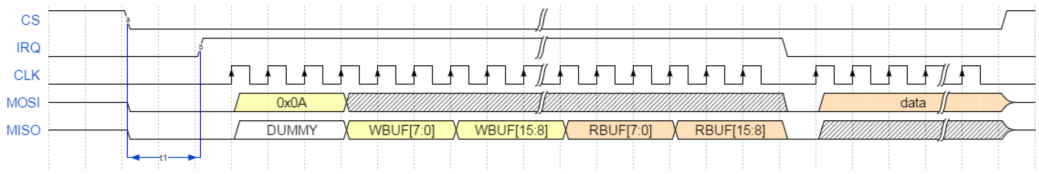
1. An event has been generated by the STM32WB05xN (point a in Figure 12).
2. The external microcontroller lowers the SPI CS signal to initiate a transaction (point b in Figure 12).
3. Since the SPI IRQ signal is high, the external microcontroller initiates a data transfer after t2. The external microcontroller transfers five bytes as follows [0x0B, XX, XX, XX, XX]. The WBUF and RBUF sizes are read from the SPI MISO signal.
4. The external microcontroller performs the read data transaction for RBUF bytes. (Note: if RBUF is 0, this is an unexpected condition since the STM32WB05xN indicates that data is available; in any case the transaction needs to be completed by reading no bytes).
5. The STM32WB05xN lowers the SPI IRQ signal after the five bytes of the header are transferred.

6. The external microcontroller raises the SPI CS signal to mark the end of the transaction.

Write transaction

- A write transaction is performed by the external microcontroller to send a command to the STM32WB05xN. The STM32WB05xN can be awake or sleeping when the SPI CS signal is lowered by the external uc. The assertion of the SPI CS signal wakes up the STM32WB05xN, if asleep.

Figure 13. SPI write transaction



- The transaction is performed as follows:

1. The external microcontroller lowers the SPI CS signal to initiate a transaction.
2. The STM32WB05xN raises the SPI IRQ signal to indicate that it is ready with t1 >= 0.
3. The external microcontroller waits for the SPI IRQ signal to become high and starts a transfer of five bytes sending the code of the intended operation and reading the read buffer and write buffer size. The external microcontroller transfers five bytes as follows [0x0A, XX, XX, XX, XX]. The WBUF and RBUF values are sampled in the SPI MISO signal.
4. The STM32WB05xN lowers the SPI IRQ after the five bytes of the header are transferred.
5. The external microcontroller checks whether the WBUF allows sending the command. If yes, it performs the data transaction, otherwise the external microcontroller must wait.
6. The external microcontroller waits for the SPI IRQ signal to be low before closing the communication.
7. The external microcontroller raises the SPI CS to mark the end of the transaction.

Error transaction

- This section lists the STM32WB05xN firmware behavior when some error transactions are performed:

1. Incomplete header transaction (0 to 4): the STM32WB05xN ignores the transaction.
2. The external microcontroller does not wait for the SPI IRQ signal to become low before raising the SPI CS signal: the STM32WB05xN lowers the SPI IRQ signal when the SPI CS signal is high.
3. The external microcontroller does not wait for the SPI IRQ signal to become high before starting the SPI clock: the result is an acquisition of corrupted data on both the master and slave side.
4. Incomplete read transaction: the master loses the event.
5. Incomplete write transaction: the STM32WB05xN stores the bytes written by the external uc. During the next write operation, the STM32WB05xN gets the new bytes trying to get a complete frame according to Bluetooth® protocol.
6. Two commands in a row without reading an event for command: the STM32WB05xN parses the two commands and then it generates the corresponding events.

SPI state machine

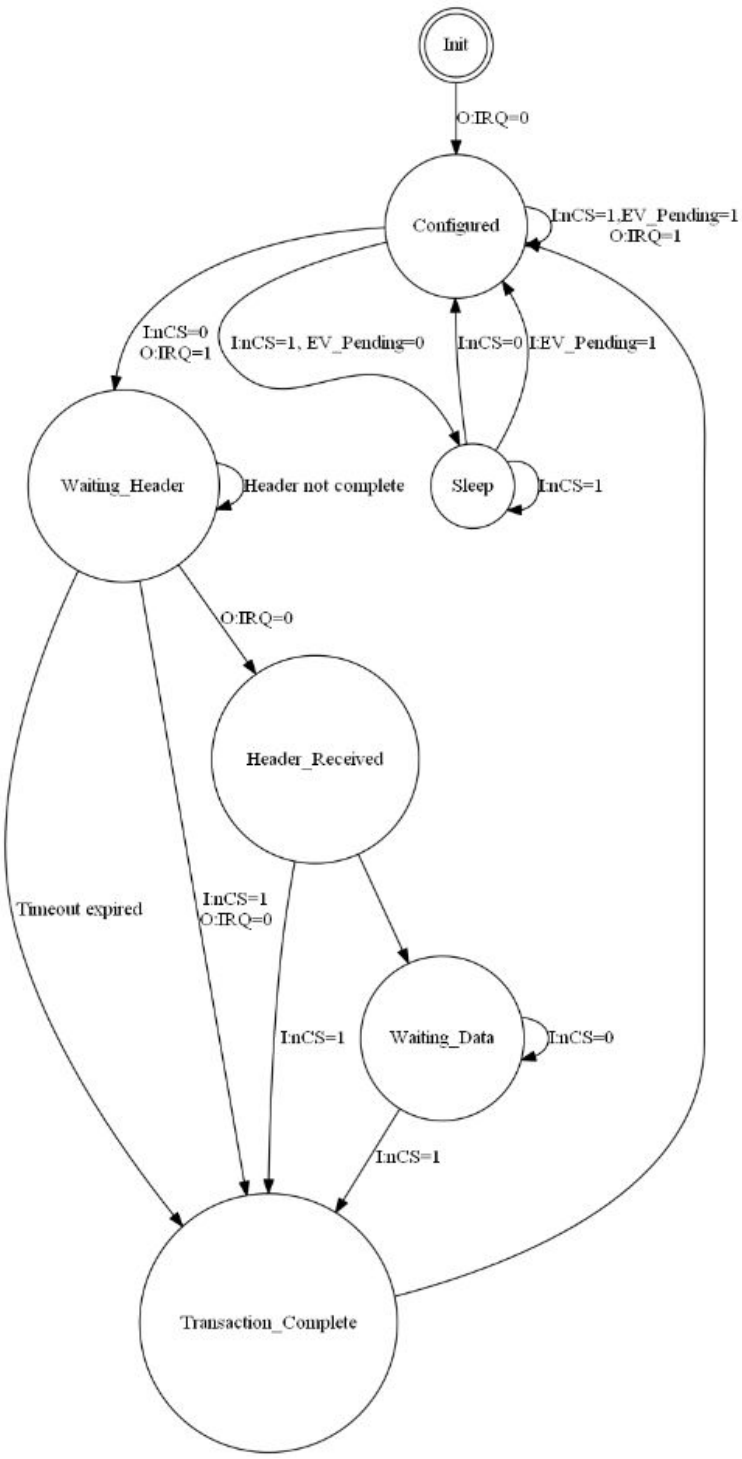
- Hereafter the description of the STM32WB05xN SPI state machine.

Table 6. STM32WB05xN SPI state machine states

State	Description	Input	Output	Next State
Init	Boot/transient state. Hardware initialization	-	IRQ=0	Configured
Configured	Ready to transfer information, 5 bytes header frozen	CS = 0	IRQ=1	Waiting Header
		CS = 1 Event pending = 0	IRQ=0	Sleep
		CS = 1 Event pending = 1	IRQ=1	Configured

State	Description	Input	Output	Next State
Sleep	Sleep state with almost all logic off	CS = 1 Event pending = 0	IRQ=0	Sleep
		CS = 1 Event pending = 1	IRQ=0	Configured
		CS = 0	IRQ=0	Configured
Waiting_Header	Receiving 5 bytes header from SPI master	CS = 0	IRQ=1	When 5 bytes are received goes to Header_Received
		CS = 1	IRQ=1	Transaction_Complete
Header_Received	5 bytes header received	CS = 0	IRQ=0	Waiting_Data
		CS = 1	IRQ=0	Transaction_Complete
Waiting_Data	Receiving payload	CS = 0	IRQ=0	Waiting_Data
		nCS = 1	IRQ=0	Transaction_Complete
Transaction_Complete	Transitional	nCS = 1	IRQ=0	Configured

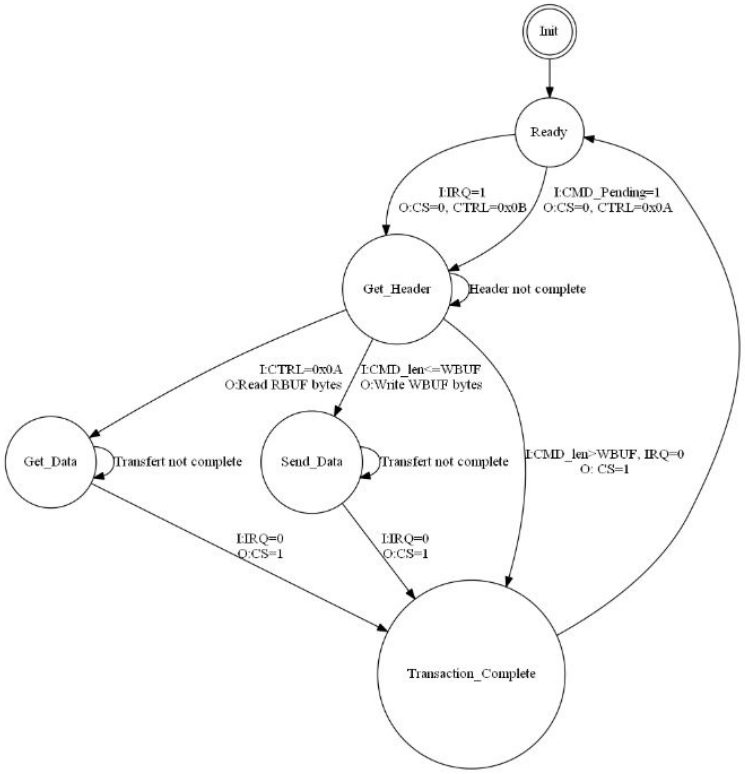
Figure 14. **SPI protocol state machine**



External microcontroller behavior

- The external microcontroller must act as follows according to the information from the STM32WB05xN:
  - SPI IRQ signal
  - Information from header frame WBUF and RBUF

Figure 15. **Expected microcontroller SPI protocol state machine**



Waveform acquisition

Figure 16. **HCI_READ_LOCAL_VERSION_INFORMATION SPI waveform**

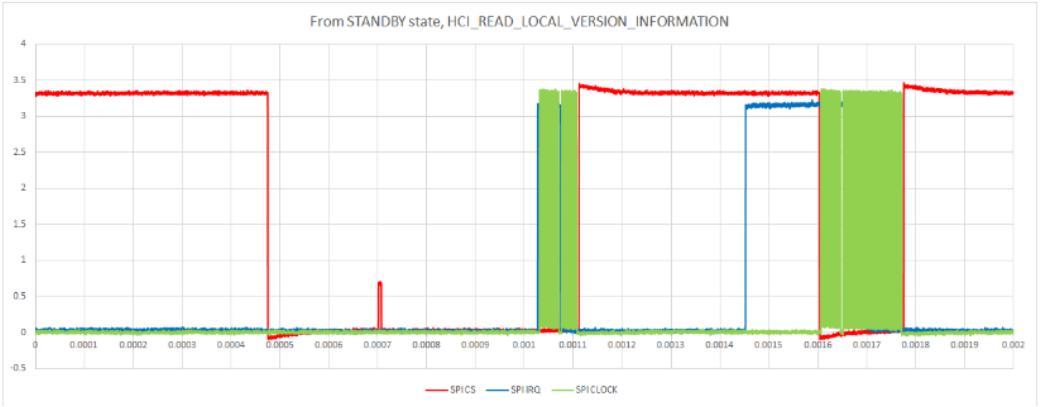


Figure 17. **HCI_READ_LOCAL_VERSION_INFORMATION SPI waveform zoom**

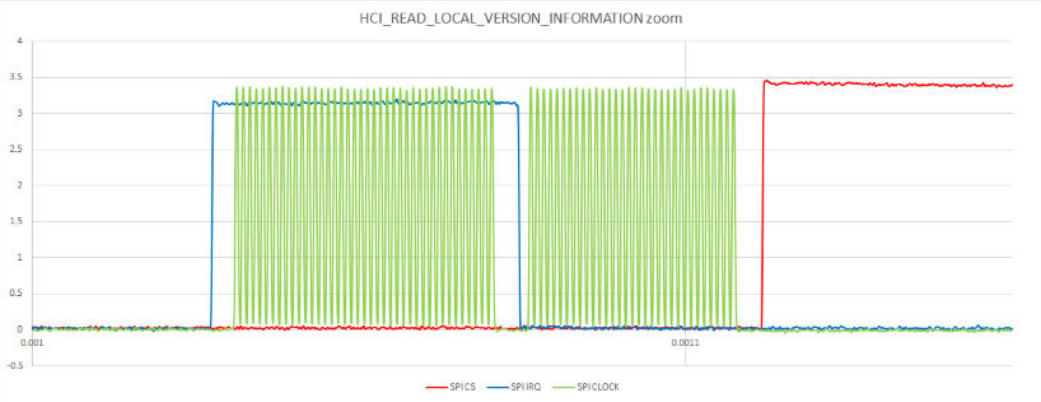
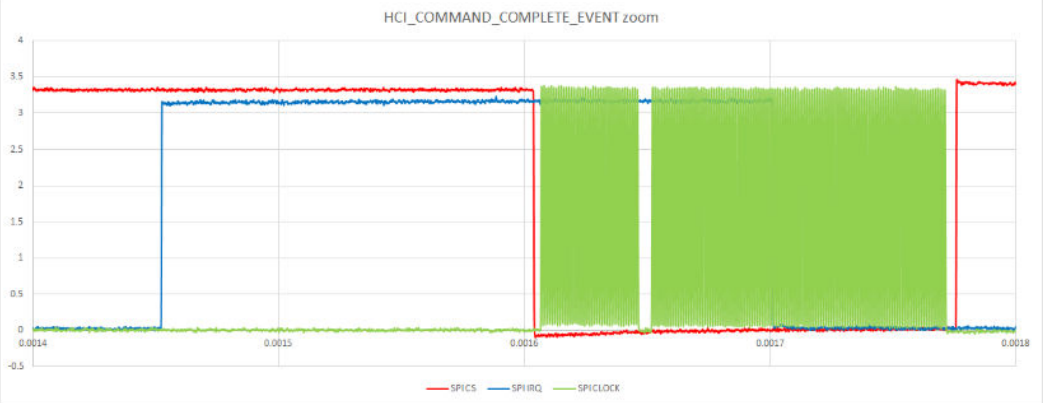


Figure 18. **HCI_COMMAND_COMPLETE_EVENT SPI waveform**



3.6 Building a new project without using STM32CubeMX

It is strongly recommended to use the STM32CubeMX to create new projects as it helps the user to avoid setting project configurations like file paths, folder structure, pin, and clock initialization, etc. Still, if a user needs to start a new project without using the STM32CubeMX, refer to the below guide as a reference:

1. Create a new project in the selected IDE–EWARM/MDK-ARM/STM32CubeIDE.
2. Select the STM32 device part number on which the application runs.
3. Add required files at the project location.
   a. Drivers
      i. CMSIS
      ii. STM32xx_HAL_Driver
   b. IDE startup file – startup_stm32xx.s
   c. Middleware
   d. Application
4. In the IDE link these files' path to the project. For more details, refer to the respective IDE’s user manual.
5. Compile the project.

4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo

STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from. The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V3 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples for different IDEs (IAR EWARM, Keil® MDK-ARM, STM32CubeIDE, Mbed™, and GCC/ LLVM).

All STM32 Nucleo users have free access to the Mbed™ online resources (compiler, C/C++ SDK, and developer community) at www.mbed.org to easily build complete applications.

Figure 19. **STM32 Nucleo board**

Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

4.1.2 X-NUCLEO-WB05KN1 expansion board

The X-NUCLEO-WB05KN1 expansion board provides Bluetooth® Low Energy connectivity for developer applications and can be plugged into an STM32 Nucleo development board (for example NUCLEO-U575ZI-Q) through its ARDUINO® Uno V3 connectors.

The expansion board features the Bluetooth® v5.4 compliant and FCC-certified STM32WB05KN. This SoC manages the complete Bluetooth® Low Energy stack and protocols on its Arm® Cortex®-M0+ core and programmable flash memory.

The STM32WB05KN supports central and peripheral modes and increased transfer rates with data length extension (DLE).

The X-NUCLEO-WB05KN1 interfaces with the STM32 Nucleo microcontroller via UART (default) with and without hardware flow control. Full duplex SPI with an interrupt line is also available. The firmware loaded on the module defines the host interface and, to modify it, simply changes the firmware without modifying the hardware.

Table 7. **Board description for X-NUCLEO-WB05KN1**

Order code	Board references	Description
X-NUCLEO-WB05KN1	MB2160	ARDUINO® interface board
	MB2032	MCU RF mini board

Figure 20. **X-NUCLEO-WB05KN1 Bluetooth LE expansion board**

X-NUCLEO-WB05KN1 global view.
Picture is not contractual.

Information about the X-NUCLEO-WB05KN1 expansion board is available at http://www.st.com/x-nucleo.

4.2 Software description

The following software components are required in order to establish a suitable development environment for creating applications for the STM32 Nucleo equipped with the Bluetooth LE expansion board:

- X-CUBE-WB05N: an STM32Cube expansion for sensor application development. The X-CUBE-WB05N firmware and associated documentation is available on www.st.com..
- Development toolchain and compiler. The STM32Cube expansion software supports the three following environments:
  - IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-LINK.
  - RealView Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK.
  - STM32CubeIDE + ST-LINK.

4.3 Hardware setup

The following hardware components are required:

1. One STM32 Nucleo development platform (suggested order code: NUCLEO-U575ZI-Q).
2. One STM32WB05xN module expansion board setup, MB2160 and MB2032 (order code: X-NUCLEO-WB05KN1).
3. One USB type A to micro-B USB cable to connect the STM32 Nucleo to a PC.

4.4 Nucleo and Bluetooth LE expansion board setup

The STM32 Nucleo board integrates the STLINK/V2-1, STLINK-V3E, or STLINK-V3EC debugger/programmer.

The X-NUCLEO-WB05KN1 expansion board can be easily connected to the STM32 Nucleo motherboard through the Arduino UNO R3 extension connector. The STM32WB05xN communicates with the STM32 microcontroller, hosted on the STM32 Nucleo development board, through an SPI or USART link available on the Arduino® UNO R3 connector.

Figure 21. X-NUCLEO-WB05KN1 expansion board

Revision history

Table 8. **Document revision history**

Date	Revision	Changes
29-Oct-2024	1	Initial release.
13-May-2025	2	Updated Table 2. Bluetooth LE stack supported features, Section 3.3: How to emulate the STM32WB05xN device with X-NUCLEO-WB05KN1 and Section 3.4: How to upgrade the STM32WB05xN firmware image.

Contents

List of tables

List of figures