Getting started with X-CUBE-WBA Bluetooth® LE software expansion for STM32Cube

Introduction

The X-CUBE-WBA Expansion Package provides a software framework to access the Bluetooth® LE stack features offered by an STM32WBA series device (STM32WBA25CE, STM32WBA55CG, or STM32WBA65RI), configured as a either a network coprocessor or a radio coprocessor.

The software package features the X-NUCLEO-WBA25A1 expansion board, including an STM32WBA25CE device, plugged onto the NUCLEO-U385RG-Q development board. The same applications examples can be used on the NUCLEO-U385RG-Q development board connected to an STM32WBA55xx or an STM32WBA65xx device over proper serial interface such as SPI or UART.

No dedicated Nucleo expansion boards are available with the STM32WBA55xx and STM32WBA65xx device variants.

The external STM32 microcontroller interfaces with the STM32WBA coprocessor through:

- Either a UART
- Or a full duplex SPI with an interrupt line interface

This document includes the description of the provided sample applications, the steps required to configure a generic project using the Bluetooth® LE middleware, and instructions for configuring and using the sample applications included in the package.

1 General information

X-CUBE-WBA interacts with STM32WBA series devices (STM32WBA25CE, STM32WBA55CG, and STM32WBA65RI), which run the Bluetooth® LE stack, and with the STM32U385RG microcontroller.They are based on the Arm® Cortex®-M33 processor.

For information on Bluetooth®, refer to www.bluetooth.com.

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

The Arm word and logo are trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved.

1.1 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Term	Description
ATT	Attribute protocol
BLE_CP	Bluetooth® LE network or radio coprocessor middleware
BSP	Board support package
HAL	Hardware abstraction layer
HCI	Host controller interface
LED	Light-emitting diode
MCU	Microcontroller unit
MTU	Maximum transmission unit
NCP	Network coprocessor
RCP	Radio coprocessor
SPI	Serial peripheral interface
UART	Universal asynchronous receiver transmitter
USB	Universal serial bus

2 X-CUBE-WBA presentation

This chapter describes all system requirements to run X-CUBE-WBA software, as well as its installation procedure.

2.1 Overview

The X-CUBE-WBA software package expands STM32Cube functionality to provide Bluetooth® LE connectivity.

The key features are:

- Complete middleware to build Bluetooth® LE applications using an STM32WBA series device (STM32WBA25CE, STM32WBA55CG, or STM32WBA65RI) configured as a network coprocessor or a radio coprocessor
- Various examples to aid in the comprehension of Bluetooth® connectivity applications
- Free, user-friendly license terms

Using X-CUBE-WBA requires the following minimum specifications of the personal computer:

- Intel® or AMD processor running at least the Windows® 11 Microsoft® operating system
- At least 2 Gbytes of RAM
- USB ports

2.2 Architecture

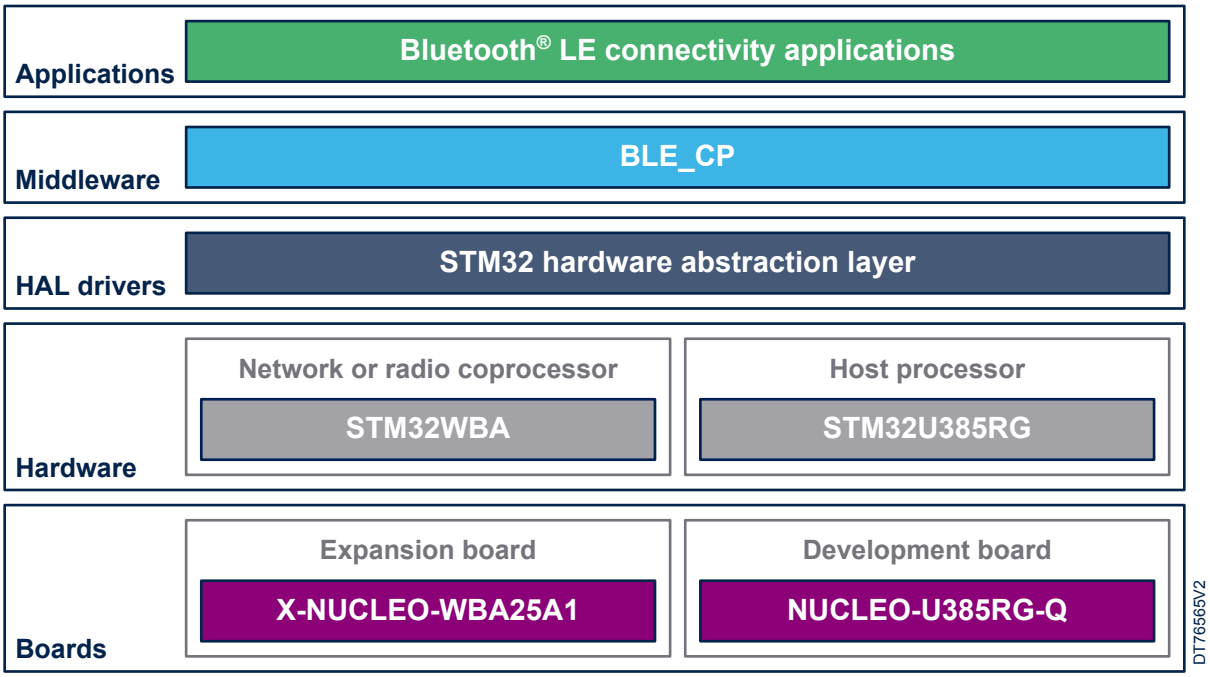
The X-CUBE-WBA software is based on the hardware abstraction layer for the STM32 microcontroller, STM32CubeHAL. The package extends STM32Cube by providing complete middleware for the Bluetooth® LE expansion board and several sample applications.

The software layers used by the application to access the STM32WBA series device (STM32WBA25CE, STM32WBA55CG, and STM32WBA65RI) are:

- The STM32Cube HAL driver layer that offers a straightforward, generic, and multi-instance set of APIs (application programming interfaces) designed to facilitate interaction with upper layers, including applications, libraries, and stacks. It comprises both generic and extension APIs. This layer is built on a generic architecture, enabling the layers above it, such as the middleware layer, to implement their functionalities independently of the specific hardware configuration of a given microcontroller unit (MCU). This architecture enhances library code reusability and ensures high portability across different devices.
- The board support package (BSP) layer contains support software for the peripherals on the STM32 Nucleo development board, excluding the microcontroller unit (MCU). It provides a set of APIs that serve as a programming interface for specific board peripherals, such as the LED and user button, and facilitate the identification of the specific board version.

Figure 1 presents the top-level software architecture of X-CUBE-WBA.

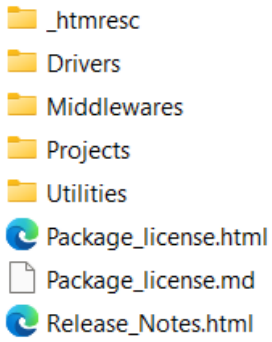
Figure 1. X-CUBE-WBA software architecture



The following folders, presented in Figure 2, are included in the software package:

- `Drivers`: contains the HAL drivers, the board-specific drivers for each supported board or hardware platform, including those for the on-board components and the CMSIS layer, which is a vendor-independent hardware abstraction layer for the Cortex®-M processor series.
- `Middlewares`: contains libraries and protocols related to the host microcontroller software and applications to interface with the STM32WBA coprocessor.
- `Projects`: contains some sample applications, showing how to use the STM32WBA series device, for the NUCLEO-U385RG-Q platform, with development environments such as IAR Embedded Workbench®, Keil® MDK-ARM, and STM32CubeIDE.
- `Utilities`: STM32WBA BLE_TransparentMode (UART interface) and BLE_TransparentMode_SPI (SPI interface) binary images to be loaded on the selected STM32WBA series device according to the selected configuration (network coprocessor or radio coprocessor).

Figure 2. X-CUBE-WBA package folders



Release notes and license files are located in the root folder.

3 Sample applications

This chapter describes the main functions of the application examples provided within the X-CUBE-WBA Expansion Package:

- Ready-to-use projects, running on a NUCLEO-U385RG-Q connected with an STM32WBA network or radio coprocessor through a UART or an SPI interface.
- Demonstration of the Bluetooth® LE APIs, provided by the STM32WBA Bluetooth® LE network or radio coprocessor middleware (BLE_CP), for correctly initializing and using the STM32WBA Bluetooth® LE device.

3.1 SensorDemo_BLESensor-App

This application is an example showing how to implement an application tailored for interacting with the "ST BLE Sensor" app for Android™ and iOS™ devices.

The "ST BLE Sensor" app is freely available on both the Google Play™ store and Apple Store®.

The source code of the "ST BLE Sensor" app is also available on GitHub for both iOS™ and Android™ devices.

After establishing the connection between the STM32 board setup and the smartphone:

- The emulated values of the temperature and pressure are sent by the STM32 board setup to the mobile device and are shown in the *ENVIRONMENTAL* tab
- The plot of the emulated data (temperature, pressure, accelerometer, gyroscope, and magnetometer) sent by the board is shown in the *PLOT DATA* tab
- The RSSI value is shown in the *RSSI & Battery* tab

3.2 SampleApp

This sample application shows how to use the Bluetooth® LE stack in a simple way. It provides the user with a complete example demonstrating how to perform an ATT MTU exchange procedure, so that the server and the client can agree on the supported MTU.

Testing this application requires two STM32 Nucleo development boards with their respective X-NUCLEO-WBA25A1 Nucleo expansion boards.

After programming both the development boards, one board configures itself as a Bluetooth® LE Server-Peripheral device, while the other one becomes a Bluetooth® LE Client-Central device.

Once the connection between the two boards is established (signaled by the LD2 LED blinking on the Client-Central device), pressing the user button on one board toggles the LD2 LED on the other board and vice versa.

If only one STM32 Nucleo development board is available, use the “BLE Scanner” app (available for both Android™ and iOS™ devices) as a Bluetooth® LE Client-Central device.

3.3 Beacon

This example application shows how to use an STM32 Nucleo development board with the X-NUCLEO-WBA25A1 Nucleo expansion board to implement a beacon device.

A beacon device is a smart Bluetooth® LE device that transmits a small data payload at regular intervals using Bluetooth® advertising packets.

Beacons are used to mark important places and objects. Typically, a beacon is visible to a user's device from a range of a few meters, allowing for highly context-sensitive use cases.

To locate the beacon, it is necessary to have a beacon scanner application running on a Bluetooth® LE -capable smartphone, such as the “Beacon Scanner” app (available for both Android™ and iOS™ devices).

3.4 Virtual_COM_Port

Virtual_COM_Port is the application to be used with a version of STM32CubeWiSE - Bluetooth® LE Explorer (STM32CubeWiSEbe) that supports the STM32WBA series devices. STM32CubeWiSE - Bluetooth® LE Explorer is a software tool to evaluate and test the radio performance of STM32WBA-based hardware devices on the Bluetooth® LE technology.

4 X-CUBE-WBA software configuration

4.1 STM32WBA binary images

Two types of binary images are available in the `X-CUBE-WBA/Utilities` folder for configuring the STM32WBA as a Bluetooth® network coprocessor or radio coprocessor (UART and SPI modes):

- Host + controller configuration (network coprocessor configuration (NCP))
- HCI controller-only configuration (radio coprocessor configuration (RCP))

The following binary images are available:

- BLE_TransparentMode_NCP
  - `BLE_TransparentMode_SPI_STM32WBA2_NCP.bin`
  - `BLE_TransparentMode_SPI_STM32WBA5_NCP.bin`
  - `BLE_TransparentMode_SPI_STM32WBA6_NCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA2_NCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA5_NCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA6_NCP.bin`
- BLE_TransparentMode_RCP
  - `BLE_TransparentMode_SPI_STM32WBA2_RCP.bin`
  - `BLE_TransparentMode_SPI_STM32WBA5_RCP.bin`
  - `BLE_TransparentMode_SPI_STM32WBA6_RCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA2_RCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA5_RCP.bin`
  - `BLE_TransparentMode_UART_STM32WBA6_RCP.bin`

4.2 UART and SPI interfaces

The STM32WBA series devices provide a hardware interface to the external host microcontroller using two widely adopted protocols:

- SPI target protocol with interrupt signal
- UART

The UART and SPI pins are described in Table 2.

Table 2. STM32WBA UART and SPI pins

Device	UART interface		SPI interface				
STM32WBA25CE	PA12 USART1_RX	PA6 USART1_TX	PA8 SPI3_SCK	PA10 SPI3_MISO	PA9 SPI3_MOSI	PA5 SPI_CS	PH3 SPI_IRQ
STM32WBA55CG	PA8 USART1_RX	PB12 USART1_TX	PB4 SPI1_SCK	PB3 SPI1_MISO	PA15 SPI1_MOSI	PA12 SPI_CS	PH3 SPI_IRQ
STM32WBA65RI	PA8 USART1_RX	PB12 USART1_TX	PB10 SPI2_SCK	PA9 SPI2_MISO	PC3 SPI2_MOSI	PB9 SPI_CS	PH3 SPI_IRQ

4.3 How to load a binary image onto the X-NUCLEO-WBA25A1

To download the selected binary image for the STM32WBA25CE coprocessor on the X-NUCLEO-WBA25A1 expansion board, a standard STLINK-V3 debugger such as the STLINK-V3SET can be used as follows:

1. Connect the X-NUCLEO-WBA25A1 CN8 pins and the CN6 pins of the STLINK-V3SET adapter board (MB1440) as indicated in Table 3:

Table 3. Connection between X-NUCLEO-WBA25A1 and STLINK-V3SET (MB1440)

Pin name	X-NUCLEO-WBA25A1 expansion board CN8 pin number	STLINK-V3SET adapter board (MB1440) CN6 pin number
VDD	1	1
SWCLK	2	2
GND	3	3
SWDIO	4	4
RSTN	5	5

2. Download and unpack the X-CUBE-WBA Expansion Package, which contains the STM32U3 firmware image.
3. Download and install the STM32CubeProgrammer tool (STM32CubeProg).
4. Connect the STLINK-V3SET debugger to the PC.
5. Open STM32CubeProgrammer
   a. Select the *STlink-V3SET* SWD probe
   b. Select the [**Connect**] icon
6. Load the selected BLE_TransparentMode firmware contained in `X-CUBE-WBA/Utilities/` and press the [**Start Programming**] button.

The image programmed on the STM32U3 device can be used with the X-CUBE-WBA software package.

4.4 SPI protocol

The SPI protocol for the STM32WBA Bluetooth® LE network and radio coprocessor framework targets the following:

- Power efficiency
- Code efficiency
- Fast data transfer

The SPI protocol requires five pins:

- SPI CLK
- SPI MOSI
- SPI MISO
- SPI CS
- SPI IRQ

The adopted timing diagram is CPOL 1 and CPHA 1. Data are captured on the rising edge of the SPI clock and output on the rising edge.

The SPI CS pin also functions as a wake-up pin for the STM32WBA series device. If the SPI CS pin is low, the external microcontroller selects the STM32WBA series device for communication. The STM32WBA series device wakes up if it was in Sleep mode.

The STM32WBA series device notifies pending events to the external microcontroller through the SPI IRQ pin. If the SPI IRQ pin is high, the STM32WBA series device has at least one event for the external microcontroller.

4.4.1 SPI communication protocol

This section describes how to format data on the SPI interface to enable communication with STM32WBA series devices. An SPI transaction is defined as the period from a rising edge of the SPI CS signal to the next rising edge of the SPI CS signal. Each SPI transaction must include one data frame. Each data frame must contain a header of at least five bytes and zero to N bytes of data.

Figure 3. Generic SPI transaction

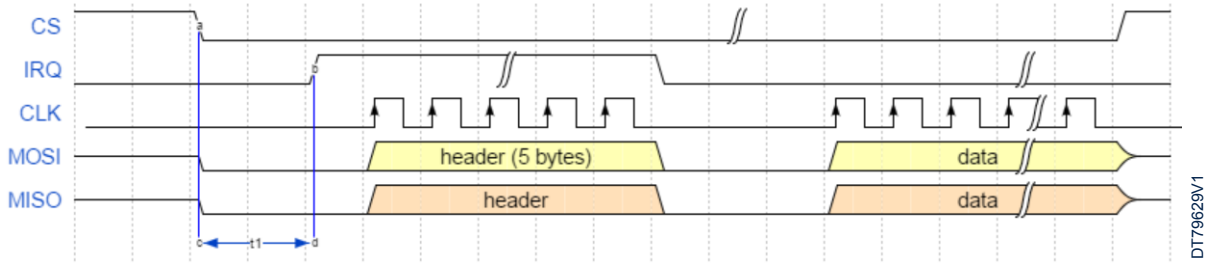


Figure 3 shows a generic SPI transaction, which consists of the following steps:

1. The external microcontroller lowers the SPI CS signal to initiate communication.
2. The STM32WBA series device raises the SPI IRQ signal to indicate readiness for communication. The time t1 varies depending on the state of the STM32WBA series device. The time t1 can include the wake-up of the STM32WBA series device and the preparation of the header section of the frame.
3. The external microcontroller must wait for the SPI IRQ signal to go high, and then transfer five header bytes, including the control field that specifies the intended operation. Additionally, the external microcontroller reads five bytes from the STM32WBA series device. These bytes include information about the actual size of the read buffer and the write buffer.
4. The external microcontroller performs the data transaction after the SPI IRQ signal goes low (the STM32WBA series device is ready for data transfer).
5. The STM32WBA series device lowers the SPI IRQ signal when the header bytes are transferred and the next data transaction has been prepared as requested by the external microcontroller.
6. The external microcontroller completes the data transaction and then raises the SPI CS signal to indicate the end of the communication.

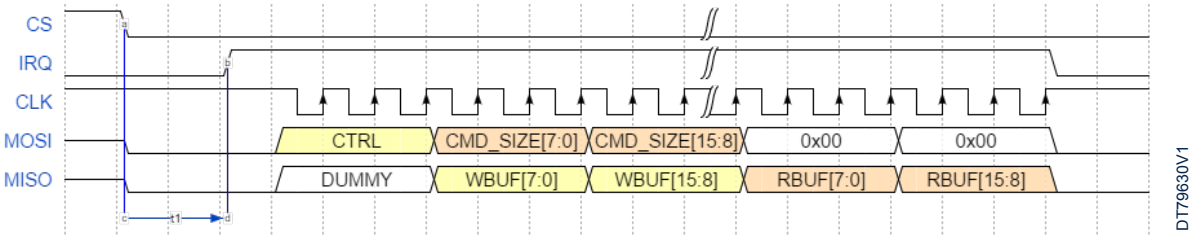
Remarks about the SPI transaction

- Setting the SPI CS signal low wakes up the STM32WBA series device from Sleep mode.
- If the SPI IRQ signal is low before setting the SPI CS signal low, the STM32WBA series device has no data events for the external microcontroller. Therefore, the read buffer size is zero (RBUF=0).
- The time t1 is the interval between wake-up (see point a in Figure 3) and when the STM32WBA series device is ready to execute the SPI transaction (see point b in Figure 3). The time t1 ranges from a minimal value, when the STM32WBA series device is already awake when the SPI CS is asserted, to a maximum value that involves a wake-up sequence and software boot.
- The SPI CS signal indicates the start and end of the transaction.

4.4.2 SPI header

The header of the external microcontroller, which acts as the SPI controller, is on the MOSI line. The header consists of one control byte (CTRL), two bytes indicating the size of the command to be sent (only with CTRL = 0x0A, SPI write), and two 0x00 bytes. The CTRL byte can have only the following values: 0x0A (SPI write) or 0x0B (SPI read).

Figure 4. SPI header format

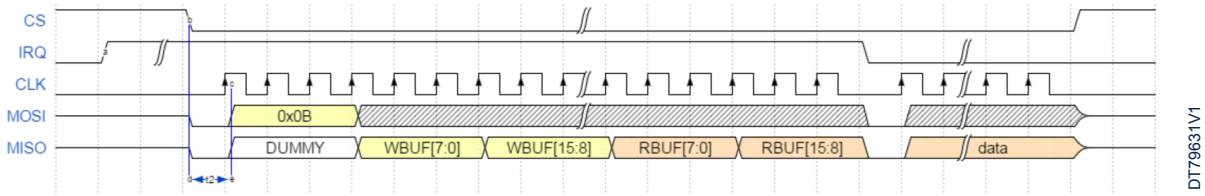


The STM32WBA device returns the header on the MISO line simultaneously. When the STM32WBA device asserts the SPI IRQ signal, the device is ready. Otherwise, the STM32WBA device remains uninitialized. The external microcontroller must wait for the IRQ signal to go high, then perform a five-byte transaction. The five bytes on the MISO line consist of one byte for the start frame, two bytes that indicate the size of the write buffer (WBUF), and two bytes that indicate the size of the read buffer (RBUF). The endianness for WBUF and RBUF is LSB first. The value in WBUF indicates how many bytes the controller can write to the STM32WBA device. The value in RBUF indicates how many bytes in the STM32WBA device are waiting to be read by the external microcontroller.

4.4.3 SPI read transaction

A read transaction normally occurs when the STM32WBA device raises the SPI IRQ signal before the external microcontroller lowers the SPI CS signal.

Figure 5. SPI read transaction



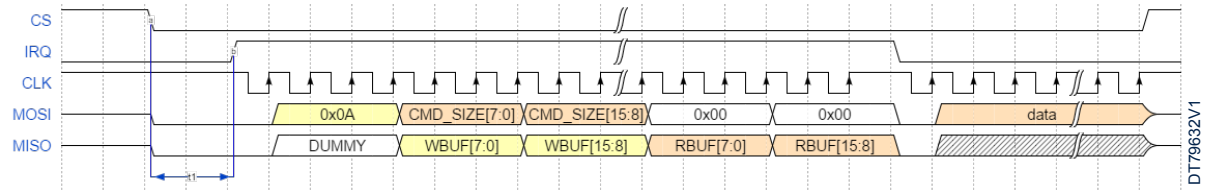
The SPI IRQ signal is high. This state indicates that the STM32WBA device is awake and ready to perform the SPI transaction, after a hardware-dependent setup time, t2. The transaction is performed as follows:

1. The STM32WBA device generates an event (see point a in Figure 5. SPI read transaction).
2. The external microcontroller lowers the SPI CS signal to initiate a transaction (see point b in Figure 5. SPI read transaction).
3. The external microcontroller transfers five bytes in the following order: [0x0B, XX, XX, XX, XX].
   The WBUF and RBUF sizes are read from the SPI MISO signal.
4. The STM32WBA device lowers the SPI IRQ signal after the five header bytes are transferred.
5. After the SPI IRQ signal is low, the external microcontroller performs the read data transaction for RBUF bytes.
   Note: RBUF = 0 is an unexpected condition because the STM32WBA device indicates that data is available. In this case, the transaction must be completed by reading zero bytes.
6. The external microcontroller raises the SPI CS signal to indicate the end of the transaction.

4.4.4 SPI write transaction

The external microcontroller performs a write transaction to send a command to the STM32WBA device. The STM32WBA device can be awake or in Sleep mode when the external microcontroller lowers the SPI CS signal. The assertion of the SPI CS signal wakes up the device if it was in Sleep mode.

Figure 6. SPI write transaction



The SPI write transaction occurs as follows:

1. The external microcontroller lowers the SPI CS signal to initiate a transaction.
2. The STM32WBA device raises the SPI IRQ signal to indicate readiness.

3. The external microcontroller waits for the SPI IRQ signal to go high. It then starts a transfer of five bytes, sending the code for the intended operation, the size of the data to send and reading the read buffer size and the write buffer size. The external microcontroller transfers five bytes in the following order: [0x0A, CMD_SIZE[7:0], CMD_SIZE[15:8], XX, XX].
   The WBUF and RBUF values are sampled in the SPI MISO signal.
4. The STM32WBA device lowers the SPI IRQ signal after the five header bytes are transferred.
5. The external microcontroller checks whether the WBUF can send the command. If the WBUF can send the command, the external microcontroller performs the data transaction after the SPI IRQ signal goes low. Otherwise, the external microcontroller must wait.
6. The external microcontroller raises the SPI CS to indicate the end of the transaction.

4.4.5 SPI transaction error

The STM32WBA device firmware operates as described in Table 4 when transaction errors occur.

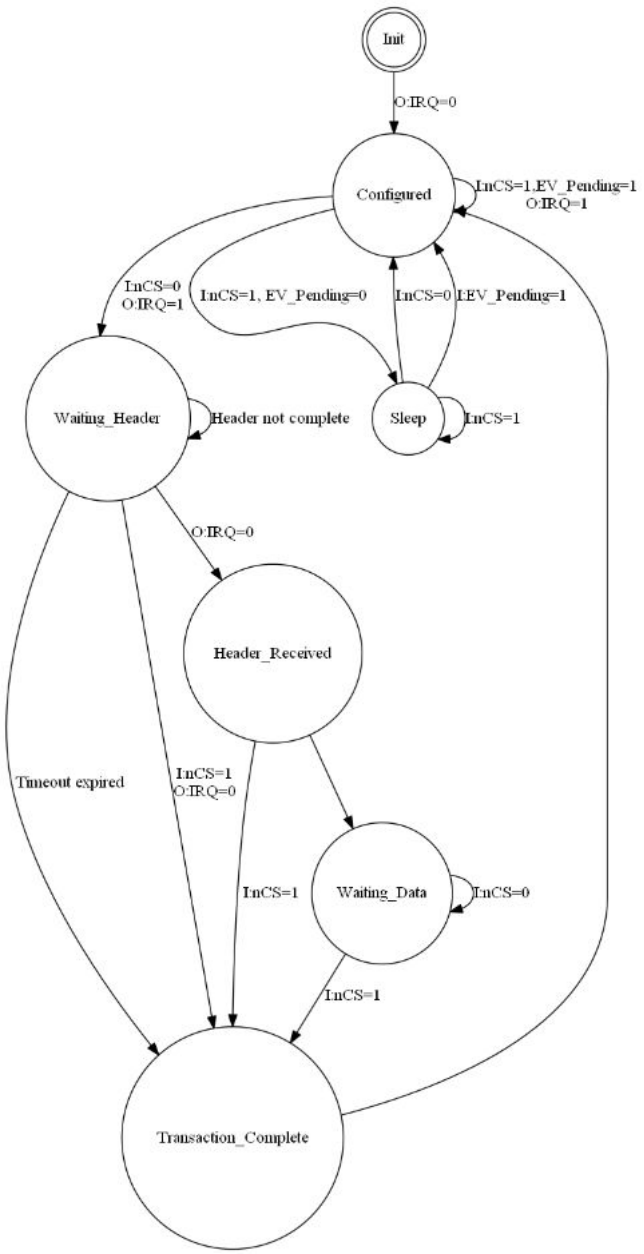
Table 4. Transaction error handling

Transaction error	Error handling
Incomplete header transaction (zero to four bytes).	The device ignores the transaction.
The external microcontroller does not wait for the SPI IRQ signal to go high before starting the SPI clock.	On both the controller and the target sides, the data is corrupted.
Incomplete read transaction.	The controller misses the event.
Incomplete write transaction.	The device stores the bytes written by the external microcontroller. During the next write operation, the device receives the new bytes and attempts to complete the frame according to the Bluetooth® protocol.
Two commands executed consecutively without a read event command.	The device parses the two commands and generates the corresponding events.

4.4.6 SPI state machine

Figure 7. SPI protocol state machine describes the SPI state machine of the STM32WBA series device. Table 5 details the related states and transition conditions.

Figure 7. SPI protocol state machine



DT79633V1

Table 5. 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-byte header frozen.	CS = 0	IRQ = 1	Waiting_Header
		CS = 1; Event pending = 0	IRQ = 0	Sleep
		CS = 1; Event pending = 1	IRQ = 1	Configured
Sleep	Sleep state with almost all logic off.	CS = 1; Event pending = 0	IRQ = 0	Sleep

State	Description	Input	Output	Next State
Sleep	Sleep state with almost all logic off.	CS = 1; Event pending = 1	IRQ = 0	Configured
		CS = 0	IRQ = 0	Configured
Waiting_Header	Receiving 5-byte header from SPI controller	CS = 0	IRQ = 1	When 5 bytes are received, goes to Header_Received
		CS = 1	IRQ = 1	Transaction_Complete
Header_Received	5-byte 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

4.4.7 External microcontroller behavior

The external microcontroller must operate as described in Table 6 and Figure 8 according to the information it receives from the STM32WBA device:

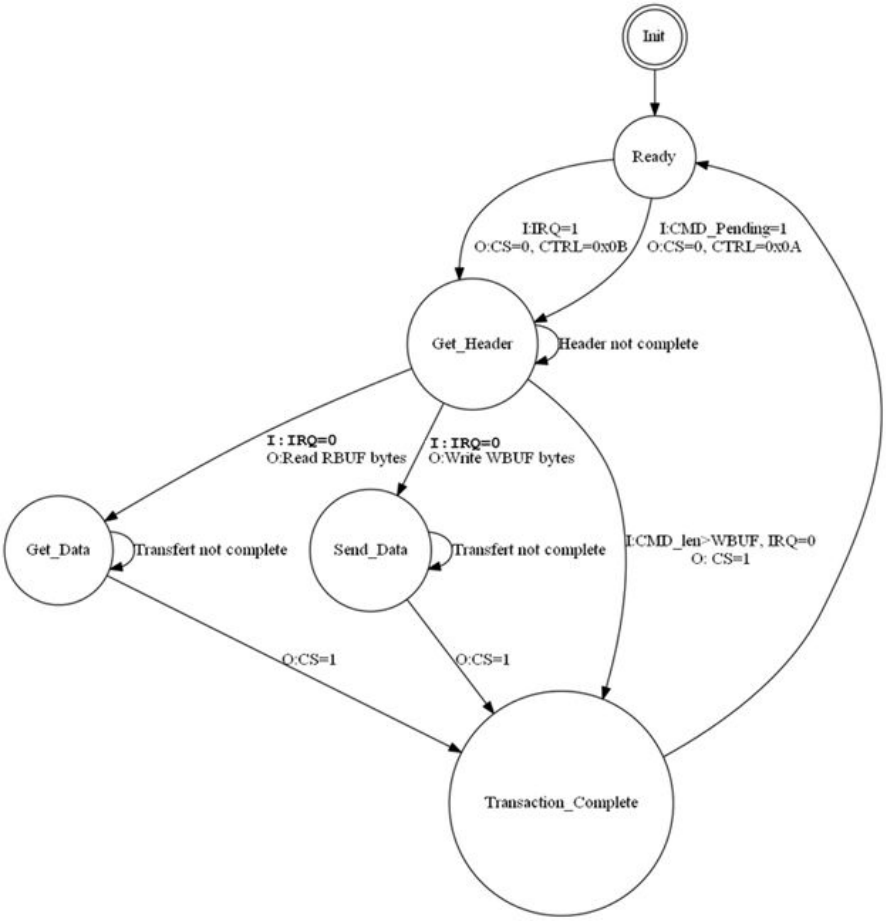
- SPI IRQ signal
- Information from WBUF and RBUF header frames

Table 6. External microcontroller SPI input from STM32WBA

Input	Meaning	External
IRQ = 1, RBUF is not 0	Read operation is required, at least one event is pending.	Read operation can be performed.
IRQ = 0, RBUF is 0	No event is pending.	Nothing to do.
IRQ = 0, RBUF is 0, WBUF is N	No event is pending.	If the number of bytes to write is less than or equal to N, then the write operation is acceptable. Otherwise, the external microcontroller must wait.

Figure 8 describes the expected SPI protocol state machine on the external microcontroller.

Figure 8. **Expected SPI protocol state machine on external microcontroller**



DT79634V1

5 X-CUBE-WBA hardware configuration

5.1 STM32 Nucleo development board

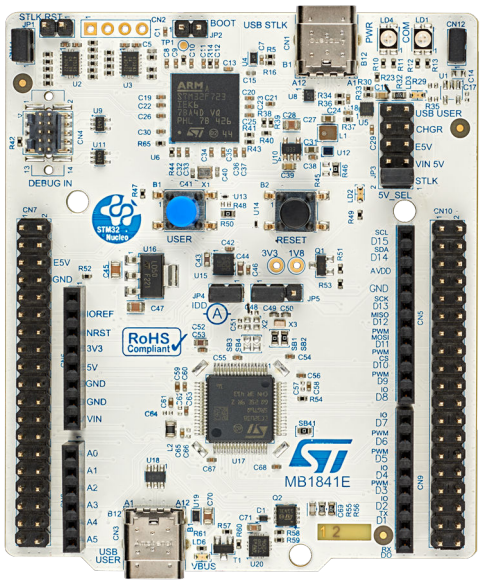
The STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller series.

The ARDUINO® Uno V3 connectivity support and the ST morpho headers allow the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized expansion boards.

The NUCLEO-U385RG-Q STM32 Nucleo board does not require separate probes as it integrates the STLINK-V3EC debugger/programmer.

The NUCLEO-U385RG-Q comes with the comprehensive STM32U3 software HAL library together with various packaged software examples for various IDEs such as IAR Embedded Workbench® (EWARM), Keil® MDK-ARM, and STMicroelectronics STM32CubeIDE.

Figure 9. STM32 Nucleo development board (NUCLEO-U385RG-Q)



DT76594V1

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

5.2 X-NUCLEO-WBA25A1 expansion board

The X-NUCLEO-WBA25A1 expansion board provides Bluetooth® Low Energy connectivity to the developer's applications. It can be plugged onto an STM32 Nucleo development board, such as the NUCLEO-U385RG-Q, through its ARDUINO® Uno V3 connectors. The coprocessor of the X-NUCLEO-WBA25A1 manages the complete Bluetooth® LE stack and protocols on its Arm® Cortex®-M33 core and programmable flash memory.

By default, the X-NUCLEO-WBA25A1 interfaces with the microcontroller of the development board via a UART, with or without hardware flow control. Full-duplex SPI with an interrupt line is also available. The firmware loaded on the coprocessor defines the host interface. To modify it, simply change the firmware; no modification of the hardware is required.

As shown in Table 7, the X-NUCLEO-WBA25A1 is composed of two printed circuit boards.

Table 7. X-NUCLEO-WBA25A1 board descriptions

Order code	Board reference	Description
X-NUCLEO-WBA25A1	MB2160	ARDUINO® interface board
	MB2293	MCU RF board

Table 8 shows how to connect the X-NUCLEO-WBA25A1 expansion board with the NUCLEO-U385RG-Q development board.

Table 8. ARDUINO® Uno V3 connection between X-NUCLEO-WBA25A1 and NUCLEO-U385RG-Q

Left connectors					Right connectors				
X-NUCLEO-WBA25A1 connector	Pin number	Pin name	X-NUCLEO-WBA25A1 signal	NUCLEO-U385RG-Q MCU port	X-NUCLEO-WBA25A1 signal	NUCLEO-U385RG-Q MCU port	Pin name	Pin number	X-NUCLEO-WBA25A1 connector
					-	-	D15	10	CN3 Digital
					-	-	D14	9	
					-	-	VDD	8	
					GND	GND	GND	7	
CN5 Power	1	-	-	-	PA8/ SPI_SCK	PA5/ SPI_SCK	D13	6	
	2	IOREF	IOREF	IOREF	PA10/ SPI_MISO	PA6/ SPI_MISO	D12	5	
	3	NRST	RSTN	-	PA9/ SPI_MOSI	PA7/ SPI_MOSI	D11	4	
	4	3V3	3V3	3V3	PA5/ SPI_CS	PC9/ SPI_CS	D10	3	
	5	5V	-	5V	-	-	D9	2	
	6	GND	GND	GND	-	-	D8	1	
	7	GND	GND	GND					
	8	VIN	-	VIN	RSTN	PA8	D7	8	CN4 Digital
					-	-	D6	7	
CN6 Analog	1	A0	PH3/ SPI_IRQ	PA0/ SPI_IRQ	-	-	D5	6	
	2	A1	-	-	-	-	D4	5	
	3	A2	-	-	-	-	D3	4	
	4	A3	-	-	-	-	D2	3	
	5	A4	-	-	PA12/ USART1_RX	PA2/ LPUART_TX	D1	2	
	6	A5	-	-	PA6/ USART1_TX	PA3/ LPUART_RX	D0	1	

Similar hardware setups can also be built for the NUCLEO-WBA55CG (STM32WBA55xx) and NUCLEO-WBA65RI (STM32WBA65xx) boards as described in Table 9 and Table 10 respectively. The hardware setups use the X-NUCLEO-WBA25A1 ARDUINO® interface board with the MCU RF board of the corresponding NUCLEO-WBA55CG and NUCLEO-WBA65RI boards.

Table 9. ARDUINO® Uno V3 connection between STM32WBA55xx hardware setup and NUCLEO-U385RG-Q

Left connectors					Right connectors				
STM32WBA55xx HW setup(1) connector	Pin number	Pin name	STM32WBA55xx HW setup(1) signal	NUCLEO-U385RG-Q MCU port	STM32WBA55xx HW setup(1) signal	NUCLEO-U385RG-Q MCU port	Pin name	Pin number	STM32WBA55xx HW setup(1) connector
					-	-	D15	10	CN3 Digital
					-	-	D14	9	
					-	-	VDD	8	
					-	-	GND	7	
CN5 Power	1	-	-	-	PB4/ SPI_SCK	PA5/ SPI_SCK	D13	6	
	2	IOREF	IOREF	IOREF	PB3/ SPI_MISO	PA6/ SPI_MISO	D12	5	
	3	NRST	RSTN	NRST	PA15/ SPI_MOSI	PA7/ SPI_MOSI	D11	4	
	4	3V3	3V3	3V3	PA12/ SPI_CS	PC9/ SPI_CS	D10	3	
	5	5V	-	5V	-	-	D9	2	
	6	GND	GND	GND	-	-	D8	1	
	7	GND	GND	GND					
	8	VIN	-	VIN	RSTN	PA8	D7	8	CN4 Digital
					-	-	D6	7	
CN6 Analog	1	A0	PH3/ SPI_IRQ	PA0/ SPI_IRQ	-	-	D5	6	
	2	A1	-	-	-	-	D4	5	
	3	A2	-	-	-	-	D3	4	
	4	A3	-	-	-	-	D2	3	
	5	A4	-	-	PA8/ USART1_RX	PA2/ LPUART_TX	D1	2	
	6	A5	-	-	PB12/ USART1_TX	PA3/ LPUART_RX	D0	1	

1. *NUCLEO-WBA55CG MCU RF board connected to X-NUCLEO-WBA25A1 ARDUINO® interface board.*

Table 10. ARDUINO® Uno V3 connection between STM32WBA65xx hardware setup and NUCLEO-U385RG-Q

Left connectors					Right connectors				
STM32WBA65xx HW setup(1) connector	Pin number	Pin name	STM32WBA65xx HW setup(1) signal	NUCLEO-U385RG-Q MCU port	STM32WBA65xx HW setup(1) signal	NUCLEO-U385RG-Q MCU port	Pin name	Pin number	STM32WBA65xx HW setup(1) connector
					-	-	D15	10	CN3 Digital
					-	-	D14	9	
					-	-	VDD	8	
					-	-	GND	7	
CN5 Power	1	-	-	-	PB10/ SPI_SCK	PA5/ SPI_SCK	D13	6	
	2	IOREF	IOREF	IOREF	PA9/ SPI_MISO	PA6/ SPI_MISO	D12	5	
	3	NRST	RSTN	NRST	PC3/ SPI_MOSI	PA7/ SPI_MOSI	D11	4	
	4	3V3	3V3	3V3	PB9/ SPI_CS	PC9/ SPI_CS	D10	3	
	5	5V	-	5V	-	-	D9	2	
	6	GND	GND	GND	-	-	D8	1	
	7	GND	GND	GND					
	8	VIN	-	VIN	RSTN	PA8	D7	8	CN4 Digital
					-	-	D6	7	
CN6 Analog	1	A0	PH3/ SPI_IRQ	PA0/ SPI_IRQ	-	-	D5	6	
	2	A1	-	-	-	-	D4	5	
	3	A2	-	-	-	-	D3	4	
	4	A3	-	-	-	-	D2	3	
	5	A4	-	-	PA8/ USART1_RX	PA2/ LPUART_TX	D1	2	
	6	A5	-	-	PB12/ USART1_TX	PA3/ LPUART_RX	D0	1	

1. *NUCLEO-WBA65RI MCU RF board connected to X-NUCLEO-WBA25A1 ARDUINO® interface board.*

5.3 Software description

The following software components are required as a suitable development environment to create applications for the STM32 Nucleo development board equipped with the Bluetooth® LE expansion board:

- X-CUBE-WBA: an STM32Cube Expansion Package for Bluetooth® LE network/radio coprocessor applications development.
- Development toolchain and compiler. X-CUBE-WBA supports the following environments:
  - IAR Embedded Workbench® for Arm® (EWARM) toolchain
  - STMicroelectronics STM32CubeIDE
  - STMicroelectronics STM32CubeProgrammer (STM32CubeProg)

5.4 Hardware setup

The following hardware components are required:

- One STM32 Nucleo development board, such as the NUCLEO-U385RG-Q.
- One expansion board setup based on the STM32WBA25CE coprocessor, such as the X-NUCLEO-WBA25A1, which is composed of an ARDUINO® interface board (MB2160) and an MCU RF board (MB2293). No dedicated Nucleo expansion boards are available for the STM32WBA55xx and STM32WBA65xx device variants.
  Alternatively, a NUCLEO-WBA25CE1 can be used. It is a Nucleo board that is also based on the STM32WBA25CE coprocessor. Connect it to the NUCLEO-U385RG-Q by wiring the required lines. Follow the selected hardware configuration (UART or SPI), as described in Section 5.2: X-NUCLEO-WBA25A1 expansion board.
  A similar approach is also applicable to the NUCLEO-WBA55CG based on the STM32WBA55CG device, and to the NUCLEO-WBA65RI based on the STM32WBA65RI device. The connections are described respectively in Table 9. ARDUINO® Uno V3 connection between STM32WBA55xx hardware setup and NUCLEO-U385RG-Q and Table 10. ARDUINO® Uno V3 connection between STM32WBA65xx hardware setup and NUCLEO-U385RG-Q.
- One USB Type-A to USB Micro-B cable to connect the STM32 Nucleo development board to a PC.
- A standard STLINK-V3 debugger such as the STLINK-V3SET.

5.5 Nucleo development board and Bluetooth® LE expansion board setup

The STM32 Nucleo boards integrate the ST-LINK debugger/programmer, such as the ST-LINK/V2-1, STLINK-V3E, or STLINK-V3EC. The X-NUCLEO-WBA25A1 expansion board can easily be connected to the STM32 Nucleo development board through the ARDUINO® Uno V3 extension connector. The STM32WBA25CE coprocessor communicates with the host microcontroller of the STM32 Nucleo development board through an SPI or UART link available on the ARDUINO® Uno V3 connector.

6 Reference

Table 11. **Reference**

Document	Location	Description
DS15003	*www.st.com*	STM32WBA2xxx datasheet
DS14127		STM32WBA5xxx datasheet
DS14736		STM32WBA6xxx datasheet
UM3608		X-NUCLEO-WBA25A1 expansion board user manual
UM3062		NUCLEO-U385RG-Q Nucleo-64 board user manual
UM3610		NUCLEO-WBA25CE1 Nucleo-64 board user manual
UM3301		NUCLEO-WBA55CG Nucleo-64 board user manual
UM3448		NUCLEO-WBA65RI Nucleo-64 board user manual

Revision history

Table 12. **Document revision history**

Date	Revision	Changes
10-Mar-2026	1	Initial release.

Contents

List of tables

List of figures