UM2463

User manual

Getting started with the X-CUBE-BLEMESH1 software expansion for STM32Cube

Inroduction

The X-CUBE-BLEMESH1 expansion software package for STM32Cube runs on the STM32 and provides easy-to-use networking APIs based on a Mesh profile library and a BLE stack.

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

The software lets you easily create your own application for extending BLE Mesh networks (by offering a ready-to-use Mesh core library), a complete set of compatible APIs, and a lighting reference design demo application running on the X-NUCLEO-IDB05A1 expansion board connected to a NUCLEO-L152RE, NUCLEO-L476RG or NUCLEO-F401RE development board.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
BLE	Bluetooth low energy
BSP	Board support package
CMSIS	Cortex® microcontroller software interface standard
GATT	Generic attribute profile
HAL	Hardware abstraction layer
SPI	Serial peripheral interface

2 What is STM32Cube?

STM32Cube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

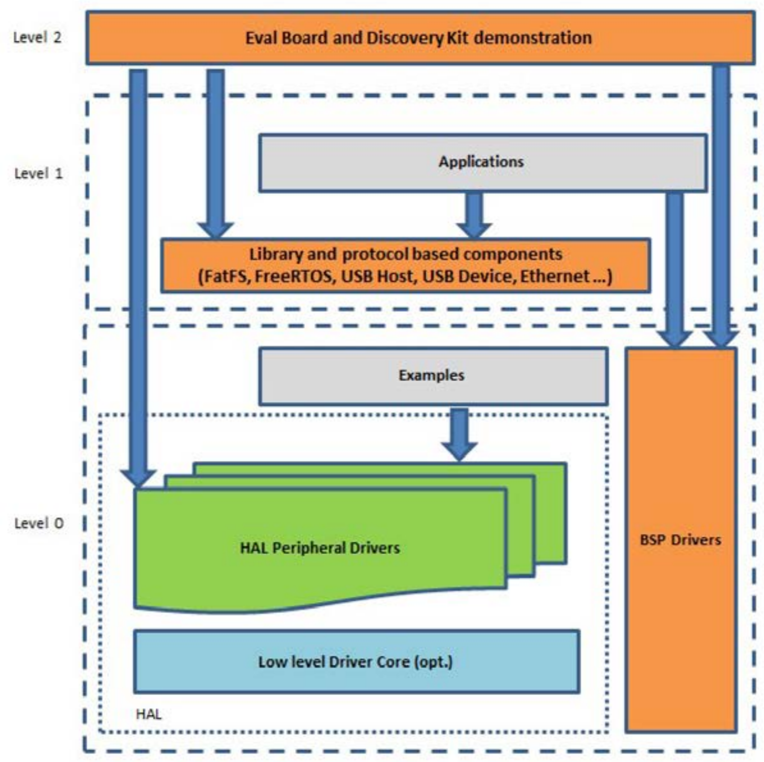
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32Cube for the STM32 series), which includes:
  – the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  – a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  – all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below.

Figure 1. Firmware architecture


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components
(FatFS, FreeRTOS, USB Host, USB Device, Ethernet ...)
Examples
Level 0
HAL Peripheral Drivers
BSP Drivers
Low level Driver Core (opt.)
HAL

Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…); it is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines. It is composed of two parts:

- Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
- BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().

- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I^2C, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-BLEMESH1 software expansion for STM32Cube

3.1 Overview

The X-CUBE-BLEMESH1 software package extends STM32Cube functionality and features:

- Complete software to build Mesh network with Bluetooth low energy (BLE) nodes, extending network coverage to large areas up to 32767 nodes and 126 hops
- BT SIG Mesh Profile 1.0 Certification
- Use of BLE enabled smartphones to monitor and control multiple BLE nodes via proxy protocol and legacy BLE GATT connectivity
- Two-layer security, thanks to 128-bit AES CCM encryption and 256-bit ECDH protocol, ensuring protection against multiple attacks, including Replay, Bit-Flipping, Eavesdropping, Man-in-the-Middle and Trashcan
- Generic Model, Lighting Model and Vendor model examples included
- Sample implementation available on the X-NUCLEO-IDB05A1 expansion board connected to a NUCLEO-L152RE, NUCLEO-L476RG, or NUCLEO-F401RE development board
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

It integrates BlueNRG-MS product in a powerful, range-extending BLE Mesh network with true full-duplex communication. The solution contains the core functionality for secure communication and provides the flexibility you need to build applications.

3.2 Architecture

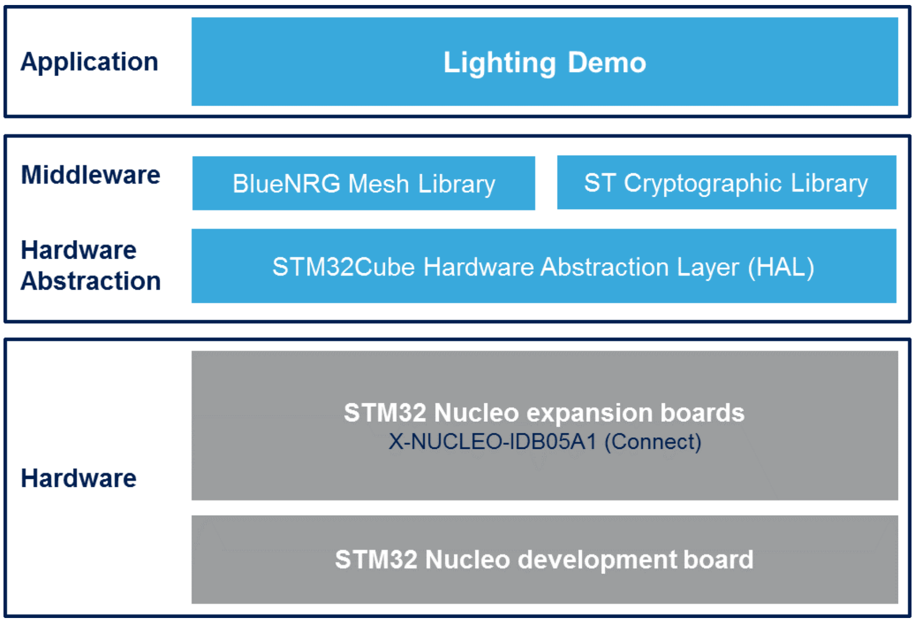
This fully compliant STM32Cube software expansion enables development of BLE-Mesh applications using the BlueNRG-MS device.

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller and extends STM32Cube by providing the middleware library for the BLE-Mesh application using the Bluetooth low energy expansion board.

The software layers used by the application software to access and use the Bluetooth low energy expansion board are:

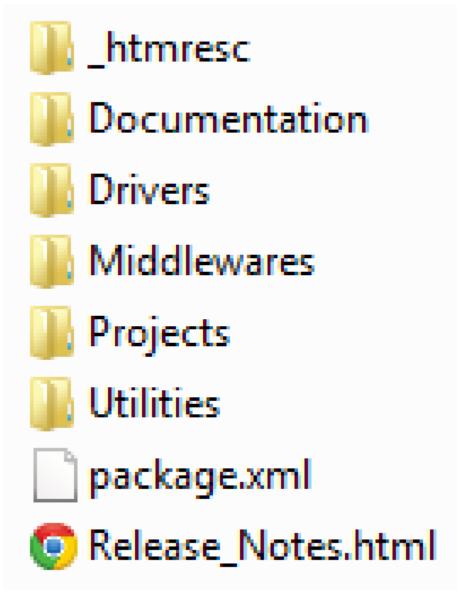
- **STM32Cube HAL layer**: provides a generic, multi-instance set of simple APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). These generic and extension APIs are based on a common architecture and the layers above them, like the middleware layer, can function without requiring specific hardware configuration data for a given microcontroller unit (MCU). This structure improves the library code reusability and guarantees easy portability across other devices.
- **Board support package (BSP) layer**: supports the peripherals on the STM32 Nucleo board, except for the MCU. This limited set of APIs provides a programming interface for certain board specific peripherals (like the user button, the reset button, etc.) and helps in identifying the specific board version.
- **Middleware**: includes the Mesh model implementation as source code and Mesh profile implementation as library. The Mesh models include generic model, lighting model and vendor model implementation examples. The crypto library is also used for encryption and decryption.

Figure 2. **X-CUBE-BLEMESH1 architecture**


Application
Lighting Demo
Middleware
BlueNRG Mesh Library
ST Cryptographic Library
Hardware Abstraction
STM32Cube Hardware Abstraction Layer (HAL)
Hardware
STM32 Nucleo expansion boards
X-NUCLEO-IDB05A1 (Connect)
STM32 Nucleo development board

3.3 Folder structure

Figure 3. **X-CUBE-BLEMESH1 package folder structure**



The software is packaged in the following main folders:

- **Documentation**: with a compiled HTML file generated from the source code that details the software components and APIs.
- **Drivers**: contains the HAL drivers, the board specific drivers for each supported board or hardware platform, including the on-board components ones and the CMSIS layer which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.
- **Middlewares**: the Mesh model implementation as source code and Mesh profile implementation as library. The Mesh models include generic model, lightness model and vendor model implementation examples. The crypto library is also used for encryption and decryption.

- **Projects**: a sample application used for a lighting application demo on BLE-Mesh. The projects are provided for the NUCLEO-L152RE, NUCLEO-L476RG and NUCLEO-F401RE platforms with three development environments (IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), and System Workbench for STM32 (SW4STM32).
- **Utilities**: contains three different folders called “STM32F4_MAC”, “STM32L4_MAC”, “STM32L1_MAC” which provide some examples of the external MAC address.

3.4 APIs

Detailed function and parameter descriptions for the user-APIs are compiled in an HTML file in the software package Documentation folder.

3.5 Sample application description

The function APIs available for application development based on Mesh network over Bluetooth Low Energy devices are described below.

The Mesh over BLE library features:

- Mesh network creation between nodes
- unicast, multicast (group), broadcast addressing
- relay feature management: all the packets whose destination address is for another node are re-transmitted
- communication with devices for advanced features, such as provisioning and proxy service

The user application handles:

- Mesh stack initialization
- user callbacks required for the application
- application handling

3.5.1 Initialization of application callbacks

An example application using the X-NUCLEO-IDB05A1 expansion board with either NUCLEO-L152RE, NUCLEO-L476RG and NUCLEO-F401RE boards is provided in the “Projects” directory. Ready-to-build projects are available for multiple IDEs.he application starts by initializing the callbacks required for the different events and functionalities.

These callbacks are used in the BlueNRG-Mesh library to call the functions based on specific events or the library state machine.

```
const MOBLE_VENDOR_CB_MAP vendor_cb =
{
  Vendor_WriteLocalDataCb,
  Vendor_ReadLocalDataCb,
  Vendor_OnResponseDataCb
};

const Appli_Vendor_cb_t VendorAppli_cb =
{
  /*Vendor Commads*/
  Appli_Vendor_LEDControl,
  Appli_Vendor_DeviceInfo,
  Appli_Vendor_Test,
  Appli_LedCtrl
};

/* Callbacks used by BlueNRG-Mesh library */
BluenrgMesh_SetVendorCbMap(&vendor_cb);
```

The structure MOBLE_VENDOR_CB_MAP is used to initialize the vendor model for the application implementation. The function `BluenrgMesh_SetVendorCbMap(&vendor_cb);` is used to initialize the vendor callbacks in the library.

3.5.2 Initialization and main application loop

To develop an application for Mesh over BLE on the BlueNRG-MS platform follow the procedure below.

Step 1. Call the `InitDevice()` API.

It automatically calls the `SystemInit()` API to initialize the device vector table, interrupt priorities and clock.

Step 2. Call the `Appli_CheckBdMacAddr()` API to check the validity of the MAC address.

If the MAC address is not valid, the firmware is stuck in while (1) loop with the LED blinking continuously.

Step 3. Initialize the hardware callback functions for the BLE hardware by updating `MOBLE_USER_BLE_CB_MAP user_ble_cb =`.

```
{
  Appli_BleStackInitCb,
  Appli_BleSetTxPowerCb,
  Appli_BleGattConnectionCompleteCb,
  Appli_BleGattDisconnectionCompleteCb,
  Appli_BleUnprovisionedIdentifyCb,
  Appli_BleSetUUIDCb,
  Appli_BleSetNumberOfElementsCb
};
```

Step 4. To create an application interface for BLE radio initialization and TxPower configuration:

Step 4a. initialize GATT connection and disconnection callbacks for the application interface

Step 4b. call `BluenrgMesh_BleHardwareInitCallBack(&user_ble_cb)` to complete the initialization of hardware callbacks

Step 5. Initialize the Mesh library by calling `BluenrgMesh_Init(&BLEMeshlib_Init_params).`

If an error occurs, the demo firmware shows the message *Could not initialize BlueNRG-Mesh library!* on the terminal window, making the LED blink continuously. The terminal interface can be opened using the VCOM port created by the USB connection available on the boards.

Step 6. Check whether the device has been provisioned with the `BluenrgMesh_IsUnprovisioned()` API. A provisioned device has network keys and other parameters configured in the internal flash memory.

If the node is unprovisioned, the `BluenrgMesh_InitUnprovisionedNode()` API initializes it, otherwise it helps to initialize the device.

Step 7. Print the messages to the terminal window for the nodes that are being initialised.

The message also prints the MAC address assigned to the node.

Step 8. Initialize the BlueNRG-Mesh models using the `BluenrgMesh_ModelsInit()` API.

Step 9. To initialize the node to the unprovisioned state, hold down the user button for at least 5 seconds.

It erases all the network parameters configured in the device internal memory. Once unprovisioning is complete, you have to reset the board.

Step 10. Call the `BluenrgMesh_Process()` in while(1) loop as frequently as possible, so that it calls `BLE_StackTick()` internally to process BLE communication.

Other APIs called in while(1) loop are `BluenrgMesh_ModelsProcess()` and `Appli_Process().`

Any application implementation is performed in the state-machine by non-blocking functions with frequent calls to `BluenrgMesh_Process().`

Step 11. Check for user inputs or buttons regularly for any action to take.

3.5.3 Node transmit power setup

The `Appli_BleSetTxPowerCb()` calls the aci function to set the power `aci_hal_set_tx_power_level(uint8_t En_High_Power, uint8_t PA_Level).`

By default, +4 dbm is configured, but it can be changed by the user.

The following different settings are available in the firmware:

```
 /* MACROS for Power Level definitions */
#define POWER_LEVEL_LOW              0
```

```
#define TX_POWER_LEVEL_MINUS_18DBM 0 // = -18 dBm,
#define TX_POWER_LEVEL_MINUS_15DBM 1 // = -15 dBm,
#define TX_POWER_LEVEL_MINUS_12DBM 2 // = -12 dBm,
#define TX_POWER_LEVEL_MINUS_9DBM  3 // = -9 dBm,
#define TX_POWER_LEVEL_MINUS_6DBM  4 // = -6 dBm,
#define TX_POWER_LEVEL_MINUS_2DBM  5 // = -2 dBm,
#define TX_POWER_LEVEL_0DBM        6 // =  0 dBm,
#define TX_POWER_LEVEL_PLUS_5DBM   7 // =  5 dBm.
#define POWER_LEVEL_HIGH           1
#define TX_POWER_LEVEL_MINUS_14DBM 0 // = -14 dBm,
#define TX_POWER_LEVEL_MINUS_11DBM 1 // = -11 dBm,
#define TX_POWER_LEVEL_MINUS_8DBM  2 // = -8 dBm,
#define TX_POWER_LEVEL_MINUS_5DBM  3 // = -5 dBm,
//#define TX_POWER_LEVEL_MINUS_2DBM  4 // = -2 dBm,
#define TX_POWER_LEVEL_PLUS_2DBM   5 // =  2 dBm,
#define TX_POWER_LEVEL_PLUS_4DBM   6 // =  4 dBm,
#define TX_POWER_LEVEL_PLUS_8DBM   7 // =  8 dBm
```

3.5.4 Firmware UART interface

The boards can be connected to a PC via USB. Any terminal software (Hercules, Putty, etc.) can be used to open the serial communication port on the PC to check the messages from the board.

The UART of the controller on the board is connected to the PC via a VCOM (virtual communication) port that can be opened with the following settings:

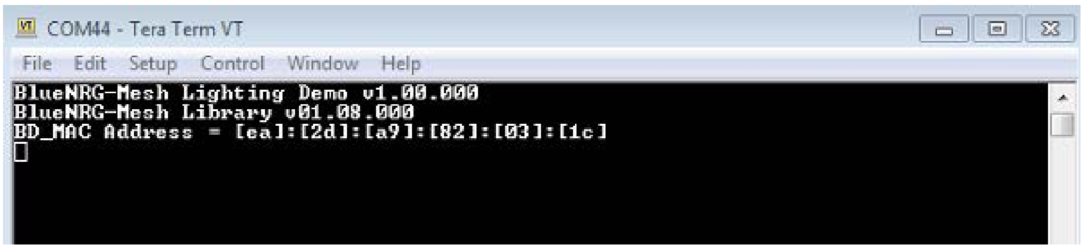
- Baud: 115200
- data size: 8
- parity: none
- stop bits: 1
- no hardware control

From the firmware main.c file, you can see certain messages are printed to the VCOM by the following code:

```
#if !defined(DISABLE_TRACES)
/* Prints the MAC Address of the board */
printf("BlueNRG-Mesh Lighting Demo v%s\n\r", BLUENRG_MESH_APPLICATION_VERSION);
printf("BlueNRG-Mesh Library v%s\n\r", BluenrgMesh_GetLibraryVersion());
printf("BD_MAC Address = [%02x]:[%02x]:[%02x]:[%02x]:[%02x]:[%02x] \n\r",
bdaddr[5],bdaddr[4],bdaddr[3],bdaddr[2],bdaddr[1],bdaddr[0] );
#endif
```

After the board is connected and the terminal window is opened, press the reset button. If the firmware starts successfully, the following messages appear in the virtual com window.

Figure 4. **Virtual com window after startup**



```
BlueNRG-Mesh Lighting Demo v1.00.000
BlueNRG-Mesh Library v01.08.000
BD_MAC Address = [ea]:[2d]:[a9]:[82]:[03]:[1c]
```

3.5.5 GATT connection/disconnection

Each node in the network can estabilsh a connection with the smartphone through GATT interface. When connected, the node becomes a proxy to bridge the messages and responses between the Mesh network and the smartphone.

The detection of connection and disconnection with the smartphone is managed by the following callbacks, initialized during the main loop:

- `Appli_BleGattConnectionCompleteCb;`

- `Appli_BleGattDisconnectionCompleteCb;`

During provisioning, the GATT connection is established with the node which needs to be provisioned. If the smartphone moves out of the range of the proxy node, it establishes a new connection with another available node.

3.6 External MAC address utilities

By default, a static random address is generated internally in each node. For specific usage or testing, you can program the nodes with a known unique MAC address. This external address is stored at a specific location in the flash memory of the MCU. Some examples of such MAC addresses are available in the folders (Utilities/STM32F4_MAC, Utilities/STM32L4_MAC, and Utilities/STM32L1_MAC) depending on the MCU.

To use an external MAC address, it is necessary to uncomment the EXTERNAL_MAC_ADDR_MGMT macro in mesh_cfg.h file in the middleware.

Demo application firmware and MAC address are flashed independently, that is, you don't have to update a MAC address if another firmware has been flashed.

The MAC address has to be unique in each node for the Mesh network.

When using an external MAC address, it is recommended to do full erase and then program:

1. STM32L1_BNRG_MS_Lighting_Demo_Ext_MAC.bin file in Binary/STM32L152RE-Nucleo folder, and then program the MAC addresses from Utilities/STM32L1_MAC (for NUCLEO-L152RE)
2. STM32F4_BNRG_MS_Lighting_Demo_Ext_MAC.bin file in Binary/STM32F401RE-Nucleo folder and then program the MAC addresses from Utilities/STM32F4_MAC (for NUCLEO-F401RE)
3. STM32L4_BNRG_MS_Lighting_Demo_Ext_MAC.bin file in Binary/STM32L476RG-Nucleo folder and then program the MAC addresses from Utilities/STM32L4_MAC (for NUCLEO-L476RG)

4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo platform

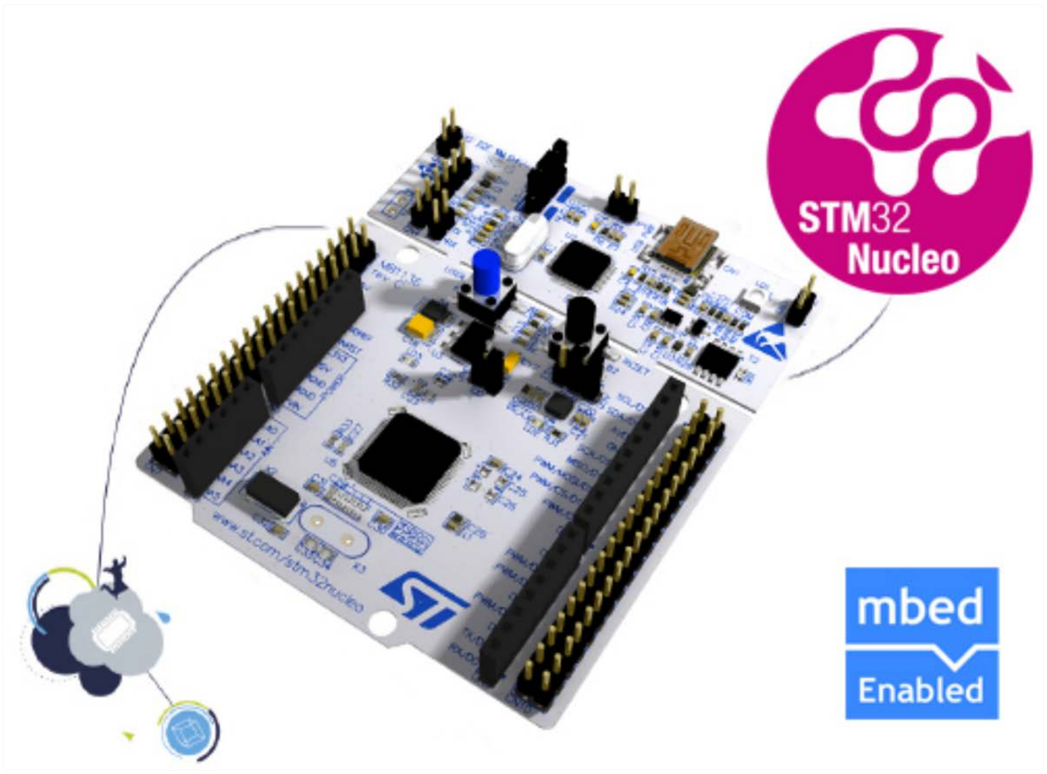
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

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/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Figure 5. STM32 Nucleo board



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

4.1.2 X-NUCLEO-IDB05A1 expansion board

The X-NUCLEO-IDB05A1 is a Bluetooth low energy expansion board based on the SPBTLE-RF RF module, built around the BlueNRG-MS network processor, to allow expansion of the STM32 Nucleo boards. The SPBTLE-RF module is FCC (FCC ID: S9NSPBTLERF) and IC certified (IC: 8976C-SPBTLERF). The BlueNRG-MS is a very low power Bluetooth low energy (BLE) single-mode network processor, compliant with Bluetooth specification v4.2. X-NUCLEO-IDB05A1 is compatible with the ST morpho and Arduino™ UNO R3 connector layout. This expansion board can be plugged into the Arduino UNO R3 connectors of any STM32 Nucleo board.

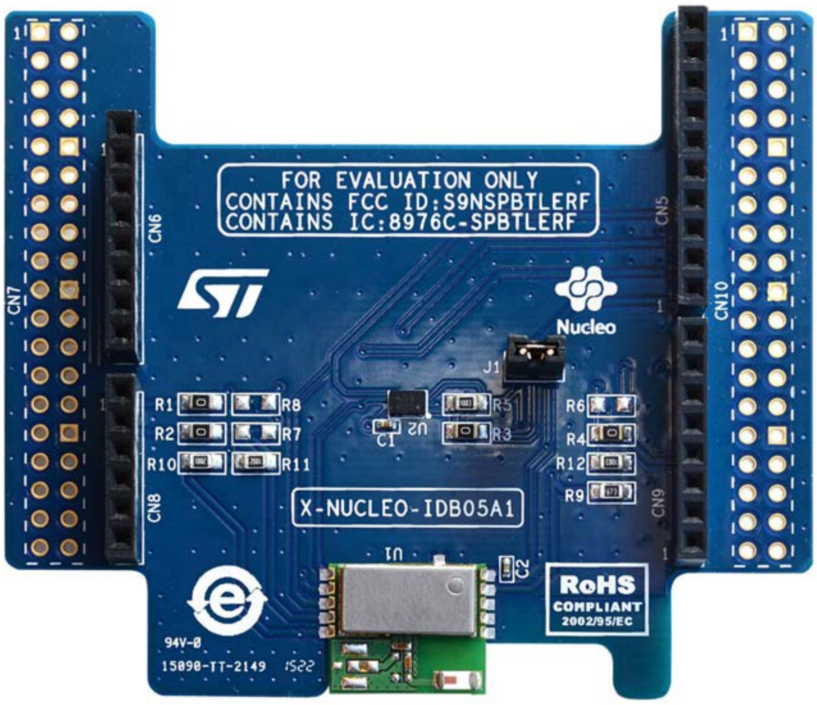


Figure 6. **X-NUCLEO-IDB05A1 expansion board**

Information about the X-NUCLEO-IDB05A1 expansion board is available on www.st.com at http://www.st.com/x-nucleo

4.2 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo development platform (suggested order code: either NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L152RE)
2. One BLE expansion board (order code: X-NUCLEO-IDB05A1)
3. One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

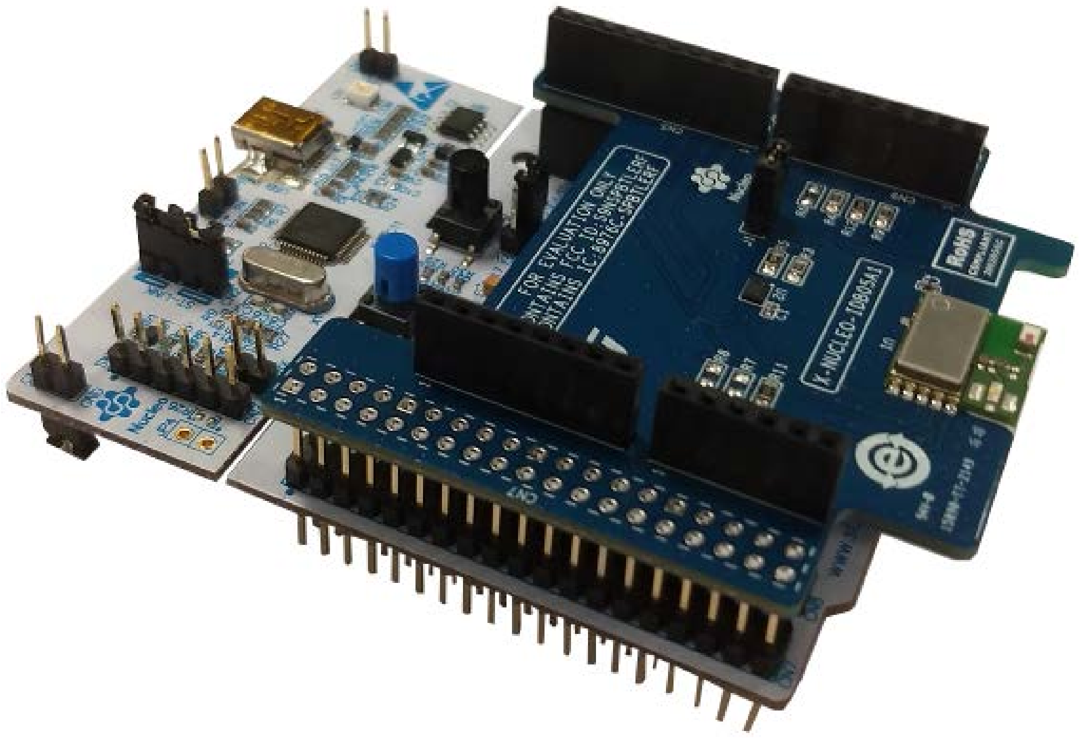
4.2.1 STM32 Nucleo and BLE expansion board setup

The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer.

The developer can download the ST-LINK/V2-1 USB driver by looking for the STSW-LINK009 software on www.st.com.

The X-NUCLEO-IDB05A1 BLE expansion board interfaces the external STM32 microcontroller on STM32 Nucleo via SPI. It can be easily connected to the STM32 Nucleo through the Arduino UNO R3 extension connector as shown below.

Figure 7. X-NUCLEO-IDB05A1 expansion board plus STM32 Nucleo



4.3 Software setup

The following software components are needed for a suitable development environment for creating applications for the STM32 Nucleo equipped with the BLE expansion board:

- X-CUBE-BLEMESH1 expansion for STM32Cube dedicated to Bluetooth Mesh application development. The X-CUBE-BLEMESH1 firmware and related documentation is available on www.st.com.
- One of the following development tool-chain and compilers:
  - Keil RealView Microcontroller Development Kit (MDK-ARM-STM32) + ST-LINK
  - IAR Embedded Workbench for ARM (IAR-EWARM) + ST-LINK
  - OpenSTM32 System Workbench for STM32 (SW4STM32) + ST-LINK

A References

1. Mesh over Bluetooth Low Energy
   https:www.st.com/en/embedded-software/stsw-bnrg-mesh.html
2. Bluetooth Mesh Networking Specifications
   https://www.bluetooth.com/specifications/mesh-specifications
3. Bluetooth Mesh Model Specification
   www.bluetooth.com/specifications/adopted-specifications

Revision history

Table 2. **Document revision history**

Date	Version	Changes
24-Aug-2018	1	Initial release.

Contents

List of tables

List of figures

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2018 STMicroelectronics – All rights reserved