Getting started with FP-SNS-BLEMESH1 function pack for IoT node with BLE Mesh connectivity and sensor model

Introduction

FP-SNS-BLEMESH1 is an STM32Cube function pack which lets you connect BLE nodes to a smartphone via BLE, through a suitable Android™ or iOS™ application, to view real-time environmental and motion sensor data using the BLE Mesh sensor model.

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 sensor demo application running on the X-NUCLEO-IDB05A1 and X-NUCLEO-IKS01A3 expansion boards connected to a NUCLEO-L476RG development board, on the X-NUCLEO-BNRG2A1 and X-NUCLEO-IKS01A3 connected to a NUCLEO-L476RG development board, and on SensorTile.box and SensorTile.box PRO.

The software runs on the STM32 microcontroller and includes all the necessary drivers to recognize the devices on the STM32 Nucleo development board and expansion boards.

1 Acronyms and abbreviations

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

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.

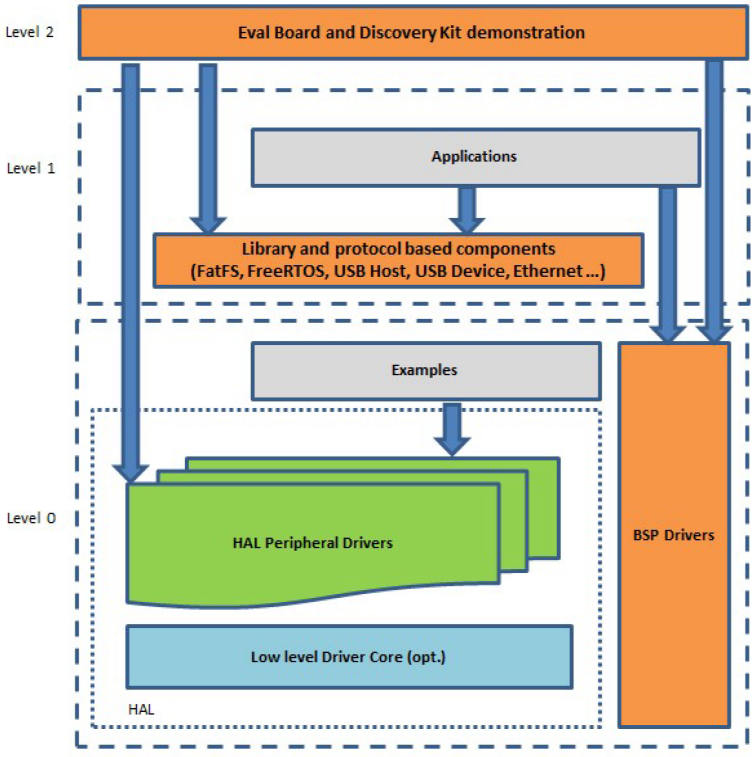
STM32Cube version 1.x includes:

- 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 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 (I²C, 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 FP-SNS-BLEMESH1 software expansion for STM32Cube

3.1 Overview

The FP-SNS-BLEMESH1 software package expands STM32Cube functionality.

The key features of the package are:

- Complete software to build Mesh network with Bluetooth low energy (BLE) nodes supporting the BLE Mesh sensor model, defined in BLE Mesh specification V1.0
- Environmental and motion sensors values read by the MCU on the STM32 Nucleo board can be sent to virtual communication port when the board is connected to a computer. The values can be read through the BlueNRG-Mesh Android and iOS app using sensor model.
- Compatible with BLE enabled smartphones to monitor and control multiple BLE nodes, using proxy protocol and legacy BLE GATT connectivity
- Two-layer security, thanks to 128-bit AES CCM encryption and 256-bit ECDH protocol, ensuring protection from multiple attacks, including Replay, Bit-Flipping, Eavesdropping, Man-in-the-Middle and Trashcan
- Sample implementation available on:
  - X-NUCLEO-IDB05A1 board and X-NUCLEO-IKS01A3 connected to a NUCLEO-L476RG development board
  - X-NUCLEO-BNRG2A1 and X-NUCLEO-IKS01A3 connected to a NUCLEO-L476RG development board
  - SensorTile.box
  - SensorTile.box PRO
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

The function pack software includes the temperature, humidity, pressure and motion sensor drivers for the HTS221, LPS22HB, LSM6DSL and LSM303AGR devices when the X-NUCLEO-IKS01A3 expansion board is mounted on STM32 Nucleo.

The package is compatible with the BlueNRG-Mesh Android/iOS application available at GooglePlay/iTunes stores, which can be used to display information sent via BLE.

It integrates BlueNRG products with embedded BLE communication in a powerful, range-extending Mesh network with real full-duplex communication and provides the flexibility necessary to build your own application.

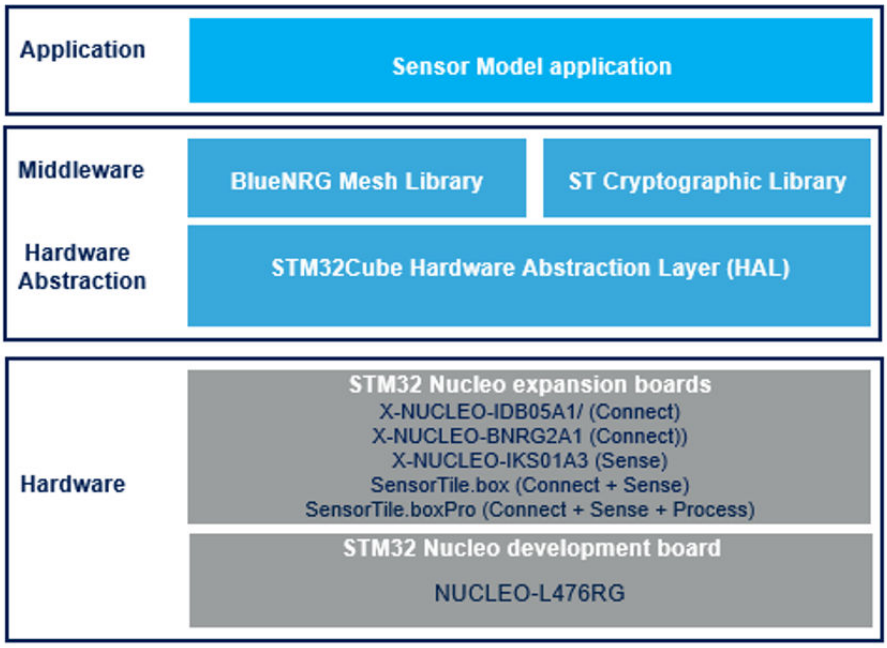
3.2 Architecture

The software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a Board Support Package (BSP) to enable development of applications using inertial and environmental sensors.

The software layers used by the application software to access and use the expansion boards are:

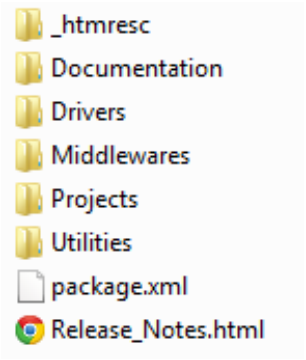
- the **STM32Cube HAL layer**, which provides a simple, generic, multi-instance set of application programming interfaces (APIs) to interact with the upper application, library and stack layers. It has generic and extension APIs and is directly built around a generic architecture and allows successive layers like the middleware layer to implement functions without requiring specific hardware configurations for a given microcontroller unit (MCU). This structure improves library code reusability and guarantees an easy portability on other devices.
- the **board support package** (BSP) layer supports all the peripherals on the STM32 Nucleo except the MCU. This limited set of APIs provides a programming interface for certain board-specific peripherals like the LED, the user button, etc. This interface also helps in identifying the specific board version.

Figure 2. **FP-SNS-BLEMESH1 software architecture**



3.3 Folder structure

Figure 3. **FP-SNS-BLEMESH1 package folder structure**



The following folders are included in the software package:

- **Documentation**: contains a compiled HTML file generated from the source code which details the software components and APIs.
- **Drivers**: contains the HAL drivers and the board-specific drivers for each supported board or hardware platform, including the on-board components and the CMSIS vendor-independent hardware abstraction layer for ARM Cortex-M processor series.
- **Middlewares**: contains libraries and protocols related to the serial communication of sensor data with a connected PC application.
- **Projects**: contains a sample application used to access sensors data, provided for the NUCLEO-L476RG, SensorTile.box and SensorTile.box PRO platforms with three development environments, IAR Embedded Workbench for ARM (IAR-EWARM), RealView Microcontroller Development Kit (MDK-ARM-STM32), and System Workbench for STM32 (SW4STM32).
- **Utilities**: contains three different folders called *STM32F4_MAC*, *STM32L4_MAC STM32L1_MAC* where an external MAC address is provided.

3.4 APIs

Detailed technical information with full user API function and parameter description are in a compiled HTML file in the “Documentation” folder.

3.5 Sample application description

3.5.1 Initialization of application callbacks

A sample application using the X-NUCLEO-IDB05A1 and X-NUCLEO-IKS01A3 expansion boards with NUCLEO-L476RG boards is provided in the “Projects” directory. Ready-to-build projects are available for multiple IDEs.

This application starts by initializing the callbacks required for the different events and functionalities. The callbacks are used in the BlueNRG-Mesh library to call the functions based on specific events or by the Mesh library state machine.

```
const MODEL_Sensor_cb_t Model_Sensor_cb =
{
  SensorModelServer_GetOpcodeTableCb,
  SensorModelServer_GetStatusRequestCb,
  SensorModelServer_ProcessMessageCb
};
```

```
const Appli_Sensor_cb_t SensorAppli_cb =
{
  /* Sensor Model callbacks */
  Appli_Sensor_Cadence_Set,
  Appli_Sensor_Data_Status,
  Appli_Sensor_Descriptor_Status
```

```
/* Callbacks used by BlueNRG-Mesh library */
BluenrgMesh_SetModelSensorCbMap(&Model_Sensor_cb);
```

The structure `MODEL_Sensor_cb_t` is used to initialize the sensor model for the application implementation. The function `BluenrgMesh_SetModelSensorCbMap(&Model_Sensor_cb)` is used to initialize the different callbacks in the library.

3.5.2 Initialization and main application loop

This procedure develops an application for Mesh over BLE on BlueNRG platforms.

Step 1. Call the `InitDevice()` API, which 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 and the LED continuously blinks.

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_BleSetProductInfoCB,
Appli_BleSetNumberOfElementsCb,
Appli_BleDisableFilterCb
};
```

Step 4. To rely on an application interface for BLE radio initialization and TxPower configuration, initialize GATT connection and disconnection callbacks for the application interface.

Step 5. Call `BluenrgMesh_BleHardwareInitCallBack(&user_ble_cb)` to complete the initialization of hardware callbacks.

Step 6. Initialize the BlueNRG-Mesh library by calling `BluenrgMesh_Init(&BLEMeshlib_Init_params)`.

If an error occurs, the demo firmware prints a message (*Could not initialize BlueNRG-Mesh library!*) on the terminal window opened for the VCOM port created by the USB connection available on the boards making the LED blink continuously.

Step 7. Check whether the device has been provisioned or not.

A provisioned device has network keys and other parameters configured in the internal flash memory. Checks can be performed with the `BluenrgMesh_IsUnprovisioned()` API. If the node is unprovisioned, `BluenrgMesh_InitUnprovisionedNode()` API initializes it. If the device is already provisioned , the `BluenrgMesh_InitprovisionedNode()` API helps to initialize the device.

Step 8. Print the messages to the terminal window for the nodes that are being initialized. The message also prints the MAC address assigned to the node.

Step 9. Initialize the BlueNRG-Mesh models using the `BluenrgMesh_ModelsInit()` API.

Step 10. To initialize the node to the unprovisioned state, hold down the user button.

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

Step 11. Initialize the sensors mounted on X-NUCLEO-IKS01A3. SensorTile.box and SensorTile.box PRO by calling `MX_X_CUBE_MEMS1_Init()/InitMemsSensors()`.

The application must call `BluenrgMesh_Process()` in while(1) loop as frequently as possible. This function calls `BLE_StackTick()` internally to process BLE communication. The `BluenrgMesh_ModelsProcess()` (model processing) and `Appli_Process()` APIs are also called in while(1) loop.
Any application implementation is performed in the state-machine by non-blocking functions with frequent calls to `BluenrgMesh_Process()`.

Step 12. Check for user inputs or buttons for any action to take.

3.5.3 GATT connection/disconnection node

Each node in the network can connect to a smartphone through GATT interface. When this connection is established, the node becomes a proxy which acts as a bridge between the Mesh network commands and the smartphone responses.

You can detect the smartphone connection and disconnection by the following callbacks:

- `Appli_BleGattConnectionCompleteCb;`
- `Appli_BleGattDisconnectionCompleteCb;`

These are initialized during the main loop.

During provisioning, the GATT connection is established with the node which needs to be provisioned.

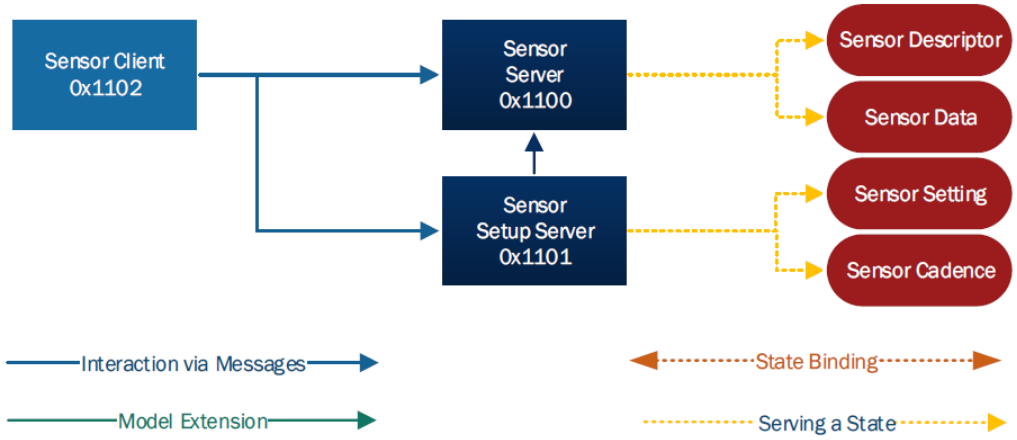
Note: If the smartphone moves out of the proxy node range, it establishes a new connection with the available node.

3.5.4 Sensor model

The sensor model shows:

- interactions via messages between client models and server models (represented respectively by blue and dark blue rectangles in the figure below)
- hierarchy of models extending other models
- server models serving states (represented by red rounded rectangles in the figure below)
- bindings between states

Figure 4. **Sensor Get and Sensor Status message packet structure (refer to Mesh model specifications)**



Sensor Get is an acknowledged message used to get the sensor data state. The response to the **Sensor Get** message is a **Sensor Status** message.

Table 1. **Sensor Get message structure**

Field	Size (octets)	Notes
Property ID	2	Property of the sensor (optional)

Sensor Status is an unacknowledged message used to report the sensor data state of an element.

Table 2. **Sensor Status message structure**

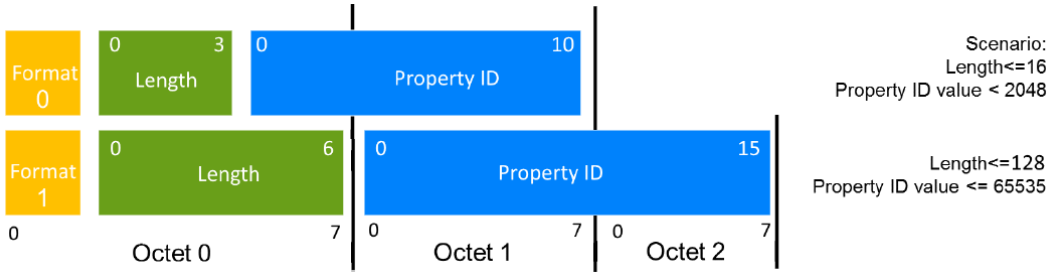
Field	Size (octets)	Notes
Marshalled sensor data	Variable	Sensor data state (optional)

The message is sent as a response to the **Sensor Get** message.

The marshalled sensor data field represents the marshalled sensor data state.

Special marshalling is used to facilitate forward compatibility and to optimize the message payload.

Figure 5. **Marshalling to optimize message payload**



Sensor Data state is a sequence of one or more pairs of **Sensor Property ID** and raw value fields: each raw value field size and representation is defined by the **Sensor Property ID** characteristics.

Table 3. FP-SNS-BLEMESH1: Sensor Property ID and raw values

Field	Size (octets)	Notes
Property ID 1	2	ID of the 1st device property of the sensor
Raw value 1	Variable	Raw value field with a size and representation defined by the 1st device property
Property ID 2	2	ID of the 2nd device property of the sensor
Raw value 2	Variable	Raw value field with a size and representation defined by the 2nd device property
...		
Property ID n	2	ID of the nth device property of the sensor
Raw value n	Variable	Raw value field with a size and representation defined by the nth device property

Sensor Property ID is a 2-octet value referencing a device property that describes the meaning and the format of data reported by a sensor.

Table 4. FP-SNS-BLEMESH1: sensors and related Property ID (standard or custom)

Sensor name	Property ID
TEMPERATURE_PID	0x0071
PRESSURE_PID	0x2A6D
HUMIDITY_PID	0x2A6F
MAGNETOMETER_PID	0x2AA1
ACCELEROMETER_PID	0x2BA1
GYROSCOPE_PID	0x2BA2

3.5.5 External MAC address utilities

The **Utilities** folder contains three different folders called STM32F4_MAC, STM32L4_MAC and STM32L1_MAC in which a hex file of an external MAC address is provided.

To use this address, you have to uncomment the `EXTERNAL_MAC_ADDR_MGMT` macro in the mesh_cfg.h file in the **Middleware** folder.

Demo application firmware and MAC address are flashed independently, that is, you do not have to update the firmware if the other firmware has already been flashed.

The MAC address is flashed just the first time and at every full chip erase.

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.

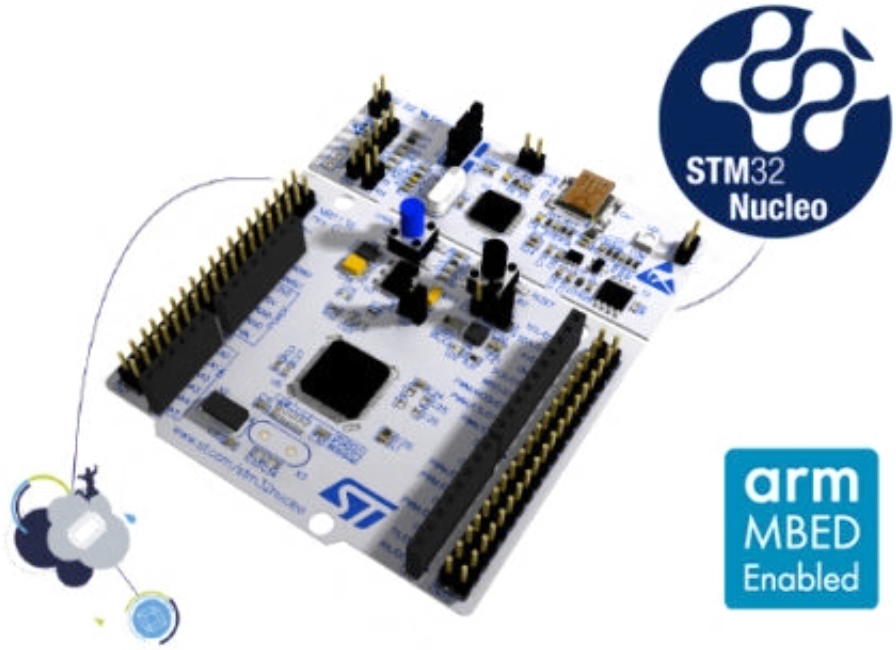
The Arduino connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 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.

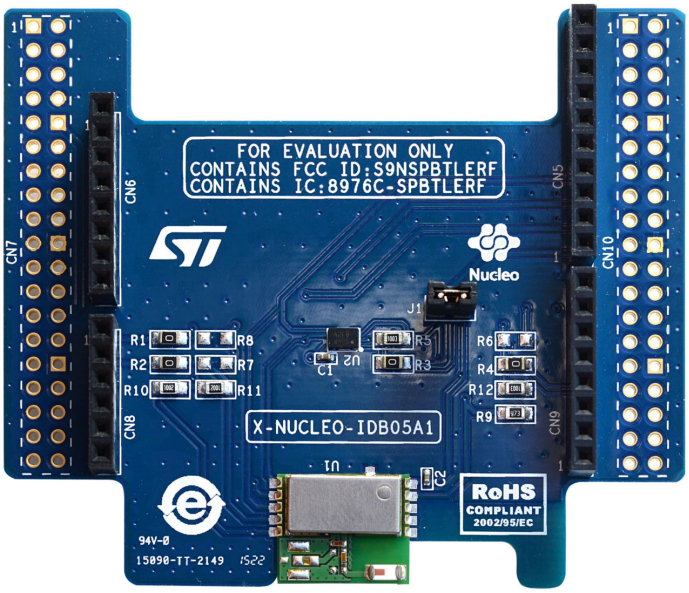
Figure 6. STM32 Nucleo board



4.1.2 X-NUCLEO-IDB05A1 expansion board

The X-NUCLEO-IDB05A1 is a Bluetooth low energy evaluation board based on the SPBTLE-RF BlueNRG-MS RF module 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 7. X-NUCLEO-IDB05A1 expansion board



4.1.3 X-NUCLEO-BNRG2A1 expansion board

The X-NUCLEO-BNRG2A1 expansion board provides Bluetooth low energy connectivity for developer applications and can be plugged onto an STM32 Nucleo development board (e.g., NUCLEO-L476RG with ultra-low power STM32 microcontroller) through its Arduino UNO R3 connectors. The expansion board features the Bluetooth® v5.0 compliant and FCC certified BlueNRG-M2SP application processor module based on the ST BlueNRG-2 System-on-Chip. This SoC manages the complete Bluetooth low energy stack and protocols on its Cortex-M0 core and programmable Flash, which can accommodate custom applications developed using the SDK. The BlueNRG-M2SP module supports master and slave modes, increased transfer rates with data length extension (DLE), and AES-128 security encryption.

The X-NUCLEO-BNRG2A1 interfaces with the STM32 Nucleo microcontroller via SPI connections and GPIO pins, some of which can be configured by the hardware.

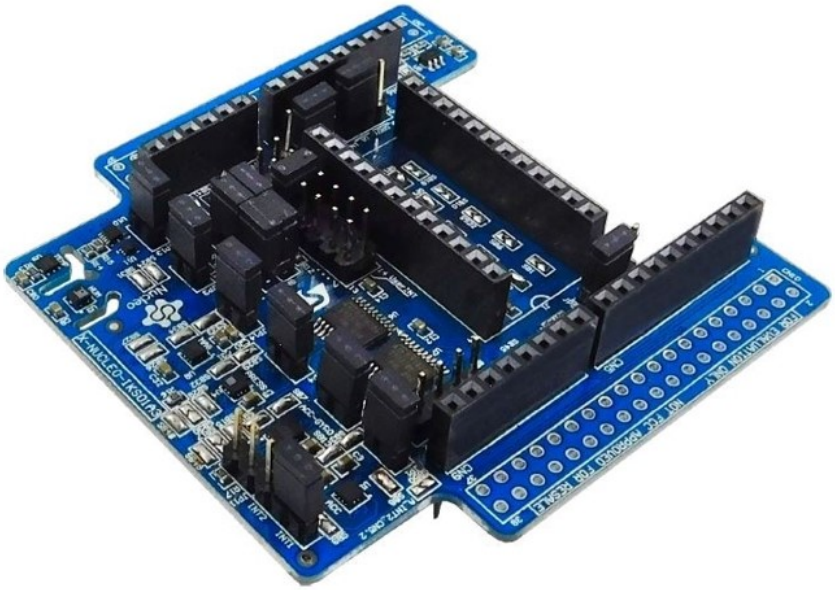
Figure 8. X-NUCLEO-BNRG2A1 expansion board

4.1.4 X-NUCLEO-IKS01A3 expansion board

The X-NUCLEO-IKS01A3 is a motion MEMS and environmental sensor evaluation board system. It is compatible with the Arduino UNO R3 connector layout and features the LSM6DSO 3-axis accelerometer + 3-axis gyroscope, the LIS2MDL 3-axis magnetometer, the LIS2DW12 3-axis accelerometer, the HTS221 humidity and temperature sensor, the LPS22HH pressure sensor, and the STTS751 temperature sensor.

The X-NUCLEO-IKS01A3 interfaces with the STM32 microcontroller via the I²C pin, and it is possible to change the default I²C port

Figure 9. X-NUCLEO-IKS01A3 expansion board



4.1.5 STEVAL-MKSBOX1V1

The STEVAL-MKSBOX1V1 (SensorTile.box) is a ready-to-use box kit with wireless IoT and wearable sensor platform to help you use and develop apps based on remote motion and environmental sensor data, regardless of your level of expertise.

Figure 10. STEVAL-MKSBOX1V1

4.1.6 STEVAL-MKBOXPRO

The STEVAL-MKBOXPRO (SensorTile.box PRO) is the new ready-to-use programmable wireless box kit for developing any IoT application based on remote data gathering and evaluation, exploit the full kit potential by leveraging both motion and environmental data sensing, along with a digital microphone, and enhance the connectivity and smartness of whatever environment you find yourself into.

Figure 11. STEVAL-MKBOXPRO

4.2 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo board (order code: NUCLEO-L476RG)
2. One BLE expansion board (order code: X-NUCLEO-IDB05A1)
3. One sensor expansion board (order code: X-NUCLEO-IKS01A3)
4. One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

4.3 Software setup

The following software components are required for the setup of a suitable development environment to create applications for the STM32 Nucleo board with the sensor expansion board:

- FP-SNS-BLEMESH1: an STM32Cube function pack for IoT node with BLE Mesh connectivity and sensor model. The firmware and related documentation are available on www.st.com.
- Development tool-chain and Compiler. The STM32Cube expansion software supports the three following environments to select from:
  - IAR Embedded Workbench for ARM® (IAR-EWARM) toolchain + ST-LINK
  - RealView Microcontroller Development Kit (MDK-ARM-STM32) toolchain + ST-LINK
  - System Workbench for STM32 (SW4STM32) + ST-LINK

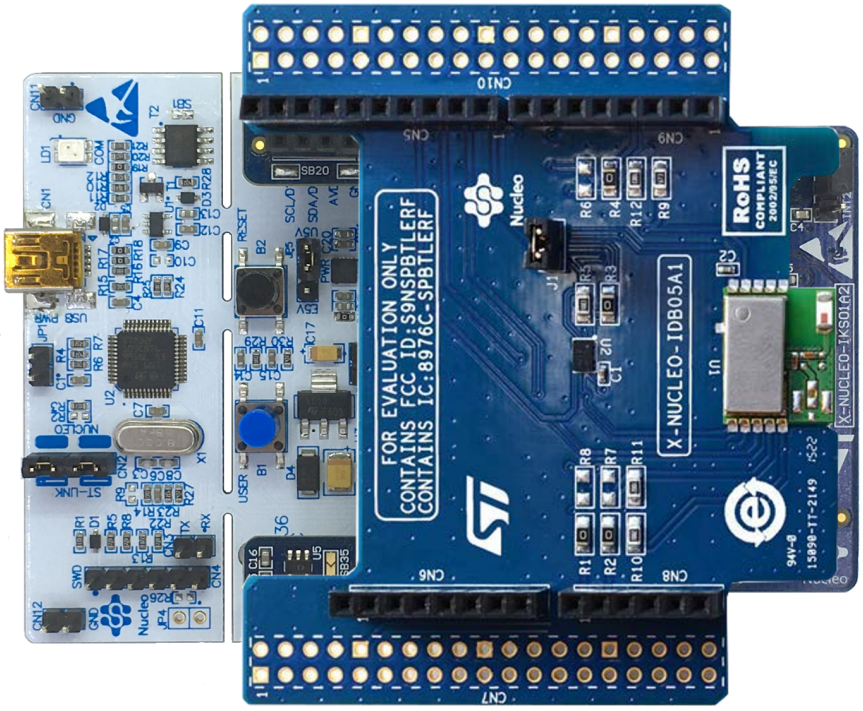
4.4 System setup

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 at the STSW-LINK009 software on www.st.com.

The X-NUCLEO-IDB05A1 and the X-NUCLEO-IKS01A3 expansion boards can be easily connected to the STM32 Nucleo through the Arduino UNO R3 extension connector.

Figure 12. STM32 Nucleo, X-NUCLEO-IDB05A1 and X-NUCLEO-IKS01A3 stack



Appendix A References

1. Mesh over Bluetooth low energy
STSW-BNRG-Mesh
2. Bluetooth Mesh networking specifications
https://www.bluetooth.com/specifications/mesh-specifications
3. Bluetooth Mesh model specification
https://www.bluetooth.com/specifications/adopted-specifications

Revision history

Table 5. **Document revision history**

Date	Version	Changes
21-Jan-2019	1	Initial release.
01-Sep-2023	2	Updated Section Introduction, Section 3.1 Overview, Section 3.2 Architecture, Section 3.3 Folder structure and Section 3.5.2 Initialization and main application loop. Added Section 4.1.3 X-NUCLEO-BNRG2A1 expansion board, Section 4.1.4 X-NUCLEO-IKS01A3 expansion board, Section 4.1.5 STEVAL-MKSBOX1V1 and Section 4.1.6 STEVAL-MKBOXPRO.

Contents

List of tables

List of figures