UM2666

User manual

Getting started with the X-CUBE-BLE2 Bluetooth Low Energy software expansion for STM32Cube

Introduction

The X-CUBE-BLE2 expansion software package for STM32Cube runs on the STM32 and includes drivers for BlueNRG-2and BlueNRG-2N Bluetooth® low energy devices.

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-BNRG2A1 when connected to a NUCLEO-L476RG 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
BLE	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	Micro controller 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-BLE2 software expansion for STM32Cube

2.1 Overview

The X-CUBE-BLE2 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 BlueNRG-2 and BlueNRG-2N 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 new innovative applications (fitness, security, healthcare, etc.) using devices which run on coin cell batteries, and can remain indefinetely operative 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).

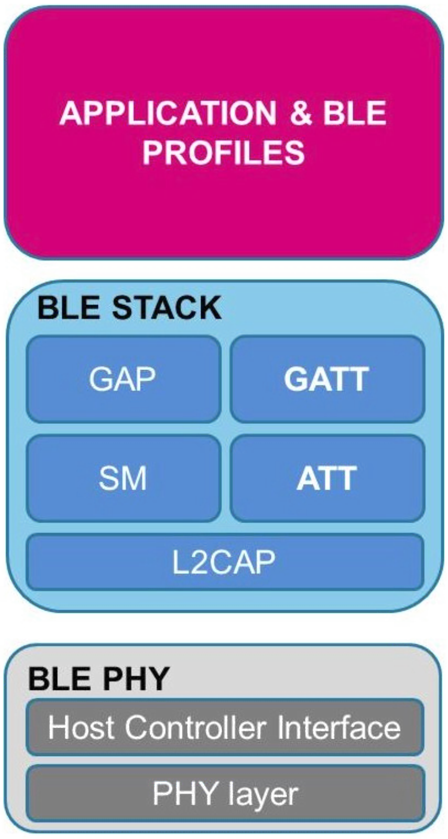
Instead, a single-mode (Bluetooth smart) device supports the BLE protocol only.

2.1.1.2 Software partitioning

A typical BLE 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 an HCI and other higher protocol layers (e.g. L2CAP, SM, ATT/GATT and GAP)

Figure 1. **BLE 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. BLE operates in the 2.4 GHz Industrial Scientific Medical (ISM) band and defines 40 radio frequency (RF) channels with 2 MHz channel spacing.

In BLE, 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 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.

BLE 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 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 BlueNRG-2 and BlueNRG-2N, this layer is implemented through the SPI hardware interface.

In BLE, the main goal of 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 BLE 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, GAP handles the initiation of security features. The BLE 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.

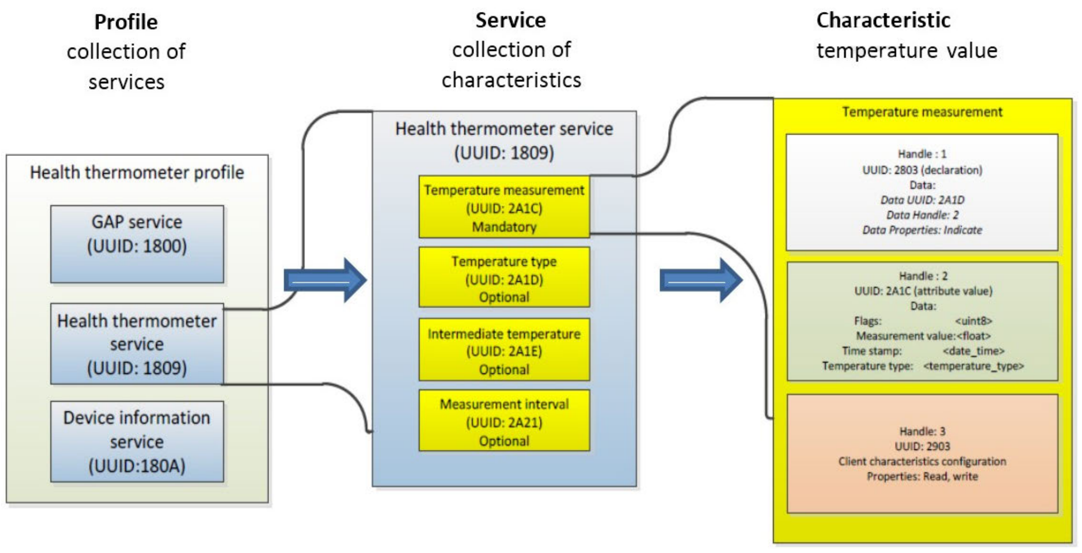
The GATT defines a framework that uses the ATT for the discovery of services, and the exchange of characteristics from one device to another. GATT specifies the structure of profiles. In BLE, all pieces of data that are being used by a profile or service are called characteristics. A characteristic is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

2.1.1.3 *Profiles and services*

The BLE protocol stack is used by the applications through its GAP and GATT profiles. The GAP profile is used to initialize the stack and setup 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


Profile
collection of services
Service
collection of characteristics
Characteristic
temperature value
Health thermometer profile
GAP service (UUID: 1800)
Health thermometer service (UUID: 1809)
Device information service (UUID:180A)
Health thermometer service (UUID: 1809)
Temperature measurement (UUID: 2A1C) Mandatory
Temperature type (UUID: 2A1D) Optional
Intermediate temperature (UUID: 2A1E) Optional
Measurement interval (UUID: 2A21) Optional
Temperature measurement
Handle : 1
UUID: 2803 (declaration)
Data:
Data UUID: 2A1D
Data Handle: 2
Data Properties: Indicate
Handle : 2
UUID: 2A1C (attribute value)
Data:
Flags: <uint8>
Measurement value:<float>
Time stamp: <date_time>
Temperature type: <temperature_type>
Handle: 3
UUID: 2903
Client characteristics configuration
Properties: Read, write

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*

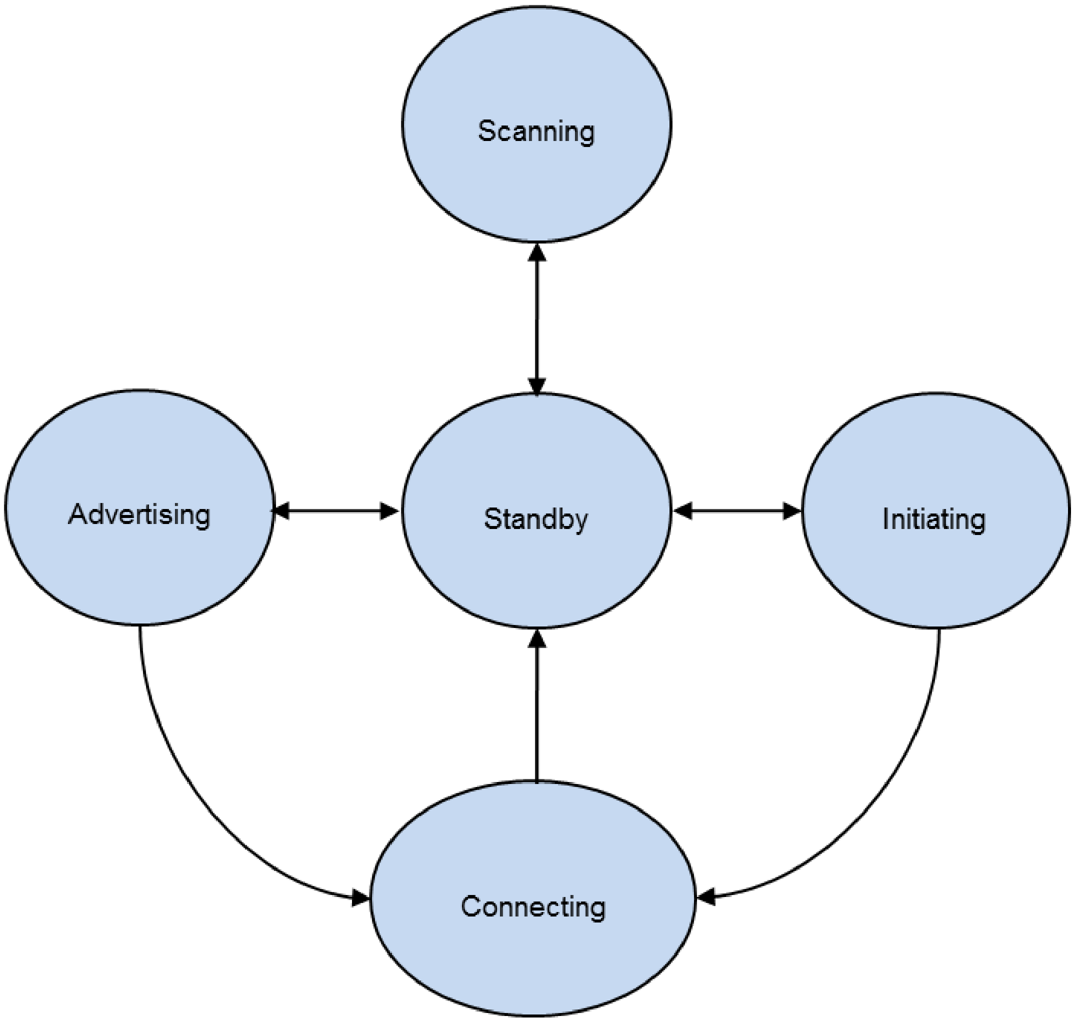


Figure 3. **State machine during BLE operations**

During normal operation, a BLE 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 to, 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 connection to the advertiser and is 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

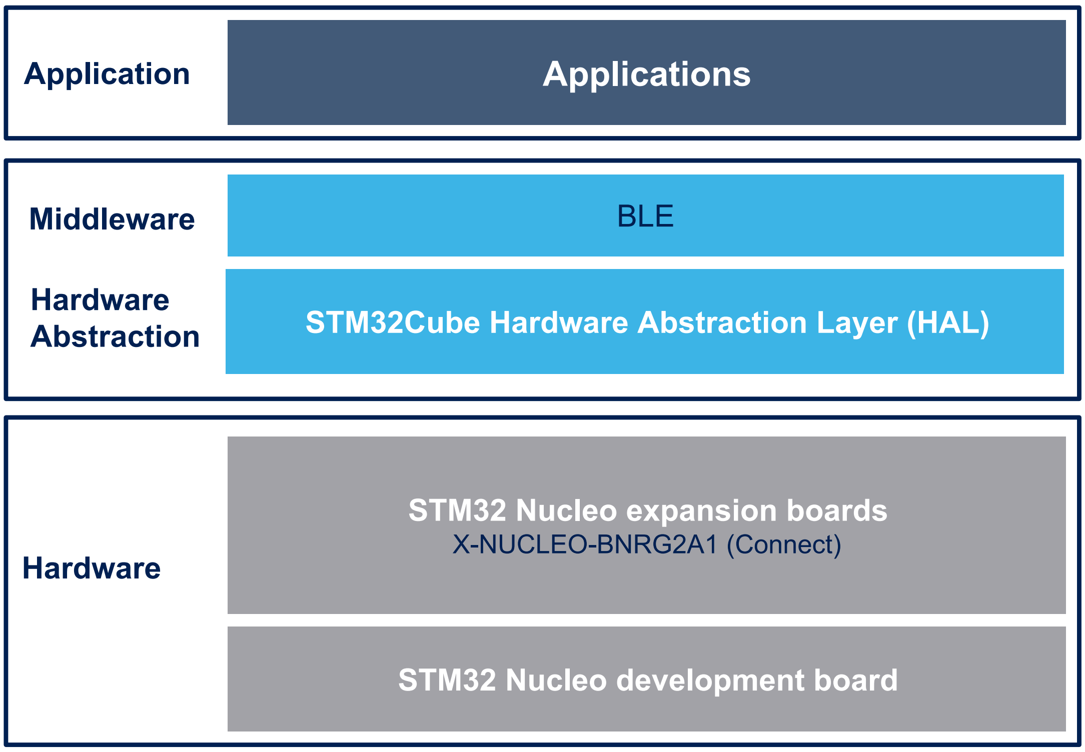
This software is a fully compliant expansion for STM32Cube enabling development of applications using the Bluetooth Low Energy connectivity.

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 BlueNRG-M2SP module expansion board are:

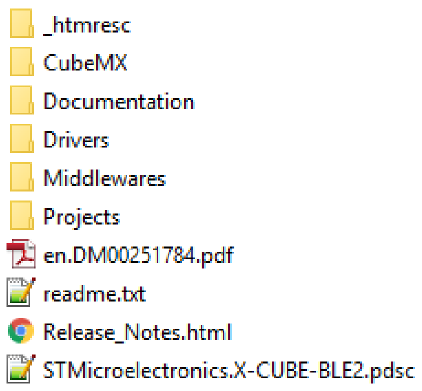
- The STM32Cube HAL driver layer provides a simple, generic and multi-instance set of APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It includes generic and extension APIs and is based on a generic architecture which allows the layers 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 guarantees 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 (e.g. the LED, the user button etc.) and allow identification of the specific board version.

Figure 4. **X-CUBE-BLE2 software architecture**


Application
Applications
Middleware
BLE
Hardware Abstraction
STM32Cube Hardware Abstraction Layer (HAL)
Hardware
STM32 Nucleo expansion boards
X-NUCLEO-BNRG2A1 (Connect)
STM32 Nucleo development board

2.3 Folders

Figure 5. **X-CUBE-BLE2 package folder structure**


_htmresc
CubeMX
Documentation
Drivers
Middlewares
Projects
en.DM00251784.pdf
readme.txt
Release_Notes.html
STMicroelectronics.X-CUBE-BLE2.pdsc

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 on-board components and the CMSIS layer, which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.
- **Middlewares**: libraries and protocols related to host software and applications to interface the BlueNRG-2 and BlueNRG-2N controller.
- **Projects**: contains some sample applications, showing how to use the BlueNRG-2 and BlueNRG-2N devices, for the NUCLEO-L476RG 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_BLE2.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-BNRG2A1 expansion board with a NUCLEO-L476RG 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 BlueNRG-2 and BlueNRG-2N library APIs. Each sample application folder contains a readme.txt file describing the application and how to use it.

The main functions to be called when creating an application using the BlueNRG-2 and BlueNRG-2N are listed below.

- `hci_init()`: initializes the Host Controller Interface (HCI) and must be called before any data can be received from the BLE controller. It takes as input the pointer to a callback for the ACI events processing and the pointer to a configuration structure. In the ACI events processing function, implemented at application level, each event must be identified and parsed.
- `hci_reset()`: sends an HCI command to the BlueNRG-2 and BlueNRG-2N devices resetting its previous state. After the reset is completed, the current operational state is lost, the controller enters standby mode and the controller automatically reverts to the default values for the parameters for which default values are defined in the specification. The [**Reset**] command does not necessarily perform hardware reset.
- `aci_hal_write_config_data()`: writes a value to a low level configure data structure. It is useful to set up directly some low level parameters for the system in the runtime. This command is not called if a command different than `Stack Init, HCI_RESET, ACI_HAL_WRITE_CONFIG_DATA or ACI_HAL_READ_CONFIG_DATA` has already been called.
- `aci_gatt_init()`: initializes the GATT layer for server and client roles. It adds also the GATT service with Service Changed Characteristic. Until this command is issued, the GATT channel does not process any command even if the connection is opened. You must enter this command before using any of the GAP features.
- `aci_gap_init()`: initializes the GAP layer. It registers the GAP service with the GATT. All the standard GAP characteristics are also added:
  - Device Name
  - Appearance
  - Peripheral Preferred Connection Parameters (peripheral role only)

Warning:
A section of the Flash memory is used by this procedure. Do not power off the device while this function is writing into Flash memory.

- `aci_gap_set_authentication_requirement()`: sets the authentication requirements for the device. Enter this command only when the device is not in a connected state.

- `aci_gatt_add_service()`: adds a service to GATT Server. When a service is created in the server, the host needs to reserve the handle ranges for this service using `Max_Attribute_Records` parameter, which specifies the maximum number of attribute records that can be added to this service (including the service attribute, include attribute, characteristic attribute, characteristic value attribute and characteristic descriptor attribute). Handle of the created service is returned in command complete event. Service declaration is taken from the service pool. The attributes for characteristics and descriptors are allocated from the attribute pool.
- `aci_gatt_add_char()`: adds a characteristic to a service.
- `hci_user_evt_proc()`: must be called after an event is received from the HCI interface, where the callback for the ACI events processing is called.
- `aci_gap_set_discoverable()`: puts the device in general discoverable mode (as defined in Bluetooth Specification v.4.1, Vol. 3, Part C, section 9.2.4). The device is discoverable until the host issues the `aci_gap_set_non_discoverable` command.
- `aci_gap_start_general_discovery_proc()`: starts the general discovery procedure. The controller is commanded to start active scanning. The procedure is terminated when either the upper layers issue a command to terminate the procedure by issuing the command `aci_gap_terminate_gap_proc` or a timeout happens. When the procedure is terminated due to any of the above reasons, `aci_gap_proc_complete_event` event is returned. The device found when the procedure is ongoing is returned to `hci_le_advertising_report_event`.
- `aci_gap_create_connection()`: starts the direct connection establishment procedure. On procedure termination, an `hci_le_connection_complete_event` event is returned. The procedure can be explicitly terminated by the upper layer by issuing the command `aci_gap_terminate_gap_proc`.

3 X-CUBE-BLE2 software package for BlueNRG-2N

3.1 BlueNRG-2N overview

The X-CUBE-BLE2 software package contains drivers for the BlueNRG-2N ultra low power (ULP) network coprocessor solution for Bluetooth® low energy applications.

The device embeds the STMicroelectronics state-of-art RF radio IPs combining unparalleled performance with extremely long battery lifetime and supporting enhanced features (security, privacy and extended packet length for faster data transfer up to 700 kbps at application level).

The BlueNRG-2N is Bluetooth® 5.0 certified and ensures interoperability with the latest generation of smartphones and other host devices.

The Bluetooth Low Energy stack runs on the embedded ARM Cortex-M0 core.

The BlueNRG-2N comes pre-programmed with a production-ready and certified Bluetooth LE stack image. STMicroelectronics BLE stack is stored into the on-chip non-volatile Flash memory and can be easily upgraded via SPI, UART interface (through the dedicated BlueNRG software tools) or FOTA.

The BlueNRG-2N supports multi-master to multi-slave communication: 2 masters to 6 slaves can be managed simultaneously, handling up to 8 simultaneous connections.

The device can be emulated using the BlueNRG-2 mounted on the X-NUCLEO-BNRG2A1 expansion board.

3.2 How to emulate the BlueNRG-2N device with X-NUCLEO-BNRG2A1

To emulate the BlueNRG-2N device, the BlueNRG-2N firmware image contained in the STSW-BNRG2N-V320 software package must be loaded onto the BlueNRG-2 device mounted on the X-NUCLEO-BNRG2A1 expansion board.

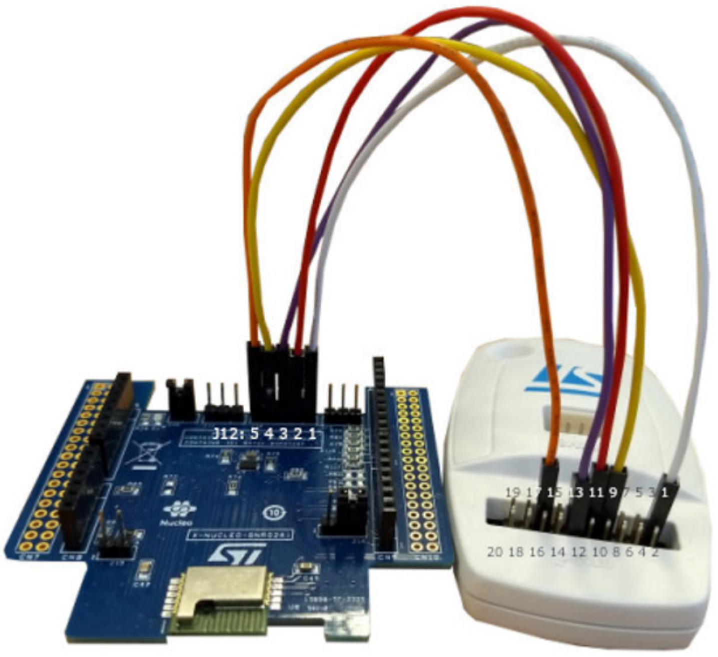
This operation can be performed using a standard ST-LINK/V2 debugger and following the procedure below.

Step 1. Connect the X-NUCLEO-BNRG2A1 J12 pins and the ST-LINK/V2 pins as per the following table and figure.

Table 2. **Jumper connection between X-NUCLEO-BNRG2A1 and ST-LINK/V2**

Pin name	J12 pin no.	ST-LINK/V2 pin no.
VDD	1	1
SWTCK	2	9
GND	3	12
SWDIO	4	7
RST	5	15

Figure 6. **X-NUCLEO-BNRG2A1 connected to ST-LINK/V2**



Step 2. Download and unpack the STSW-BNRG2N-V320 containing the BlueNRG-2N firmware image.

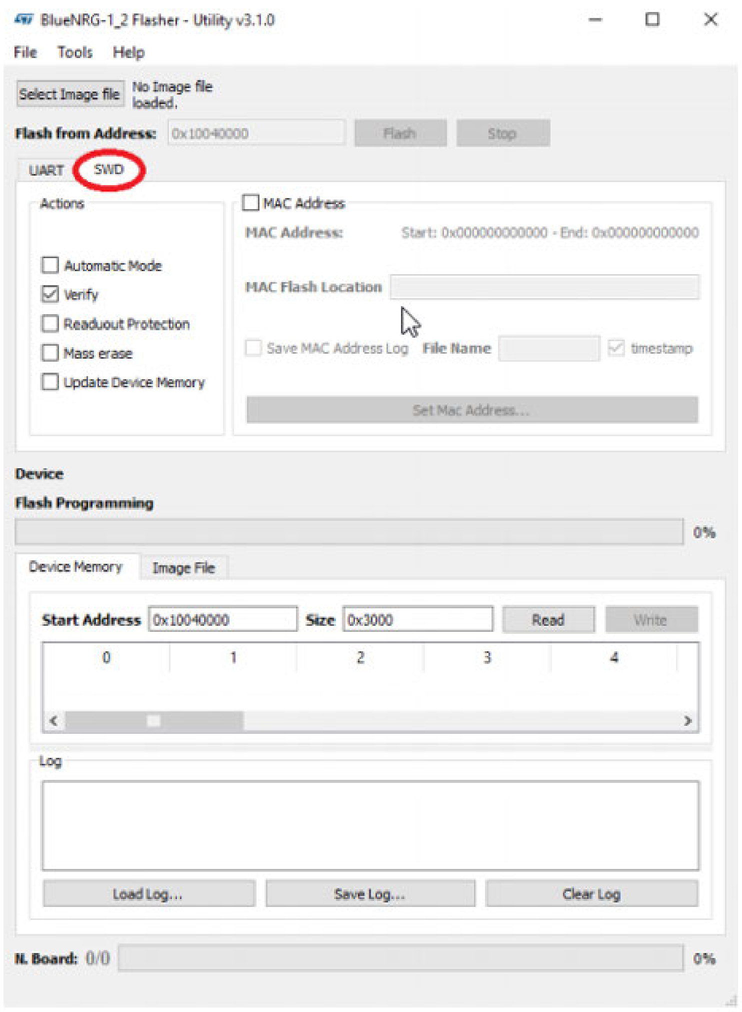
Step 3. Download and install the STSW-BNRGFLASHER (available for Windows only).

Step 4. Connect the ST-LINK/V2 debugger to your PC.

Step 5. Open the BlueNRG-1_2 Flasher Utility (STSW-BNRGFLASHER) and:

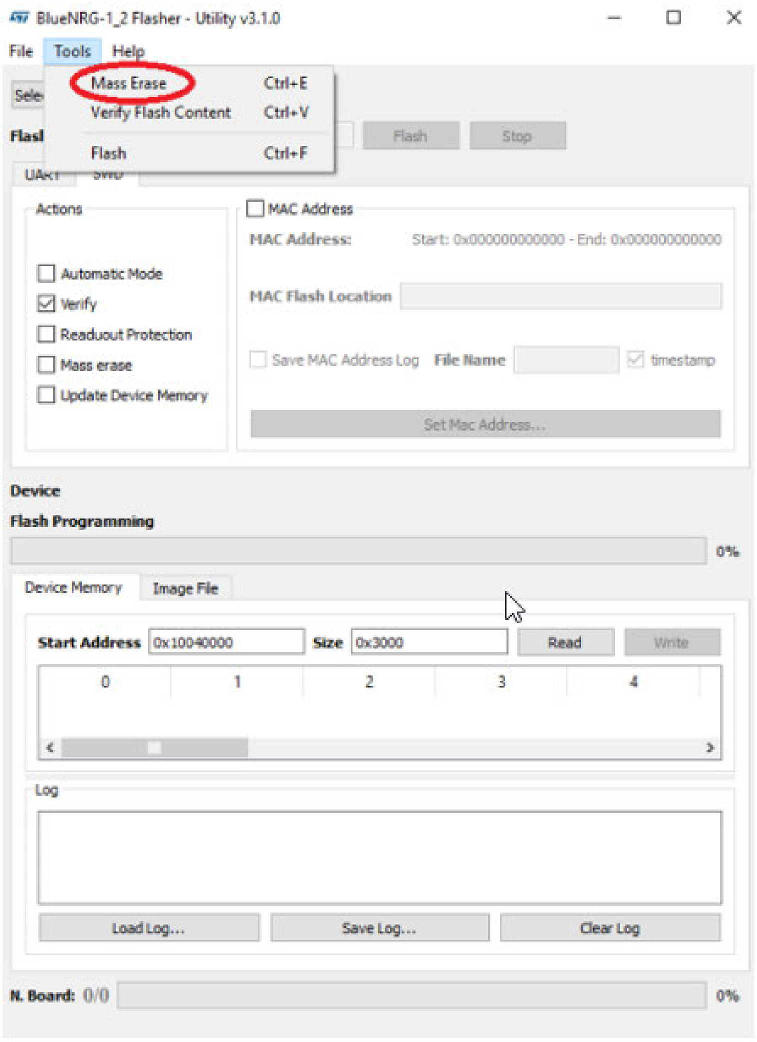
Step 5a. Select the SWD tab.

Figure 7. **STSW-BNRGFLASHER - SWD tab**



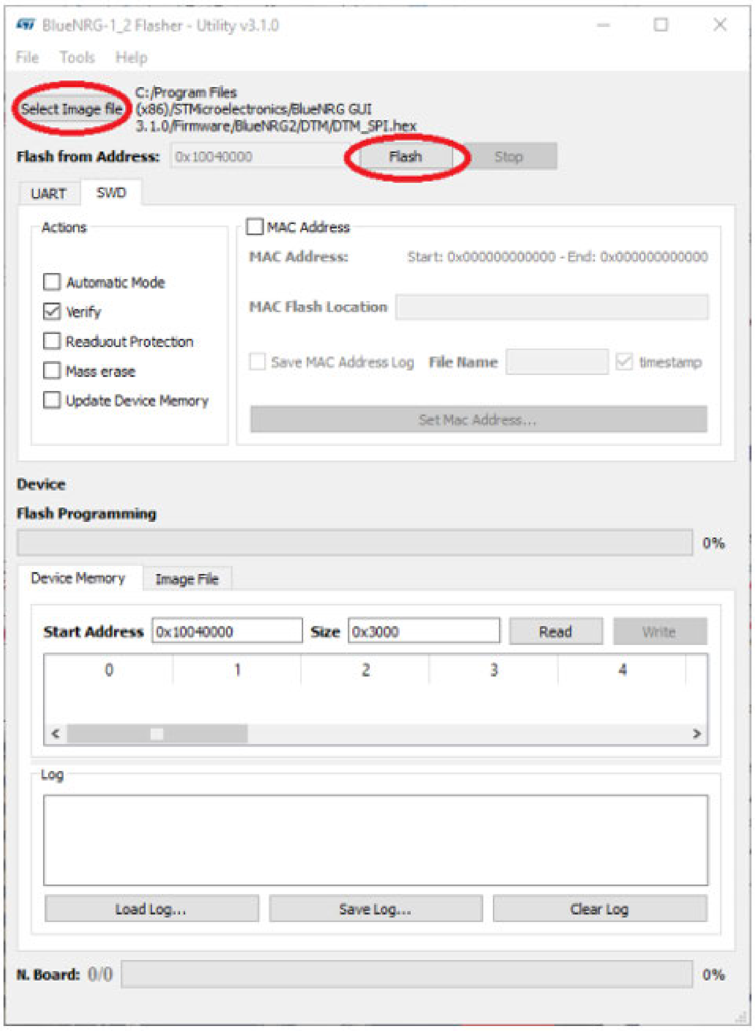
Step 5b. Erase the BlueNRG-2 Flash memory from [**Tools**]>[**Mass Erase**].

Figure 8. **STSW-BNRGFLASHER - Mass Erase**



Step 5c. Load the BlueNRG_2N_FW_V3_2_0.hex firmware contained in the STSW-BNRG2N-V320 and press the [**Flash**] button.

Figure 9. **STSW-BNRGFLASHER - image flashing**



C:/Program Files (x86)/STMicroelectronics/BlueNRG GUI 3.1.0/Firmware/BlueNRG2/DTM/DTM_SPI.hex

Flash from Address: 0x10040000

- [ ] Automatic Mode
- [x] Verify
- [ ] Readuout Protection
- [ ] Mass erase
- [ ] Update Device Memory

- [ ] MAC Address

MAC Address: Start: 0x000000000000 - End: 0x000000000000

MAC Flash Location

- [ ] Save MAC Address Log File Name
- [x] timestamp

Device

Flash Programming 0%

Device Memory | Image File

Start Address 0x10040000 Size 0x3000

0 1 2 3 4

Log

N. Board: 0/0 0%

The image flashed on the BlueNRG-2 emulates the BlueNRG-2N device and can be used with the X-CUBE-BLE2 software package.

3.3 How to restore the BlueNRG-2 firmware image

To restore the BlueNRG-2 firmware image on the BlueNRG-2 device:

Step 1. Download and install STSW-BNRGUI.

Step 2. Open the BlueNRG-1_2 Flasher Utility (STSW-BNRGFLASHER).

Step 3. Select the SWD tab.

Step 4. Erase the BlueNRG-2 Flash memory from [**Tools**]>[**Mass Erase**].

Step 5. Load the DTM_SPI.hex firmware contained in the STSW-BNRGUI installation folder (usually C:\Program Files (x86)\STMicroelectronics\BlueNRG GUI 3.2.1\Firmware\BlueNRG2\DTM for version 3.2.1).

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.

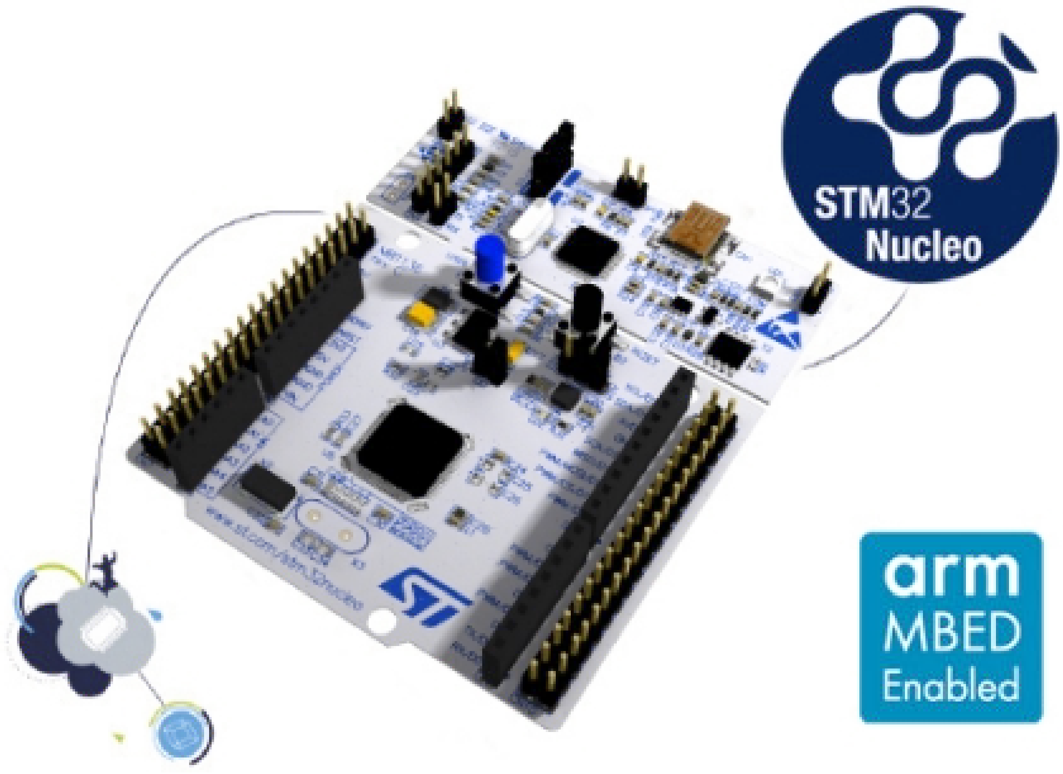
The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

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 10. **STM32 Nucleo board**



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

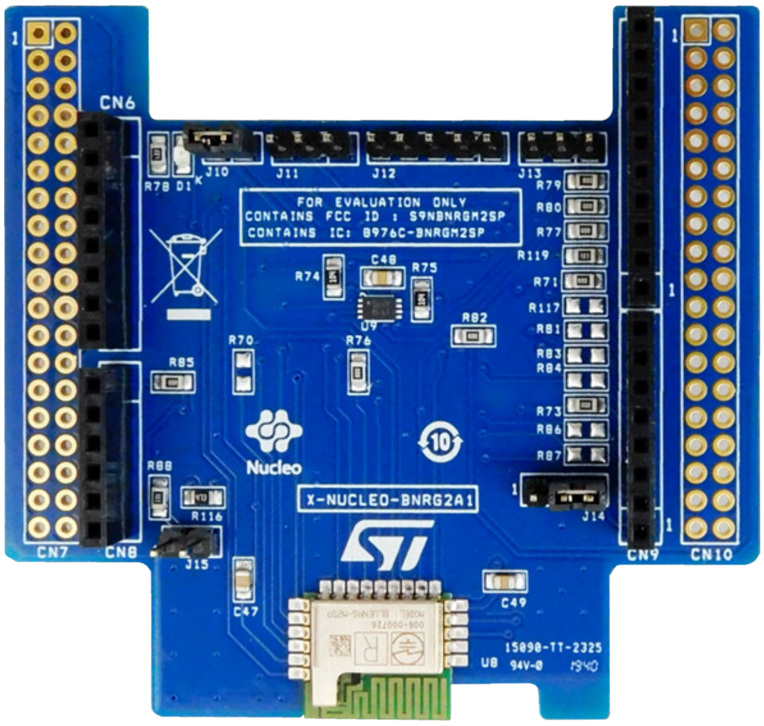
4.1.2 X-NUCLEO-BNRG2A1 expansion board

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 11. X-NUCLEO-BNRG2A1 BLE expansion board



Information about the X-NUCLEO-BNRG2A1 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 BLE expansion board:

- X-CUBE-BLE2: an STM32Cube expansion for sensor application development. The X-CUBE-BLE2 firmware and associated documentation is available on www.st.com.
- Development tool-chain 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-L476RG)
2. One BlueNRG-M2SP module expansion board (order code: X-NUCLEO-BNRG2A1)
3. One USB type A to mini-B USB cable to connect the STM32 Nucleo to a PC

4.4 STM32 Nucleo and BLE expansion board setup

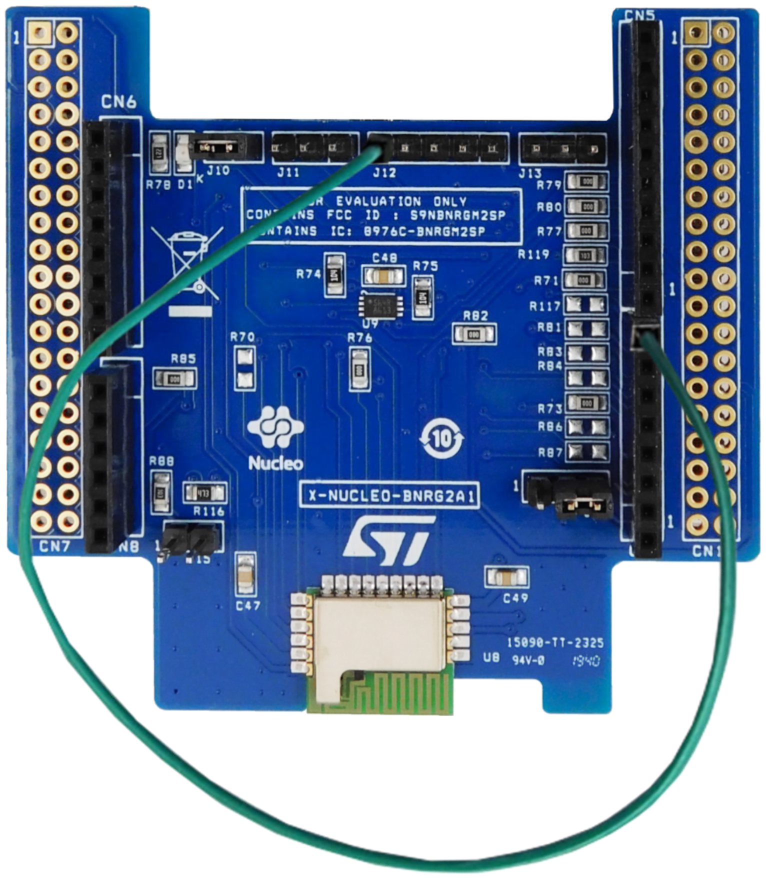
The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer. Developers can download the relevant version of the ST-LINK/V2-1 USB driver at STSW-LINK008 or STSW-LINK009 (according to your version of Windows).

The X-NUCLEO-BNRG2A1 expansion board can be easily connected to the STM32 Nucleo motherboard through the Arduino UNO R3 extension connector. The BlueNRG-2 processor hosted in the BlueNRG-M2SP module communicates with the STM32 microcontroller, hosted on the STM32 Nucleo development board, through an SPI link available on the Arduino UNO R3 connector.

Warning:

To correctly set the RESET on pin D7 a 0 Ohm resistor must be soldered on R117. Alternatively, the Arduino connector D7 pin and the J12 pin 5 on the X-NUCLEO-BNRG2A1 expansion board must be bridged as shown in the picture below.

Figure 12. ***X-NUCLEO-BNRG2A1 expansion board: bridge between Arduino connector (D7 pin) and J12 (pin 5)***



Revision history

Table 3. **Document revision history**

Date	Revision	Changes
11-Dec-2019	1	Initial release.
09-Jul-2020	2	Updated Introduction, Section 2.1 Overview, Section 2.2 Architecture, Section 4.1.1 STM32 Nucleo and Section 4.3 Hardware setup. Added Section 3.1 BlueNRG-2N overview, Section 3.2 How to emulate the BlueNRG-2N device with X-NUCLEO-BNRG2A1 and Section 3.3 How to restore the BlueNRG-2 firmware image. Added references to BlueNRG-2N device.

Contents

1 **Acronyms and abbreviations** **2**

2 **X-CUBE-BLE2 software expansion for STM32Cube** **3**

2.1 Overview 3

2.1.1 Bluetooth Low Energy 3

2.2 Architecture 6

2.3 Folders 7

2.4 APIs 8

2.5 Sample application description 8

3 **X-CUBE-BLE2 software package for BlueNRG-2N** **10**

3.1 BlueNRG-2N overview 10

3.2 How to emulate the BlueNRG-2N device with X-NUCLEO-BNRG2A1 10

3.3 How to restore the BlueNRG-2 firmware image 14

4 **System setup guide** **15**

4.1 Hardware description 15

4.1.1 STM32 Nucleo 15

4.1.2 X-NUCLEO-BNRG2A1 expansion board 15

4.2 Software description 16

4.3 Hardware setup 16

4.4 STM32 Nucleo and BLE expansion board setup 17

Revision history **18**

List of figures

Figure 1. BLE protocol stack 4
Figure 2. Structure of a GATT-based profile 5
Figure 3. State machine during BLE operations 6
Figure 4. X-CUBE-BLE2 software architecture 7
Figure 5. X-CUBE-BLE2 package folder structure 7
Figure 6. X-NUCLEO-BNRG2A1 connected to ST-LINK/V2 11
Figure 7. STSW-BNRGFLASHER - SWD tab 12
Figure 8. STSW-BNRGFLASHER - Mass Erase 13
Figure 9. STSW-BNRGFLASHER - image flashing 14
Figure 10. STM32 Nucleo board 15
Figure 11. X-NUCLEO-BNRG2A1 BLE expansion board 16
Figure 12. X-NUCLEO-BNRG2A1 expansion board: bridge between Arduino connector (D7 pin) and J12 (pin 5) 17

List of tables

Table 1. List of acronyms . . . 2
Table 2. Jumper connection between X-NUCLEO-BNRG2A1 and ST-LINK/V2 . . . 11
Table 3. Document revision history . . . 18