AN4493
Application note

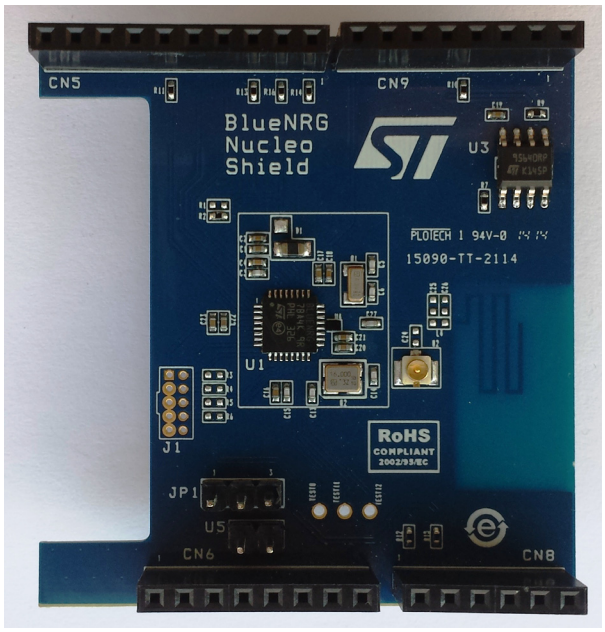
X-NUCLEO-IDB04A1 Bluetooth low energy expansion board based on BlueNRG for STM32 Nucleo

By Antonio Vilei

Introduction

This document describes the use of the X-NUCLEO-IDB04A1 with the STM32 Cube Driver API allowing the STM32 Nucleo boards to support the BlueNRG Shield. The BlueNRG is a very low power Bluetooth low energy (BLE) single-mode network processor, compliant with Bluetooth specifications core 4.0.

Figure 1. X-NUCLEO-IDB04A1 expansion board



Contents

1 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronyms	Description
ACI	Application controller interface
BLE	Bluetooth low energy
HAL	Hardware abstraction layer
HCI	Host controller interface
IDE	Integrated development environment
SPI	Serial peripheral interface

2 Getting started

This section describes all the software requirements for:

- Installation and setup procedure
- Running the sample BLE application
- Using the STM32 Cube F4 firmware for customized application development

Hardware requirements:

1. Two STM32 Nucleo Development platforms (suggested order code: NUCLEO-F401RE)
2. Two BlueNRG Shields (see *Figure 1*, order code: X-NUCLEO-IDB04A1)
3. Optionally, one STM32 Nucleo Board can be replaced by a BlueNRG USB Dongle (see *Figure 4*, order code: STEVAL-IDB003V1)

In order to explain the use of the BlueNRG, a use case is introduced describing a testing scenario where a sample application can be exploited to get two remote BlueNRG devices connected and have them exchange messages through a simple ACI protocol.

Please be sure to download from st.com the latest version of the firmware package distributed along this document, as it will include improvements and additions.

The user is advised to refer to the README file included in the firmware package for more detailed information.

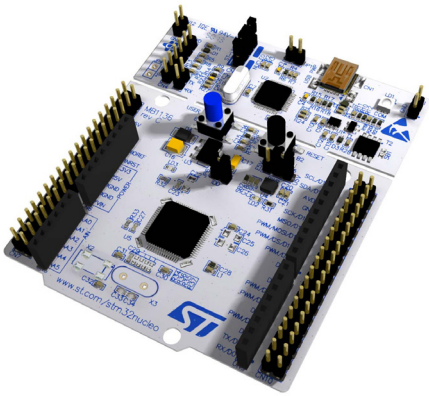
2.1 Hardware description

- 2 x STM32F401RE
- 2 x X-NUCLEO-IDB04A1
- 2 x USB type A to Mini-B USB cable to connect the Nucleo to the PC.

2.1.1 STM32F401RE Nucleo

The STM32F401RE Nucleo board (refer to http://www.st.com/web/en/catalog/tools/FM147/CL1794/SC961/SS1743/PF259243) belongs to the STM32 Nucleo family and provides an affordable and flexible means for users to experiment with new ideas and build prototypes with any of the STM32 microcontroller lines. The Arduino™ connectivity support and ST Morpho headers make it easy to expand the functionality of the Nucleo open development platform with a wide selection of specialized shields. The STM32 Nucleo board does not require any separate probe, as it integrates the ST-LINK/V2-1 debugger/programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library, together with various packaged software examples.

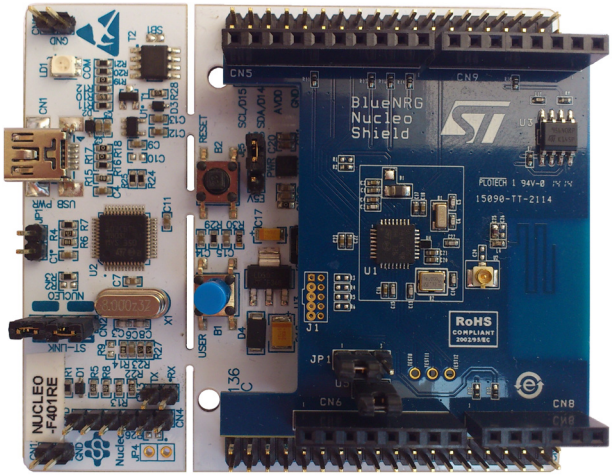
Figure 2. STM32 Nucleo board



2.1.2 BlueNRG Shield

The BlueNRG shield is a BLE single-mode network processor, compliant with Bluetooth specification v4.0. The BlueNRG can act as master or slave. The entire Bluetooth low energy stack runs on the embedded Cortex M0 core. The BlueNRG offers the option of interfacing with external microcontrollers using SPI transport layer.

Figure 3. BlueNRG Shield connected to STM32 Nucleo F4

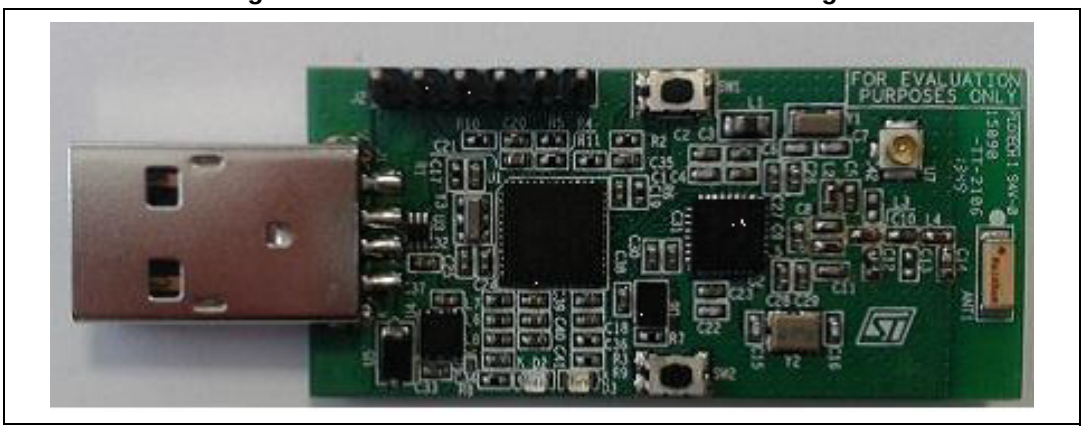


2.1.3 BlueNRG USB dongle

The STEVAL-IDB003V1 is an evaluation board based on the BlueNRG low power Bluetooth smart IC, which is compliant with the Bluetooth 4.0 specification and supports both master and slave roles.

The STEVAL-IDB003V1 has a USB connector for PC GUI interaction and firmware update.

Figure 4. STEVAL-IDB003V1 BlueNRG USB dongle



3 System requirements

This section lists the minimum requirements for the developer to set up the SDK, run the sample testing scenario based on the GUI utility and customize applications through the API provided by the STM32 Cube F4 firmware.

3.1 Development toolchains and compilers

IAR Embedded Workbench for ARM (EWARM) toolchain v7.10

The IAR toolchain has the following minimum requirements:

- PC with Intel® or AMD® processor running one of the following Microsoft® operating systems
  - Windows XP SP3
  - Windows Vista
  - Windows 7

3.2 STM32 cube F4

The STM32Cube is a development framework which provides tools and libraries to develop C applications on STM32 series platforms.

STM32Cube comprises the STM32CubeF4 (refer to http://www.st.com/web/en/catalog/tools/FM147/CL1794/SC961/SS1743/PF259243) platform which includes the STM32Cube HAL (an STM32 abstraction layer embedded software, ensuring maximized portability across STM32 portfolio), plus a consistent set of middleware components (RTOS, USB, FatFS and STM32 touch sensing). All embedded software utilities come with a full set of examples.

The STM32CubeF4 gathers in one single package all the generic embedded software components required to develop an application on STM32F4 microcontrollers. STM32CubeF4 is fully compatible with STM32CubeMX (http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF259242) code generator that allows generation of the initialization code. The package includes a low level hardware abstraction layer (HAL) which covers the microcontroller hardware, together with an extensive set of examples running on STMicroelectronics boards. The HAL is available in open-source BSD license for developer convenience.

4 System setup guide

4.1 BlueNRG USB dongle setup

The BlueNRG USB dongle allows easy addition of BLE functionalities to user the PC by simply plugging it into a PC USB port. The USB dongle can be used as a simple interface between the BlueNRG and a GUI application on the PC. The STM32L microcontroller on the board can also be programmed, so the board can be used to develop applications for use with the BlueNRG. The board can be powered through the USB connector, which can also be used for I/O interaction with a USB host. The board also has two buttons and two LEDs for user interaction.

The reader can refer to http://www.st.com/web/catalog/tools/FM116/SC1075/PF260386 and related documentation (UM1686) for additional details.

4.2 BlueNRG GUI setup

The BlueNRG GUI included in the software package is a graphical user interface that can be used to interact and evaluate the capabilities of the remote BlueNRG network processor through the local BlueNRG USB dongle.

This utility can send standard and vendor-specific HCI commands to the controller and receive events from it. It lets the user configure each field of the HCI command packets to be sent and analyzes all received packets. In this way BlueNRG can be easily managed at low level.

In order to use the BlueNRG GUI, make sure you have correctly set up your hardware and software (BlueNRG GUI installed) according to the requirements described in UM1686, section 3.1

4.3 STM32 Nucleo F4 board setup

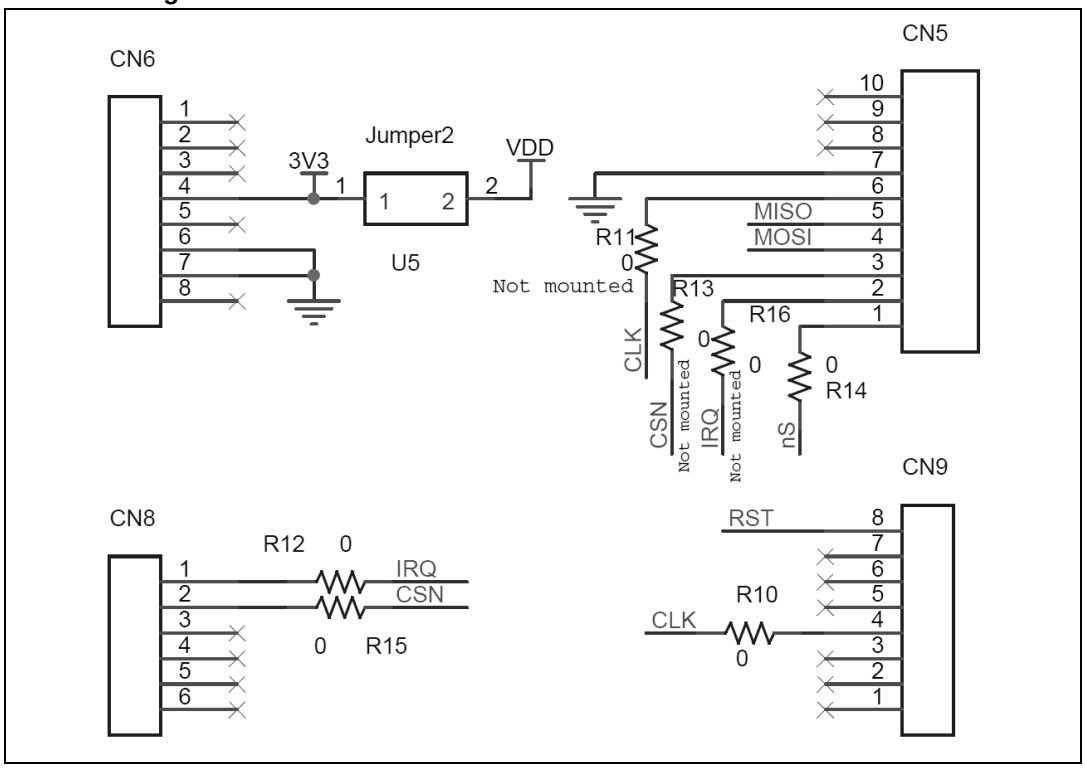
The STM32 Nucleo F4 development motherboard allows the exploitation of the BLE capabilities provided by the BlueNRG Shield network processor.

The Nucleo board integrates the ST-LINK/V2-1 debugger/programmer. The developer can download the relevant version of the ST-LINK/V2-1 USB driver (ST-SW-LINK008 or STSW-LINK009, depending on the user's MS Windows OS) from *www.st.com*.

The BlueNRG Shield can be easily connected to the Nucleo motherboard through the Arduino UNO R3 extension connector (see *Figure 5*). The BlueNRG Shield is capable of interfacing with the external STM32F4 microcontroller on Nucleo using the SPI transport layer.

Note: It is assumed that the BlueNRG Shield should be preprogrammed with BlueNRG firmware image version V6.3

Figure 5. BlueNRG Shield connection over STM32 Nucleo board



Once the USB driver has been installed and the BlueNRG Shield has been connected, the STM32 Nucleo board can be plugged into a PC USB port through a USB cable.

The IAR embedded workbench IDE (see *Figure 6*) allows developing and building the final applications for the STM32 Nucleo F4 board. The IAR workspace file of the sample application described in this document is shown in *Figure 7*. The user is recommended to “Rebuild All” the project (see *Figure 8*).

The IDE also provides the commands for Flash download and debug as well as the setting of compiler and assembler options (*Figure 8*). The reader can refer to the IAR Project management and building guide for full and extensive documentation of the toolchain.

Figure 6. IAR IDE

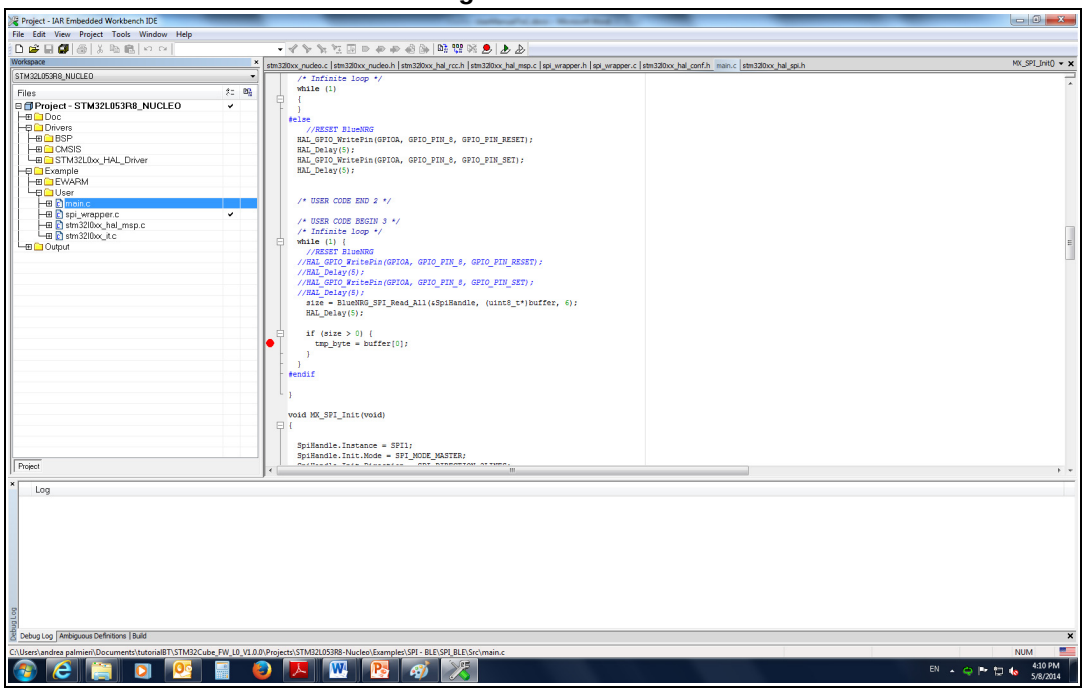


Figure 7. IAR Workspace file

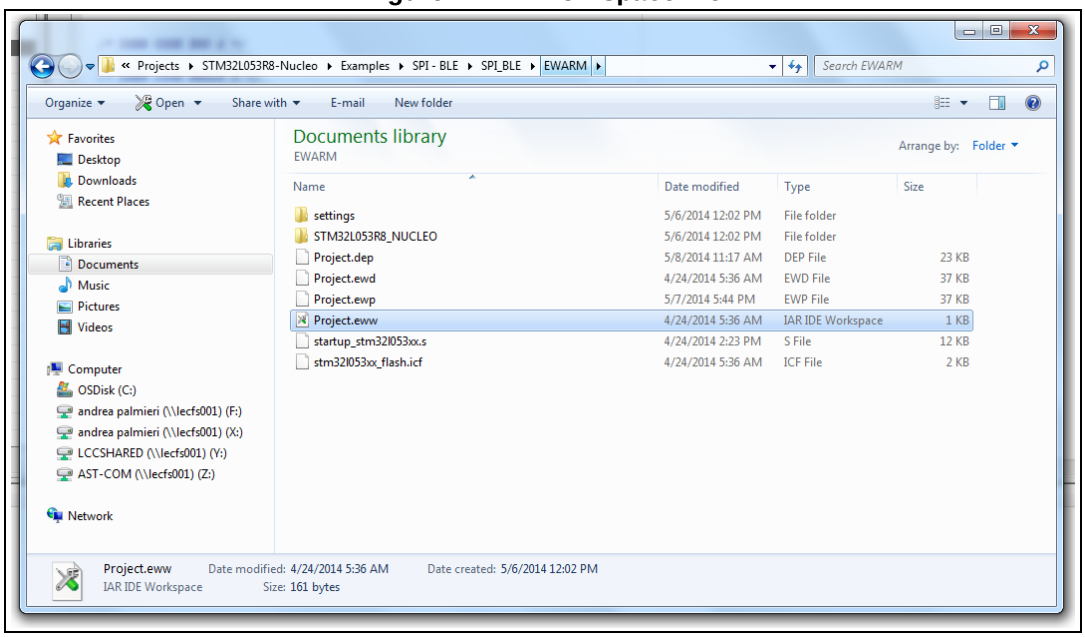
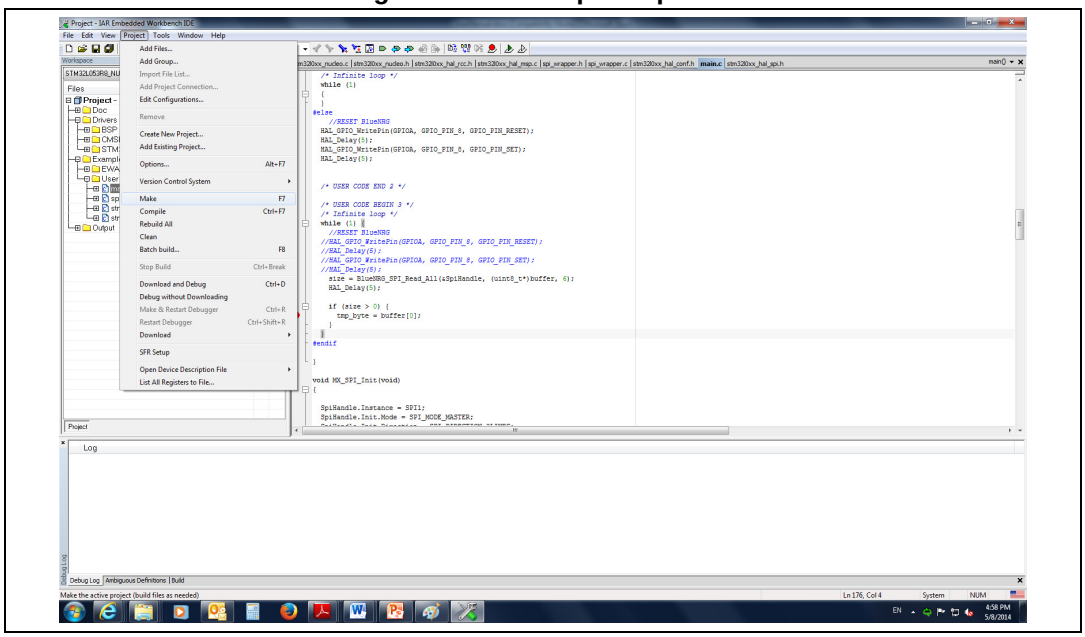


Figure 8. IAR compiler options



5 Sample application

This section describes a sample application developed for testing the BT connection setup and message passing between the two STM32 Nucleo F4 boards, each equipped with the BlueNRG Shield. Alternatively, the sample application can be run using a BlueNRG USB Dongle and an STM32 Nucleo F4 board equipped with its BlueNRG Shield.

In the sample application, the BLE devices interact with each other in order to set up point-to-point wireless communication. One Nucleo F4 board acts as server-peripheral and the second as a client-central.

5.1 Initialization and services characteristics

BlueNRG's stack must be correctly initialized before establishing a connection with another Bluetooth LE device. This is done with two commands

aci_gatt_init();

- BLE “Server” role:
  - aci_gap_init(uint8_t role, uint16_t* service_handle, uint16_t* dev_name_char_handle, uint16_t* appearance_char_handle); where role=GAP_PERIPHERAL_ROLE (0x01)
- BLE “Client” role:
  - aci_gap_init(uint8_t role, uint16_t* service_handle, uint16_t* dev_name_char_handle, uint16_t* appearance_char_handle); where role=GAP_CENTRAL_ROLE (0x03)

The sample service is added on the BLE server role device using the following command:

- aci_gatt_add_serv(UUID_TYPE_128, service_uuid, PRIMARY_SERVICE, 7, &sampleServHandle);

Where service_uuid is the private service UUID 128bits allocated for the sample service (primary service).

The command will return the service handle in sampleServHandle

5.2 Security requirements

BlueNRG exposes a command that the application can use to specify its security requirements. If a characteristic has security restrictions, a pairing procedure must be initiated by the central in order to access that characteristic. On the BLE sample demo, a fixed pin (123456) is used as follows:

- aci_gap_set_auth_requirement(MITM_PROTECTION_REQUIRED, OOB_AUTH_DATA_ABSENT, NULL, 7, 16, USE_FIXED_PIN_FOR_PAIRING, 123456, BONDING);

5.3 Connectable mode

On the BLE Server role device use GAP ACI commands to enter the general discoverable mode:

- aci_gap_set_discoverable(ADV_IND, 0, 0, PUBLIC_ADDR, NO_WHITE_LIST_USE,8, local_name, 0, NULL, 0, 0);

5.4 Connection with central device

Once the BLE Server role device is put into discoverable mode, it can be seen by the BLE Client role device in order to create a BLE connection.

On the BLE Client role device use the GAP ACI commands to connect with the BLE Server role device in advertising mode:

- aci_gap_create_connection(0x4000, 0x4000, PUBLIC_ADDR, bdaddr, PUBLIC_ADDR, 9, 9, 0, 60, 1000 , 1000);

where bdaddr is the peer address of the BLE Client role device.

Once the 2 devices are connected the BLE communication will work as follows:

1. On BLE Server role device the following command will be performed:
   – aci_gatt_update_char_value(chatServHandle, TXCharHandle, 0, len, (tHalUint8 *)cmd+j)
2. On the BLE Client role device, the following command will be performed:
   – aci_gatt_write_without_response(connection_handle, RX_HANDLE+1, len, (tHalUint8 *)cmd+j)

Where connection_handle is the handle returned on connection creation as a parameter of the EVT_LE_CONN_COMPLETE event.

6 SPI HAL API

The communication between the STM32 Nucleo F4 motherboard and the BlueNRG Shield is managed via the SPI peripheral.

The sample application running on the STM32 Nucleo F4 motherboard leverages the SPI HAL driver which provides firmware functions to manage the following functionalities of the SPI peripheral:

- Initialization and de-initialization functions
- IO operation functions
- Peripheral control functions
- Peripheral state functions

The SPI HAL driver can be used as follows:

1. Declare a SPI_HandleTypeDef handle structure, for example:

```
SPI_HandleTypeDef SpiHandle;
```

2. Initialize the SPI low level resources by implementing the HAL_SPI_MspInit() API:
   a) Enable the SPIx interface clock
   b) SPI pins configuration
   – Enable the clock for the SPI GPIOs
   – Configure these SPI pins as alternate function push-pull
   c) NVIC configuration if you need to use interrupt process
   – Configure the SPIx interrupt priority
   – Enable the NVIC SPI IRQ handle
   d) DMA Configuration if you need to use DMA process
   – Declare a DMA_HandleTypeDef handle structure for the transmit or receive stream
   – Enable DMAx interface clock use
   – Configure the DMA handle parameters
   – Configure the DMA Tx or Rx stream
   – Associate the initialized hdma_tx handle to the hspi DMA Tx or Rx handle
   – Configure the priority and enable the NVIC for the transfer complete interrupt on the DMA Tx or Rx stream
3. Program the Mode, Direction, Data size, Baudrate Prescaler, NSS management, Clock polarity and phase, FirstBit and CRC configuration in the hspi Init structure.
4. Initialize the SPI registers by calling the HAL_SPI_Init()API. This API configures also the low level Hardware GPIO, CLOCK, CORTEX...etc) by calling the customized HAL_SPI_MspInit(&SpiHandle) API.

In the sample application described in *Section 5*, the HAL_SPI_Init()API is invoked by the code fragment reported in the following:

Figure 9. SPI Init function

```
void MX_SPI_Init(void)
{

  SpiHandle.Instance = SPI1;
  SpiHandle.Init.Mode = SPI_MODE_MASTER;
  SpiHandle.Init.Direction = SPI_DIRECTION_2LINES;
  SpiHandle.Init.DataSize = SPI_DATASIZE_8BIT;
  SpiHandle.Init.CLKPolarity = SPI_POLARITY_LOW;
  SpiHandle.Init.CLKPhase = SPI_PHASE_1EDGE;
  SpiHandle.Init.NSS = SPI_NSS_SOFT;
  SpiHandle.Init.FirstBit = SPI_FIRSTBIT_MSB;
  SpiHandle.Init.TIMode = SPI_TIMODE_DISABLED;
  SpiHandle.Init.CRCPolynomial     = 7;

  SpiHandle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_4;
  SpiHandle.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLED;

  HAL_SPI_Init(&SpiHandle);

}
```

Once the SPI peripheral has been initiated, and the direction has been programmed as SPI_DIRECTION_2LINES (full duplex), the communication between the Nucleo board and BlueNRG Shield is handled by the following API:

```
HAL_StatusTypeDef HAL_SPI_TransmitReceive(SPI _HandleTypeDef *hspi,
                                          uint8_t *pTxData,
                                          uint8_t *pRxData
                                          uint16_t Size,
                                          uint32_t Timeout);
```

This API allows transmitting and receiving of an amount of data in blocking mode.

A full set of the SPI HAL Driver API is reported in the following list:

Figure 10. SPI HAL API

HAL_StatusTypeDef	**HAL_SPI_Transmit** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout) Transmit an amount of data in blocking mode.
HAL_StatusTypeDef	**HAL_SPI_Receive** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size, uint32_t Timeout) Receive an amount of data in blocking mode.
HAL_StatusTypeDef	**HAL_SPI_TransmitReceive** (SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size, uint32_t Timeout) Transmit and Receive an amount of data in blocking mode.
HAL_StatusTypeDef	**HAL_SPI_Transmit_IT** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size) Transmit an amount of data in no-blocking mode with Interrupt.
HAL_StatusTypeDef	**HAL_SPI_Receive_IT** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size) Receive an amount of data in no-blocking mode with Interrupt.
HAL_StatusTypeDef	**HAL_SPI_TransmitReceive_IT** (SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size) Transmit and Receive an amount of data in no-blocking mode with Interrupt.
HAL_StatusTypeDef	**HAL_SPI_Transmit_DMA** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size) Transmit an amount of data in no-blocking mode with DMA.
HAL_StatusTypeDef	**HAL_SPI_Receive_DMA** (SPI_HandleTypeDef *hspi, uint8_t *pData, uint16_t Size) Receive an amount of data in no-blocking mode with DMA.
HAL_StatusTypeDef	**HAL_SPI_TransmitReceive_DMA** (SPI_HandleTypeDef *hspi, uint8_t *pTxData, uint8_t *pRxData, uint16_t Size) Transmit and Receive an amount of data in no-blocking mode with DMA.
void	**HAL_SPI_IRQHandler** (SPI_HandleTypeDef *hspi) This function handles SPI interrupt request.
__weak void	**HAL_SPI_TxCpltCallback** (SPI_HandleTypeDef *hspi) Tx Transfer completed callbacks.
__weak void	**HAL_SPI_RxCpltCallback** (SPI_HandleTypeDef *hspi) Rx Transfer completed callbacks.
__weak void	**HAL_SPI_TxRxCpltCallback** (SPI_HandleTypeDef *hspi) Tx and Rx Transfer completed callbacks.
__weak void	**HAL_SPI_ErrorCallback** (SPI_HandleTypeDef *hspi) SPI error callbacks.

7 Revision history

Table 2. Document revision history

Date	Revision	Changes
28-May-2014	1	Initial release.