UM1686
User manual

BlueNRG development kits

Introduction

This document describes the BlueNRG development kits and related hardware and software components. The BlueNRG is a very low power Bluetooth® low energy (BLE) single-mode network processor, compliant with Bluetooth specifications core 4.0. The BlueNRG can act as master or slave.

There are two types of BlueNRG kits:

1. BlueNRG development platform (order code: STEVAL-IDB002V1)
2. BlueNRG USB dongle (order code: STEVAL-IDB003V1)

The BlueNRG software package includes a graphical user interface application to control the BlueNRG through a simple ACI protocol.

Contents

1 Getting started

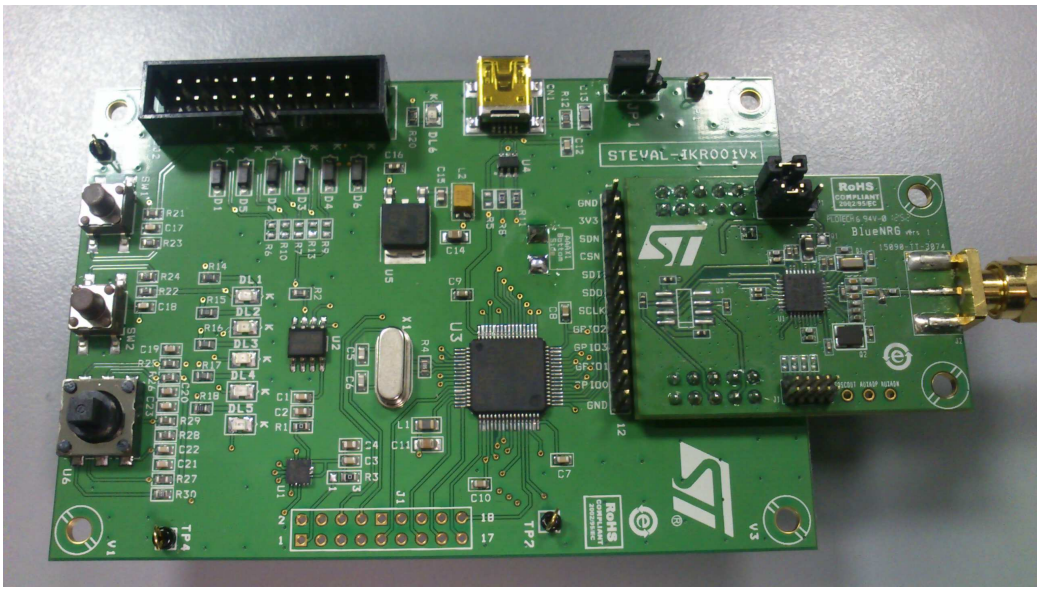
This section describes all the software and hardware requirements for running the BlueNRG GUI utility as well as the related installation procedure.

1.1 STEVAL-IDB002V1 kit contents

This kit is composed of the following items:

- 1 development motherboard
- 1 BlueNRG daughterboard
- 1 2.4 GHz Bluetooth antenna
- 1 USB cable

Figure 1. BlueNRG kit motherboard with the STEVAL-IDB002V1 daughterboard connected

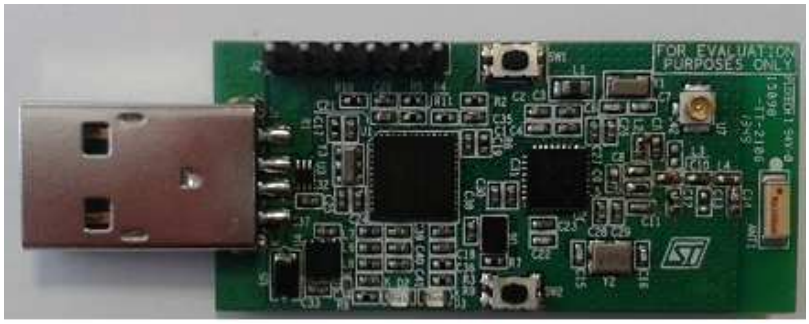


1.2 STEVAL-IDB003V1 kit

This kit is composed of the following items:

- 1 USB dongle

Figure 2. STEVAL-IDB003V1 BlueNRG USB dongle



1.3 System requirements

The BlueNRG graphical user interface utility 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
- At least 128 Mb of RAM
- 2 USB ports
- 40 Mb of hard disk space available
- Adobe Acrobat Reader 6.0 or later

1.4 BlueNRG development kit setup

- Extract the content of the BlueNRG_DK_-x.x.x-Setup.zip file into a temporary directory.
- Launch the BlueNRG-DK-x.x.x-Setup.exe file and follow the on-screen instructions.

Note: EWARM Compiler 7.40.3 or later version is required for building the BlueNRG_DK_x.x.x demonstration applications.

2 Hardware description

The following sections describe the components of the kits.

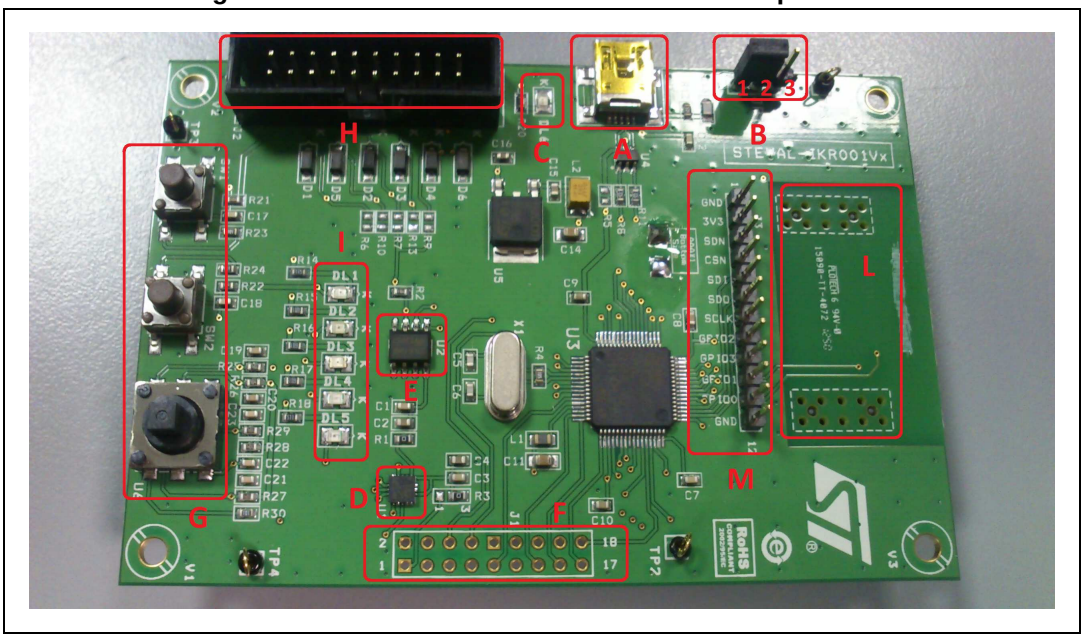
2.1 STEVAL-IDB002V1 motherboard

The motherboard included in the development kit allows testing of the functionality of the BlueNRG processor. The board 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 using the BlueNRG. A connector on the motherboard (*Figure 1*) allows access to the JTAG interface for programming and debugging. The board can be powered through a mini-USB connector that can also be used for I/O interaction with a USB Host. The board includes sensors, and buttons and a joystick for user interaction. The RF daughterboard can be easily connected through a dedicated interface.

This is a list of some of the features that are available on the boards:

- STM32L151RBT6 64-pin microcontroller
- Mini USB connector for power supply and I/O
- JTAG connector
- RF daughterboard interface
- One RESET button and one USER button
- One LIS3DH accelerometer
- One STLM75 temperature sensor
- One joystick
- 5 LEDs
- One PWR LED
- One battery holder for 2 AAA batteries
- One row of test points on the interface to the RF daughterboard

Figure 3. Motherboard for the BlueNRG development kit



2.1.1 Microcontroller and connections

The board features an STM32L151RB microcontroller, which is an ultra low-power microcontroller with 128 KB of Flash memory, 16 KB of RAM, 32-bit core ARM cortex-M3, 4 KB of data EEPROM, RTC, LCD, timers, USART, I^2C, SPI, ADC, DAC and comparators.

The microcontroller is connected to various components such as buttons, LEDs and connectors for external circuitry. The following table shows what functionality is available on each microcontroller pin.

Table 1. MCU pin description versus board function

Pin name	Pin	Board function							
		LEDs	DB connector	Buttons / joystick	Acceler.	Temperature sensor	USB	JTAG	Ext. conn
VLCD	1								
PC13	2		DB_SDN_RST						
PC14	3								3
PC15	4								5
OSC_IN	5								
OSC_OUT	6								
NRST	7			RESET					7
PC0	8	LED1							
PC1	9	LED2							
PC2	10		DB_PIN3						
PC3	11								9

Table 1. MCU pin description versus board function (continued)

Pin name	Pin	Board function							
		LEDs	DB connector	Buttons / joystick	Acceler.	Temperature sensor	USB	JTAG	Ext. conn
VSSA	12								
VDDA	13								
PA0	14								11
PA1	15								13
PA2	16								15
PA3	17								17
VSS_4	18								
VDD_4	19								
PA4	20				SPI1_NSS				
PA5	21				SPI1_SCK				
PA6	22				SPI1_MISO				
PA7	23				SPI1_MOSI				
PC4	24	LED4							
PC5	25	LED5							
PB0	26			JOY_DOWN					
PB1	27			JOY_RIGHT					
PB2	28								18
PB10	29				INT1				
PB11	30				INT2				
VSS_1	31								
VDD_1	32								
PB12	33		DB_CSN(1)						
PB13	34		DB_SCLK(1)						
PB14	35		DB_SDO(1)						
PB15	36		DB_SDI(1)						
PC6	37			PUSH_BTN					
PC7	38		DB_IO0(1)						
PC8	39		DB_IO1(1)						
PC9	40		DB_IO2(1)						
PA8	41			JOY_LEFT					
PA9	42			JOY_CENTER					

Table 1. MCU pin description versus board function (continued)

Pin name	Pin	Board function							
		LEDs	DB connector	Buttons / joystick	Acceler.	Temperature sensor	USB	JTAG	Ext. conn
PA10	43			JOY_UP					
PA11	44						USB_DM		
PA12	45						USB_DP		
PA13	46							JTMS	16
VSS_2	47								
VDD_2	48								
PA14	49							JTCK	14
PA15	50							JTDI	12
PC10	51		DB_IO3_IRQ(1)						
PC11	52		DB_PIN1						
PC12	53		DB_PIN2						
PD2	54	LED3							
PB3	55							JTDO	10
PB4	56							JNTRST	8
PB5	57					TSEN_INT			
PB6	58					I2C1_SCL			
PB7	59					I2C1_SDA			
BOOT0	60								
PB8	61								4
PB9	62								6
VSS_3	63								
VDD_3	64								

1. These lines are also available on the test point row

2.1.2 Power

The board can be powered either by the mini USB connector CN1 (A in *Figure 3*) or by 2 AAA batteries. To power the board through USB bus, jumper JP1 must be in position 1-2, as in *Figure 3* (B). To power the board using batteries, 2 AAA batteries must be inserted in the battery holder at the rear of the board, and jumper JP1 set to position 2-3.

When the board is powered, the green LED DL6 is on (C).

If needed, the board can be powered by an external DC power supply. Connect the positive output of the power supply to the central pin of JP1 (pin 2) and ground to one of the four test point connectors on the motherboard (TP1, TP2, TP3 and TP4).

2.1.3 Sensors

Two sensors are available on the motherboard:

- LIS3DH, an ultra-low power high performance three-axis linear accelerometer (D in *Figure 3*). The sensor is connected to the STM32L through the SPI interface. Two lines for interrupts are also connected.
- STLM75, a high precision digital CMOS temperature sensor, with I^2C interface (E in *Figure 3*). The pin for the alarm function is connected to one of the STM32L GPIOs.

2.1.4 Extension connector

There is the possibility to solder a connector on the motherboard to extend its functionality (F in *Figure 3*). 16 pins of the microcontroller are connected to this expansion slot (*Table 1*).

2.1.5 Push-buttons and joystick

For user interaction the board has two buttons. One is to reset the microcontroller, while the other is available to the application. There is also a digital joystick with 4 possible positions (left, right, up, down) (G in *Figure 3*).

2.1.6 JTAG connector

A JTAG connector on the board (H in *Figure 3*) allows the programming and debugging of the STM32L microcontroller on board[(a)], using an in-circuit debugger and programmer such as the ST-LINK/V2.

2.1.7 LEDs

Five LEDs are available (I in *Figure 3*).

- DL1: green
- DL2: orange
- DL3: red
- DL4: blue
- DL5: yellow

2.1.8 Daughterboard interface

The main feature of the motherboard is the capability to control an external board, connected to the J4 and J5 connectors (L in *Figure 3*). *Table 1* shows which pins of the microcontroller are connected to the daughterboard.

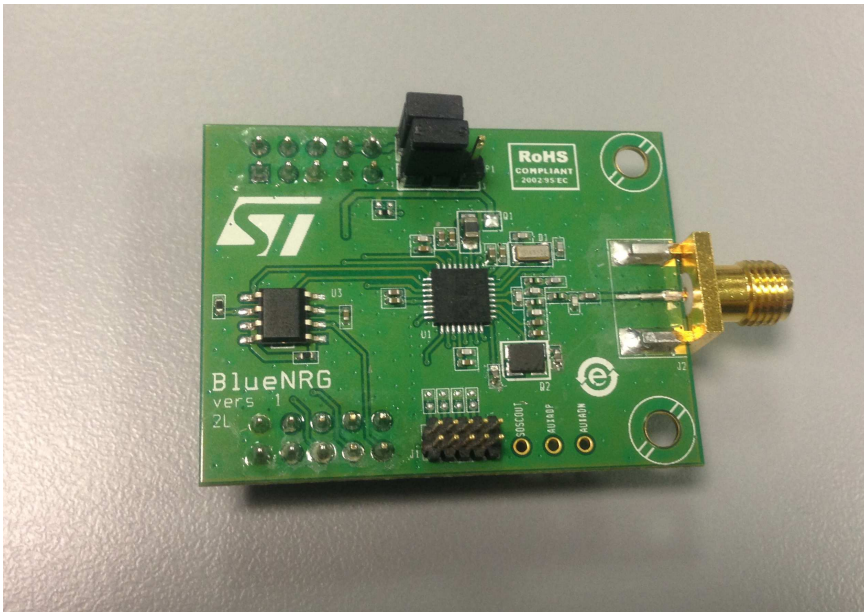
Some of the lines are connected also to a row of test points (M).

a. The STM32L is preprogrammed with a DFU firmware that allows the downloading of a firmware image without the use of a programmer. If an user accidentally erases DFU firmware, he can reprogram it through STLink using the hex image DFU_Bootloader.hex available on BlueNRG DK SW package, firmware folder.

2.2 BlueNRG daughterboard

The BlueNRG daughterboard (*Figure 4*) included in the development kit is a small circuit board to be connected to the main board. It contains the BlueNRG network processor (in a QFN32 package), an SMA antenna connector, discrete passive components for RF matching and balun, and small number of additional components required by the BlueNRG for proper operation (see the schematic diagram in *Figure 24*).

Figure 4. BlueNRG daughterboard



The main features of the BlueNRG daughterboard are:

- BlueNRG low power network processor for Bluetooth low energy (BLE), with embedded host stack
- High frequency 16 MHz crystal
- Low frequency 32 kHz crystal for the lowest power consumption
- Balun, matching network and harmonic filter
- SMA connector

The daughterboard is also equipped with a discrete inductor for the integrated high-efficiency DC-DC converter, for best-in-class power consumption. It is still possible to disable the DC-DC converter. In this case the following changes must be performed on the daughterboard (see *Figure 24*):

- Remove inductor from solder pads 1 and 2 of D1
- Place a 0 ohm resistor between pads 1 and 3
- Move resistor on R2 to R1

For proper operation, jumpers must be set as indicated in *Figure 4*.

The following tables show the connections between the daughterboard and the main board.

Table 2. Connections between BlueNRG board and motherboard on left connector

Pin	J4 motherboard	J3 daughterboard
1	DB_PIN1	NC
2	3V3	3V3
3	DB_PIN3	NC
4	NC	NC
5	GND	GND
6	DB_PIN2	nS
7	GND	GND
8	3V3	U2 pin 1
9	DB_SDN_RST	RST
10	3V3	U2 pin 1

Table 3. Connections between BlueNRG board and motherboard on right connector

Pin	J5 motherboard	J4 daughterboard
1	GND	GND
2	GND	GND
3	DB_CSN	CSN
4	DB_IO3_IRQ	IRQ
5	DB_SCLK	CLK
6	DB_IO2	NC
7	DB_SDI	MOSI
8	DB_IO1	NC
9	DB_SDO	MISO
10	DB_IO0	NC

2.2.1 Current measurements

To monitor power consumption of the entire BlueNRG daughterboard, remove the jumper from U2 and insert an ammeter between pins 1 and 2 of the connector. Since power consumption of the BlueNRG during most operation time is very low, an accurate instrument in the range of few microamps may be required.

2.2.2 Hardware setup

1. Plug the BlueNRG daughterboard into J4 and J5 connectors as in *Figure 1*.
2. Ensure the jumper configuration on the daughterboard is as in *Figure 1*
3. Connect the motherboard to the PC with an USB cable (through connector CN1).
4. Verify the PWR LED lights is on.

2.2.3 STM32L preprogrammed application

The STM32L on STEVAL-IDB002V1 motherboard is preprogrammed with the sensor demo application when the kits components are assembled (refer to *Section 5* for the application description).

2.3 STEVAL-IDB003V1 USB dongle

The BlueNRG USB dongle allows users to easily add BLE functionalities to their PC by plugging it into a USB port. The USB dongle can be used as a simple interface between the BlueNRG and a GUI application on the PC. The on-board STM32L microcontroller can also be programmed, so the board can be used to develop applications that use 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.

Below is a list of some of the main features that are available on the board (see *Figure 2*):

- BlueNRG network coprocessor
- STM32L151CBU6 48-pin microcontroller
- USB connector for power supply and I/O
- One row of pins with SWD interface
- Chip antenna
- Two user buttons (SW1, SW2)
- Two LEDs (D2, D3)

2.3.1 Microcontroller and connections

The board utilizes an STM32L151CBU6, which is an ultra low-power microcontroller with 128 KB of Flash memory, 16 KB of RAM, 32-bit core ARM cortex-M3, 4 KB of data EEPROM, RTC, timers, USART, I^2C, SPI, ADC, DAC and comparators.

The microcontroller is connected to various components such as buttons, LEDs and connectors for external circuitry. The following table shows which functionality is available on each microcontroller pin.

Table 4. MCU pin description versus board function

Pin name	Pin num.	Board function				
		LEDs	BlueNRG	Buttons	USB	SWD
VLCD	1		VBAT			
PC13	2					
PC14	3					
PC15	4					
OSC_IN	5					
OSC_OUT	6					
NRST	7					
VSS_A	8					
VDD_A	9					
PA0	10					
PA1	11			Button SW2		
PA2	12					
PA3	13					
PA4	14					
PA5	15					
PA6	16					
PA7	17					
PB0	18	Led D2				
PB1	19	Led D3				
PB2	20			Button SW1		
PB10	21		BlueNRG_IRQ			
PB11	22					
VSS1	23					
VDD1	24					
PB12	25		SPI2_CS			
PB13	26		SPI2_CLK			
PB14	27		SPI2_MISO			
PB15	28		SPI2_MOSI			
PA8	29					
PA9	30		EEPROM_CS			
PA10	31					
PA11	32				USB_DM	

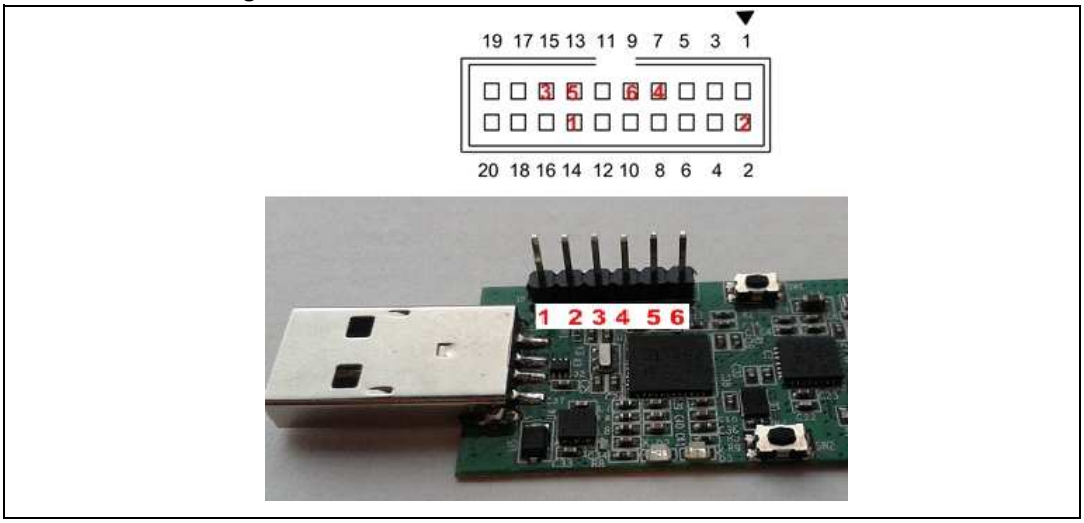
Table 4. MCU pin description versus board function (continued)

Pin name	Pin num.	Board function				
		LEDs	BlueNRG	Buttons	USB	SWD
PA12	33				USB_DP	
PA13	34					SWDIO
VSS2	35					
VDD2	36					
PA14	37					SWCLK
PA15	38					
PB3	39					SWO
PB4	40					
PB5	41					
PB6	42					
PB7	43					
BOOT0	44					
PB8	45					
PB9	46					
VSS_3	47					
VDD_4	48					

2.3.2 SWD interface

The SWD interface is available through the J2 pins. The SWD interface allows programming and debugging of the STM32L microcontroller on the board, using an in-circuit debugger and programmer like the ST-LINK/V2. In *Figure 5* the connection scheme illustrating how to connect the ST-LINK/V2 with the board pins is shown.

Figure 5. SWD connection scheme with ST-LINK/V2



The signals available on the STEVAL-IDB003V1 are:

1. GND
2. VDD
3. nRESET
4. SWDIO
5. SWO/TRACE
6. SWCLK

The connection to the ST-LINK/V2 interface is given in the table below, as shown in *Figure 5*:

Table 5. SWD connection

Signal name	STEVAL-IDS001Vx pin number	ST-LINK/V2 pin number
GND	1	14 /6
VDD	2	2 / 1
nRESET	3	15
SWDIO	4	7
SWO/TRACE	5	13
SWCLK	6	9

2.3.3 RF connector

The STEVAL-IDB003V1 provides two different RF connections: antenna (chip antenna, default configuration) and UFL connector. Although the default configuration allows communication on air, it can be useful to switch to the UFL connector in order to connect the STEVAL-IDB003V1 to RF equipment such as a spectrum analyzer or RF signal generator.

To switch from antenna to UFL connector, capacitor C10 must be removed and capacitor C42 must be soldered. To restore the default configuration and use the antenna, capacitor C42 must be removed and capacitor C10 must be soldered. Both capacitors C10 and C42 have the same value: 56 pF. In *Figure 6*, the two pads for C10 and C42 are shown together with the chip antenna and UFL connector.

Figure 6. RF connector scheme

2.3.4 Push-buttons

For user interaction the board has two buttons, both available to the application

- SW1
- SW2

Note: SW1 is the DFU button. The BlueNRG USB dongle is preprogrammed with a DFU application allowing upgrades to the STM32L firmware image through USB and using the BlueNRG GUI. To activate the DFU, press button SW1 and plug the BlueNRG USB dongle into a PC USB port.

2.3.5 User LEDs

Two LEDs are available:

- D2: red
- D3: orange

Note: When DFU is activated, LED D3 is blinking

2.3.6 Hardware setup

Plug the BlueNRG USB dongle into a PC USB port.

2.3.7 STM32L preprogrammed application

The STM32L on the STEVAL-IDB003V1 motherboard is preprogrammed with the BlueNRG_VCOM_x_x.hex application when the kits components are assembled (refer to *Section 3.1* for the application description).

3 GUI software description

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 BlueNRG network processor.

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.

3.1 Requirements

In order to use the BlueNRG GUI, make sure you have correctly set up your hardware and software (BlueNRG GUI installed). The STM32L in the STEVAL-IDB002V1 kit has been preprogrammed with a demo application (see *Section 5*). Hence, new firmware must be loaded into the STM32L. Firmware images can be found within the firmware folder. The firmware image that must be programmed is latest BlueNRG_VCOM_x_x.hex available within the BlueNRG DK SW package. The GUI has the ability to Flash new firmware.

In order to download binary images into the internal Flash of the STM32L, the microcontroller must be put into a special DFU (device firmware upgrade) mode. To enter DFU mode:

1. BlueNRG development platform (STEVAL-IDB002V1)
   - Power up the board
   - Press and hold USER button
   - Reset the board using RESET button (keep USER button pressed while resetting) The orange LED DL2 will start to blink
   - Release USER button
   - Use BlueNRG GUI to Flash the device with new firmware (Tools -> Flash motherboard FW).
2. BlueNRG USB Dongle (order code: STEVAL-IDB003V1)
   - Press and hold SW1 button
   - Plug the USB dongle on a PC USB port. The orange LED D3 will start to blink.
   - Use BlueNRG GUI to Flash the device with a new firmware (Tools -> Flash Motherboard FW).

3.2 The BlueNRG graphical user interface

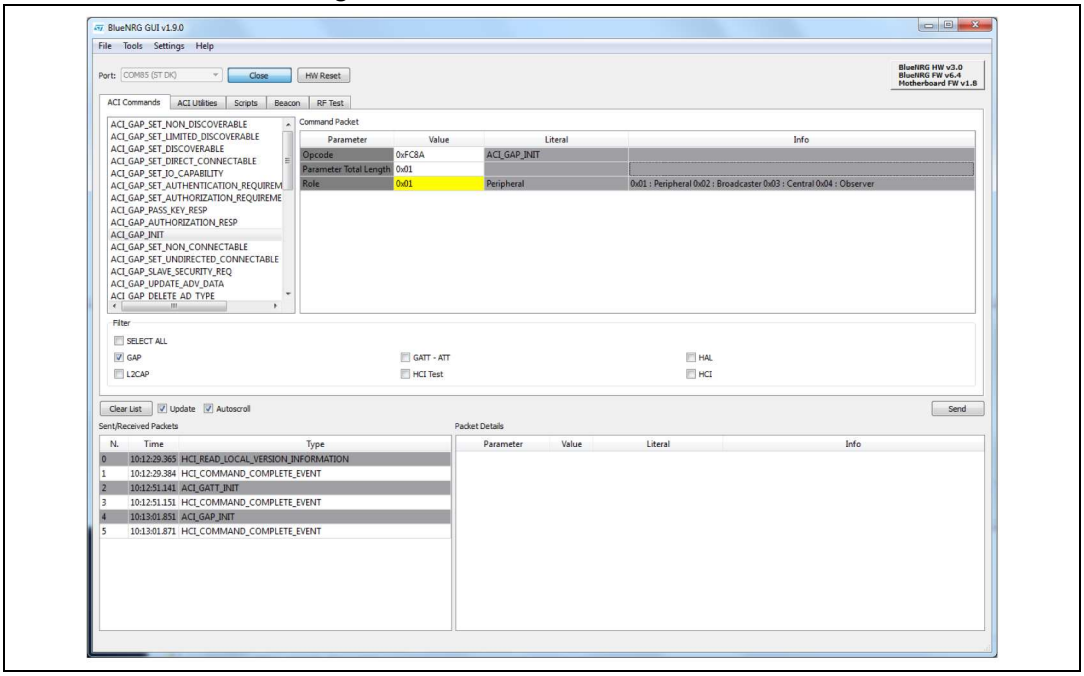
This section describes the main functions of BlueNRG GUI application.

You can run this utility by clicking on the BlueNRG GUI icon on the Desktop or under:

Start ⟶ STMicroelectronics ⟶ BlueNRG DK X.X.X ⟶ BlueNRG GUI

3.2.1 GUI main window

Figure 7. BlueNRG GUI main window



The BlueNRG GUI main window is characterized by different zones. Some of these zones can be resized.

Port and interface selection

The uppermost zone allows the user to open the COM port associated to the BTLE controller.

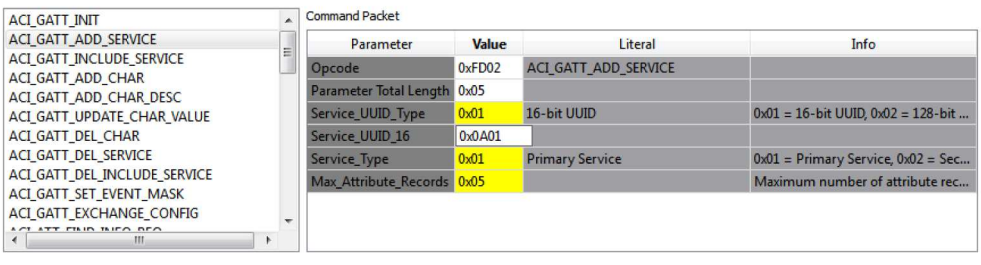
When a COM port is opened the following information are displayed:

– BlueNRG HW version
– BlueNRG FW version
– STM32L motherboard GUI firmware (VCOM) version

HCI commands

The HCI Commands tab contains a list of all the available HCI commands. Commands can be filtered by checking/unchecking boxes under the filter section. After clicking on one of the commands, all the packet fields will be displayed on the command packet table in the upper-right section of the tab (see *Figure 8*).

Figure 8. Command packet table



The command packet table contains four columns:

- **Parameter**: name of the packet field as they are named in volume 2, part E of Bluetooth specification.
- **Value**: field value represented in hexadecimal format (right-click on a cell to change its representation format).
- **Literal**: meaning of the current field value.
- **Info**: description of the corresponding field.

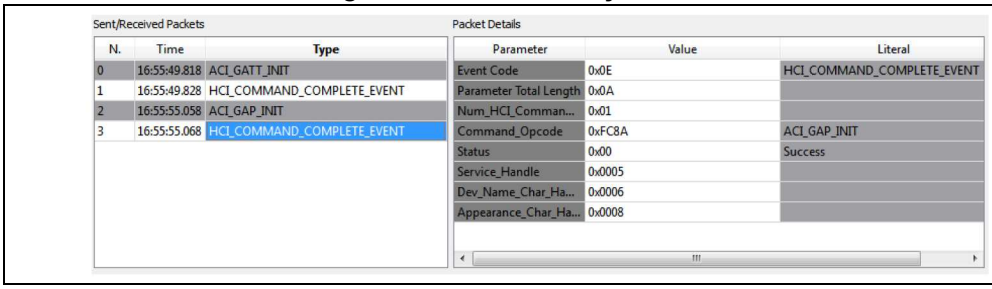
Only the yellow cells of this table can be modified by the user. The Parameter Total Length is fixed or automatically calculated after modifying cell content.

After the fields have been modified (if required) the command can be sent using the Send button.

HCI Packet history and details

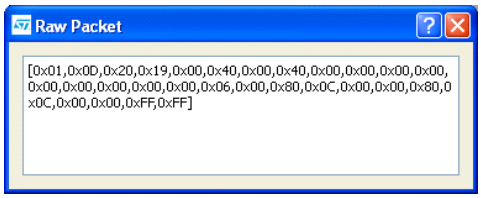
At the bottom of the main window, two tables show packets sent to and received from the BTLE controller, as well as other events. The left table (sent/received packets) holds a history of all packets (see *Figure 9*). The right one (packet details) shows all the details of the selected packet as is done in the command packet table (*Figure 9*).

Figure 9. Packet history and details



Double-clicking on a row of the sent/received packets table shows the raw packet.

Figure 10. Raw packet dump



Some events (displayed in yellow cells) can provide other information. HCI packets sent towards the BTLE controller are displayed in gray cells while received packets are shown inside white cells.

The Sent/received packets table can be cleared by clicking on clear list button. Update and auto-scrolling check boxes enable or disable updating and auto-scrolling of the Sent/received packets table while new packets are sent or received (however, information will still be printed).

The sent/received packets can be stored and later reloaded on the GUI, by using the utilities provided on File menu:

1. Save History... : it saves the current list of sent commands and received events on a CSV file
2. Load History... : it loads a list of sent commands and received events, previously stored on a CSV file.
3. Save as Python Script... : it allows to store the current list of sent commands and received events as a script file (Python format). This script file can be used on GUI Script window, after proper customization (by adding specific code for handling events, parameters, ...), in order to address an user application scenario (refer to *Section 3.2.5: GUI Beacon window*).

3.2.2 Tools

The BlueNRG GUI has some functions that can be accessed through the tools menu. These tools are described in this section.

BlueNRG updater

This tool can be used to update the firmware inside the BlueNRG by using its internal bootloader. VCOM firmware must be present on the STM32L and COM port must be open, in order to use this function.

1. Go to Tools -> BlueNGR updater
2. Select the correct stack firmware (.img)
3. Press update to start the update procedure. If the procedure completes with no errors, the new firmware has been loaded into the BlueNRG internal Flash.

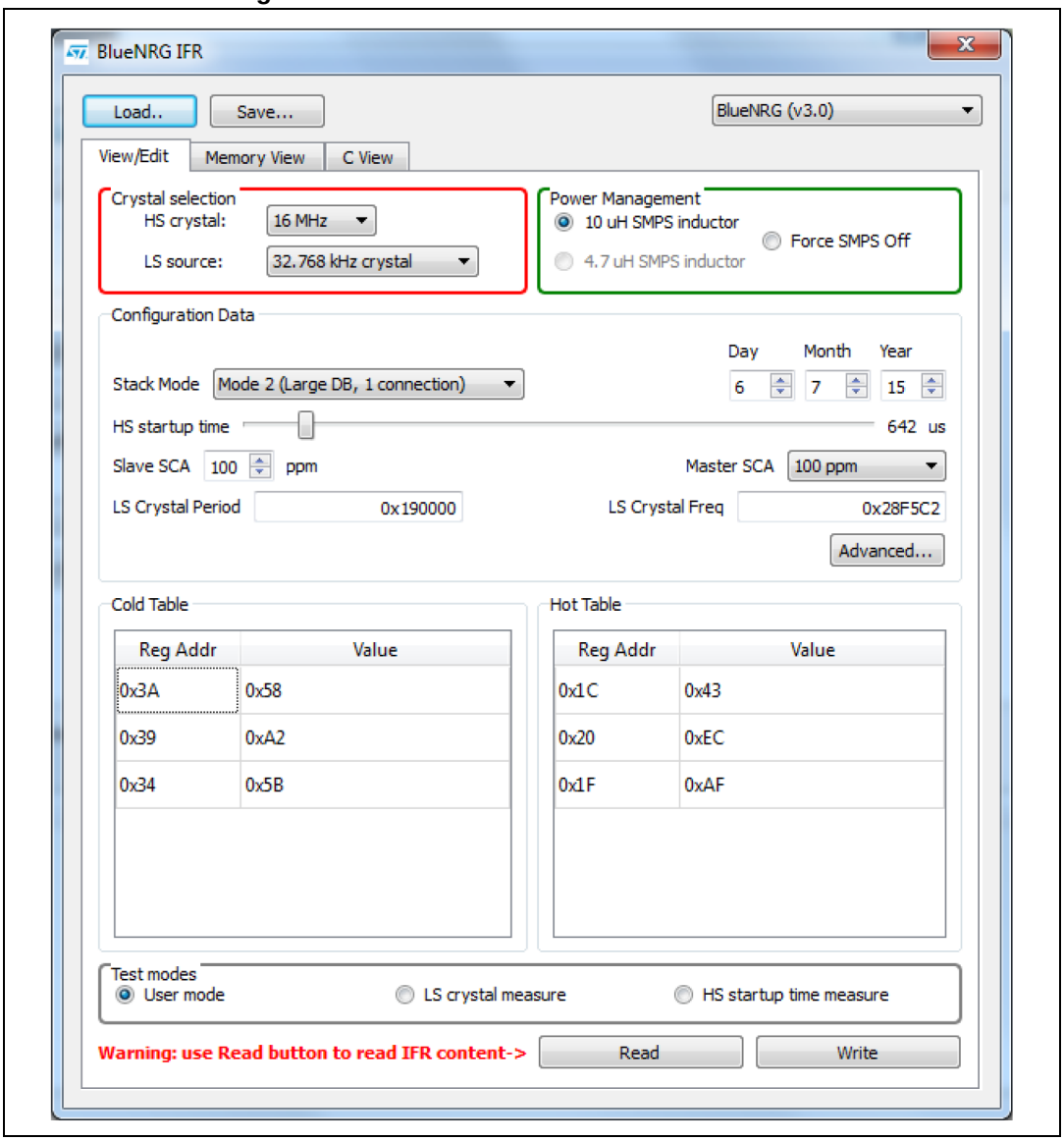
BlueNRG IFR

To preserve BlueNRG's flexibility, its firmware uses a table of configurable parameters. This table resides in a sector of the Flash called Information Register (IFR). The BlueNRG IFR tool can read and modify this portion of BlueNRG's Flash. This tool is available in BlueNRG GUI, Tools, BlueNRG IFR... item.

The BlueNRG GUI IFR utility is a tool that allow the customer to define the IFR data in a controller way. Using this utility is the only supported mode to define IFR data based on customer needs. The utility provides the following windows:

- View/Edit view: displays the IFR regions with related fields and description. The user can modify some of these fields according to his needs.
- Memory view: displays the IFR field memory addresses and related values that are generated by BlueNRG GUI according to the specified values.
- C view: displays the C language structure related to the IFR configuration data region matching the View/Edit and Memory view.

Figure 11. BlueNRG GUI IFR tool: View/Edit view



In the View/Edit view, the following operations are available:

- Select the high speed (HS) crystal (16 or 32 MHz) and the low speed oscillator source (32 kHz or the internal ring oscillator)
- Set the Power Management options (SMPS inductor or SMPS off configuration)
- Change stack mode. Each mode has a different functionality:
  - Mode 1: slave/master, 1 connection only, small GATT database (RAM2 off during sleep)
  - Mode 2: slave/master, 1 connection only, large GATT database (RAM2 on during sleep)
  - Mode 3: only master, 8 connections, small GATT database (RAM2 on during sleep)
- Change HS startup time parameter. This parameter control the time offset between the wakeup of the device and the start of RX/TX phase. It must be big enough to allow the device to be ready to transmit or receive after wakeup from sleep. This time depends on the startup time of the high speed crystal.

- Change sleep clock accuracy. This must reflect the actual clock accuracy, depending on the low speed oscillator or crystal in use.
- Set low speed (LS) crystal period and frequency
- View/change date to distinguish between different versions of configurations.
- View registers that are written into the radio (hot and cold table)
- Set some test modes for specific tests
- Read IFR content from BlueNRG.
- Write IFR configuration to BlueNRG IFR.

The following general utilities are also available:

- Load button: allows to load a configuration file.
- Save button: allows to save the current parameters into a configuration file.

Flash motherboard firmware

The BlueNRG GUI embeds a utility that allows to Flash firmware to the STM32L microcontroller on the motherboard without a JTAG/SWD programmer. This utility uses a bootloader that has been programmed in the first 12 KB of the Flash. Any application to be programmed to the STM32L by this tool must first consider that the lower area of the Flash is used by the bootloader[(b)].

OTA bootloader

OTA bootloader is a tool that allows to Flash new firmware to the STM32L of a remote device via Bluetooth low energy technology. Refer to the dedicated application note for more information.

Get production data

From the tools menu it is possible to retrieve production information from the BlueNRG daughterboard. This data is stored in the EEPROM on the daughterboard.

Get version

The Get version tool is used to retrieve the version of the BlueNRG GUI firmware (VCOM) on the STM23L, and hardware and firmware version from the BlueNRG.

Settings

This tool allows to configure the firmware stack version to be used from the GUI (when no device is actually connected to a PC USB port). Further, it allows to configure the GUI serial baud rate (valid only for communication over serial UART and not through USB Virtual COM).

In order to use this function:

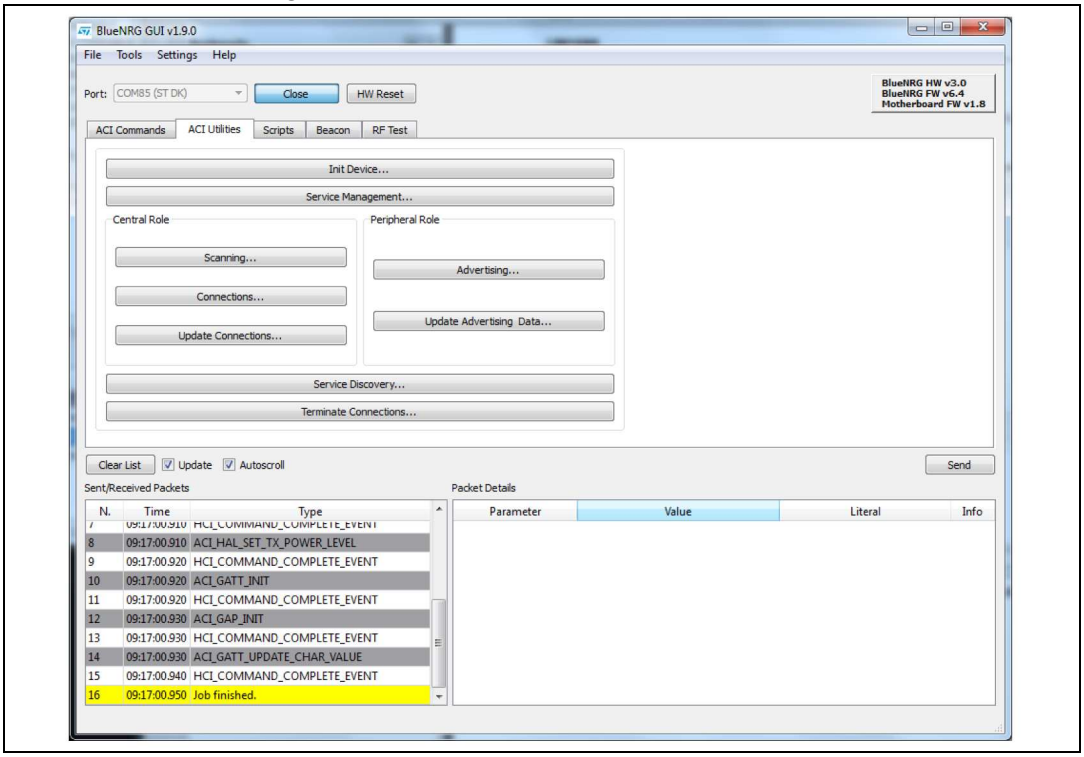
1. Go to Settings --> FW 6.4 for BlueNRG device
2. Go to Settings --> select Set Baud Rate… and choose the value (default is 115200)

b. Two precautions must be taken for any firmware: 1) change memory regions in linker script (vector table and Flash must start at 0x08003000); 2) Change the vector table offset (NVIC_SetVectorTable())

3.2.3 GUI ACI utilities window

The BlueNRG GUI ACI utilities window provides several tabs to allow testing of some BlueNRG application scenarios.

Figure 12. BlueNRG GUI ACI utilities window



Central and Peripheral roles are supported with the BLE operations described in *Table 6*, *Table 7* and *Table 8*.

Table 6. GUI ACI utilities window: available general operations

Operation	Associated actions	Notes
Init Device…	Allows to initialize a device by selecting: - Role - Stack Mode (1,2,3); - Address type (Public, Random) and value - Tx power level - Power mode - Device Name	
Service Management …	Allows to add a service by selecting: - UUID type (16 or 128 bits) - Service Type (Primary or Secondary) - Set max number of records For each service, it allows to add a characteristic by selecting: - UUID type (16 or 128 bits) - Properties - Security permissions - Variable length or not - Length - GATT Event mask - Encryption key size	After a characteristic is defined, the user can edit its parameters and/or delete it. Once a service and its characteristics have been defined, click OK to add them.
Service Discovery...	Allows to discover all services and related characteristics of available connections.	Service start handle, end handle and UUID are showed. For each selected Service the related Characteristics information are showed (attribute handle, property, value handle and UUID). For the available characteristic with Notify or Indication Property it's possible to enable the Notification/Indication.
Terminate Connection...	Allows to terminate the available connections	

Table 7. GUI ACI utilities window: available central operations

Operation	Associated actions	Notes
Scanning	Allows to put device in scanning mode by selecting: - GAP Procedure (Limited, general, general-connection establishment and terminate general-connection establishment procedures) - Enable or Disable filters - Set own address type - Set passive or active scan - Set Scanning interval and Window	
Connection	Allows to connect to a peer device by: - Searching for devices in Advertising - Select the device to which to connect - Select the connection parameters - Peer address and type - Scan Interval and Window - Connection Interval (min & max) - Latency - Supervision timeout - Connection event length (min & max)	The addresses of the detected advertising devices are displayed
Update Connections	Allows to update the connection parameters of available connections by: - Selecting the specific connection to be updated - Set the new connection parameters - Connection interval (min & max) - Latency - Supervision timeout - Connection event length (min & max)	

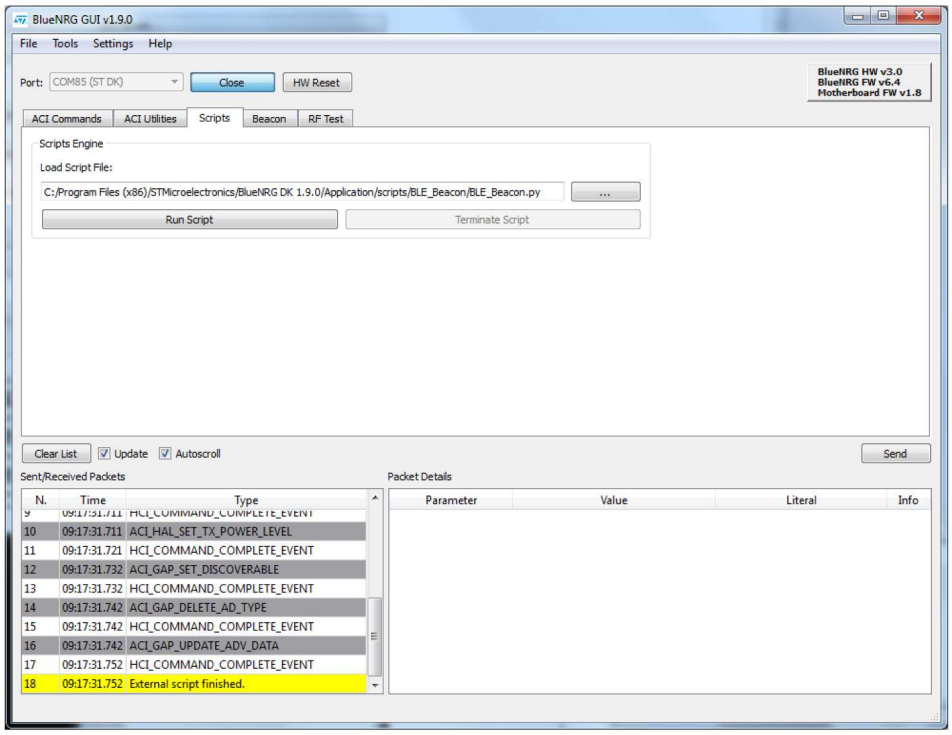
Table 8. GUI ACI utilities window: available peripheral operations

Operation	Associated actions	Notes
Advertising	Allows to put a Peripheral device in Advertising mode by selecting: - Discoverable mode (limited, non discoverable and general discoverable) - Type (ADV_IND, ADV_SCAN_IND, ADV_NONCONN_IND) - Set Local name and type (complete or short) - Advertising intervals (min & max) - Policy: - Allow scan request from any, allow connect request from any - Allow scan request from white list only, allow connect - request from any - Allow scan request from any, allow connect request from white list only	
Update Advertising Data	It allows to update the advertising data; It allows to set the scan response data; It allows to update the location UUID, major and minor number defined on the Beacon window	

3.2.4 GUI Scripts window

The GUI Scripts window allows the user to load and run a Python script built using the available set of BlueNRG ACI commands and the related events. For a list of supported HCI and ACI script commands and related parameters, refer to the commands available in the BlueNRG GUI ACI Commands window.

Figure 13. BlueNRG GUI Scripts window



Moreover, the script engine supports other utility commands:

Table 9. GUI Scripts window: utility commands

Command name	Parameters	Description
HW_BOOTLOADER	None	Hardware bootloader activation
HW_RESET	None	HW reset
INFO	String to be displayed	Opens a message window and shows the input parameter. Script is blocked until user presses OK button
ERROR	User message	Raises an exception with a user-defined debug message
GET_CHAR	None	Allows user to enter a specific char as input (such as the C get_char() API)
GET_FILE	None	Allows the selection of a specific file as input
GET_NAME	None	Returns the device name within an advertising packet
GET_VALUE	Array of bytes	Converts the array of bytes to an integer value. Example: X = [0x33,0x22] GET_VALUE(X) = 0x2233

Table 9. GUI Scripts window: utility commands (continued)

Command name	Parameters	Description
GET_LIST	Integer, Number of bytes	Converts the integer value to an array of bytes. Example: X = 0x2233 GET_LIST(X, 2) = [0x33,0x22]
GET_STACK_VERSION	None	Returns the device information (HW version & FW version) as (hw, fw)
GET_RAND_KEY	None	Returns a random number between 0 and 999999
INSERT_PASS_KEY	None	Allows to enter a pass key value used for the security pass key method
PRINT	String	Print utility: displays information on GUI Sent/Received Packets
RESET	None	SW reset
SLEEP	time	It sleeps for “time” in milliseconds
SET_MODE	Mode	Set stack mode (1,2,3)
SET_PUBLIC_ADDRESS	Public address	Set public address (optional)
SENSORDEMO_GET_TEMPERATURE	None	Allows to obtain the temperature value from the ACI_ATT_READ_RESP_EVENT event (only for the SensorDemo_Central script)
SENSORDEMO_GET_ACCELERATION	None	Allows to obtain the acceleration values (x,y,z) from the ACI_GATT_NOTIFICATION_EVENT event (only for the SensorDemo_Central script)
TIME	None	Returns the time as a floating point number expressed in seconds since the epoch, in UTC

The following pseudo code describes how to initialize a BlueNRG device as a peripheral using a simple Python script:

```
# Reset BlueNRG
HW_RESET()

# Init GATT
ACI_GATT_INIT()

# Init GAP as central device
ACI_GAP_INIT(Role=CENTRAL)
```

When a script is calling a command which generates specific events, the script can detect them by using the WAIT_EVENT (event_code=None, timeout=None, continueOnEvtMiss=False, **param_checks) command.

Table 10. WAIT_EVENT macro-command

Command name	Description	Parameters	Return
WAIT_EVENT	Waits for an event with 'Event Code' parameterequal to event_code. If no event_code is indicated, the macro-command waits for any event. Optional filtering parameters allow definition of additional filters on event fields	event_code = None (default)	An event with its parameters
		timeout = None (default)	None, if a timeout occurs and the input parameter “continueOnEvtMiss” is set to True
		continueOnEvtMiss = False (default)	An HCITimeoutError error exception is raised when a timeout occurs
		param_checks = optional filtering parameters	evt.get_param(“parameter_name”).val is used for getting the specific event

The WAIT_EVENT macro-command waits for an event with 'Event Code' parameter equal to event_code. If no event_code is indicated, the macro-command waits for any event.

The timeout parameter allows to set the event timeout. If no timeout is set, the macro-command waits until an event occurs. If a timeout (greater than zero) is set and continueOnEvtMiss is False and no event occurs before the timeout, an HCITimeoutError error occurs. Otherwise, if the input parameter continueOnEvtMiss is True and a timeout (greater than zero) is set, the macro-command returns the value None even when no event occurs before the timeout.

If one or more optional filtering parameters are specified, the macro-command performs a check on them and returns only the first detected event that satisfies these parameters. The events received before the one returned are discarded.

The WAIT_EVENT() command return value can be:

- an event
- None, if a timeout occurs and the input parameter “continueOnEvtMiss” is set to True

An HCITimeoutError error exception is raised when a timeout occurs

The event_code parameter can be one of the following values:

Table 11. WAIT_EVENT: event codes with related event parameter types

event_code	Event parameter type	Event parameter type value
HCI_LE_META_EVENT	Subevent_Code	HCI_LE_CONNECTION_COMPLETE_EVENT
		HCI_LE_ADVERTISING_REPORT_EVENT
		HCI_LE_CONNECTION_UPDATE_COMPLETE_EVENT
		HCI_LE_READ_REMOTE_USED_FEATURES_COMPLETE_EVENT
		HCI_LE_LONG_TERM_KEY_REQUEST_EVENT

Table 11. WAIT_EVENT: event codes with related event parameter types (continued)

event_code	Event parameter type	Event parameter type value
HCI_VENDOR_EVENT	Ecode	ACI_BLUE_INITIALIZED_EVENT
		ACI_GAP_LIMITED_DISCOVERABLE_EVENT
		ACI_GAP_PAIRING_COMPLETE_EVENT
		ACI_GAP_PASS_KEY_REQ_EVENT
		ACI_GAP_AUTHORIZATION_REQ_EVENT
		ACI_GAP_SLAVE_SECURITY_INITIATED_EVENT
		ACI_GAP_BOND_LOST_EVENT
		ACI_GAP_DEVICE_FOUND_EVENT
		ACI_GAP_PROC_COMPLETE_EVENT
		ACI_L2CAP_CONNECTION_UPDATE_RESP_EVENT
		ACI_L2CAP_PROC_TIMEOUT_EVENT
		ACI_L2CAP_CONNECTION_UPDATE_REQ_EVENT
		ACI_GATT_ATTRIBUTE_MODIFIED_EVENT
		ACI_GATT_PROC_TIMEOUT_EVENT
		ACI_ATT_EXCHANGE_MTU_RESP_EVENT
		ACI_ATT_FIND_INFO_RESP_EVENT
		ACI_ATT_FIND_BY_TYPE_VALUE_RESP_EVENT
		ACI_ATT_READ_BY_TYPE_RESP_EVENT
		ACI_ATT_READ_RESP_EVENT
		ACI_ATT_READ_BLOB_RESP_EVENT
		ACI_ATT_READ_MULTIPLE_RESP_EVENT
		ACI_ATT_READ_BY_GROUP_TYPE_RESP_EVENT
		ACI_ATT_WRITE_RESP_EVENT
		ACI_ATT_PREPARE_WRITE_RESP_EVENT
		ACI_ATT_EXEC_WRITE_RESP_EVENT
		ACI_GATT_INDICATION_EVENT
		ACI_GATT_NOTIFICATION_EVENT
		ACI_GATT_PROC_COMPLETE_EVENT
		ACI_GATT_ERROR_RESP_EVENT
		ACI_GATT_DISC_READ_CHAR_BY_UUID_RESP_EVENT
		ACI_GATT_WRITE_PERMIT_REQ_EVENT
		ACI_GATT_READ_PERMIT_REQ_EVENT
		ACI_GATT_READ_MULTI_PERMIT_REQ_EVENT
HCI_DISCONNECTION_COMPLETE_EVENT		
HCI_ENCRYPTION_CHANGE_EVENT		

Table 11. WAIT_EVENT: event codes with related event parameter types (continued)

event_code	Event parameter type	Event parameter type value
HCI_READ_REMOTE_VERSION_INFORMATION_COMPLETE_EVENT		
HCI_COMMAND_COMPLETE_EVENT		
HCI_COMMAND_STATUS_EVENT		
HCI_HARDWARE_ERROR_EVENT		
HCI_NUMBER_OF_COMPLETED_PACKETS_EVENT		
HCI_DATA_BUFFER_OVERFLOW_EVENT		
HCI_ENCRYPTION_KEY_REFRESH_COMPLETE_EVENT		

Below are some code examples using the WAIT_EVENT() macro-command:

Example 1

```
# Wait any events
evt = WAIT_EVENT()
if evt.event_code == HCI_LE_META_EVENT:
# User specific code ……
elif evt.event_code==HCI_VENDOR_EVENT:
# User specific code ……
```

Example 2

```
# Wait an HCI_LE_META_EVENT
evt = WAIT_EVENT(HCI_LE_META_EVENT)
# Using evt.get_param('Subevent_Code').val it's possible to identify the specific HCI_LE_META_EVENT
# parameter type value
evtCode = evt.get_param('Subevent_Code').val
# Check if received event is HCI_LE_CONNECTION_COMPLETE_EVENT
if (evtCode == HCI_LE_CONNECTION_COMPLETE_EVENT):
    # If Connection Complete Status is success, get connection handle
    if evt.get_param('Status').val==0x00:
        conn_handle= evt.get_param('Connection_Handle').val
```

Example 3

```
# Wait HCI_VENDOR_EVENT event_code
evt = WAIT_EVENT(HCI_VENDOR_EVENT)
#Using evt.get_param('Ecode').val it's possible to identify the specific
HCI_VENDOR_EVENT event parameter type value
evtCode = evt.get_param('Ecode').val
if (evtCode == ACI_GATT_NOTIFICATION_EVENT):
    conn_handle=evt.get_param('Connection_Handle').val
```

Example 4

```
# Wait the Ecode ACI_GATT_PROC_COMPLETE_EVENT (HCI_VENDOR_EVENT
#event_code).
# if no event occurs within the selected timeout, an exception is raised
WAIT_EVENT(HCI_VENDOR_EVENT, timeout=30,
Ecode=ACI_GATT_PROC_COMPLETE_EVENT)
```

Note: If no timeout parameter is specified, it waits until the ACI_GATT_PROC_COMPLETE_EVENT event occurs.

Example 5

```
# Wait an event for 10 seconds with continueOnEvtMiss set to True
# If no event occurs, the script continues (no exception is raised).
WAIT_EVENT(timeout=10, continueOnEvtMiss =True)
```

Note: If the continueOnEvtMiss parameter is set to False and no event within the selected timeout occurs, an exception is raised.

Example 6

```
# Wait the HCI_DISCONNECTION_COMPLETE_EVENT event_code
WAIT_EVENT(HCI_DISCONNECTION_COMPLETE_EVENT)
```

Example 7

```
# Create a Connection and wait for the HCI_LE_CONNECTION_COMPLETE_EVENT
ACI_GAP_CREATE_CONNECTION(Peer_Address=[0x12, 0x34, 0x00, 0xE1, 0x80,
0x02])
event = WAIT_EVENT(HCI_LE_META_EVENT,
timeout=30,Subevent_Code=HCI_LE_CONNECTION_COMPLETE_EVENT)
if event.get_param('Status').val==0x00:
# Store the connection handle
    conn_handle= event.get_param('Connection_Handle').val
    # User defined code …
```

GUI script engine loading and running steps

To load and run a Python script using the BlueNRG GUI script engine, the following steps must be observed:

1. In the BlueNRG GUI, Scripts window, Script Engine section, click on tab "…", browse to the script location and select the script
2. Click on the "Run Script" tab to run the script. The execution flow (commands and events) will be displayed in the BlueNRG GUI "Sent/Received Packets" section

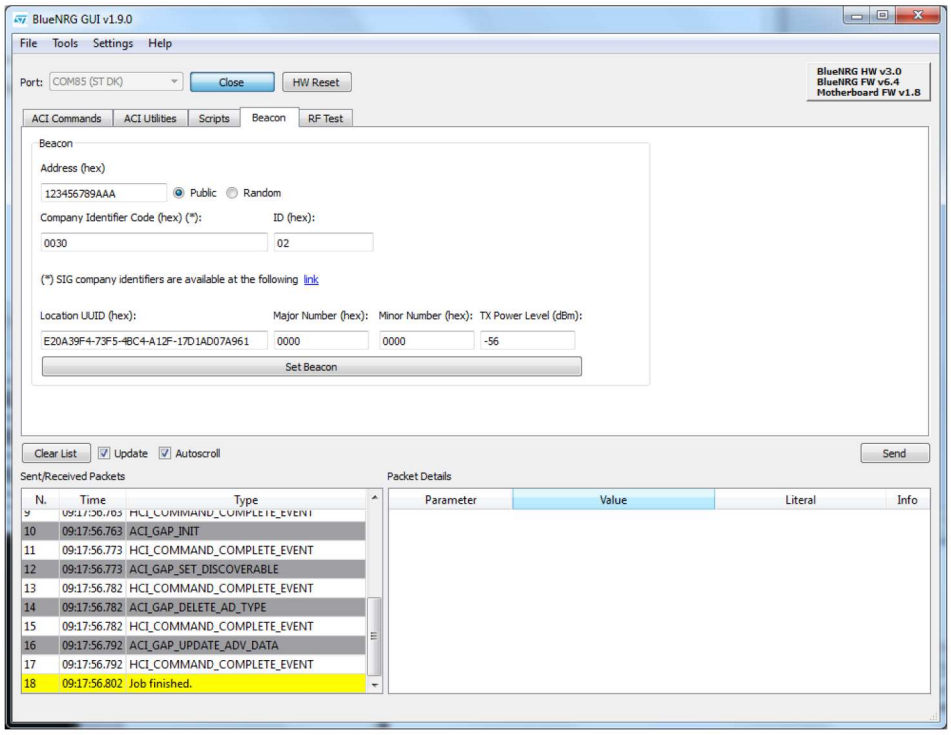
In the BlueNRG DK 1.7.0 and future versions, some reference BlueNRG scripts are available in the GUI/scripts folder.

Note: It is worthy of note that in order to write and use the BlueNRG scripts, the user is required to have some knowledge of the Python language (Python 2.7.6), and a good understanding of the BlueNRG ACI commands and related events.

3.2.5 GUI Beacon window

The BlueNRG GUI Beacon window provides some tabs allowing configuration of a BlueNRG device as a BLE Beacon device which transmits advertising packets with specific manufacturer data.

Figure 14. BlueNRG GUI Beacon window



The user can configure the following advertising data fields for the BLE Beacon device, through the BlueNRG GUI Beacon window configuration parameters.

Table 12. BlueNRG GUI beacon window configuration parameters

Data field	Description	Notes
Address	Device address	
Public or Random	Device address type	
Company Identifier Code	SIG company identifier	Default is 0x0030 (STMicroelectronics)
ID	Beacon ID	Fixed value

Table 12. BlueNRG GUI beacon window configuration parameters (continued)

Data field	Description	Notes
Location UUID	Beacons UUID	Used to distinguish specific beacons from others
Major number	Identifier for a group of beacons	Used to group a related set of beacons
Minor number	Identifier for a single beacon	Used to identify a single beacon
Tx Power Level	2's complement of the Tx power	Used to establish how far you are from device

To configure a BlueNRG platform as a BLE beacon device, click on “Set Beacon” tab.

3.2.6 GUI RF Test window

The BlueNRG GUI provides the RF Test window that permits the performance of the following tests:

1. Start/Stop a tone on a specific BLE RF channel
2. Perform BLE Packer Error Rate (PER) tests using BLE Direct Test Mode (DTM) commands

Start/Stop a tone

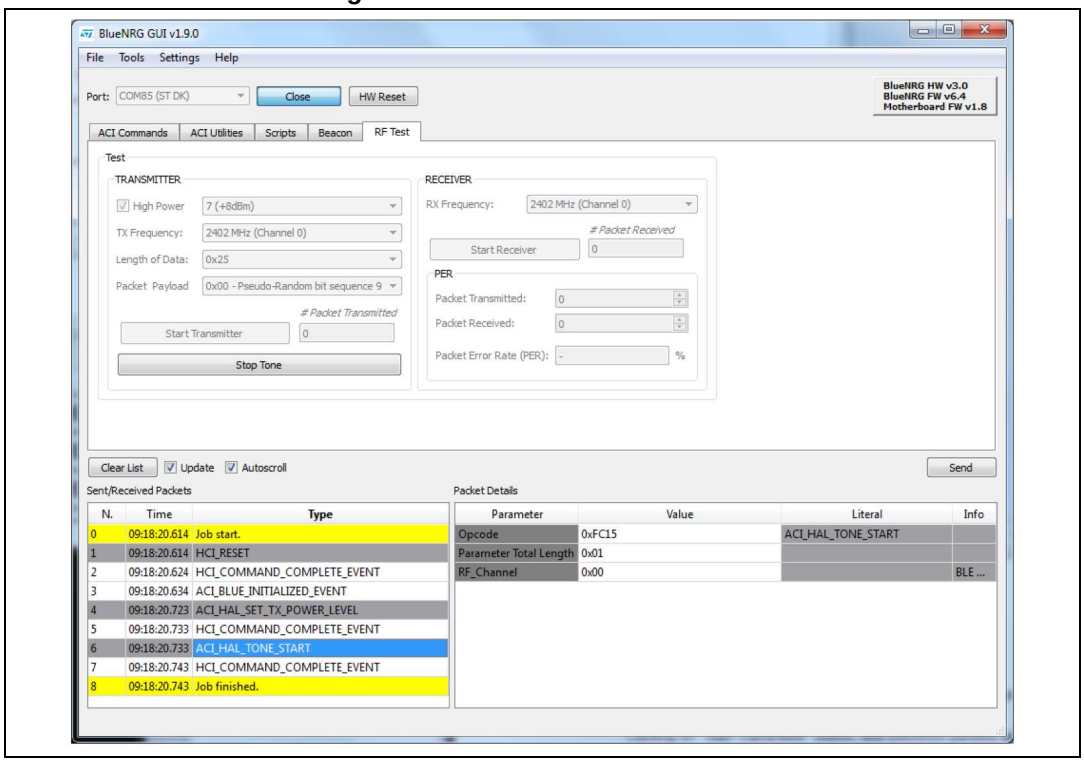
To start a tone on a specific RF BLE channel, perform these steps:

1. Connect a BlueNRG platform to a PC USB port
2. Launch an instance of the BlueNRG GUI
3. Open related COM port
4. Go to RF Test window and in the TRANSMITTER section:
   – Set the BLE channel using the TX Frequency combo box
   – Set TX power in the related combo box
   – Click on the “Start Tone” button

To stop a tone on a specific RF BLE channel, perform these steps:

1. Go to RF Test window and in the TRANSMITTER section:
   – Click on the Stop Tone button (the Stop button is available only when a tone is started)

Figure 15. GUI RF test: Start a tone



Direct Test Mode (DTM) tests

The BlueNRG GUI provides an RF test using the BLE Direct Test Modes commands that allows users to target a packet error rate test scenario.

Two sections are available:

1. TRANSMITTER section for transmitting reference packets at a fixed interval
2. RECEIVER section for receiving reference packets at a fixed interval

TRANSMITTER section

This section permits to set the following items:

- The power level of the transmitter
- The Frequency of the transmitter
- Length of data to transmit in each packet

Packet payload format as defined in the Bluetooth Low Energy specification, Direct Test Mode section

By clicking on the “Start Transmitter” button, test reference packets will be sent at a fixed interval.

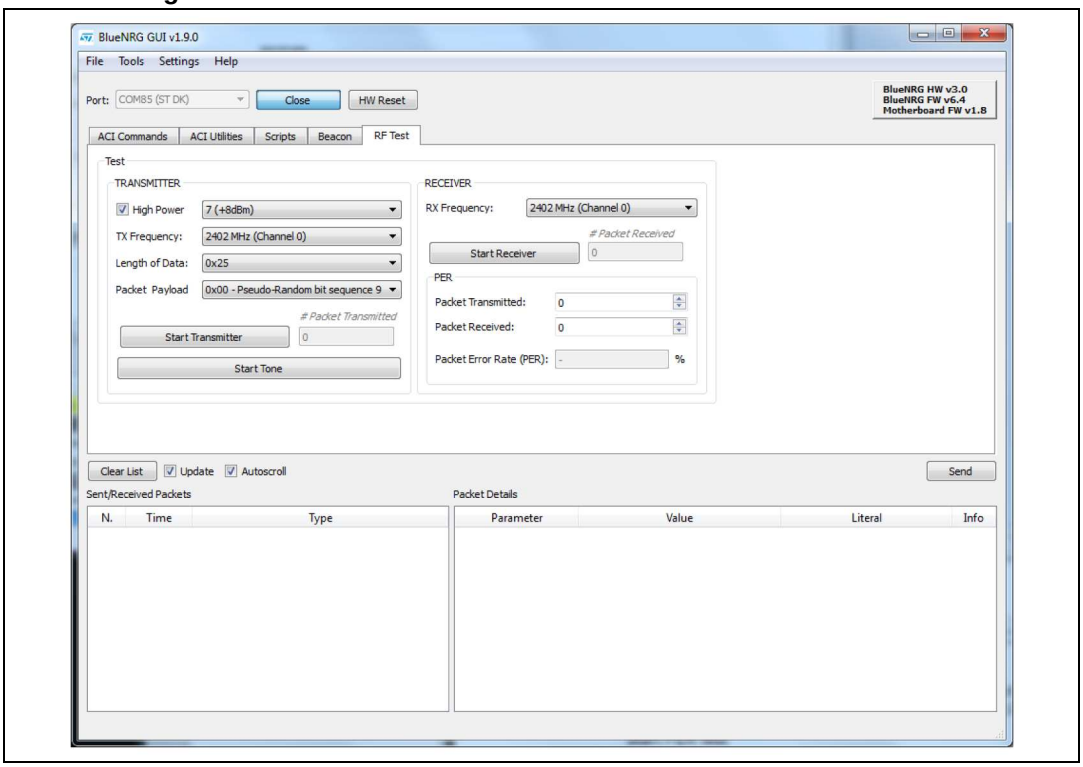
RECEIVER section

This section permits to set the following items:

- The Frequency of the receiver

By clicking on the “Receiver Test” button, test reference packets will be received at a fixed interval.

Figure 16. GUI RF Test: TRANSMITTER and RECEIVER sections



Packet Error Rate (PER) test procedure

To perform a Packet Error Rate test using standard BLE Direct Test Mode commands (HCI_LE_Transmitter_Test, HCI_LE_Receiver_Test and HCI_LE_Test_End), it is necessary to perform the following procedure:

Start PER test

1. Connect two BlueNRG platforms (TX and RX) to PC USB ports
2. Open two instances of BlueNRG GUI (one for TX and RX BlueNRG devices)
3. In each instance of the BlueNRG GUI, Open the COM port related to TX/RX BlueNRG device
4. Ensure that the antennas are plugged into the BlueNRG devices, where applicable
5. In the BlueNRG GUI related to the RX BlueNRG device,

Go to the RF Test window, RECEIVER section:

- Set the RX frequency
- Click on “Start Receiver” button to start the Receiver test

6. In the BlueNRG GUI related to TX BlueNRG device,

Go to RF Test window, TRANSMITTER section:

- Set TX power
- Set TX frequency
- Set Length of data
- Set Packet payload format
- Click on the “Start Transmitter” button, to start the Transmitter test

Stop PER test

1. In the BlueNRG GUI related to TX BlueNRG device,
   – Go to the RF Test window, TRANSMITTER section
   – Click on “Stop Transmitter” button. The number of transmitted packets is displayed in the #Packet Transmitted field
2. In the BlueNRG GUI related to Rx BlueNRG device,

Go to the RF Test window, RECEIVER section:

– Click on the “Stop Receiver” button. The number of received packets is displayed in the #Packet Received field.

Get PER (Packet Error Rate) value

1. In the BlueNRG GUI related to RX BlueNRG device,
   – Go to the RF Test window, RECEIVER section
   – In the PER section, insert the number of transmitted packet from TX device into the Packet Transmitted field (read this value from TRANSMITTER section in the BlueNRG GUI related to TX device)
   – The PER (packet error rate) value is shown in the Packet Error Rate field

Figure 17. GUI RF Test, PER test: TX device

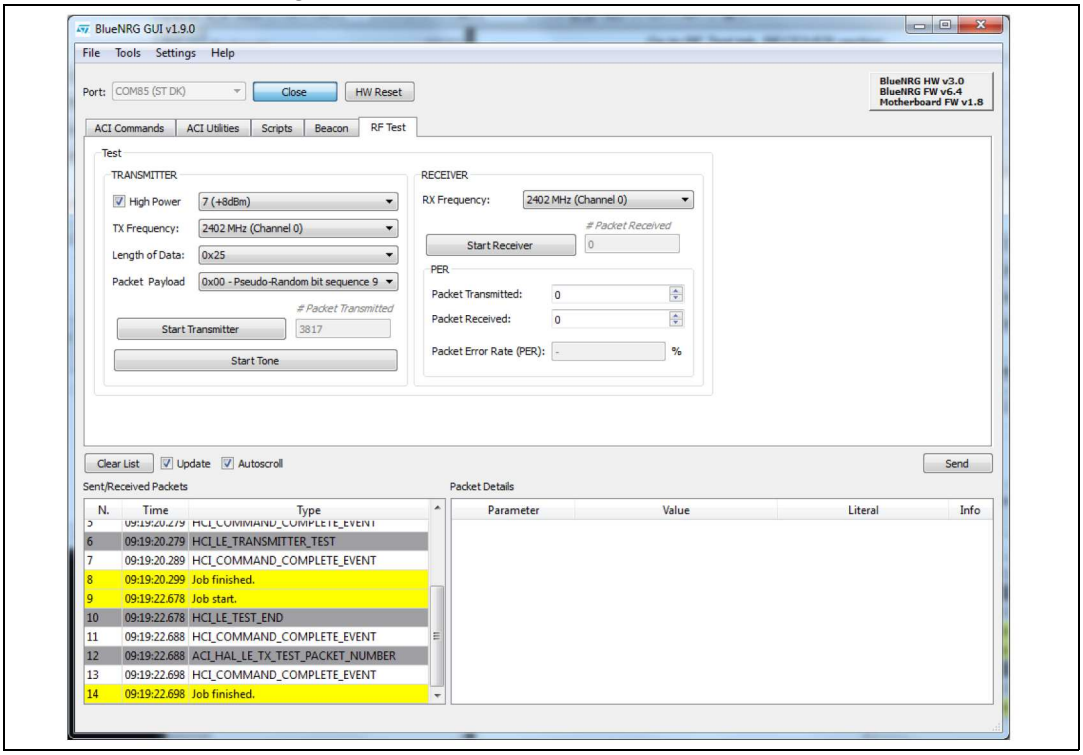
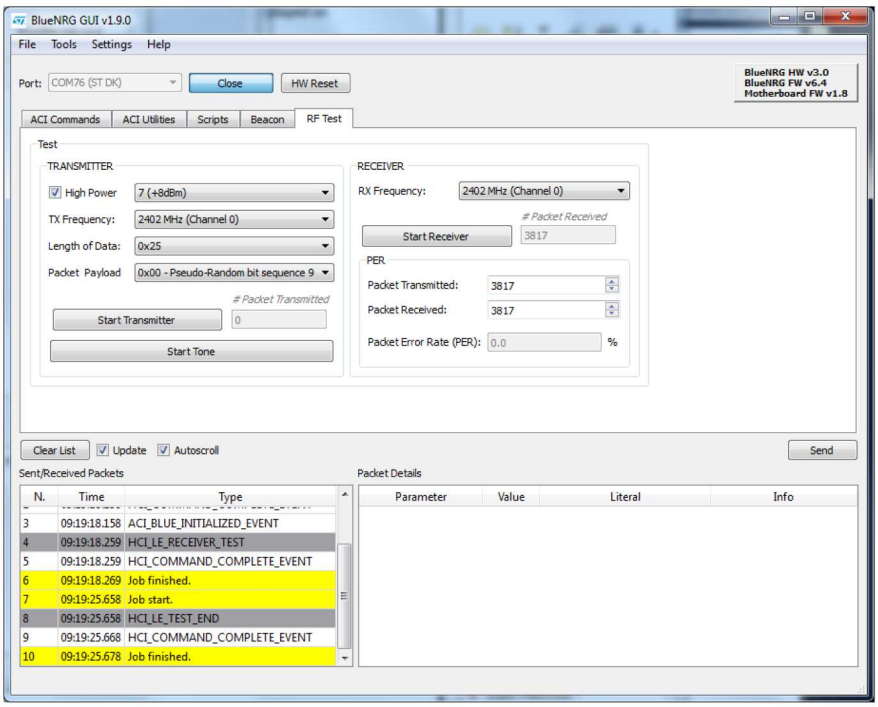


Figure 18. GUI RF Test, PER test: RX device

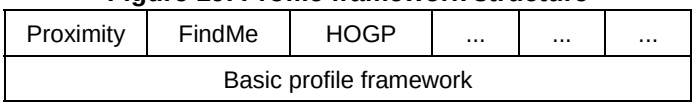


4 Programming with BlueNRG network processor

The BlueNRG provides a high level interface to control its operation. This interface is called ACI (application-controller interface). The ACI is implemented as an extension to the standard Bluetooth HCI interface. Since BlueNRG is a network processor, the stack runs inside the device itself. Hence, no library is required on the external microcontroller, except for profiles and all the functions needed to communicate with the BlueNRG SPI interface.

The development kit software includes sample code that shows how to configure BlueNRG and send commands or parsing events. The source library is called simple BlueNRG HCI to distinguish it from the library for the complete profile framework (not present in the software development kit). This library is able to handle multiple profiles at the same time and supports several Bluetooth GATT-based profiles for BlueNRG. Documentation on the ACI is provided in a separate document.

Figure 19. Profile framework structure



4.1 Requirements

In order to communicate with BlueNRG network processor very few resources are needed by the main processor. These are listed below:

- SPI interface
- Platform-dependent code to write/read to/from SPI
- A timer to handle SPI timeouts or to run Bluetooth LE Profiles

Minimum requirements in terms of Flash and RAM space largely depend on the functionality needed by the application, on the microprocessor that will run the code and on the compiler toolchain used to build the firmware.

On the STM32L (Cortex-M3 core), the memory footprint for the code interfacing the BlueNRG requires few kilobytes of Flash and RAM (typically 2-4 KB of Flash, and 0.8-1.5 KB of RAM). So a complete simple application (like the BlueNRG sensor demo) could require just 15 KB of Flash and 2 KB of RAM.

If using the complete BlueNRG profile framework, the memory footprint is around 9 KB of code and 3 KB of data for just the ACI interface and the profile framework functions. The memory required for the profiles can vary depending on the complexity of the profile itself. For example, code for HID-over-GATT host is around 6 KB, while for heart rate monitor is around 2.3 KB.

4.2 Software directory structure

The Projects folder contains some sample code that can be used on the application processor to control the BlueNRG. Platform-dependent code is also provided for STM32L1 platforms. The example project provided in the package will run “as is” on the development kit.

The files are organized using the following folder structure:

- **Drivers.** It contains all the STM32L1xx Cube library framework files.
- **Middlewares\ST\STM32_BlueNRG\SimpleBlueNRG_HCI.** Contains the code that is used to send ACI commands to the BlueNRG network processor. It contains also definitions of BlueNRG events.
- **platform.** Contains all the platform-dependent files (only on STM32L1xx standard library framework). These can be taken as an example to build applications that can be run on other platforms.
- **Project_Cube, Projects_STD_Library.** Contains source based, respectively, on STM32L1xx Cube library and on STM32L1xx standard library frameworks, that will use the Bluetooth technology with the BlueNRG. Project files for IAR embedded workbench are also available.

5 BlueNRG sensor profile demo

The software development kit contains an example, which implements a proprietary Bluetooth profile: the sensor profile. This example is useful for building new profiles and applications that use the BlueNRG network processor. This GATT profile is not compliant to any existing specification. The purpose of this project is simply to show how to implement a given profile.

This profile exposes two services: acceleration service and environmental service. *Figure 20* shows the whole GATT database, including the GATT and GAP services that are automatically added by the stack.

One of the acceleration service's characteristics has been called free-fall characteristic. This characteristic cannot be read or written but can be notified. The application will send a notification on this characteristic (with value equal to 0x01) if a free-fall condition has been detected by the LIS3DH MEMS sensor (the condition is detected if the acceleration on the 3 axes is near zero for a certain amount of time). Notifications can be enabled or disabled by writing on the related client characteristic configuration descriptor.

The other characteristic exposed by the service gives the current value of the acceleration that is measured by the accelerometer. The value is made up of six bytes. Each couple of bytes contains the acceleration on one of the 3 axes. The values are given in mg. This characteristic is readable and can be notified if notifications are enabled.

Another service is also defined. This service contains characteristics that expose data from some environmental sensors: temperature, pressure and humidity[(c)]. For each characteristic, a characteristic format descriptor is present to describe the type of data contained inside the characteristic. All of the characteristics have read-only properties

c. An expansion board with LPS25H pressure sensor and HTS221 humidity sensor can be connected to the motherboard through the expansion connector (F in *Figure 3*). If the expansion board is not detected, only temperature from STLM75 will be used.

Figure 20. BlueNRG sensor demo GATT database

#	Handle	UUID (16 or 128bit)	Attribute Type	Properties	Initial Parameter Value	Comment
1	0001	2800	Primary Service		{Service=0x1801 ("Attribute Profile")}	
2	0002	2803	Characteristic	X	{handle=0x0003, UUID=0x2A05}	
3	0003	2A05	**Service Changed**		{start handle=0x0001, end handle=0xFFFF}	
4	0004	2902	Client Characteristic Configuration		0x0000	
5	0005	2800	Primary Service		{Service=0x1800 ("Generic Access Profile")}	
6	0006	2803	Characteristic	X X X X	{handle=0x0007, UUID=0x2A00}	
7	0007	2A00	**Device Name**		"bluenrg"	
8	0008	2803	Characteristic	X X X	{handle=0x0009, UUID=0x2A01}	
9	0009	2A01	**Appearance**		0x0000	
16	0010	2800	Primary Service		{Service=0x02366E80CF3A11E19AB40002A5D5C51B ("Acc Service")}	
17	0011	2803	Characteristic	X	{handle=0x0012, UUID=0xE23E78A0CF4A11E18FFC0002A5D5C51B}	
18	0012	E23E78A0CF4A11E18FFC0002A5D5C51B	**Free Fall**		0x00	Indication with value 1 when a freel fall condition is detected
19	0013	2902	Client Characteristic Configuration		0x0000	
20	0014	2803	Characteristic	X X	{handle=0x0015, UUID=0x340A1B80CF4B11E1AC360002A5D5C51B}	
21	0015	340A1B80CF4B11E1AC360002A5D5C51B	**Acceleration**		0x000000000000	X-Axis (2bytes) Y-Axis (2bytes) Z-Axis (2bytes)
22	0016	2902	Client Characteristic Configuration		0x0000	
23	0017	2800	Primary Service		{Service=0x42821A40E47711E282D00002A5D5C51B ("Env Service")}	
24	0018	2803	Characteristic	X	{handle=0x0019, UUID=0xA32E5520E47711E2A9E30002A5D5C51B}	
25	0019	A32E5520E47711E2A9E30002A5D5C51B	**Temperature**		0x0000	Temperature in tenths of degree Celsius
26	001A	2904	Characteristic Format		{format=0x0E, exp=-1, unit=0x272F, n_sp=0x00, descr=0x0000}	format=sint16, unit=temperature celsius
27	001B	2803	Characteristic	X	{handle=0x001C, UUID=0xCD20C480E48B11E2840B0002A5D5C51B}	
28	001C	CD20C480E48B11E2840B0002A5D5C51B	**Pressure**		0x000000	Pressure in hundredths of millibar
29	001D	2904	Characteristic Format		{format=0x0F, exp=-5, unit=0x2780, n_sp=0x00, descr=0x0000}	format=sint24, unit=pressure bar
30	001E	2803	Characteristic	X	{handle=0x001F, UUID=0x01C50B60E48C11E2A0730002A5D5C51B}	
31	001F	01C50B60E48C11E2A0730002A5D5C51B	**Humidity**		0x0000	Humidity in tenths of RH
32	0020	2904	Characteristic Format		{format=0x06, exp=-1, unit=0x2700, n_sp=0x00, descr=0x0000}	format=uint16, unit=unitless

5.1 Supported platforms

The BlueNRG sensor profile demo is supported only on the BlueNRG development platform (STEVAL-IDB002V1).

5.2 BlueNRG app for smartphones

An application is available for smartphones (iOS and android), that works with the sensor profile demo. The development kits are preprogrammed with the sensor profile demo firmware. If the development board has been flashed with another firmware, it can be programmed with the correct firmware. Refer to *Section 4.1* for the programming procedure using the device firmware upgrade feature and BlueNRG GUI. The correct pre-compiled firmware can be found inside firmware folder (SensorDemo.hex). The source file for the demo is inside the project folder.

This app enables notifications on the acceleration characteristic and displays the value on the screen. Data from environmental sensors are also periodically read and displayed.

Figure 21. BlueNRG app

5.3 BlueNRG sensor profile demo: connection with a central device

This section describes how to interact with a central device, while BlueNRG is acting as a peripheral. The central device can be another BlueNRG acting as a master, or any other Bluetooth smart or smart-ready device.

First, BlueNRG must be set up. In order to do this, a series of ACI command need to be sent to the processor.

5.3.1 Initialization

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()
- aci_gap_init(GAP_PERIPHERAL_ROLE,&service_handle, &dev_name_char_handle, &appearance_char_handle);

Where: Role = GAP_PERIPHERAL_ROLE.

See ACI documentation for more information on these commands and on those that follow as well. Peripheral role must be specified inside the GAP_INIT command.

5.3.2 Add service and characteristics

BlueNRG's Bluetooth LE stack has both server and client capabilities. A characteristic is an element in the server database where data are exposed. A service contains one or more characteristics. Add a service using the following command. Parameters are provided only as an example.

- aci_gatt_add_serv(0x01, 0xA001, 0x01, 0x06, & Service_Handle);

Where: Service_UUID_Type=0x01, Service_UUID_16=0xA001, Service_Type=0x01, Max_Attributes_Records=0x06.

The command will return the service handle on variable Service_Handle (e.g., 0x0010). A characteristic must now be added to this service. This service is identified by the service handle.

– aci_gatt_add_char (Service_Handle, 0x01, 0xA002, 10, 0x1A,0x00, 0x01, 0x07, 0x01, &Char_Handle);

Where: Char_UUID_Type=0x01, Char_UUID_16=0xA002, Char_Value_Length=10, Char_Properties=0x1A,Security_Permissions=0x00, GATT_Evt_Mask=0x01, Enc_Key_Size=0x07, Is_Variable=0x01.

With this command a variable-length characteristic has been added, with read, write and notify properties. The characteristic handle is also returned (Char_Handle).

5.3.3 Set 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. Let's assume we want the user to insert a passcode during the pairing procedure.

– aci_gap_set_authentication_requirement (0x01, 0,0, 7, 16, 123456, 1);

Where: Char_UUID_Type=0x01, Char_UUID_16=0xA002, Char_Value_Length=10, Char_Properties=0x1A,Security_Permissions=0x00, GATT_Evt_Mask=0x01, Enc_Key_Size=0x07, Is_Variable=0x01.

5.3.4 Enter connectable mode

Use GAP ACI commands to enter one of the discoverable and connectable modes.

– aci_gap_set_discoverable (0x00, 0x800,0x900, 0x00, 0x00, 0x08, local_name, 0x00, 0x00, 0x0000, 0x0000);

Where: Advertising_Type=0x00, Advertising_Interval_Min=0x800, Advertising_Interval_Max=0x900, Own_Address_Type=0x00, Advertising_Filter_Policy=0x00, Local_Name_Length=0x08, local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,'B','l','u','e','N','R','G'}; Service_UUID_Length=0x00, Service_UUID_List=0x00, Slave_Connection_Interval_Min=0x0000, Slave_Connection_Interval_Max=0x0000.

The Local_Name parameter contains the name that will be present in advertising data, as described in Bluetooth core specification version 4.0, Vol. 3, Part C, Ch. 11.

5.3.5 Connection with central device

Once BlueNRG is put in a discoverable mode, it can be seen by a central device in scanning.

Any Bluetooth smart and smart-ready device can connect to BlueNRG, such as a smartphone. LightBlue is one of the applications in the Apple store for iPhone® 4S/5 and later versions of Apple's iPhone.

Start the LightBlue application. It will start to scan for peripherals. A device with the BlueNRG name will appear on the screen. Tap on the box to connect to the device. A list of all the available services will be shown on the screen. Touching a service will show the characteristics for that service.

BlueNRG has added two standard services: GATT Service (0x1801) and GAP service (0x1800).

Try to read the characteristic from the service just added (0xA001). The characteristic has a variable length attribute, so you will not see any value. Write a string into the characteristic and read it back.

BlueNRG can send notifications of the characteristic that has been previously added, with UUID 0xA002 (after notifications have been enabled). This can be done using the following command:

– aci_gatt_update_char_value (Service_Handle, Char_Handle, 0,0x05,'hello');

where: Val_Offset=0, Char_Value_Length=0x05, Char_Value='hello'.

Once this ACI command has been sent, the new value of the characteristic will be displayed on the phone.

5.4 BlueNRG sensor demo: central profile role

This application implements a basic version of the BlueNRG Sensor Profile Central role which emulates the BlueNRG Sensor Demo applications available for smartphones (iOS and Android).

It configures a BlueNRG device as a BlueNRG Sensor device, Central role which is able to find, connect and properly configure the free fall, acceleration and environment sensor characteristics provided by a BlueNRG development platform, configured as a BlueNRG Sensor device, Peripheral role.

This application uses a new set of APIs that allow the performance of the following operations on a BlueNRG Master/Central device:

– Master Configuration Functions
– Master Device Discovery Functions
– Master Device Connection Functions
– Master Discovery Services & Characteristics Functions
– Master Data Exchange Functions
– Master Security Functions
– Master Common Services Functions

These APIs are provided through binary libraries available on Projects\Bluetooth LE\Profile_Framework_Central\library. The master library APIs are documented in doxygen format within the SW package.

The BlueNRG Sensor Demo Central role is supported on the BlueNRG development platform (STEVAL-IDB002V1) and on the BlueNRG USB dongle (STEVAL-IDB003V1).

The sections that follow describe how to use the master library APIs for configuring a BlueNRG Sensor Demo Central device.

5.4.1 Initialization

BlueNRG's master library must be correctly initialized before establishing a connection with another

Bluetooth LE device. This is done with this command:

– Master_Init(&param)

param variable allows to set the initialization parameters (device address, name, …).

Refer to the master library doxygen documentation for more information about the command and related parameters.

On the application main loop, the Master_Process() API has to be called in order to process the Master library state machines.

5.4.2 Discovery a sensor peripheral device

In order to discover a Sensor Peripheral device, a discovery procedure has to be started with the master library command:

– Master_DeviceDiscovery(&devDiscParam);

devDiscParam variable allows to set the discovery parameters (discovery procedure, interval, window, …).

Refer to the master library doxygen documentation for more information about the command and related parameters.

The found devices are returned through the Master_DeviceDiscovery_CB() master library callback (DEVICE_DISCOVERED status).

5.4.3 Connect to discovered sensor peripheral device

Once a Sensor Peripheral device has been found, the Sensor Central device connects to it by using the following master library command:

– Master_DeviceConnection(&connParam);

connParam variable allows to set the connection parameters (connection procedure, scan duration, window,…).

Refer to the master library doxygen documentation for more information about the command and related parameters.

When the connection is established with success, the Master_Connection_CB() master library callback is called with CONNECTION_ESTABLISHED_EVT event.

5.4.4 Discovery sensor peripheral services and characteristics

Once a Sensor Peripheral device has been connected, the Sensor Central device starts discovery all primary service procedure, by using the following master library command:

– Master_GetPrimaryServices()

Refer to the master library doxygen documentation for more information about the command and related parameters.

When services are discovered, the Master_ServiceCharacPeerDiscovery_CB master library callback is called with PRIMARY_SERVICE_DISCOVERY code. In particular the sensor and environmental services are discovered.

For each discovered service, the related characteristics are discovered by using the following master library command:

– Master_GetCharacOfService()

Refer to the master library doxygen documentation for more information about the command and related parameters.

When the characteristics of a service are discovered, the Master_ServiceCharacPeerDiscovery_CB master library callback is called with GET_CHARACTERISTICS_OF_A_SERVICE code. In particular the sensor acceleration, free fall and temperature characteristics are discovered.

5.4.5 Enable sensor peripheral acceleration and free fall notifications

Once the Sensor Peripheral device sensor acceleration and free fall characteristics have been discovered, the Sensor Central device can enable the related characteristics notification by using the following master library command:

– Master_NotifIndic_Status(masterContext.connHandle, handle, TRUE, FALSE);

Refer to the master library doxygen documentation for more information about the command and related parameters.

When a characteristic notification is enabled, the Master_PeerDataExchange_CB() master library callback is called with NOTIFICATION_INDICATION_CHANGE_STATUS code. On a Sensor Central device context, the sensor acceleration and free fall characteristics notifications coming from the Sensor Peripheral device are received through the Master_PeerDataExchange_CB() master library callback, NOTIFICATION_DATA_RECEIVED code. Each received values is displayed on the connected hyper terminal (115200, 8, N, 1).

5.4.6 Read the sensor peripheral temperature sensor characteristic

Once the Sensor Peripheral device sensor temperature characteristic is discovered, the Sensor Central device can read the related characteristic value by using the following master library command:

– Master_Read_Value()

Refer to the master library doxygen documentation for more information about the command and related parameters.

The characteristic value is received though the Master_PeerDataExchange_CB() master library callback, READ_VALUE_STATUS code. Each received value is also displayed on the connected hyper terminal (115200, 8, N, 1).

6 BlueNRG chat demo application

The software development kit contains another example, which implements a simple 2-way communication between two BlueNRG devices. It shows a simple point-to-point wireless communication using the BlueNRG product.

This demo application exposes one service: chat service.

The chat service contains 2 characteristics:

- The TX characteristic: the client can enable notifications on this characteristic. When the server has data to be sent, it will send notifications which will contain the value of the TX characteristic.
- The RX characteristic: this is a writable characteristic. When the client has data to be sent to the server, it will write a value into this characteristic.
- The maximum length of the characteristic value is 20 bytes.

There are 2 device roles which can be selected through the specific EWARM workspace:

– The “Server” that exposes the chat service (BLE peripheral device).

– The “Client” that uses the chat service (BLE central device).

The application requires 2 devices to be programmed respectively with the 2 devices roles: server and client. The user must connect the 2 devices to a PC through USB and open a serial terminal on both, with the following configurations:

Table 13. Serial port configuration

Baudrate	115200	bit/sec
Data bits	8	bit
Parity	None	bit
Stop bits	1	bit

The application will listen for keys typed into one device and upon pressing the keyboard return key, it will send them to the remote device. The remote device will listen for RF messages and will output them in the serial port. In other words, anything typed in one device will be visible to the other device.

6.1 Supported platforms

The BlueNRG chat demo (server & client roles) is supported on the BlueNRG development platform (STEVAL-IDB002V1) and on the BlueNRG USB dongle (STEVAL-IDB003V1).

6.2 BlueNRG chat demo application: peripheral & central devices

This section describes how two BLE chat devices (server-peripheral & client-central) interact with each other in order to set up a point-to-point wireless chat communication.

First, BlueNRG must be set up on both devices. In order to do this, a series of ACI commands need to be sent to the processor.

6.2.1 Initialization

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 Chat, "Server" role:
  - aci_gap_init(GAP_PERIPHERAL_ROLE, &service_handle, &dev_name_char_handle, &appearance_char_handle);
- BLE Chat, "Client role:
  - aci_gap_init(GAP_CENTRAL_ROLE, &service_handle, &dev_name_char_handle, &appearance_char_handle);

Peripheral & central BLE roles must be specified inside the GAP_INIT command. See ACI documentation for more information on these commands and on those that follow.

6.2.2 Add service and characteristics

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

aci_gatt_add_serv(UUID_TYPE_128, service_uuid, PRIMARY_SERVICE, 7, &chatServHandle);

Where service_uuid is the private service UUID 128 bits allocated for the chat service (Primary service).

The command will return the service handle in chatServHandle.

The TX characteristic is added using the following command (on BLE Chat, Server role device):

aci_gatt_add_char(chatServHandle, UUID_TYPE_128, charUuidTX, 20, CHAR_PROP_NOTIFY, ATTR_PERMISSION_NONE, 0, 16, 1, &TXCharHandle);

Where charUuidTX is the private characteristic UUID 128 bits allocated for the TX characteristic (notify property). The characteristic handle is also returned (on TXCharHandle).

The RX characteristic is added using the following command (on BLE Chat, Server role device):

aci_gatt_add_char(chatServHandle, UUID_TYPE_128, charUuidRX, 20, CHAR_PROP_WRITE|CHAR_PROP_WRITE_WITHOUT_RESP, ATTR_PERMISSION_NONE, GATT_SERVER_ATTR_WRITE,16, 1, &RXCharHandle);

Where charUuidRX is the private characteristic UUID 128 bits allocated for the RX characteristic (write property). The characteristic handle is also returned (on RXCharHandle).

See ACI documentation for more information on these commands as well as those that follow.

6.2.3 Set 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 BLE chat 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);

6.2.4 Enter connectable mode

On BLE chat, server role device uses GAP ACI commands to enter into general discoverable mode:

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

The local_name parameter contains the name that will be present in advertising data, as described in the Bluetooth core specification version 4.0, Vol. 3, Part C, Ch. 11.

6.2.5 Connection with central device

Once the BLE chat, server role device is put in a discoverable mode, it can be seen by the BLE chat, client role device in order to create a Bluetooth low energy connection.

On BLE chat, client role device uses GAP ACI commands to connect with the BLE chat, 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 chat, client role device.

Once the 2 devices are connected, the user can set up a serial terminal and type into each of them. The typed characters will be respectively stored in 2 buffers and upon pressing the keyboard return key, BLE communication will work as follows:

1. On BLE chat, server role device, the typed characters will be sent to BLE chat, client role device by notifying the TX characteristic that has been previously added (after notifications have been enabled). This can be done using the following command:

aci_gatt_update_char_value(chatServHandle,TXCharHandle,0,len,(tHalUint8 *)cmd+j)

2. On BLE chat, client role device, the typed characters will be sent to the BLE chat, server role device, by writing the RX characteristic that has been previously added. This can be done using the following command:
   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.

Once these ACI commands have been sent, the values of the TX, RX characteristics are displayed on the serial terminals.

Figure 22. BLE chat client example

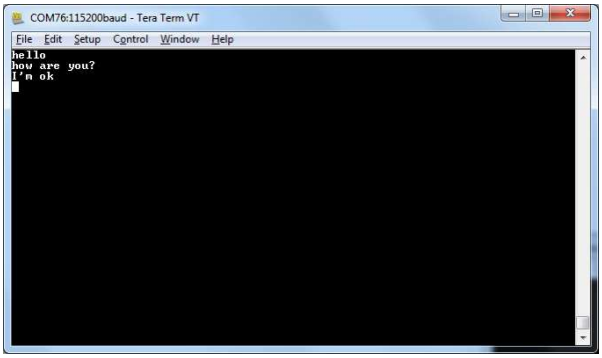
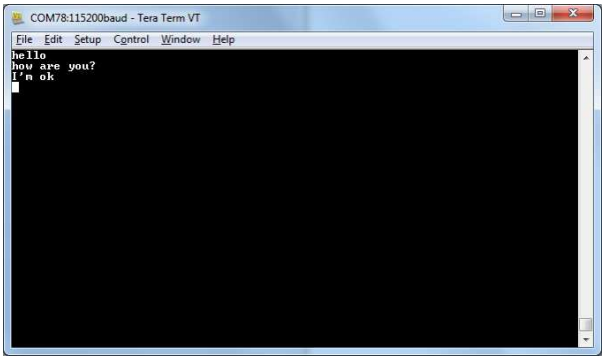


Figure 23. BLE chat server example



7 BlueNRG Beacon demonstration application

The software development kit contains another example, which shows how to configure a BlueNRG device to advertise specific manufacturing data and allow another BLE device to know if it is in the range of the BlueNRG beacon device.

7.1 Supported platforms

The BlueNRG Beacon demo is supported by the BlueNRG development platform (STEVAL-IDB002V1) and the BlueNRG USB dongle (STEVAL-IDB003V1).

7.2 BLE Beacon application setup

This section describes how to configure a BlueNRG device for acting as a beacon device.

7.2.1 Initialization

The BlueNRG stack must be correctly initialized as follows:

– aci_gatt_init()
– aci_gap_init(GAP_PERIPHERAL_ROLE, &service_handle, &dev_name_char_handle, &appearance_char_handle);

7.2.2 Define advertising data

The BLE Beacon application advertises the following manufacturing data:

Table 14. BlueNRG Beacon advertising manufacturing data

Data field	Description	Notes
Company identifier code	SIG company identifier	Default is 0x0030 (STMicroelectronics)
ID	Beacon ID	Fixed value
Location UUID	Beacons UUID	Used to distinguish specific beacons from others
Major number	Identifier for a group of beacons	Used to group a related set of beacons
Minor number	Identifier for a single beacon	Used to identify a single beacon
Tx Power	2's complement of the Tx power	Used to establish how far you are from device

Note: SIG company identifiers are available at:

https://www.bluetooth.org/en-us/specification/assigned-numbers/company-identifiers

7.2.3 Entering non-connectable mode

The BLE Beacon device uses the GAP ACI command to enter non-connectable mode as follows:

aci_gap_set_discoverable(ADV_NONCONN_IND, 160, 160, PUBLIC_ADDR, NO_WHITE_LIST_USE,0, NULL, 0, NULL, 0, 0);

In order to advertise the specific selected manufacturer data, the BLE Beacon application uses the following GAP ACIs:

```
/* Remove TX power level field from the advertising data: it is necessary to have enough space for the beacon manufacturing data */

ret = aci_gap_delete_ad_type(AD_TYPE_TX_POWER_LEVEL);

/* Define the beacon manufacturing payload */

const uint8_t manuf_data[] = {26, AD_TYPE_MANUFACTURER_SPECIFIC_DATA,
    0x30, 0x00, //Company identifier code (Default is 0x0030 - STMicroelectronics)
    0x02,       // ID
    0x15,       //Length of the remaining payload
    0xE2, 0x0A, 0x39, 0xF4, 0x73, 0xF5, 0x4B, 0xC4, //Location UUID
    0xA1, 0x2F, 0x17, 0xD1, 0xAD, 0x07, 0xA9, 0x61,
    0x00, 0x00, // Major number
    0x00, 0x00, // Minor number
    0xC8        //2's complement of the Tx power (-56dB)};
  };

/* Set the beacon manufacturing data on the advertising packet */

ret = aci_gap_update_adv_data(27, manuf_data);
```

8 BLE remote control demo application

This demo application shows how to control a remote device (like an actuator) using a BlueNRG device.This application periodically sends broadcast data (temperature values) that can be read by any device. The broadcast data is encapsulated in a manufacturer-specific AD type. The data content (besides the manufacturer ID, i.e. 0x0030 for STMicroelectronics) is as follows:

Table 15. BLE remote advertising data

Byte 0	Byte 1	Byte2
App ID (0x05)	Temperature value (little-endian)	

The temperature value is given in tenths of degrees Celsius.

The device is also connectable and exposes a characteristic used to control the LEDs on the BlueNRG platform. The value of this characteristic is a bitmap of 1 byte. Each bit controls one of the LEDs:

- bit 0 is the status of LED 1
- bit 1 is the status of LED 2.
- bit 2 is the status of LED 3.
- bit 3 is the status of LED 4.
- bit 4 is the status of LED 5.

As a consequence, a remote device can connect and write this byte to change or read the status of these LEDs (1 for LED ON, 0 for LED OFF).

The peripheral disconnects after a timeout (DISCONNECT_TIMEOUT), to prevent that a central is always connected to the device.

By default, no security is used, but it can be enabled with ENABLE_SECURITY (refer to file BLE_RC_main.h). When security is enabled the central has to be authenticated before reading or writing the device characteristic.

In order to interact with a BlueNRG device configured as a BLE Remote control, another BLE device (a BlueNRG or any SMART READY device) can be used to scan and see broadcast data.

To control one of the LEDs, the device has to connect to a BlueNRG BLE Remote Control device and write into the exposed control point characteristic. The Service UUID is ed0ef62e-9b0d-11e4-89d3-123b93f75cba. The control point characteristic UUID is ed0efb1a-9b0d-11e4-89d3-123b93f75cba.

8.1 Supported platforms

The BlueNRG BLE Remote Control is supported on the BlueNRG development platform (STEVAL-IDB002V1) and on the BlueNRG USB dongle (STEVAL-IDB003V1).

8.2 BLE remote control application setup

This section describes how to configure a BlueNRG device to acting as a remote control device.

8.2.1 Initialization

The 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()
- aci_gap_init(GAP_PERIPHERAL_ROLE, &service_handle, &dev_name_char_handle, &appearance_char_handle)

8.2.2 Define advertising data

The BLE Remote Control application advertises some manufacturing data as follows:

/* Set advertising device name as Node */

const uint8_t scan_resp_data[] =
{0x05,AD_TYPE_COMPLETE_LOCAL_NAME,'N','o','d','e'}

/* Set scan response data */

hci_le_set_scan_resp_data(sizeof(scan_resp_data),scan_resp_data);

/* Set Undirected Connectable Mode */

ret = aci_gap_set_discoverable(ADV_IND, (ADV_INTERVAL_MIN_MS*1000)/625, (ADV_INTERVAL_MAX_MS*1000)/625, PUBLIC_ADDR, NO_WHITE_LIST_USE, 0, NULL, 0, NULL, 0, 0);

/* Set advertising data */

ret = hci_le_set_advertising_data(sizeof(adv_data),adv_data);

On the BlueNRG development platform (STEVAL-IDB002V1), the temperature sensor value is set within the adv_data variable. On the BlueNRG USB dongle (STEVAL-IDB003V1), a random value is set within the adv_data variable (no temperature sensor is available on this platform).

8.2.3 Add service and characteristics

The BLE Remote Control service is added using the following command:

aci_gatt_add_serv(UUID_TYPE_128, service_uuid, PRIMARY_SERVICE, 7, &RCServHandle);

Where service_uuid is the private service 128-bit UUID allocated for the BLE remote service (ed0ef62e-9b0d-11e4-89d3-123b93f75cba).

The command returns the service handle in RCServHandle.

The BLE Remote Control characteristic is added using the following command:

#if ENABLE_SECURITY

```
ret = aci_gatt_add_char(RCServHandle, UUID_TYPE_128, controlPointUuid, 1,
CHAR_PROP_READ|CHAR_PROP_WRITE|CHAR_PROP_WRITE_WITHOUT_RESP|CH
AR_PROP_SIGNED_WRITE,
ATTR_PERMISSION_AUTHEN_READ|ATTR_PERMISSION_AUTHEN_WRITE,
GATT_NOTIFY_ATTRIBUTE_WRITE, 16, 1, &controlPointHandle);

#else

ret = aci_gatt_add_char(RCServHandle, UUID_TYPE_128, controlPointUuid, 1,
CHAR_PROP_READ|CHAR_PROP_WRITE|CHAR_PROP_WRITE_WITHOUT_RESP,
ATTR_PERMISSION_NONE, GATT_NOTIFY_ATTRIBUTE_WRITE, 16, 1,
&controlPointHandle);

#endif
```

Where controlPointUuid is the private characteristic 128-bit UUID allocated for BLE Remote Control characteristic (ed0efb1a-9b0d-11e4-89d3-123b93f75cba).

If security is enabled, the characteristic properties must be set accordingly to enable authentication on controlPointUuid characteristic read and write.

8.2.4 Connection with a BLE Central device

When connected to a BLE Central device (another BlueNRG device or any SMART READY device), the controlPointUuid characteristic is used to control the BLE Remote Control platform LED. Each time a write operation is done on controlPointUuid, the EVT_BLUE_GATT_ATTRIBUTE_MODIFIED event is raised on the HCI_Event_CB () callback and the selected LED/LEDs are turned on or off.

9 List of acronyms

Table 16. List of acronyms used in this document

Term	Meaning
BLE	Bluetooth low energy
IFR	Information register
USB	Universal serial bus

10 Available board schematics

Figure 24. STEVAL-IDB002V1 BlueNRG daughterboard

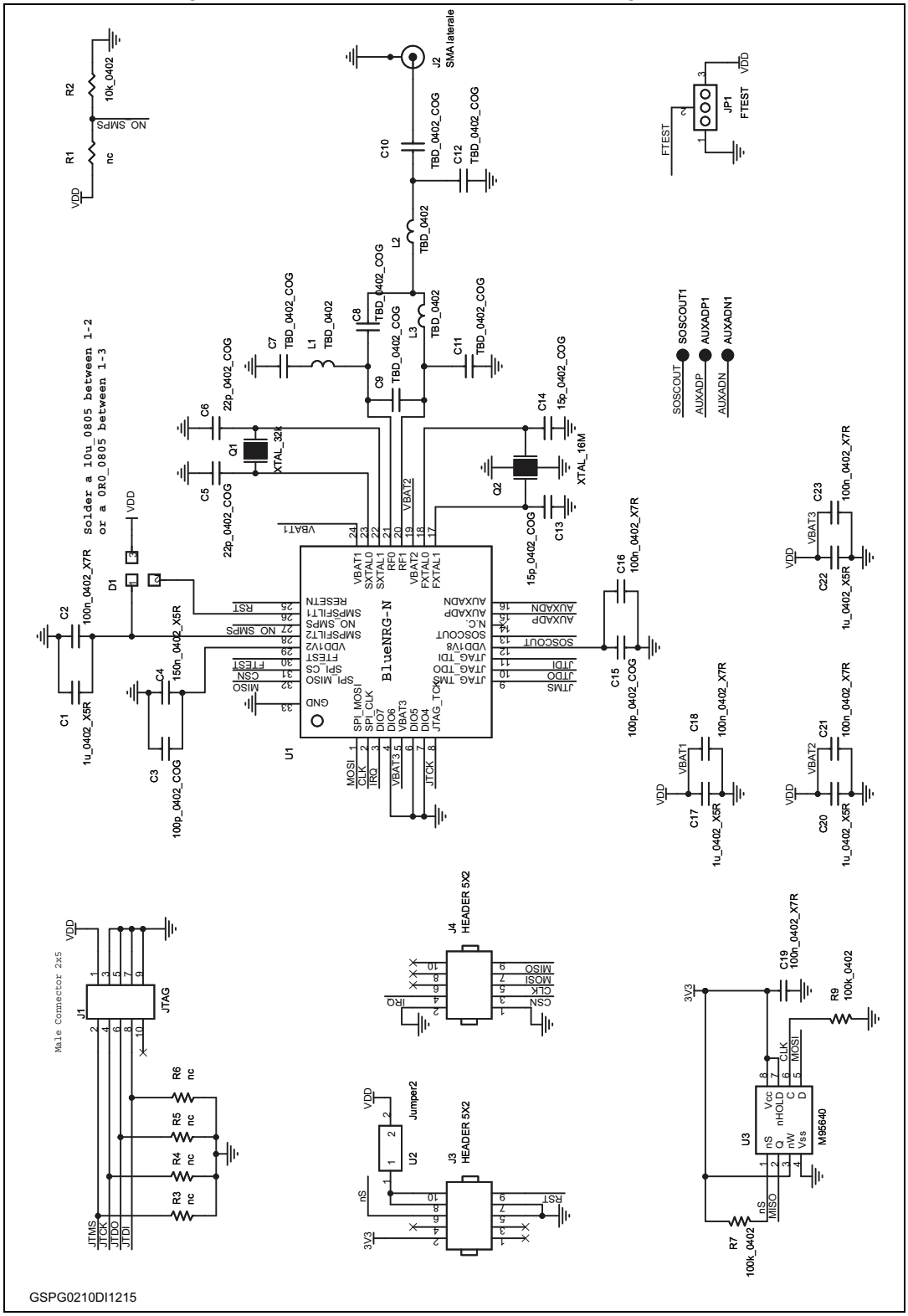


Figure 25. STEVAL-IDB002V1 temperature sensor

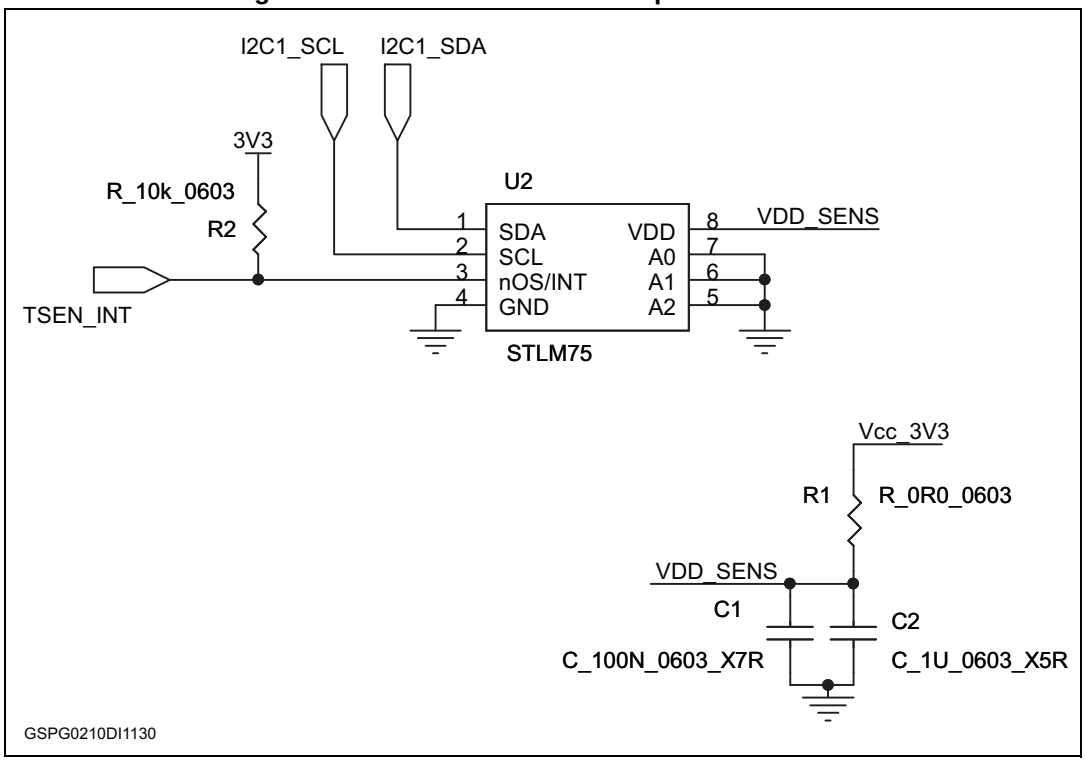


Figure 26. STEVAL-IDB002V1 accelerometer

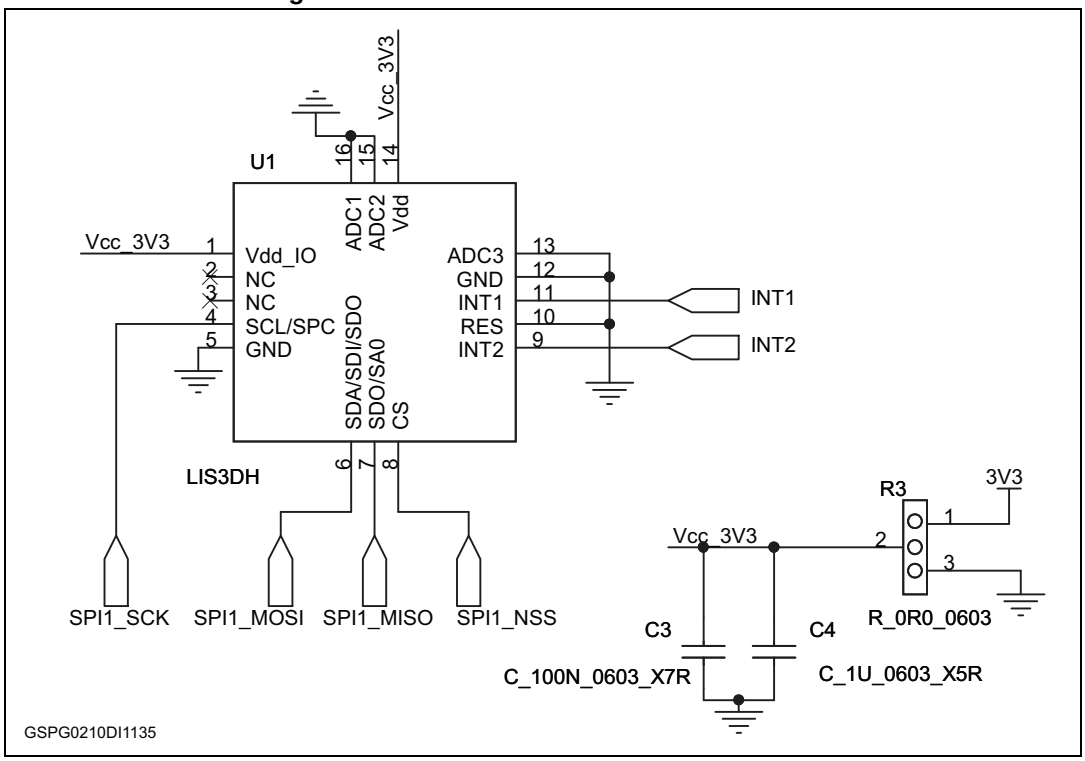


Figure 27. STEVAL-IDB002V1 MCU

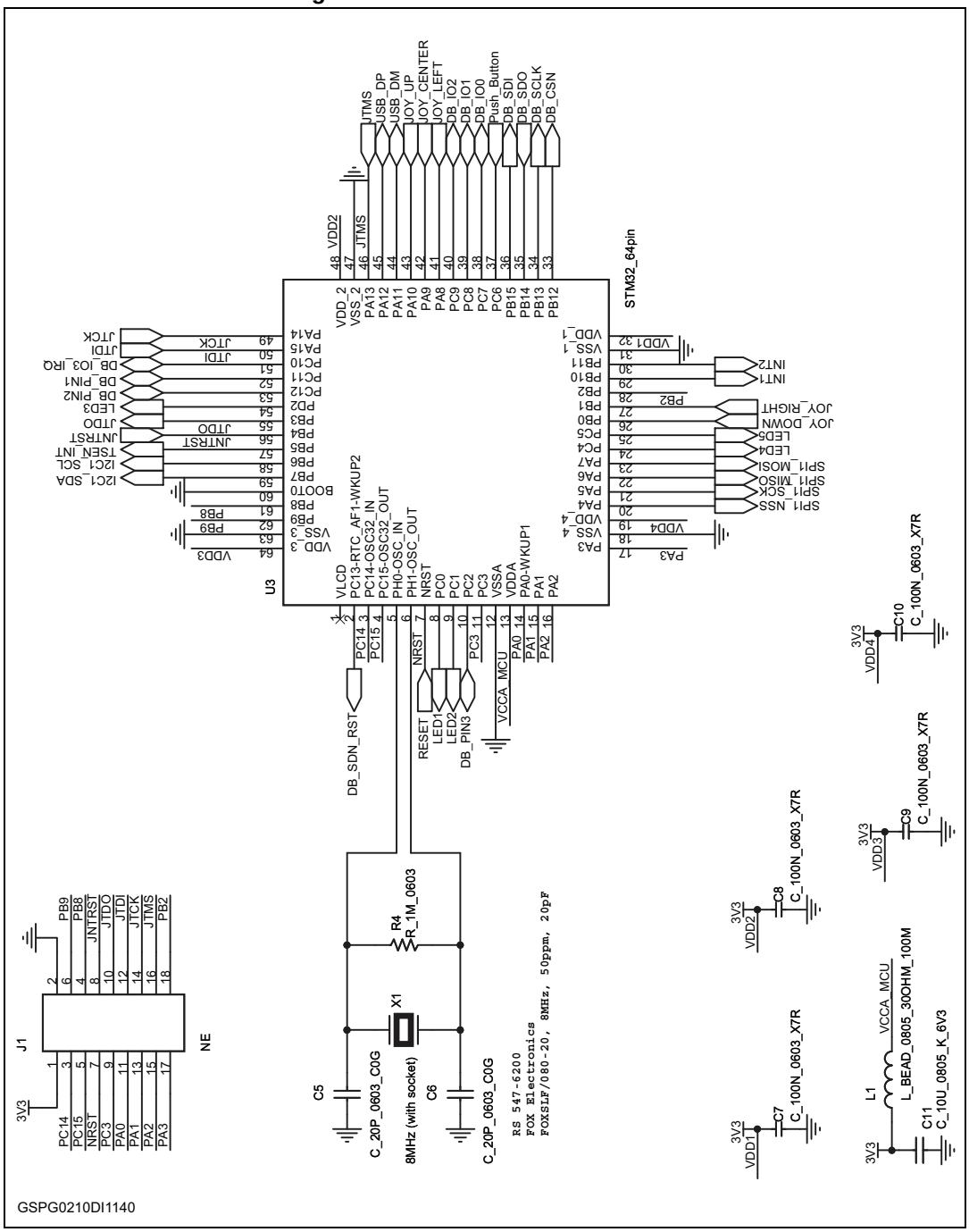


Figure 28. STEVAL-IDB002V1 JTAG/SWD

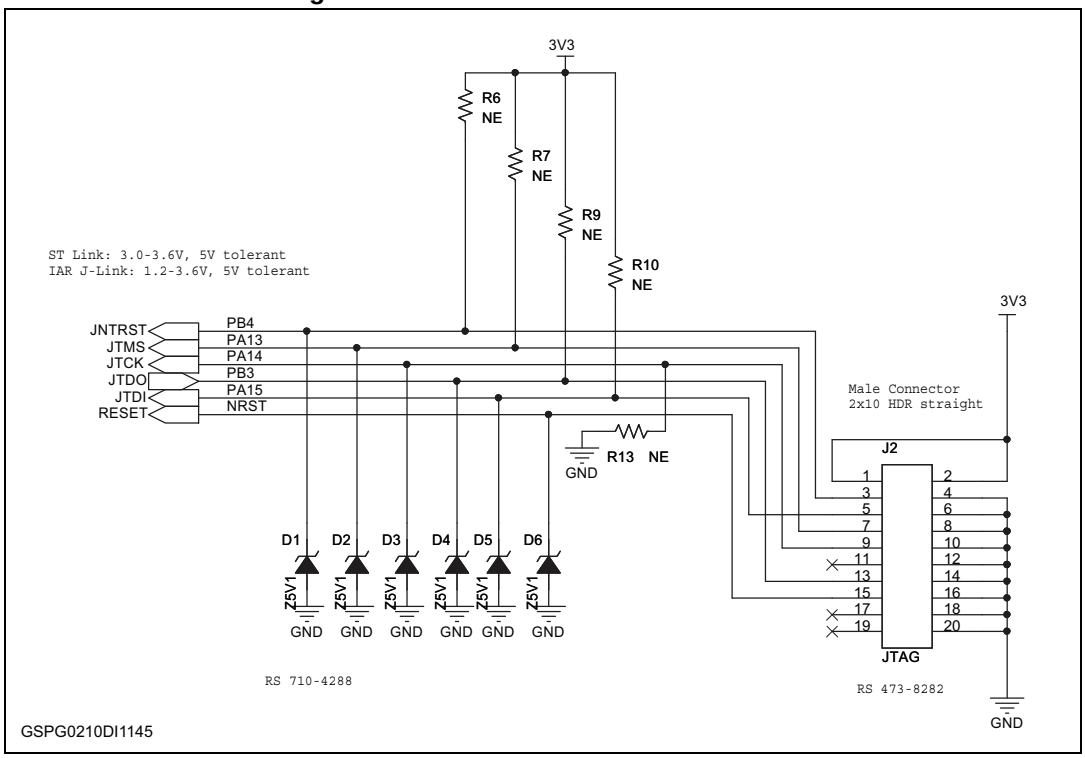


Figure 29. STEVAL-IDB002V1 USB

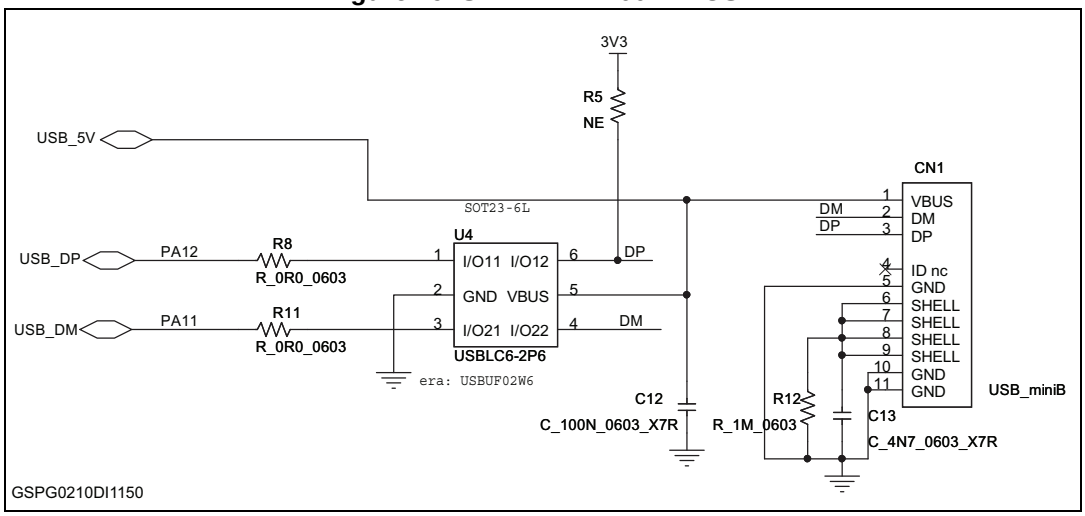


Figure 30. STEVAL-IDB002V1 LED

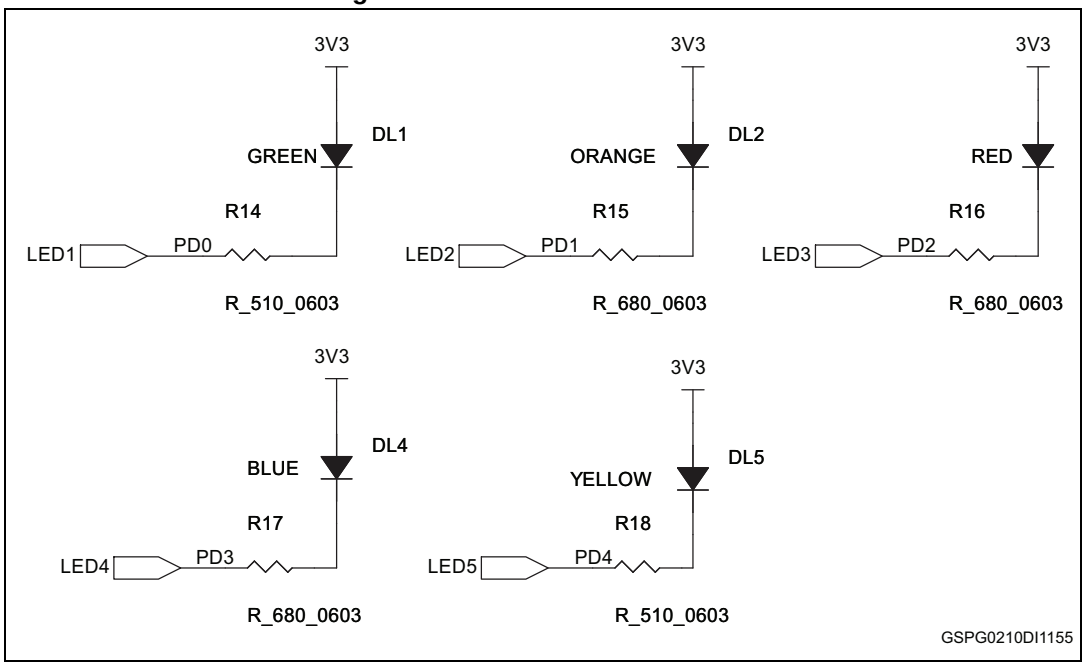


Figure 31. STEVAL-IDB002V1 power supply

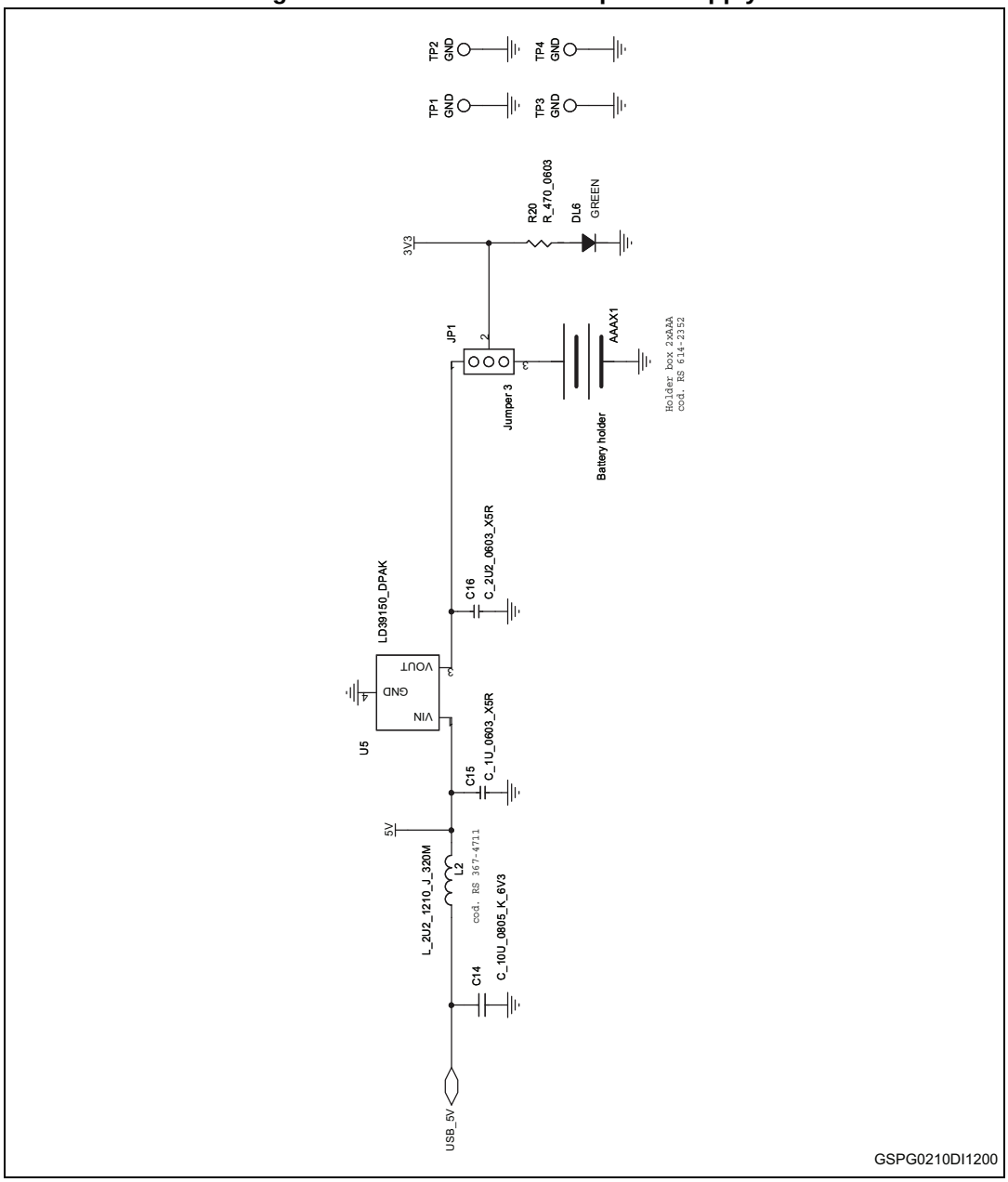


Figure 32. STEVAL-IDB002V1 button and joystick

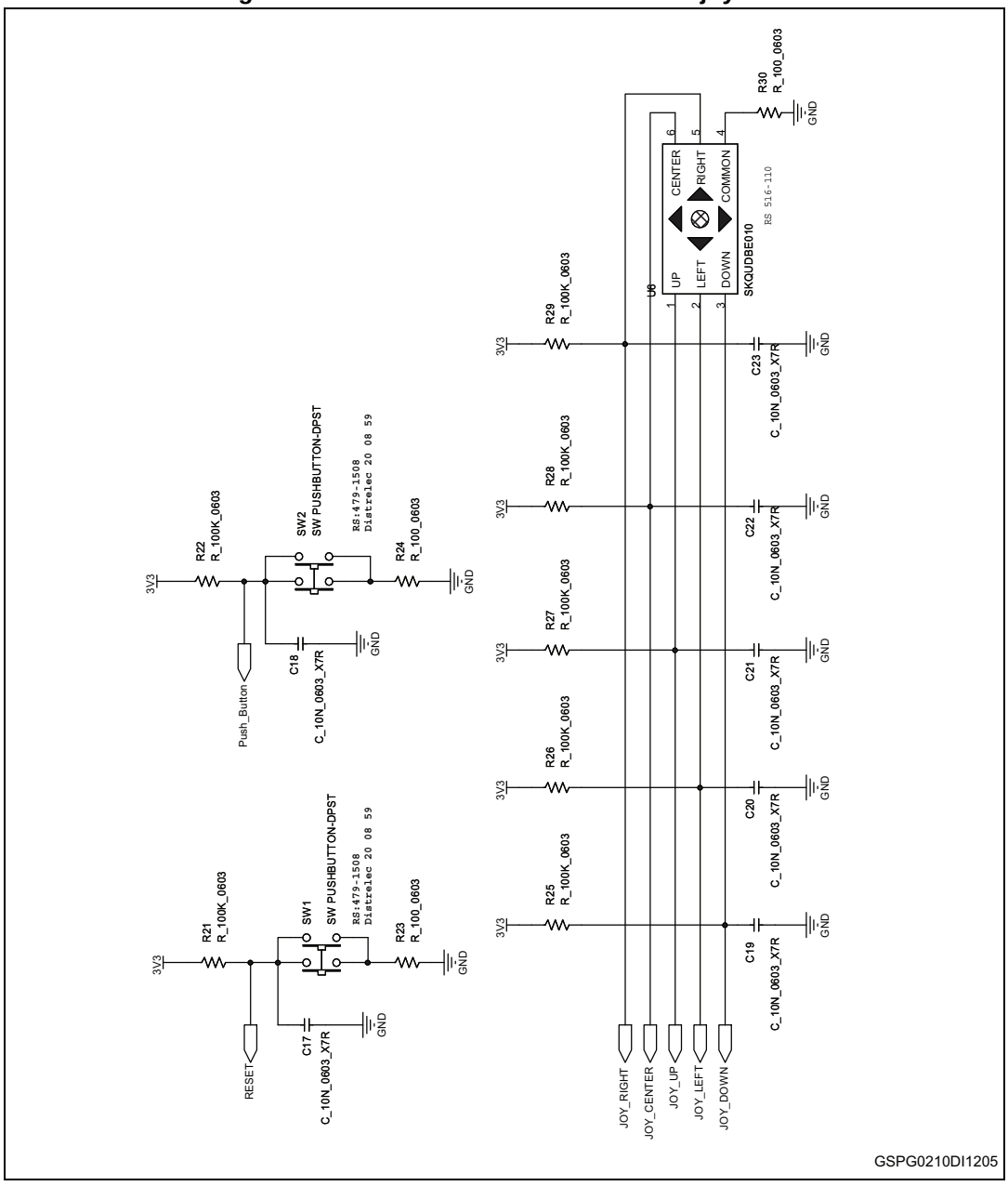


Figure 33. STEVAL-IDB002V1 daughterboard connectors

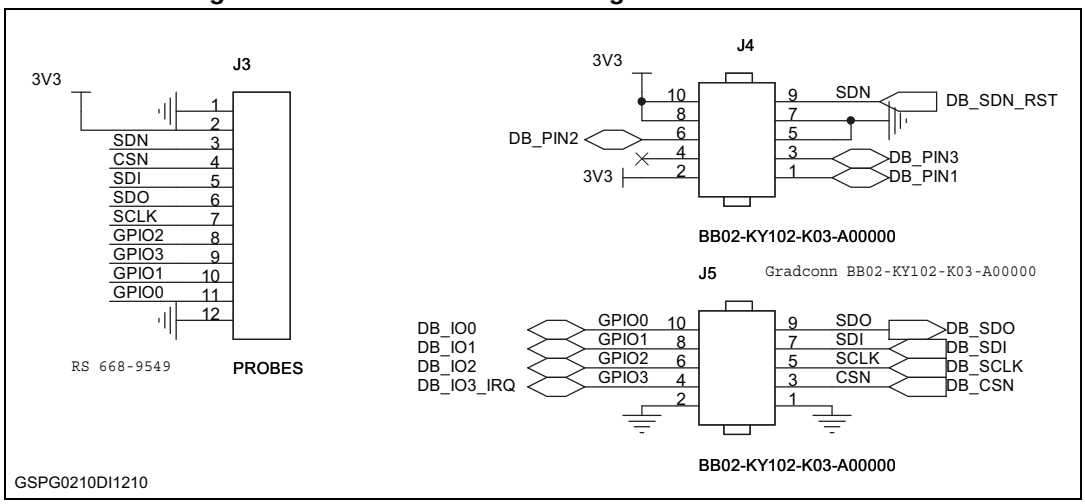
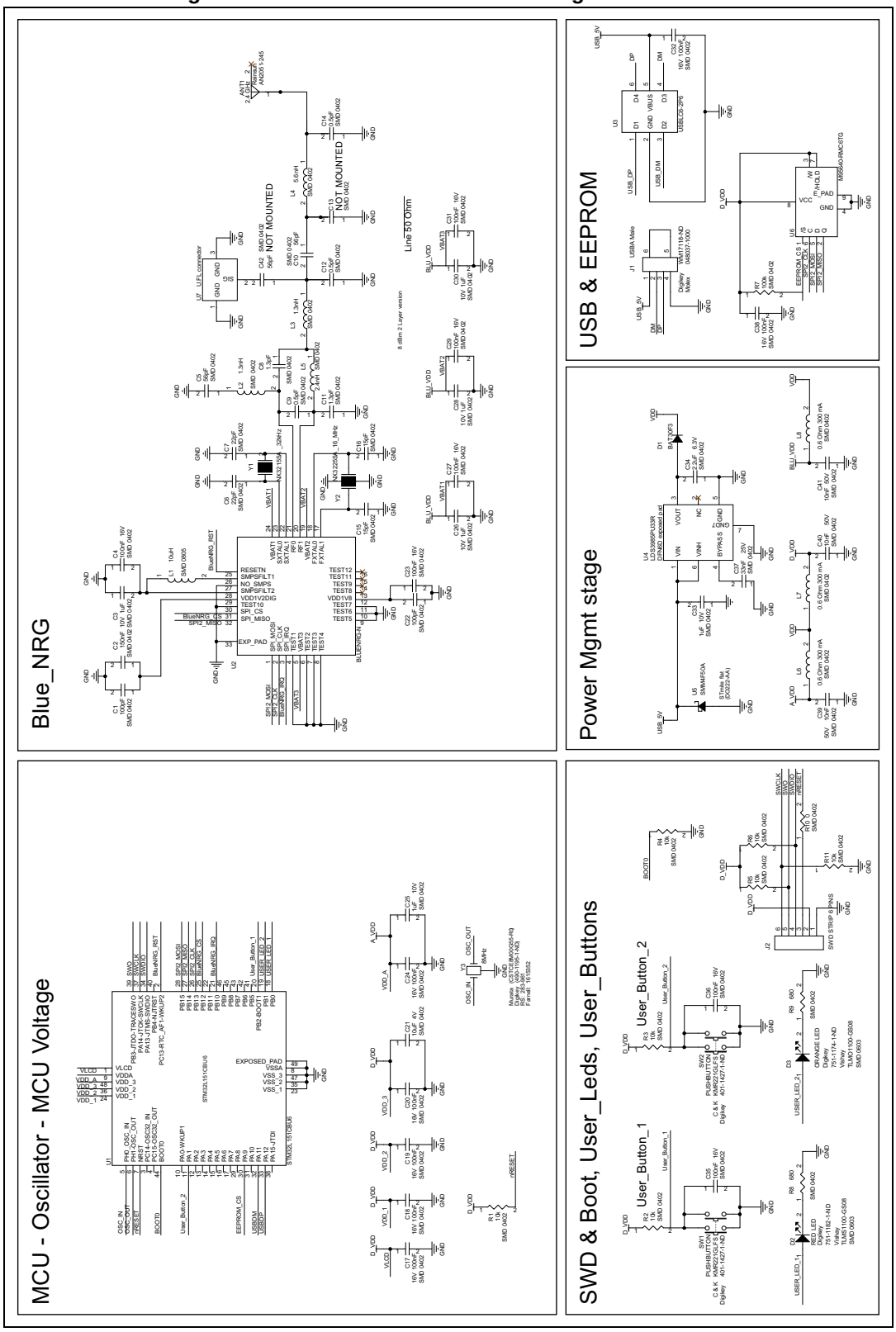


Figure 34. STEVAL-IDB003V1 USB dongle schematics



11 Revision history

Table 17. Document revision history

Date	Revision	Changes
28-Nov-2013	1	Initial release
24-Apr-2014	2	– Added reference to the STEVAL-IDB003V1 BlueNRG USB Dongle – Added: Section 6 – Added: Section 9 – Added: Section 10 – Minor text edits throughout the document
10-Dec-2014	3	– Added: Section 3.2.3 – Added Section 3.2.5 – Added Section 7 – Renamed APIs with prefix BLUEHCI_ in Section 5.3.1 to 5.3.5 and 6.2.1
11-Mar-2015	4	– Updated: Figure 7, 11, 12, 13 and 14, and caption of Figure 1 – Updated: Table 6, Table 7, Table 8 and Table 9 – Updated: Section 3.2.2 and Section 3.2.3 – Added: Table 10, Table 11 and Table 14 – Added: Section 5.4 and Section 8 – Added: Figure 26, 27, 28, 29, 30, 31, 32, 33 and 34
09-Dec-2015	5	– Updated: *Figure 7*, *Figure 11*, *Figure 12*, *Figure 13*, *Figure 14*, *Figure 15*, *Figure 16*, *Figure 17* and *Figure 18* – Updated: *Section 3.2.4: GUI Scripts window* – Updated: *Table 10* – Added: *Section 3.2.6: GUI RF Test window*