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)

Contents

1 Getting started

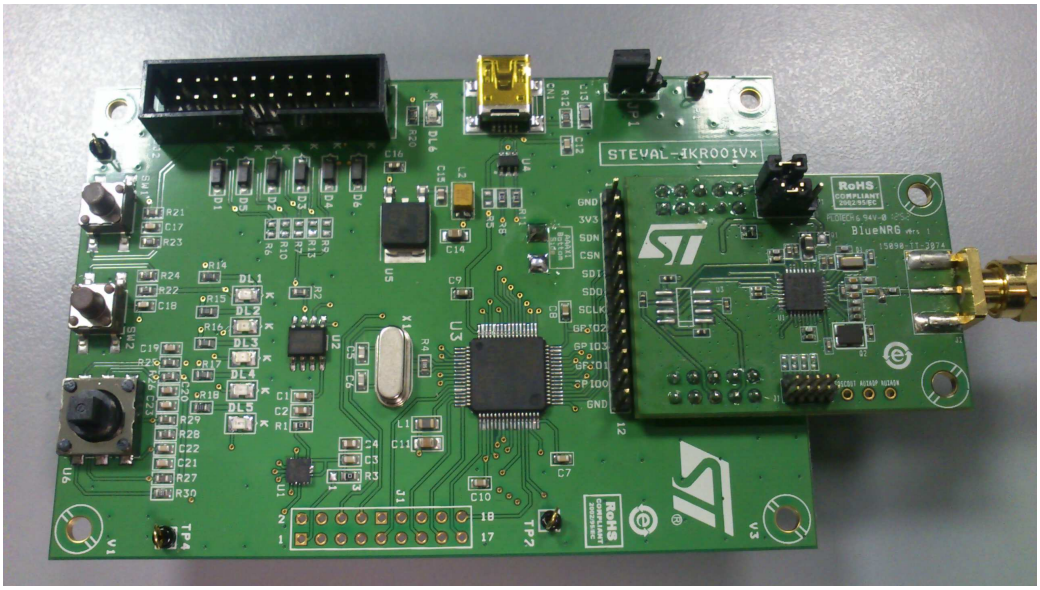
This section describes all the software and hardware requirements for installing the BlueNRG DK SW package (STSW-BLUENRG-DK) and using the related HW, SW resources.

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

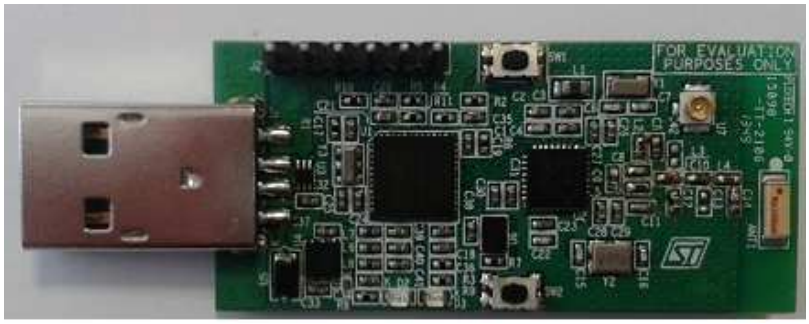


1.2 STEVAL-IDB003V1 kit

This kit is composed of the following items:

- 1 USB dongle

Figure 2. STEVAL-IDB003V1 BlueNRG USB dongle



1.3 System requirements

The BlueNRG DK SW package (STSW-BLUENRG-DK) 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.

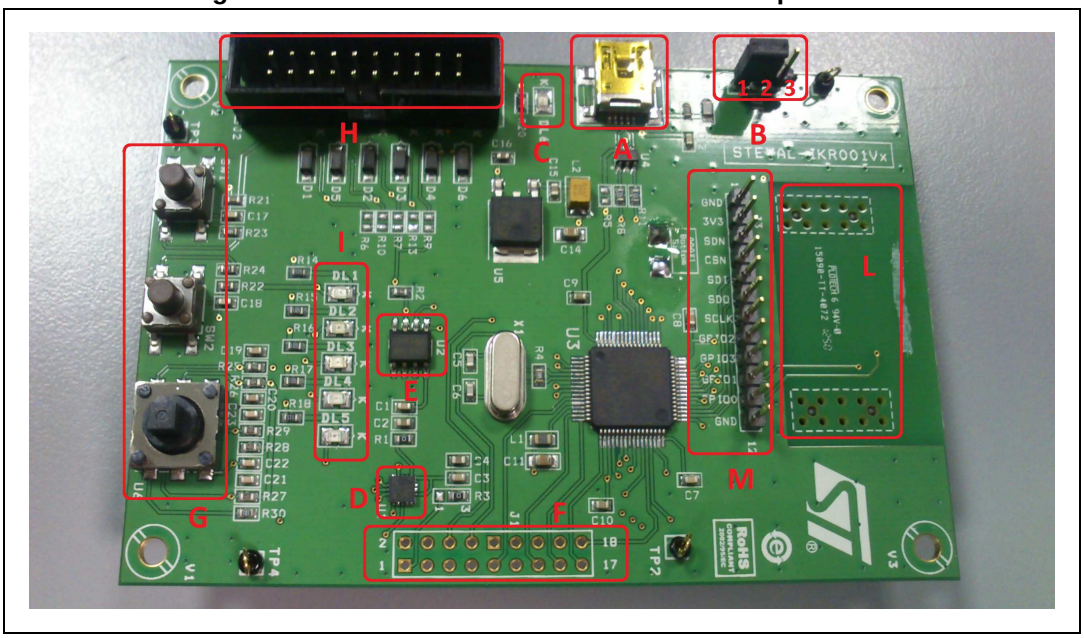
2.1 STEVAL-IDB002V1 motherboard

The motherboard included in the development kit allows testing of the functionality of the BlueNRG processor. The 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
VSSA	12								

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
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					
PA10	43			JOY_UP					
PA11	44						USB_DM		
PA12	45						USB_DP		
PA13	46							JTMS	16

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
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.

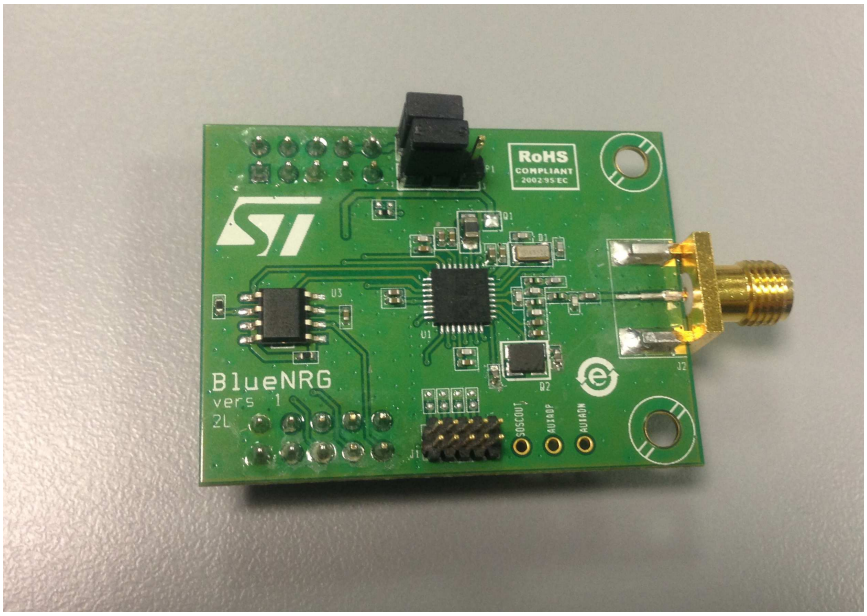
Some of the lines are connected also to a row of test points (M).

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 12*).

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 12*):

- 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 4* 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 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	

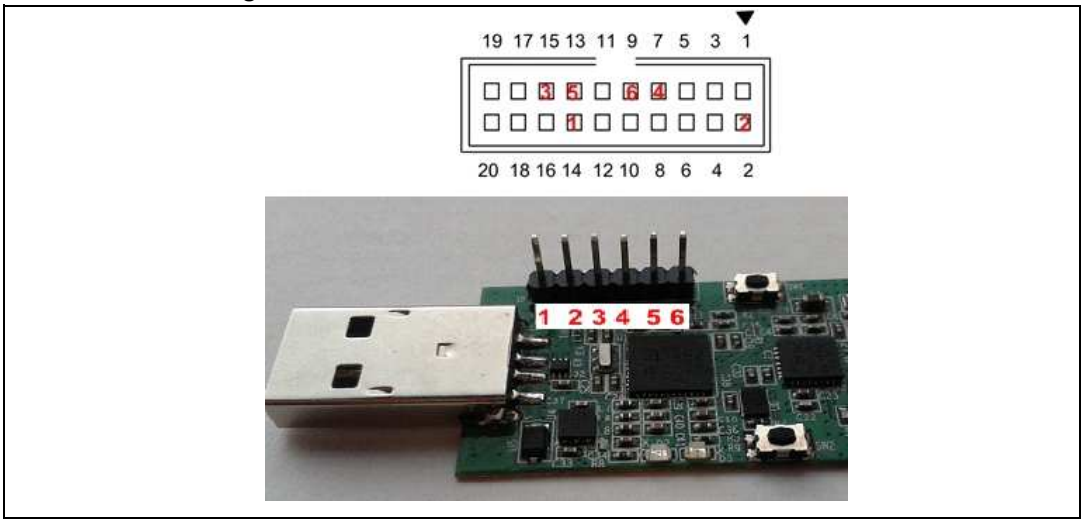
Table 4. MCU pin description versus board function (continued)

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

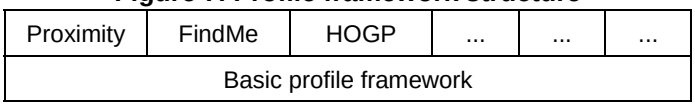
The STM32L on the STEVAL-IDB003V1 motherboard is preprogrammed with the BlueNRG_VCOM_x_x.hex application when the kits components are assembled.

3 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 7. Profile framework structure



3.1 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.

3.2 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.

4 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 8* 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[(b)]. 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

b. 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 8. 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

4.1 Supported platforms

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

4.2 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. The correct pre-compiled firmware can be found inside firmware folder (BlueNRG_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 9. BlueNRG app

4.3 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.

4.3.1 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.

4.3.2 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).

4.3.3 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.

4.3.4 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.

4.3.5 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.

4.4 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.

4.4.1 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.

4.4.2 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).

4.4.3 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.

4.4.4 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.

4.4.5 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).

4.4.6 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).

5 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 6. 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.

5.1 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).

5.2 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.

5.2.1 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.

5.2.2 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.

5.2.3 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);

5.2.4 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.

5.2.5 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 10. BLE chat client example

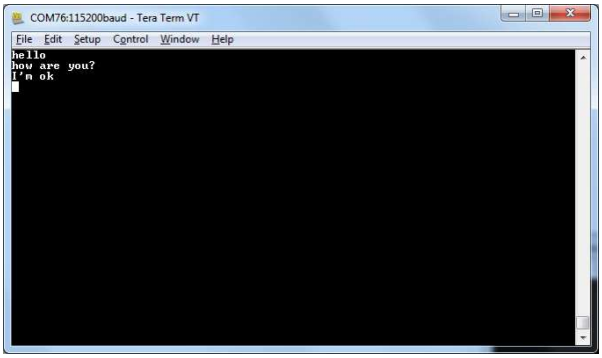
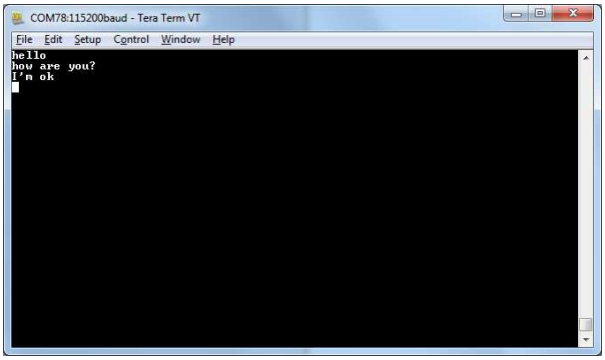


Figure 11. BLE chat server example



6 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.

6.1 Supported platforms

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

6.2 BLE Beacon application setup

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

6.2.1 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);

6.2.2 Define advertising data

The BLE Beacon application advertises the following manufacturing data:

Table 7. 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

6.2.3 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);
```

7 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 8. 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.

7.1 Supported platforms

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

7.2 BLE remote control application setup

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

7.2.1 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)

7.2.2 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).

7.2.3 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.

7.2.4 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.

8 List of acronyms

Table 9. List of acronyms used in this document

Term	Meaning
BLE	Bluetooth low energy
USB	Universal serial bus

9 References

Table 10. References table

Name	Title
STSW-BLUENRG-DK	BlueNRG SW package for BlueNRG, BlueNRG-MS kits
Bluetooth specification V4.0	Specification of the Bluetooth system v4.0
UM1755	BlueNRG Bluetooth LE stack APIs and events user manual

10 Available board schematics

Figure 12. STEVAL-IDB002V1 BlueNRG daughterboard

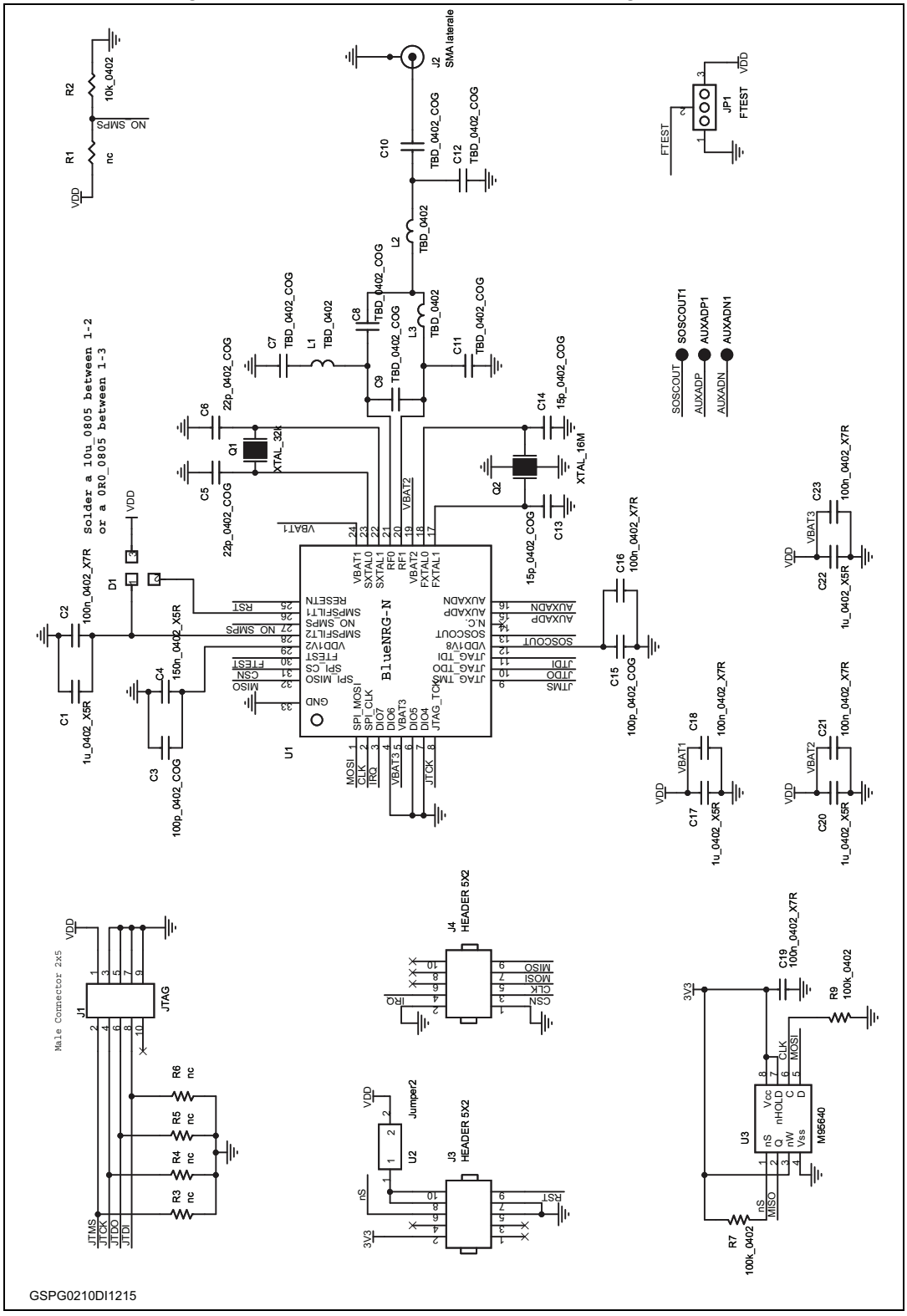


Figure 13. STEVAL-IDB002V1 temperature sensor

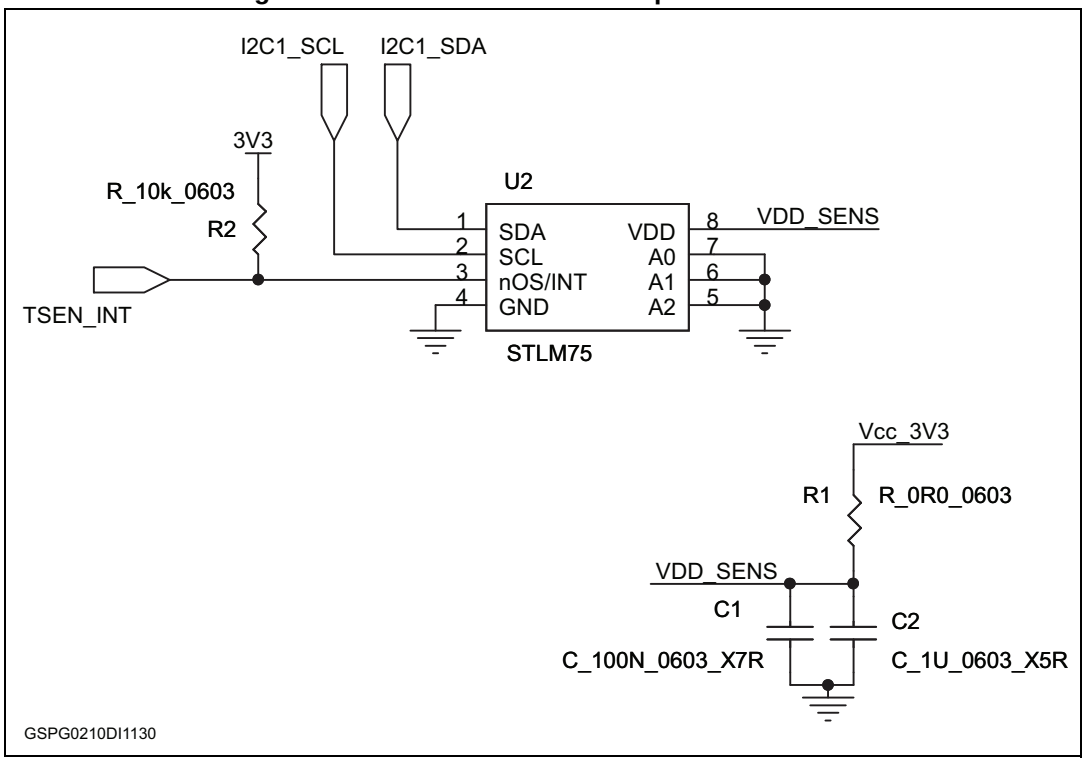


Figure 14. STEVAL-IDB002V1 accelerometer

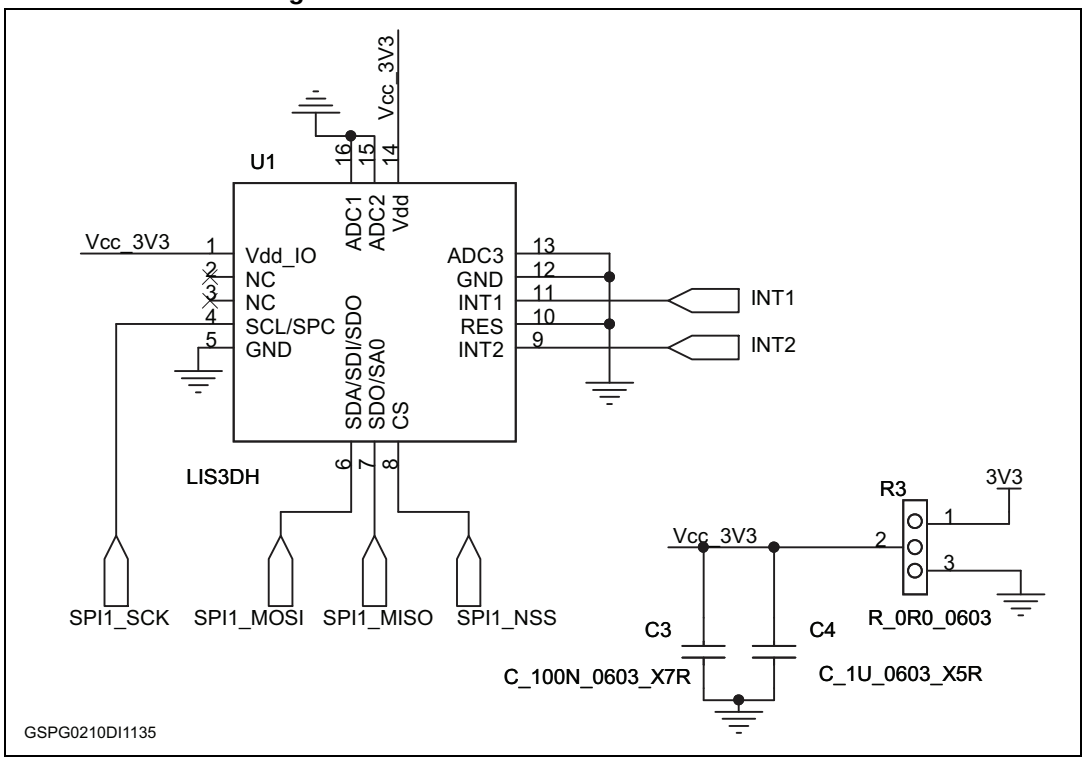


Figure 15. STEVAL-IDB002V1 MCU

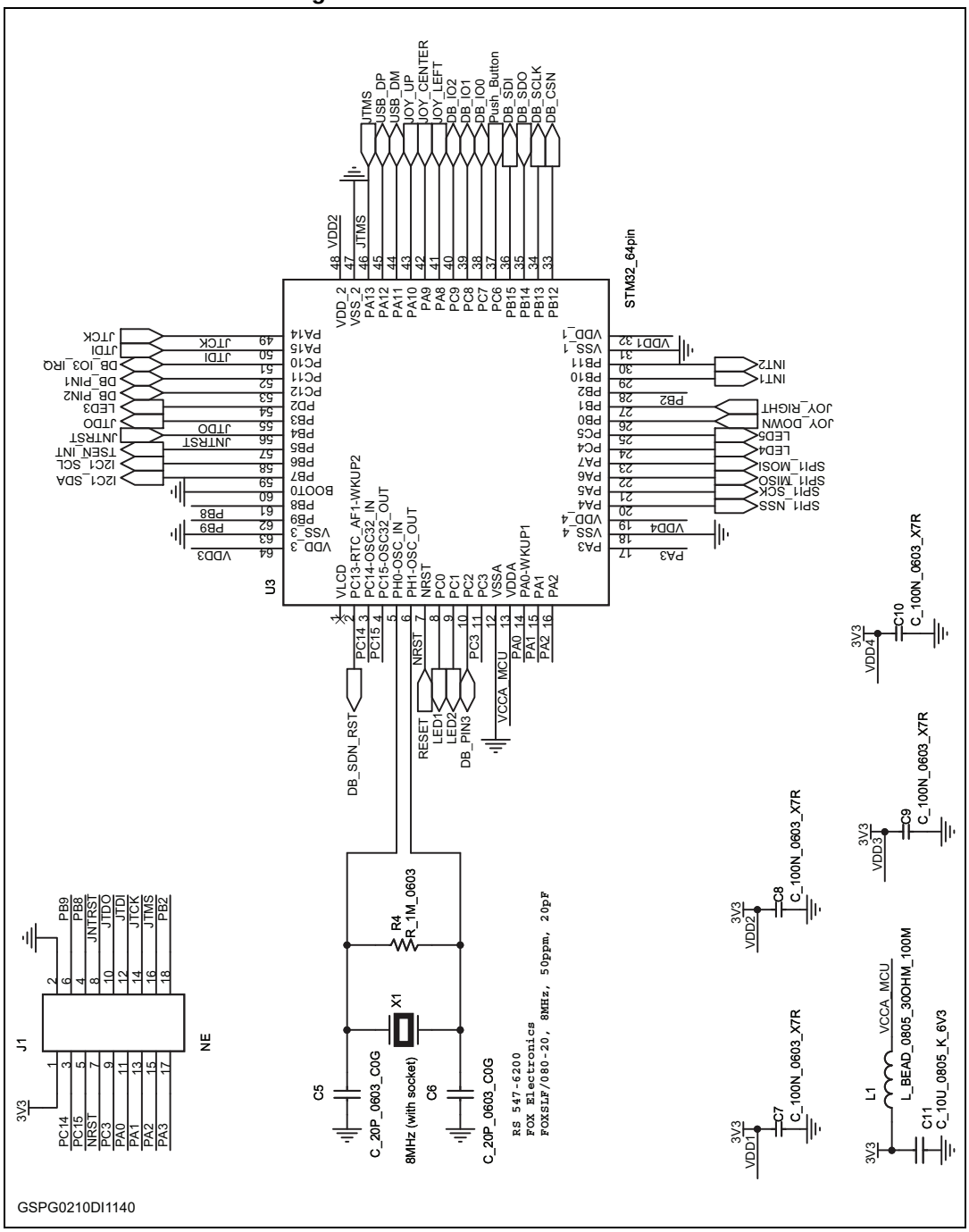


Figure 16. STEVAL-IDB002V1 JTAG/SWD

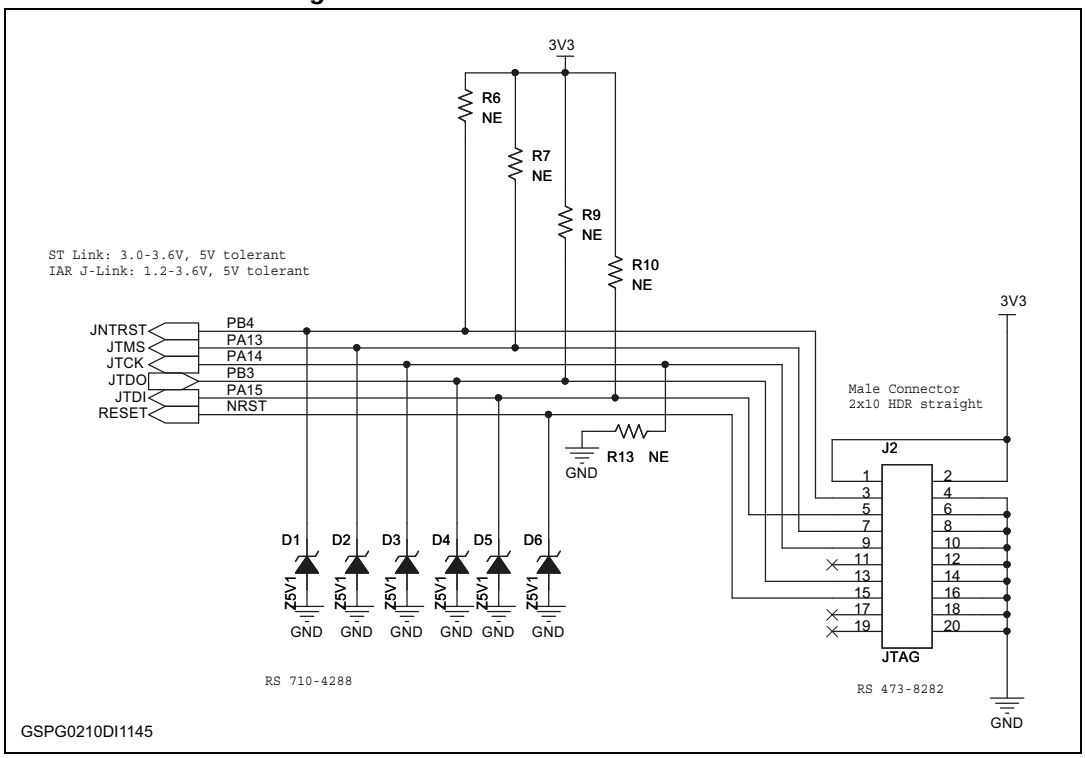


Figure 17. STEVAL-IDB002V1 USB

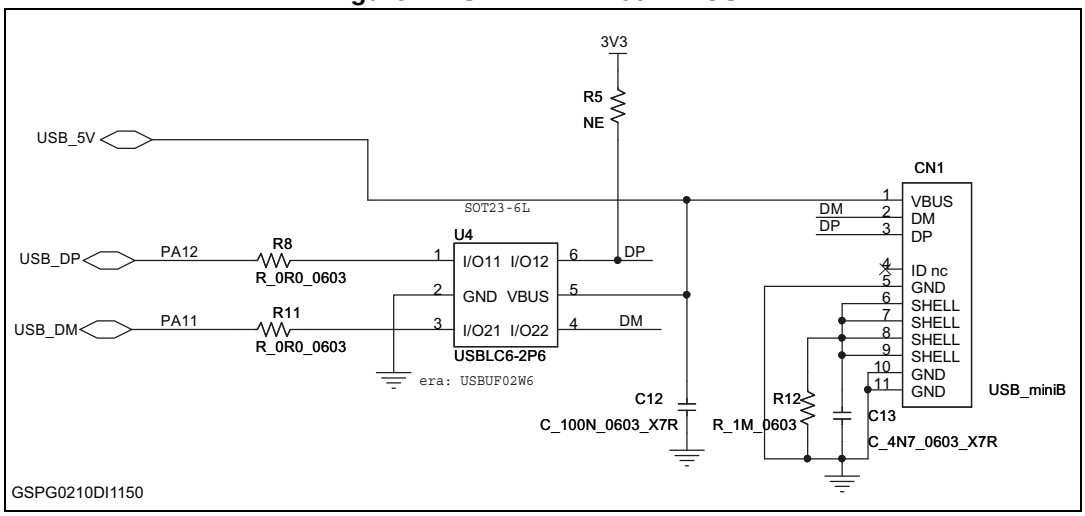


Figure 18. STEVAL-IDB002V1 LED

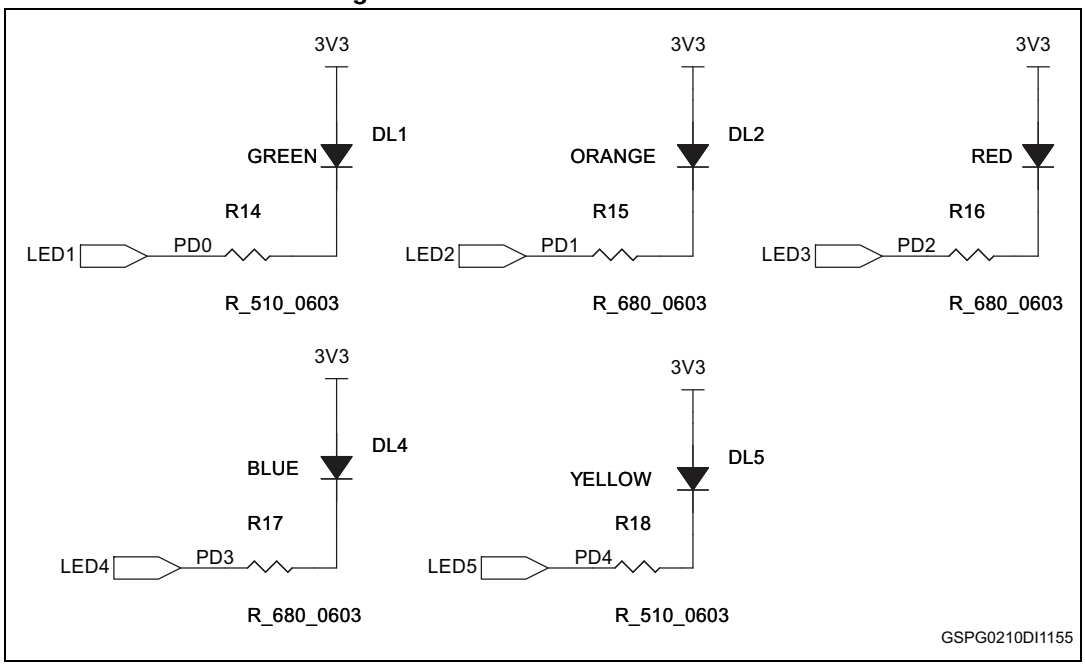


Figure 19. STEVAL-IDB002V1 power supply

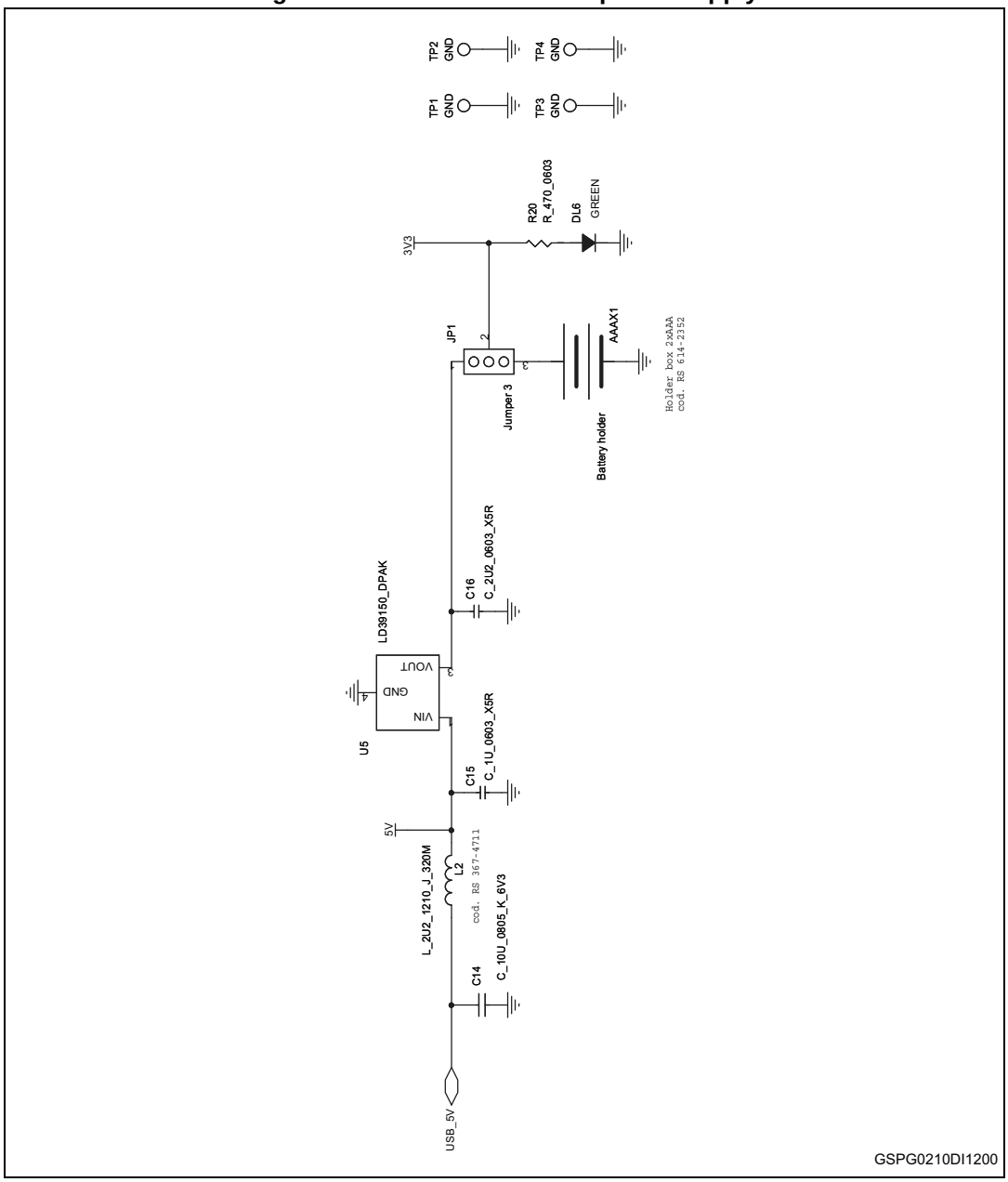


Figure 20. STEVAL-IDB002V1 button and joystick

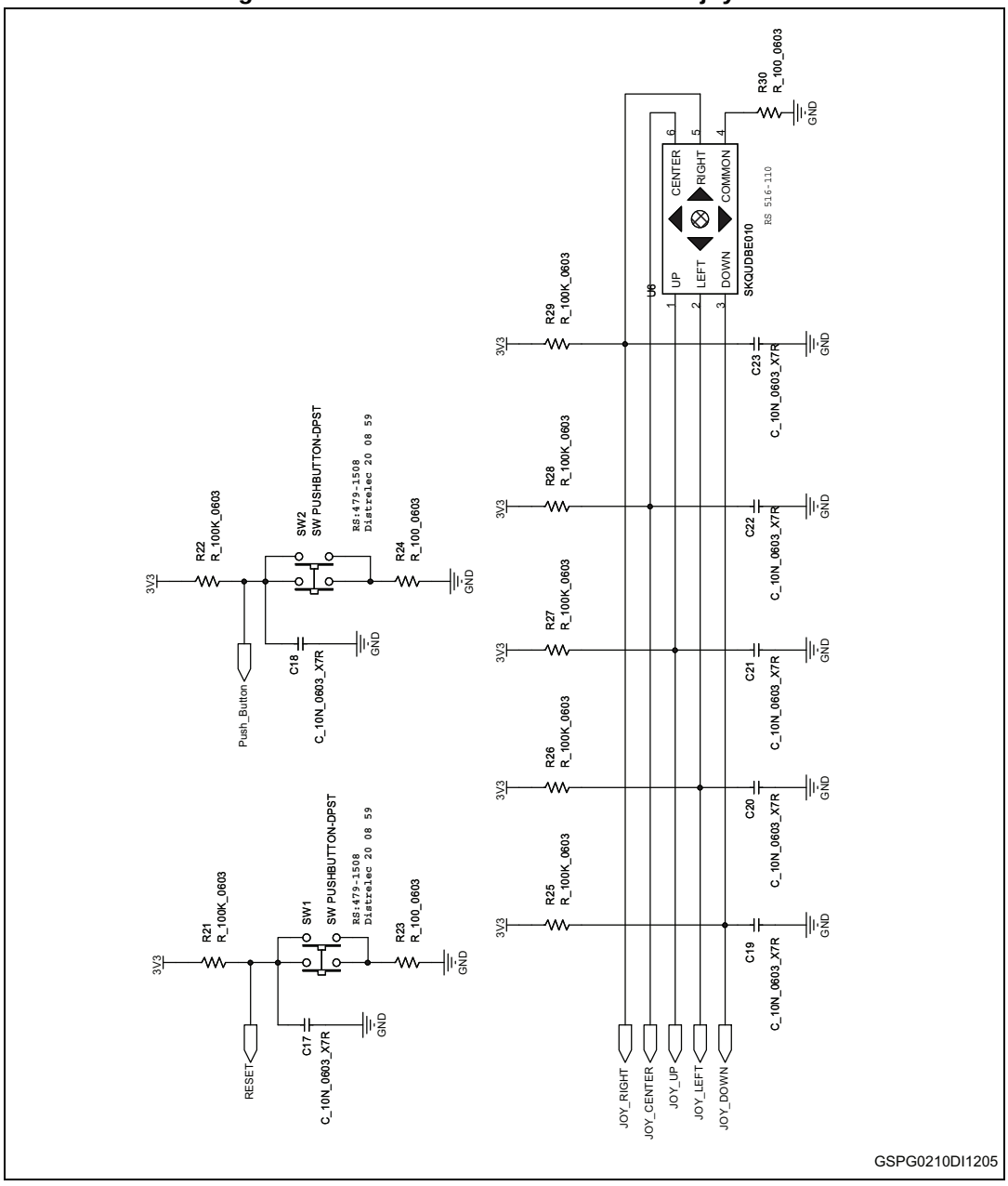


Figure 21. STEVAL-IDB002V1 daughterboard connectors

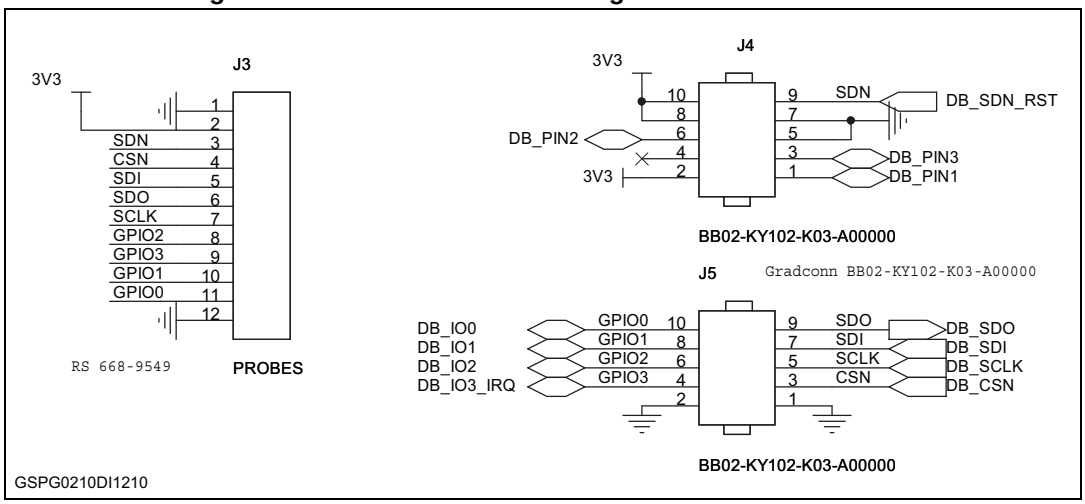
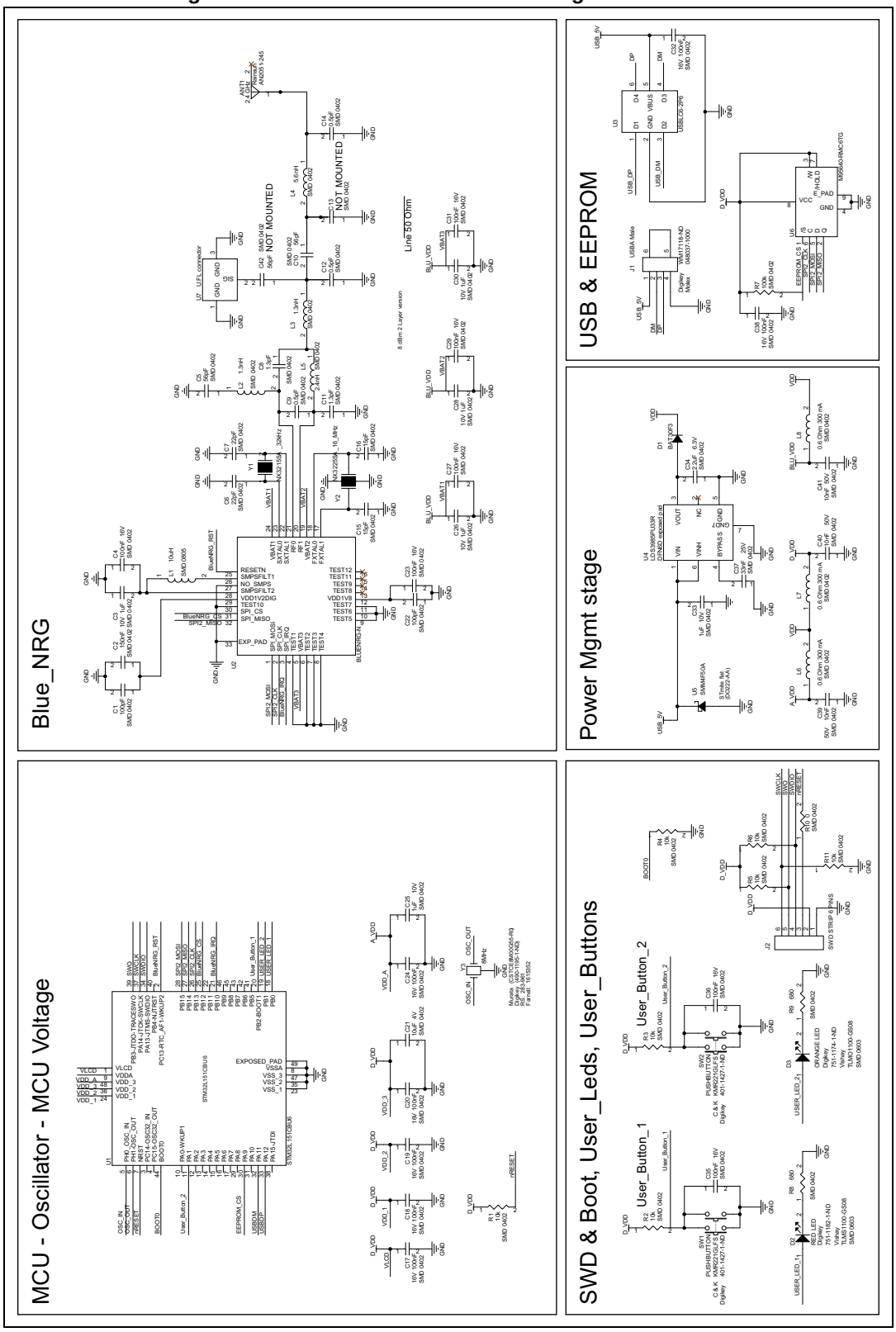


Figure 22. STEVAL-IDB003V1 USB dongle schematics



11 Revision history

Table 11. 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
26-May-2016	6	– Added: *Section 9: References* – Removed: Section 3