Introduction

The BlueNRG-LP device is a low power Bluetooth® system-on-chip, compliant with the Bluetooth® specification and supports master, slave and simultaneous master-and-slave roles. The device also supports the Bluetooth Low Energy data length extension feature, 2 Mbps, Long Range and extended advertising features.

The following BlueNRG-LP kit is available:

• STEVAL-IDB011V1 QFN48 package development platform

The STEVAL-IDB011V1 development platform embeds a CMSIS-DAP programming/debugging interface and features hardware resources for a wide range of application scenarios: sensor data (accelerometer, pressure and temperature sensor), human interface (buttons and LEDs), digital MEMS microphone and serial communication through USB virtual COM. Three power options are available (USB only, battery only and external power supply plus USB) for high application development and testing flexibility.

Figure 1. STEVAL-IDB011V1 development platform based on BlueNRG-LP
1 Getting started

1.1 Kit contents

The STEVAL-IDB011V1 kit includes:

- a BlueNRG-LP QFN48 package development platform
- a 2.4 GHz Bluetooth antenna
- a USB cable

1.2 System requirements

The BlueNRG-LP Navigator and Radio Init Parameters Wizard PC applications require:

- PC with Intel® or AMD® processor running:
  - Windows 7 - the driver installation for the Virtual COM port requires the driver files available at https://os.mbed.com/docs/mbed-os/v6.0/program-setup/windows-serial-driver.html
  - Windows 10 - no driver installation needed
- At least 128 MB of RAM
- USB ports
- Adobe Acrobat Reader 6.0 or later

1.3 BlueNRG-LP development kit setup

The STSW-BNRGLP-DK DK software package is available for BlueNRG-LP Bluetooth LE stack v3.x family.

After downloading the selected software package, extract en.stsw-bnrglp-dk.zip contents to a temporary directory, launch BlueNRG-LP-DK-x.x.x-Setup.exe and follow the on-screen instructions.

Note: EWARM Compiler 8.40.1 or later is required for building the BlueNRG-LP_DK_x.x.x demonstration applications. The Utility/EWARM_BlueNRG-LP_Flasher_2.0.0.exe must be applied to the local IAR EWARM installation path to add the BlueNRG-LP to the list of supported devices.

Note: Keil MDK-ARM toolchain is also supported. The Utility/Keil.STBlueNRG-LP_DFP.2.0.0.pack must be installed on the local Keil MDK-ARM tool to add the BlueNRG-LP device to the list of supported devices.
2 Hardware description

2.1 STEVAL-IDB011V1 board overview

The STEVAL-IDB011V1 development kit lets you experiment with BlueNRG-LP system on chip functions. It features:

- Bluetooth® low energy board based on the BlueNRG-LP Bluetooth low energy system-on-chip (QFN48 package)
- Associated development kit SW package (STSW-BNRGLP-DK) including firmware and documentation
- Bluetooth® low energy compliant, supports master, slave and simultaneous master-and-slave roles
- PCB antenna and option for adding a U.FL connector for antenna or measuring equipment
- 3 user LEDs
- 2 user buttons
- 3D digital accelerometer and 3D digital gyroscope
- MEMS pressure sensor with embedded temperature sensor
- MEMS audio sensor omnidirectional digital microphone
- Battery holder
- CMSIS-DAP debugger/programmer via micro USB connector
- USB to serial bridge to create an I/O channel with the BlueNRG-LP device
- Jumper to measure BlueNRG-LP current
- RoHS compliant

Figure 2. STEVAL-IDB011V1 board components
### Table 1. STEVAL-IDB011V1 component descriptions

<table>
<thead>
<tr>
<th>Region</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>BlueNRG-355MC (BlueNRG-LP, QFN48 package)</td>
</tr>
<tr>
<td>C</td>
<td>Micro USB connector for power supply and I/O, and CMSIS-DAP debugger/programmer</td>
</tr>
<tr>
<td>L</td>
<td>Reset button</td>
</tr>
<tr>
<td>M</td>
<td>Two user buttons</td>
</tr>
<tr>
<td>I</td>
<td>LPS22HH MEMS pressure sensor with embedded temperature</td>
</tr>
<tr>
<td>H</td>
<td>LSM6DSOX 3D digital accelerometer and 3D digital gyroscope</td>
</tr>
<tr>
<td>G</td>
<td>Power LED (DL1)</td>
</tr>
<tr>
<td>F</td>
<td>User LEDs (DL2, DL3)</td>
</tr>
<tr>
<td>Q</td>
<td>3-color LED (U5)</td>
</tr>
<tr>
<td>O</td>
<td>Programmer/debugger and communication activity (LED DL4)</td>
</tr>
<tr>
<td>Back of the PCB</td>
<td>Battery holder for two AAA batteries and footprint for soldering a battery holder for a CR2032 coin cell</td>
</tr>
<tr>
<td>J, P</td>
<td>Two rows of Arduino connectors.</td>
</tr>
<tr>
<td>N</td>
<td>USB CMSIS-DAP program/debug channel and serial bridge for I/O channel to PC communication (STM32F103xx 64-pin microcontroller)</td>
</tr>
<tr>
<td>R</td>
<td>Power options (USB, battery)</td>
</tr>
<tr>
<td>D</td>
<td>32 MHz high speed crystal</td>
</tr>
<tr>
<td>E</td>
<td>32 kHz low speed crystal</td>
</tr>
<tr>
<td>S</td>
<td>Board jumpers</td>
</tr>
<tr>
<td>T</td>
<td>MP34DT05-A digital microphone</td>
</tr>
<tr>
<td>U</td>
<td>8 MHz crystal</td>
</tr>
<tr>
<td>V</td>
<td>SMA connector</td>
</tr>
</tbody>
</table>

1. **STM32 is not intended to be programmed by users.**

### 2.2 BlueNRG-LP SoC connections

The STEVAL-IDB011V1 on-board BlueNRG-LP in QFN48 package is a very low power Bluetooth low energy (BLE) single-mode system-on-chip (Figure 2 – region A) with 256 KB of Flash, 64 KB of RAM, a 32-bit core ARM® Cortex®-M0+ processor and several peripherals (ADC, 32 GPIOs, I²C, SPI, I²S, Timers, UART, WDG and RTC).

The microcontroller is connected to various components such as buttons, LEDs and sensors.

### Table 2. BlueNRG-LP pin description with board functions

<table>
<thead>
<tr>
<th>Pin name</th>
<th>Pin no.</th>
<th>Board function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QFN48</td>
<td>LEDs</td>
</tr>
<tr>
<td>PA0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>PA1</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>PA2</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>PA3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>PA4</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>PA5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>PA6</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>PA7</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Pin name</td>
<td>Pin no.</td>
<td>Board function</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
<td>----------------</td>
</tr>
<tr>
<td>PA8</td>
<td>12</td>
<td>UART_RXD</td>
</tr>
<tr>
<td>PA9</td>
<td>11</td>
<td>UART_TXD</td>
</tr>
<tr>
<td>PA10</td>
<td>10</td>
<td>BOOT, PUSH1</td>
</tr>
<tr>
<td>PA11</td>
<td>9</td>
<td>SPI_CS</td>
</tr>
<tr>
<td>PA12</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>PA13</td>
<td>7</td>
<td>SPI_CLK</td>
</tr>
<tr>
<td>PA14</td>
<td>6</td>
<td>SPI_MISO</td>
</tr>
<tr>
<td>PA15</td>
<td>5</td>
<td>SPI_MOSI</td>
</tr>
<tr>
<td>PB0</td>
<td>4</td>
<td>INT1</td>
</tr>
<tr>
<td>PB1</td>
<td>3</td>
<td>CLK</td>
</tr>
<tr>
<td>PB2</td>
<td>2</td>
<td>DOUT</td>
</tr>
<tr>
<td>PB3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>PB4</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>PB5</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>PB6</td>
<td>46</td>
<td>PUSH2</td>
</tr>
<tr>
<td>PB7</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>PB8</td>
<td>44</td>
<td>DL2</td>
</tr>
<tr>
<td>PB9</td>
<td>43</td>
<td>DL3</td>
</tr>
<tr>
<td>PB10</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>PB11</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>PB12</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>PB13</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>PB14</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>PB15</td>
<td>27</td>
<td>SPI_CS</td>
</tr>
<tr>
<td>RSTN</td>
<td>38</td>
<td>RSTN, RESET</td>
</tr>
<tr>
<td>GND</td>
<td>49</td>
<td></td>
</tr>
</tbody>
</table>

The board section labeled BlueNRG-LP (Figure 2, region B) includes the following main components:
- BlueNRG-LP low power system-on-chip
- 32 MHz high frequency crystal
- 32 kHz low frequency crystal for the lowest power consumption
- SMA connector

### 2.3 Power supply

DL1 green LED (Figure 2, region G) signals the board is being powered, either via:
- micro USB connector CN6 (Figure 2, region C)
- two AAA batteries on BATT1 region at the rear of the board
- a CR2032 coin cell battery (if the coin battery support is soldered on BATT2 region at the rear of the board)
- an external DC power supply

**Table 3. STEVAL-IDB011V1 kit platform power supply modes**

<table>
<thead>
<tr>
<th>Power supply mode</th>
<th>JP2 settings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - USB</td>
<td>USB position</td>
<td>USB supply through CN6 connector (Figure 2, region C)</td>
</tr>
<tr>
<td>2 - Battery</td>
<td>BAT position</td>
<td>The supply voltage must be provided through batteries (at the rear of the board)</td>
</tr>
<tr>
<td>3 – External DC power supply</td>
<td>Removed</td>
<td>The supply voltage must be provided through JP2 pin 2. USB connection is not needed</td>
</tr>
</tbody>
</table>

### 2.4 Jumpers

The following jumpers are available (Figure 2, region S).

**Table 4. BlueNRG-LP kit platform jumpers**

<table>
<thead>
<tr>
<th>Jumper</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP1</td>
<td>It provides the voltage to the BlueNRG-LP circuit. It must be adapted and can be used for BlueNRG-LP current measurements.</td>
</tr>
<tr>
<td>JP2</td>
<td>It is a switch between two power domains:</td>
</tr>
<tr>
<td></td>
<td>• BAT position - to provide power from battery holder</td>
</tr>
<tr>
<td></td>
<td>• USB position - to provide power from USB connector</td>
</tr>
<tr>
<td>JP3</td>
<td>It connects the BlueNRG-LP BLE_SWCLK pin with the USB_CMSISDAP SWCLK pin. It must be adapted.</td>
</tr>
<tr>
<td>JP4</td>
<td>It connects the BlueNRG-LP BLE_SWDIO pin with the USB_CMSISDAP SWDIO pin. It must be adapted.</td>
</tr>
<tr>
<td>JP5</td>
<td>It connects the BlueNRG-LP BLE_RSTN pin a pin of the USB_CMSISDAP. It must be adapted.</td>
</tr>
</tbody>
</table>

### 2.5 Sensors

The following sensors are available on the platform:

1. An LPS22HH (Figure 2 region H) piezo resistive absolute pressure sensor which functions as a digital output barometer. The device comprises a sensing element and an IC interface which communicates through I²C from the sensing element to the application.

2. An LSM6DSOX (Figure 2, region I) 3D digital accelerometer and 3D digital gyroscope with embedded temperature sensor which communicates via SPI interface. One line for interrupt is also connected.

3. An MP34DT05-A MEMS audio sensor omnidirectional digital microphone connected to the BlueNRG-LP PDM port.

### 2.6 Extension connector

BlueNRG-LP signal test points are shared on two Arduino connector rows: CN1, CN3 (Figure 2 – region J) and CN2, CN4 (region P).

### 2.7 Push buttons

The board has one user button to reset the microcontroller (Figure 2 – region L) and two further buttons for application purposes (region N).

### 2.8 LEDs

LEDs DL1 (green power LED), DL2 (green), DL3 (red), U5 (blue) and DL4 (USB_CMSISDAP activity red LED) are available on the board (Figure 2, regions G, F, Q and O).
2.9 CMSIS-DAP and Virtual COM

The most important features of the STM32F103xx microcontroller (Figure 2, region N) are:

- CMSIS-DAP debugging/programming capability through the USB micro connector
- USB-to-serial bridge providing an I/O communication channel with the BlueNRG-LP device (to interface with a USB host device as a PC)
- drag and drop capability to program the BlueNRG-LP

**Note:** The on-board STM32F103xx microcontroller is not intended to be programmed by users. ST provides a pre-programmed firmware image (USB_CMSISDAP.hex) to interface the BlueNRG-LP with a USB host device with the highlighted features.

2.9.1 Virtual COM port driver setup for Windows

In Windows 7, the driver installation for the Virtual COM port requires the installation of the driver files available at https://os.mbed.com/docs/mbed-os/v5.7/tutorials/windows-serial-driver.html
In Windows 10, no installation is required.

2.9.2 System functionality checks

To check if the system is ready to use, follow the steps below.

**Step 1.** Check whether the CMSIS-DAP device is present in the Windows Device Manager.

**Figure 3. Windows Device Manager - CMSIS-DAP**

- DAPLink CMSIS-DAP
- HID-compliant vendor-defined device
- MBED VFS USB Device
- USB Composite Device
- USB Input Device
- USB Mass Storage Device
- USB Serial Device (COM72)
- WebUSB: CMSIS-DAP

**Note:** The composite device (WebUSB: CMSIS-DAP) installation is not required, as this functionality is not used.

**Step 2.** Check under [Devices and drives] whether the ST IDB011VX mass storage device is present (if the board is powered by USB cable and connected to a PC).

**Figure 4. ST IDB011VX mass storage device**

- Windows (C): 255 GB free of 455 GB
- ST IDB011VX (E): 63.9 MB free of 63.9 MB

**Step 3.** Check whether DL1 LED is on (if the board is powered by USB).

2.9.3 USB_CMSISDAP programming/debugging feature

The STEVAL-IDB011V1 enables the on-board programming/debugging feature (USB_CMSISDAP). To use it, you have to choose CMSIS DAP as debugger/programmer in IAR EWARM or KEIL µVision development environments.
Figure 5. IAR EWARM project - Debugger option

Options for node “BSP_Com”

Category:
- General Options
- Static Analysis
- Runtime Checking
- C/C++ Compiler
- Assembler
- Output Converter
- Custom Build
- Build Actions
- Linker

Debugger:
- Simulator
- CADD
- CMSIS DAP
- GDB Server
- I-Jet/ITASjet
- J-Link/J-Trace
- TI Stellaris
- Nu-Link
- P3 micro
- ST-LINK
- Third-Party Driver
- TI MSP430
- TI XDS

Setup:
- Driver: CMSIS DAP
- Run to: main

Setup macros:
- Use macro file(s)

Device description file:
- Override default
- $TOOLKIT_DIRS/CONFIG/debugger/ST/BlueNRG-LP.dcd

[Diagram showing the Debugger options window]

OK Cancel
2.9.4 USB_CMSISDAP firmware update

If an updated version of the USB_CMSISDAP firmware is released, you should follow the procedure below for firmware update.

Step 1. Unplug the USB cable (if plugged).
Step 2. Press and hold the [RESET] button.
Step 3. Plug the USB cable.
Step 4. Release the [RESET] button.

A new mass storage device ([MAINTENANCE]) appears.

Step 5. Copy and paste the new binary image into the [MAINTENANCE] mass storage device.
Step 6. At the end of the operation, unplug and then plug the USB cable again to start the board up.
2.10 BlueNRG-LP programming and debugging

To program and debug the BlueNRG-LP embedded in the STEVAL-IDB011V1 board, you can use an external SWD programmer/debugger tool after removing JP3, JP4 and JP5 jumpers from the BlueNRG-LP kit platform and connecting the SWD tool to the board as listed in the table below.

<table>
<thead>
<tr>
<th>SWD pins</th>
<th>BlueNRG-LP kit platform pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWDIO</td>
<td>JP4 pin 1</td>
</tr>
<tr>
<td>SWCLK</td>
<td>JP3 pin 1</td>
</tr>
<tr>
<td>NRST</td>
<td>JP5 pin 1</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>Target VCC</td>
<td>VBAT</td>
</tr>
</tbody>
</table>

Remember:
Before using the supported IDE toolchains, select the related SWD programmer/debugger tool from the IAR EWARM project (Option/Debugger) and from the KEIL µVision project (Option/Debug).

The CMSIS-DAP debugging/programming capability via the USB micro-connector can also be used to program/debug a BlueNRG-LP device on a different board. Remove JP3, JP4 and JP5 jumpers and connect the BlueNRG-LP kit platform to the board SWD pins, GND and Target VCC as follows:

<table>
<thead>
<tr>
<th>BlueNRG-LP kit platform pins</th>
<th>User board SWD pins</th>
</tr>
</thead>
<tbody>
<tr>
<td>JP4 pin 2</td>
<td>SWDIO</td>
</tr>
<tr>
<td>JP3 pin 2</td>
<td>SWCLK</td>
</tr>
<tr>
<td>JP5 pin 2</td>
<td>NRST</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>VBAT</td>
<td>Target VCC</td>
</tr>
</tbody>
</table>

Then connect the BlueNRG-LP kit platform USB to a PC USB port to start programming and debugging the BlueNRG-LP device on your board.

2.11 Current measurements

To monitor the BlueNRG-LP power consumption, you must remove the jumper from JP1 and insert an ammeter between the connector pins 1 and 2.

Since BlueNRG-LP power consumption is usually very low, an accurate instrument in the range of few microamps is recommended.
2.12 Hardware setup

Step 1. Connect an antenna to the SMA connector.
Step 2. Configure the STEVAL-IDB011V1 board to USB power supply mode as per Table 3.
Step 3. Connect the STEVAL-IDB011V1 board to a PC via USB cable (CN6 connector).
Step 4. Check whether the DL1 power indication LED is on.
The BlueNRG-LP Navigator GUI lets you select and run demonstration applications without additional hardware. It provides access to the following DK software package components:

- BlueNRG-LP Bluetooth low energy (BLE) demonstration applications
- BlueNRG-LP peripheral driver examples (LL, HAL & MIX)
- BlueNRG-LP 2.4 GHz radio proprietary examples
- BlueNRG-LP development kits
- Release notes
- License files

With BlueNRG-LP DK Navigator, you can directly download and run the selected pre-built application binary image (BLE examples or peripheral driver example) on the BlueNRG-LP platform without a JTAG interface. The interface gives demo descriptions and access to board configurations and source code if needed. You can run the utility through the BlueNRG-LP Navigator icon under: [Start]>[ST BlueNRG-LP DK X.X.X]>[BlueNRG-LP Navigator].

3.1 Demonstration applications

You can navigate the menus for the reference/demo application you want to launch. For each application, the following information is provided:

- application settings (if applicable)
- application description
- application hardware related information (e.g., LED signals, jumper configurations, etc.)
The following functions are also available for each application:

- **Flash**: to automatically download and run the available pre-built binary file to the BlueNRG-LP platform connected to a PC USB port
- **Doc**: to display application documentation (in html format)
- **Project**: to open the project folder with application headers, source and project files

The figure below shows how to run the BLE Beacon demo application; the other demos function similarly.

![Figure 9. BlueNRG-LP Navigator - BLE Beacon application](image)

When the BlueNRG-LP platform is connected to your PC USB port, you can select the [Flash & Run] tab on the application window to download and run the available pre-built application binary image on the BlueNRG-LP platform.
3.2 Basic examples

This page lists some basic sample applications to verify some of the BlueNRG-LP device modes (alive, sleeping, wake-up).
3.3 BLE demonstration and test applications

This page lists all the available Bluetooth LE demonstration applications in the DK software package. The applications provide usage examples of the Bluetooth LE stack features for the BlueNRG-LP device.
3.4 Peripheral driver examples

BlueNRG-LP Navigator includes three sets of peripheral driver examples related to the following categories:

1. LL
2. HAL
3. MIX
The 2.4 GHz radio proprietary driver provides access to the BlueNRG-LP device radio to send and receive packets without using the Bluetooth link layer.

The 2.4 GHz radio proprietary examples are built on top of the 2.4 GHz radio proprietary driver and can be used as reference examples for building other applications which use the BlueNRG-LP Radio.
3.6 Development kits

This window displays the available BlueNRG-LP DK kit platforms and corresponding resources. When you hover the mouse pointer over a specific item, the related component is highlighted on the board.
Further, user can get access to the development kit 3D view just selecting the related icon.

3.7 Release Notes and License

The Release Notes and License pages display the DK SW package Release Notes (in html format) and the DK software package license file, respectively.
3.8 Documentation Index

This page displays the DK SW package documentation index providing links to the available documentations (in html and pdf format).
4 BlueNRG-LP Radio Parameters Wizard

The BlueNRG-LP Radio Parameters Wizard is a PC application which allows defining the proper values required for the correct BlueNRG-LP Bluetooth LE radio initialization, based on the specific user application scenario. A configuration header file (*.config.h) is generated to be used on your demonstration application folder.

**Note:** The BlueNRG-LP Radio Parameters Wizard is provided only for BlueNRG-LP DK SW package (STSW-BNRGLP-DK) supporting BLE stack v3.x family.

### 4.1 How to run the Radio Parameters Wizard

User can run this utility by clicking on the BlueNRG-LP Radio Init Wizard icon under [Start] \>[ST BlueNRG-LP DK X.X.X].

**Figure 18. BlueNRG-LP Radio Parameters Wizard - general configuration**

#### 4.2 Main user interface window

In the left section of the BlueNRG-LP Radio Initialization Parameters Wizard utility, you can select the following topics to define the radio initialization parameters based on the specific Bluetooth LE application requirements:

1. General configuration
2. Radio configuration
3. Service configuration
4. Connection configuration
5. Security database configuration
6. OTA configuration
7. Privacy configuration
8. Overview
9. Output
Refer to the BlueNRG-LP Radio Parameters Wizard documentation available in the BlueNRG-LP DK SW package (STSW-BNRGLP-DK) for more details about each provided configuration section.
5 Programming with the BlueNRG-LP system on chip

The BlueNRG-LP Bluetooth low energy (BLE) stack is provided as a binary library with a set of APIs to control Bluetooth LE functionality.

Some callbacks are provided for user applications to handle Bluetooth LE stack events. You have to link the binary library to your application and use the relevant APIs to access Bluetooth LE functions and complete the stack event callbacks, managing responses according to application requirements.

A set of software driver APIs is also included to access the BlueNRG-LP SoC peripherals and resources (ADC, GPIO, I2C, timers, micro, RTC, SPI, SysTick, UART and WDG).

The development kit software includes sample code on how to configure BlueNRG-LP and use the device peripherals, Bluetooth LE APIs and event callbacks.

5.1 Software directories

The BlueNRG-LP DK software packages files are organized in the following main directories:

- **Application**: contains BlueNRG-LP Navigator and Radio Init Parameters Wizard PC applications
- **Doc**: contains doxygen BLE APIs and events, BlueNRG-LP peripheral drivers, Bluetooth LE demo applications, BlueNRG-LP Peripheral examples, BlueNRG-LP SDK and HAL driver documentation, DK release notes and license file
- **Firmware**: contains pre-built binary Bluetooth LE and peripheral driver sample applications
- **Drivers**:
  - **BSP**: SDK drivers providing an API interface to the BlueNRG-LP platform hardware resources (LEDs, buttons, sensors, I/O channel)
  - **CMSIS**: BlueNRG-LP CMSIS files
  - **External_micro**: drivers framework to support network coprocessor framework with an external microcontroller
  - **Peripherals_Drivers**: BlueNRG-LP drivers for device peripherals (ADC, clock, DMA, Flash, GPIO, I2C, timers, RTC, SPI, UART and watchdog).
- **Middlewares**:
  - **Bluetooth LE**: Bluetooth low energy stack binary library and all the definitions of stack APIs, stack and events callbacks. BLE stack v3.x configuration header and source files
  - **BLE_Application**: BLE application framework files (Bluetooth LE stack layers define values, OTA FW upgrade, Bluetooth LE utilities, master library, GATT, GAP standard profiles, ATT Prepare Write Queue framework)
  - **cryptolib**: AES crypto library
  - **External_micro**: Bluetooth LE framework to support network coprocessor framework with an external microcontroller
  - **HAL**: hardware abstraction level APIs to abstract certain BlueNRG-LP hardware features (sleep modes, clock based on SysTick, etc.).
  - **NVMDB**: non volatile memory drivers
- **Projects**
  - **BLE_Examples**: Bluetooth low energy demonstration application including headers, source files and EWARM, Keil and Atollic project files
  - **Peripheral_Examples**: with sample applications for the BlueNRG-LP peripherals LL, HAL and mixed examples including headers, source files and project files. It also includes the 2.4 GHz radio proprietary examples (in the Examples_MIX folder)
- **Utility**: contains the BlueNRG-LP patches for IAR, EWARM and KEIL, MDK-ARM toolchains and the secure bootloader utilities.
6 BLE beacon demonstration application

The BLE beacon demo is supported by the BlueNRG-LP development platforms (STEVAL-IDB011V1) and demonstrates how to configure a BlueNRG-LP device to advertise specific manufacturing data and allow another Bluetooth LE device to determine whether it is in the Bluetooth LE beacon device range.

The demo also allows using the extended advertising feature to configure the beacon on a secondary advertising channel.

6.1 BLE Beacon application setup

6.1.1 Initialization

To configure a Bluetooth LE device to act as a beacon device, you have to correctly initialize the BLE stack as follows:

```c
aci_gatt_srv_init();
aci_gap_init(GAP_PERIPHERAL_ROLE, 0, 0x08, 0,&service_handle, &dev_name_char_handle &appearance_char_handle);
```

**RELATED LINKS**

For BLE stack API documentation, refer to STSW-BNRGLP-DK sw DK package under Docs\BlueNRG-LP_aci_events folder.

6.1.2 Manufacturing data

The BLE beacon application advertises the following manufacturing data.

<table>
<thead>
<tr>
<th>Data field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company identifier code</td>
<td>SIG company identifier @&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Default is 0x0030 (for STMicroelectronics)</td>
</tr>
<tr>
<td>ID</td>
<td>Beacon ID</td>
<td>Fixed value</td>
</tr>
<tr>
<td>Location UUID</td>
<td>Beacons UUID</td>
<td>Used to identify specific beacons</td>
</tr>
<tr>
<td>Major number</td>
<td>Identifier for a group of beacons</td>
<td>Used to group a related set of beacons</td>
</tr>
<tr>
<td>Minor number</td>
<td>Identifier for a single beacon</td>
<td>Used to identify a single beacon</td>
</tr>
<tr>
<td>Tx Power</td>
<td>Complement of the Tx power</td>
<td>Used to establish the distance from the device</td>
</tr>
</tbody>
</table>


6.1.3 Non-connectable mode

The Bluetooth LE beacon device uses the GAP API commands to enter non-connectable mode.

It sets the advertising configuration parameters as follows:

```c
ret = aci_gap_set_advertising_configuration(0,GAP_MODE_GENERAL_DISCOVERABLE,
                                           ADV_PROP_LEGACY,
                                           160, 160,
                                           ADV_CH_ALL,
                                           0,NULL,
                                           ADV_NO_WHITE_LIST_USE,
                                           0, LE_1M_PHY, 0, LE_1M_PHY, 0,0);
```
To define the specific selected manufacturer data, the BLE beacon application can use the following GAP API:

```c
/* Define the beacon manufacturing payload */
static uint8_t adv_data[] = {
/* Advertising data: Flags AD Type*/
0x02,
0x01,
0x06,
/* Advertising data: manufacturer specific data*/
26, //len
AD_TYPE_MANUFACTURER_SPECIFIC_DATA, //manufacturer type
0x30, 0x00, //company identifier code (Default is 0x0030 - STMicroelectronics: To be customized
for specific identifier
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, 0x07, //Minor number
0xC8 //2's complement of the Tx power (-56dB));
};
ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);
```

The Bluetooth LE beacon device then enters advertising mode as follows:

```c
Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
/* Enable advertising */
ret = aci_gap_set_advertising_enable (ENABLE, 1, Advertising_Set_Parameters);
```

**Note:** BLE beacon with Flash Management demonstration application is also available and allows configuring a beacon device. It also shows how to properly handle Flash operations (erase and write) and preserve the Bluetooth LE radio activities by synchronizing Flash operations with the scheduled Bluetooth LE radio activities through the `aci_hal_end_of_radio_activity_event()` event callback timing information.

### 6.1.4 Extended advertising mode

A new project configuration (ExtendedAdv) shows how to use the Bluetooth LE specification v5.0 extended advertising feature to configure a beacon also on a secondary advertising channel as follow:

```c
#if EXTENDED_ADV
/* Set advertising configuration for extended advertising. */
ret = aci_gap_set_advertising_configuration(1, GAP_MODE_GENERAL_DISCOVERABLE,
ADV_PROP_NONE,
160, 160,
ADV_CH_ALL,
0,NULL, /* No peer address*/
ADV_NO_WRITE_LIST_USE,
0,/* 0 dBm */
(EXT_ADV_PHY==LE_2M_PHY)?LE_1M_PHY:EXT_ADV_PHY,
0,/* 0 skips */
EXT_ADV_PHY, /* Secondary advertising PHY */
1, /* SID */
0 /* No scan request notifications */);
ret = aci_gap_set_advertising_data(1, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);
#endif
```
The extended advertising is enabled as follows:

```c
/* Enable advertising */
#if EXTENDED_ADV
Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;
Advertising_Set_Parameters[1].Advertising_Handle = 1;
Advertising_Set_Parameters[1].Duration = 0;
Advertising_Set_Parameters[1].Max_Extended_Advertising_Events = 0;
ret = aci_gap_set_advertising_enable(ENABLE, 2, Advertising_Set_Parameters);
#endif
```

### 6.2 BLE Beacon FreeRTOS example

The BLE_Beacon_FreeRTOS example shows how to use FreeRTOS with ST BLE stack v3.x. It configures a BlueNRG-LP device in non-connectable advertising mode with specific manufacturing data. The BTLE_StackTick() function is called from a FreeRTOS task (BLETask).

A task randomly changes the Minor number in the advertising data, every 500 ms, sending a message via UART each time. Another task sends other messages via UART every 200 ms and generates a short pulse on LED3 (visible with a logic analyzer or oscilloscope).

A low priority has been assigned to the BLETask in this example.

In general, assigning a high priority to the BLETask can give better latency, especially if other tasks occupy a large amount of CPU resources.

**Note:** Apart from very short tasks involving sporadic operations while waiting for an event, most other tasks should be assigned a lower priority than the BLETask to avoid slowing Bluetooth LE operations down.
The BlueNRG-LP development platform (STEV AL-IDB011V1) supports the BLE Serial port demo (server and client roles) that implements two-way point-to-point wireless communication between two Bluetooth LE devices. This demo application exposes a single Serial port service with the following (20-byte max.) characteristic values:

- **TX characteristic** to enable notifications; before transmitting data, the server sends notifications with the value of the TX characteristic
- **RX characteristic** is a writable characteristic; when the client has to transmit data to the server, it writes a value in this characteristic

There are two device roles which can be selected through the specific project workspace:

- the server that exposes the Serial port service (Bluetooth LE peripheral device)
- the Client that uses the Serial port service (Bluetooth LE central device)

The application requires two devices to be programmed with server and client roles and have to be connected to a PC via USB with an open serial terminal for each device, with the following configuration.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baudrate</td>
<td>115200 bit/s</td>
</tr>
<tr>
<td>Data bits</td>
<td>8</td>
</tr>
<tr>
<td>Parity bits</td>
<td>None</td>
</tr>
<tr>
<td>Stop bits</td>
<td>1</td>
</tr>
</tbody>
</table>

The application listens for keys typed in one device terminal and sends them to the remote device when the return key is pressed; the remote device then outputs the received RF messages to the serial port. Therefore, anything typed in one terminal becomes visible in the other one.

### 7.1 Peripheral and central device setup

Before establishing the point-to-point wireless Serial port communication between the two Bluetooth LE Serial port devices (server-peripheral and client-central), you need to set the Bluetooth LE device up on both devices by sending a series of API commands to the processor.

#### 7.1.1 Initialization

The Bluetooth LE stack must be correctly initialized before establishing a connection with another Bluetooth LE device through the `aci_gatt_srv_init()` and `aci_gap_init()` APIs.

For `aci_gap_init()`:
- **Bluetooth LE Serial port server role**
  ```c
  aci_gap_init(GAP_PERIPHERAL_ROLE, 0, 0x08,0,&service_handle, &dev_name_char_handle, &appearance_char_handle);
  ```

- **Bluetooth LE Serial port client role**
  ```c
  aci_gap_init(GAP_CENTRAL_ROLE, 0, 0x08,0,&service_handle, &dev_name_char_handle, &appearance_char_handle);
  ```

Peripheral and central Bluetooth LE roles must be specified in the `aci_gap_init()` command.
### 7.1.2 Add service and characteristics

The Serial port service is added to the Bluetooth LE Serial port server device via

```c
aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&serial_port_service);
```

where `serial_port_service` is the private service structure allocated for the Serial port service and related characteristics.

The TX characteristic is obtained by using the following command on the Bluetooth LE Serial port server device:

```c
TXCharHandle=aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t *)&serial_port_chars[0]);
```

where `serial_port_chars[0]` is the characteristic structure allocated for the TX characteristic (notify property).

The characteristic handle is returned on the `TXCharHandle` variable.

The RX characteristic is obtained by using the following command on the Bluetooth LE Serial port server device:

```c
RXCharHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t *)&serial_port_chars[1]);
```

where `serial_port_chars[1]` is the characteristic structure allocated for the RX characteristic (write property).

The characteristic handle is returned on the `RXCharHandle` variable.

### RELATED LINKS

For BLE stack API documentation, refer to STSW-BNRGLP-DK sw DK package under Docs\BlueNRG-LP_aci_events folder

### 7.1.3 Entering connectable mode

The server device uses GAP API commands to enter the general discoverable mode.

```c
/* Configure advertising parameters */
ret = aci_gap_set_advertising_configuration(0,GAP_MODE_GENERAL_DISCOVERABLE,
    ADV_PROP_CONNECTABLE|ADV_PROP_SCANNABLE|
    ADV_PROP_LEGACY,
    ADV_INTERVAL_MIN, ADV_INTERVAL_MAX,
    ADV_CH_ALL,
    0,NUL,ADV_NO.WHITE_LIST_USE,
    0, LE_1M_PHY, 0,LE_1M_PHY, 0,0);

/* Define advertising data */
static uint8_t adv_data[] = {0x02,AD_TYPE_FLAGS,
    FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|FLAG_BIT_BR_EDR_NOT_SUPPORTED,
    16, AD_TYPE_COMPLETE_LOCAL_NAME,'S','P','o','r','t','_','L','P'};
/* Set advertising data */
ret = aci_gap_set_advertising_data(0,ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;
/* Enable advertising */
ret = aci_gap_set_advertising_enable (ENABLE, 1, Advertising_Set_Parameters);
```

### 7.1.4 Connection with the central device

Once the server device is discoverable by the Bluetooth LE Serial port client device, the client device uses the following APIs to connect to the Bluetooth LE Serial port server device:

```c
/* Set scanning configuration parameters */
ret = aci_gap_set_scan_configuration(DUPLICATE_FILTER_ENABLED,SCAN_ACCEPT_ALL,
    LE_1M_PHY_BIT, PASSIVE_SCAN, SCAN_INTERVAL, SCAN_WINDOW);
/* Set connection configuration parameters*/
ret = aci_gap_set_connection_configuration(LE_1M_PHY_BIT, CONN_INTERVAL_MIN,
    0, SUPERVISION_TIMEOUT,
    CE_LENGTH, CE_LENGTH);
```
To connect with the Bluetooth LE Serial port server device, the following API is used: 

\[
\text{ret = aci_gap_create_connection(LE_1M_PHY_BIT, PUBLIC_ADDR bdaddr);}
\]

where \(\text{bdaddr}\) is the peer address of the client device.

Once the two devices are connected, you can set up the corresponding serial terminals and type messages in both of them. The typed characters are stored in two respective buffers and when the return key is pressed:

- on the Bluetooth LE Serial port server device, the typed characters are sent to the Bluetooth LE Serial port client device by notifying the previously added TX characteristic (after notifications are enabled) with:

\[
\text{aci_gatt_srv_notify(connection_handle, TXCharHandle + 1, 0, len, (uint8_t *)(cmd + cmd_buff_start))};
\]

- on the Bluetooth LE Serial port client device, the typed characters are sent to the Bluetooth LE Serial port server device by writing the previously added RX characteristic with:

\[
\text{aci_gatt_clt_write_without_resp(connection_handle, rx_handle + 1 len, (uint8_t *) (cmd+ cmd_buff_start))};
\]

where \(\text{connection_handle}\) is the handle returned upon connection as a parameter of the connection complete event, \(\text{rx_handle}\) is the RX characteristic handle discovered by the client device.

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

**Figure 19. Bluetooth LE Serial port client**

**Figure 20. Bluetooth LE Serial port server**
8 BLE Serial port master and slave demo application

The BlueNRG-LP development platform (STEV AL-IDB011V1) supports the BLE Serial port master and slave demo that implements point-to-point wireless communication using a single application which configures the Serial port client and server roles at runtime.

The Serial port demo application configures a Bluetooth LE device as central or peripheral using the API:

```c
aci_gap_init(GAP_CENTRAL_ROLE|GAP_PERIPHERAL_ROLE, 0, 0x07, 0,
&service_handle,&dev_name_char_handle, &appearance_char_handle);
```

It then initiates a discovery procedure for another Bluetooth LE device configured with the same Serial port master and slave application image. If the device is found within a random interval, it starts a connection procedure and waits for the connection to be established. If the discovery procedure time expires without finding another Serial port master and slave device, the device enters discovery mode and waits for another Serial port master and slave device to discover and connect to it.

When the connection is established, the client and server roles are defined and the Serial port communication channel can be used.

This demo application exposes a single Serial port service with the following (20 byte max.) characteristic values:
- TX characteristic allowing the client to enable notifications; before transmitting data, the server sends notifications with the value of the TX characteristic.
- RX characteristic (writable characteristic); when the client has data to be transmitted to the server, it writes a value in this characteristic.

The application requires two devices to be programmed with the same application, with the server and client roles defined at runtime; after the devices have been connected to a PC via USB, you have to open a serial terminal on both with the configuration shown in Table 6.

The application listens for keys typed in one device terminal and sends them to the remote device when the return key is pressed; the remote device then outputs the received RF messages to the serial port. Therefore, anything typed in one terminal becomes visible in the other one.

8.1 BLE Serial port master and slave roles

To set up a point-to-point wireless Serial port, two Bluetooth LE Serial port master and slave devices interact with each other.

The BLE stack must first be set up on both devices by sending a series of API commands to the processor. The Serial port master and slave client and server roles are defined at runtime.

8.1.1 Initialization

The BLE stack must be correctly initialized before establishing a connection with another Bluetooth LE device via the following commands:

```c
aci_gatt_srv_init();
aci_gap_init(GAP_CENTRAL_ROLE|GAP_PERIPHERAL_ROLE, TRUE, 0x07, 0,
&service_handle,&dev_name_char_handle, &appearance_char_handle);
```

The Bluetooth LE peripheral and central roles are specified in the `aci_gap_init()` command.

8.1.2 Add service and characteristics

Refer to Section 7.1.2 Add service and characteristics.
8.1.3 Start discovery procedure

To find another Bluetooth LE serial port master and slave device in discovery mode, a discovery procedure must be started through the following code:

```c
/* Set discovery procedure parameters */
ret = aci_gap_set_scan_configuration(DUPLICATE_FILTER_ENABLED, SCAN_ACCEPT_ALL,
    LE_1M_PHY_BIT, PASSIVE_SCAN, DISCOVERY_PROC_SCAN_INT,
    DISCOVERY_PROC_SCAN_WIN);

/* Start general discovery procedure (0x01) */
ret = aci_gap_start_procedure(0x01, LE_1M_PHY_BIT, 0, 0);
```

8.1.4 Enter connectable mode

The following GAP API commands are used for entering general discoverable mode:

```c
/* Configure advertising parameters */
ret = aci_gap_set_advertising_configuration(0, GAP_MODE_GENERAL_DISCOVERABLE,
    ADV_PROP_CONNECTABLE|ADV_PROP_SCANNABLE|
    ADV_PROP_LEGACY,
    ADV_INTERVAL_MIN,
    ADV_INTERVAL_MAX,
    ADV_CH_ALL,
    0, NULL,
    0, LE_1M_PHY, 0, LE_1M_PHY, 0, 0);

/* Define advertising data */
static uint8_t adv_data[] = {0x02, AD_TYPE_FLAGS,
    FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|FLAG_BIT_BR_EDR_NOT_SUPPORTED,
    13, AD_TYPE_COMPLETE_LOCAL_NAME,'S','p','o','r','t','_','L','P','_','N','e','w'};

/* Set advertising data */
ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);

/* Enable advertising */
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].MaxExtendedAdvertisingEvents = 0;
ret = aci_gap_set_advertising_enable (ENABLE, 1, Advertising_Set_Parameters);
```

8.1.5 Connection with serial port master and slave client device

In the discovery and mode assignment procedures, the two serial port master and slave applications assume, respectively, client and server roles at runtime. During this initial configuration phase, when a serial port master and slave device is put in discoverable mode and is found by the other serial port master and slave device performing a discovery procedure, a Bluetooth low energy connection is created and the device roles are defined.

The following GAP API commands are used for connecting to the discovered device:

```c
/* Initialization phase: set connection configuration parameters */
ret = aci_gap_set_connection_configuration(LE_1M_PHY_BIT, 40, 40, 0, 60, 2000, 2000);

/* Create connection */
ret = aci_gap_create_connection(LE_1M_PHY_BIT,
    discovery.device_found_address_type,
    discovery.device_found_address);
```

`device_found_address_type` is the address type of the discovered serial port master and slave and `device_found_address` is the peer address of the discovered serial port master and slave device.
Once the two devices are connected, you can set up the corresponding serial terminals and type messages in either of them. The typed characters are stored in two respective buffers and when the return key is pressed:

- on the Bluetooth LE serial port master-and-slave server device, the typed characters are sent to the master-and-slave client device by notifying the previously added TX characteristic (after notifications have been enabled) through the following command:

```c
aci_gatt_srv_notify(connection_handle, TXCharHandle + 1, 0, len,
                      (uint8_t *)(cmd + cmd_buff_start))
```

- on the master-and-slave client device, the typed characters are sent to the master-and-slave server device, by writing the previously added RX characteristic through the following command:

```c
aci_gatt_clt_write_withoutResp(connection_handle, rx_handle + 1, len,
                                 (uint8_t *)(cmd + cmd_buff_start))
```

*connection_handle* is the handle returned upon connection as a parameter of the connection complete event. *rx_handle* is the RX characteristic handle discovered by the client device.

Once these API commands have been sent, the values of the TX and RX characteristics are displayed on the serial terminals.
The BlueNRG-LP development platform (STEV AL-IDB011V1) supports the BLE remote control application to control a remote device (like an actuator) using the BlueNRG-LP device. This application periodically broadcasts temperature values that can be read by any device. The data is encapsulated in a manufacturer-specific AD type and the content (besides the manufacturer ID, i.e., 0x0030 for STMicroelectronics) is as follows:

<table>
<thead>
<tr>
<th>Table 9. Bluetooth LE remote advertising data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0</td>
</tr>
<tr>
<td>App ID (0x05)</td>
</tr>
</tbody>
</table>

The temperature value is given in tenths of Celsius degrees.

The device is also connectable and exposes a characteristic used to control DL1, DL2 and DL3 LEDs on the Bluetooth LE kit 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 DL1
- bit 1 is the status of DL2
- bit 2 is the status of DL3

A remote device can therefore 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 a central device remains connected to the device indefinitely.

Security is not enabled by default, but can be changed with ENABLE_SECURITY (refer to BLE_RC_main.h file). When security is enabled, the central device must be authenticated before reading or writing the device characteristic.

To interact with a device configured as a Bluetooth LE remote control, another Bluetooth LE device (a BlueNRG-LP or any Bluetooth low energy device) can be used to detect and view broadcast data.

To control one of the LEDs, the device has to connect to a BlueNRG-LP Bluetooth LE remote control device and write in the exposed control point characteristic (service UUID = ed0ef62e-9b0d-11e4-89d3-123b93f75cba, control point characteristic UUID = ed0efb1a-9b0d-11e4-89d3-123b93f75cba).

9.1 BLE remote control application setup

To configure a BlueNRG-LP device to act as a remote control device, follow the procedure described in the following sections.

9.1.1 Initialization

The BLE stack must be correctly initialized before establishing a connection with another Bluetooth LE device through the following commands:

```c
aci_gatt_srv_init();
aci_gap_init(GAP_PERIPHERAL_ROLE, 0, 0x07, 0, &service_handle, &dev_name_char_handle, &appearance_char_handle);
```

--- RELATED LINKS ---

For BLE stack API documentation, refer to STSW-BNRLDP-DK sw DK package under Docs\BlueNRG-LP_aci_events folder

9.1.2 Define advertising data

The BLE remote control application advertises certain 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_response_data(sizeof(scan_resp_data), scan_resp_data);
/* Set advertising configuration parameters */
ret = aci_gap_set_advertising_configuration(0, GAP_MODE_GENERAL_DISCOVERABLE,
  ADV_PROP_CONNECTABLE | ADV_PROP_SCANNABLE | ADV_PROP_LEGACY,
  (ADV_INTERVAL_MIN_MS * 1000) / 625,
  (ADV_INTERVAL_MAX_MS * 1000) / 625,
  ADV_CH_ALL,
  PUBLIC_ADDR, NULL,
  ADV_NO_WHITE_LIST_USE,
  0, LE_1M_PHY, 0, LE_1M_PHY, 0, 0);
/* Define advertising data */
uint8_t adv_data[] = {0x02, AD_TYPE_FLAGS, FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|
  FLAG_BIT_BR_EDR_NOT_SUPPORTED, 0x06, AD_TYPE_MANUFACTURER_SPECIFIC_DATA, 0x30, 0x00, 0x05, 0xFF, 0xFF};
/* Set advertising data */
ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);
/* Start advertising */
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Ev = 0;
ret = aci_gap_set_advertising_enable(ENABLE, 1, Advertising_Set_Parameters);

On the development platform, the temperature sensor value is set in the adv_data variable.

9.1.3 Add service and characteristics

The BLE Remote Control service and characteristics is added via:
aci_gatt_srv_add_service((ble_gatt_srv_def_t *)&rc_service), where rc_service is the
private service structure allocated for the BLE remote service (ed0ef62e-9b0d-11e4-89d3-123b93f75cba) and
related characteristics.

The BLE remote control characteristic handle is obtained using the following command:
controlPointHandle = aci_gatt_srv_get_char_decl_handle((ble_gatt_chr_def_t *)&rc_chars[0]);

The rc_chars[0] is the private characteristics structure allocated for BLE remote control characteristic
(ed0efb1a-9b0d-11e4-89d3-123b93f75cba) and controlPointHandle is the BLE remote control characteristic
handle.

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

9.1.4 Connection with a Bluetooth LE Central device

When connected to a Bluetooth LE central device (another BlueNRG-LP device or any Bluetooth low energy
device), the controlPointHandle characteristic is used to control the Bluetooth LE remote control platform
LED.

Each time a write operation is performed on controlPointHandle, the aci_gatt_srv_write_event()
callback is raised and the selected LEDs are turned on or off.
The BLE sensor profile demo is supported on the BlueNRG-LP development platform (STEV AL-IDB011V1). It implements a proprietary, Bluetooth low energy (BLE) sensor profile. This example is useful for building new profiles and applications that use the BlueNRG-LP SoC. The GATT profile is not compliant with any existing specifications as the purpose of this project is to simply demonstrate how to implement a given profile.

This profile exposes the acceleration and environmental services. Figure 21. BLE sensor demo GATT database shows the whole GATT database, including the GATT (0x1801) and GAP (0x1800) services that are automatically added by the stack.

The acceleration service free fall characteristic cannot be read or written, but can be signaled. The application sends notification of this characteristic (with a value of 0x01) if a free fall condition is detected by the MEMS sensor (when the acceleration on the three axes is near zero for a certain amount of time). Notifications can be enabled or disabled by writing the associated client characteristic configuration descriptor.

Another service is defined, which contains characteristics that expose data from some environmental sensors: temperature and pressure. Each characteristic data type is described in a format descriptor. All of the characteristics have read-only properties.

**Figure 21. BLE sensor demo GATT database**

### 10.1 BlueNRG app for smartphones

An application is available for iOS™ and Android™ smartphones or tablets that also works with the BLE sensor profile demo. This app enables notification of the acceleration characteristic and displays the value on screen. Data from environmental sensors are also periodically read and displayed.
10.2 BLE sensor profile demo: connection with a central device

This section describes how to interact with a central device, while the BLE stack is acting as a peripheral. The central device may be another BlueNRG-LP device acting as a Bluetooth LE master, or any other Bluetooth Low Energy device.

The BLE stack must first be set up by sending a series of Bluetooth LE API commands to the processor.

10.2.1 Initialization

The BLE stack must be correctly initialized before establishing a connection with another Bluetooth LE device. This is done via:

```c
aci_gatt_srv_init();
aci_gap_init(GAP_PERIPHERAL_ROLE, 0, 0x07, &service_handle, &dev_name_char_handle, &appearance_char_handle);
```

See BLE stack API documentation for more information on these and following commands.

10.2.2 Add service and characteristics

The BlueNRG-LP BLE stack can act as server and client. A characteristic is an element in the server database where data is exposed, while a service contains one or more characteristics. The acceleration service and related characteristics are added with the following command:

```c
ret = aci_gatt_srv_add_service(&acc_service),
```

where `acc_service` is the allocated structure for the acceleration service.

The free fall and acceleration characteristics can be obtained by:

```c
freeFallCharHandle = aci_gatt_srv_get_char_decl_handle(&acc_chars[0]);
accCharHandle = aci_gatt_srv_get_char_decl_handle(&acc_chars[1]);
```

The free fall and acceleration characteristics handles are returned on `freeFallCharHandle` and `accCharHandle` variables, respectively.

Similar steps are followed for adding the environmental sensor and related characteristics.
10.2.3 Enter connectable mode

Use GAP API commands to enter one of the discoverable and connectable modes:

```c
/* Set advertising configuration parameters */
ret = aci_gap_set_advertising_configuration(0, GAP_MODE_GENERAL_DISCOVERABLE,
    ADV_PROP_CONNECTABLE|ADV_PROP_SCANNABLE|ADV_PROP_LEGACY,
    (ADV_INTERVAL_MIN_MS*1000)/625,
    (ADV_INTERVAL_MAX_MS*1000)/625,
    ADV_CH_ALL,
    0,NULL,
    ADV_NO_WHITE_LIST_USE,
    0, LE_1M_PHY, 0, LE_1M_PHY, 0,0);

/* Define advertising data */
uint8_t adv_data[] = {0x02, AD_TYPE_FLAGS,
    FLAG_BIT_LE_GENERAL_DISCOVERABLE_MODE|FLAG_BIT_BR_EDR_NOT_SUPPORTED,
    8, AD_TYPE_COMPLETE_LOCAL_NAME,'B','l','u','e','N','R','G'};

/* Set advertising data */
ret = aci_gap_set_advertising_data(0, ADV_COMPLETE_DATA, sizeof(adv_data), adv_data);

/* Enable advertising */
static Advertising_Set_Parameters_t Advertising_Set_Parameters[1];
Advertising_Set_Parameters[0].Advertising_Handle = 0;
Advertising_Set_Parameters[0].Duration = 0;
Advertising_Set_Parameters[0].Max_Extended_Advertising_Events = 0;
ret = aci_gap_set_advertising_enable(ADV_LEGACY, ENABLE, 0, NULL);
```

10.2.4 Connection with central device

Once the BLE stack is in discoverable mode, it can be detected by a central device. The smartphone app described in Section 10.1 is designed to interact with the sensor profile demos (it also supports the BlueNRG-LP device).

Any Bluetooth low energy device can connect to the BLE sensor profile demo.

For example, the LightBlue application available in the Apple Store connects the iPhone (versions 4S/5 and above) to the sensor profile device. When you use the LightBlue application, the detected devices appear on the screen with the BlueNRG name. By tapping on the box to connect to the device, a list of all the available services is shown; tapping a service shows the characteristics for that service.

The acceleration characteristic can be notified using the following command:

```c
aci_gatt_srv_notify(connection_handle, accCharHandle + 1, 0, 6, buff);
```

where `buff` is a variable containing the three-axis acceleration values.

Once this API command has been sent, the new value of the characteristic is displayed on the phone.

**Note:** A BLE Sensor demo for the ST BLE Sensor smartphone app is also provided. The demo allows configuring the BlueNRG-LP kit platform and provides a set of sensor data that can be logged to different cloud providers. Emulated sensor value configurations for the accelerometer and/or temperature pressure sensors are also supported.
11 BLE sensor profile center demo

The BLE sensor profile central demo is supported on the BlueNRG-LP development platform (STEVAL-IDB011V1). It implements a basic version of the BLE Sensor Profile Central role which emulates the Sensor Demo applications available for smartphones (iOS and Android).

This application configures a BlueNRG-LP device as a Sensor device Central role, which is able to find, connect and properly configure the free fall, acceleration and environmental sensor characteristics provided by a Bluetooth LE development platform configured as a Bluetooth LE Sensor device Peripheral role (refer to Section 10).

This application uses a new set of APIs to perform the following operations on the BlueNRG-LP 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 a binary library and they are fully documented on available doxygen documentation within the STSW-BNRGLP-DK package. The following master/central binary library is provided in the Library/BLUE_Application/Profile_Central/library folder: libmaster_library_bluenrgx.a for IAR and Keil in the STSW-BNRGLP-DK package.
12 BLE HID/HOGP demonstration applications

The BLE HID/HOGP demonstration applications are supported by the BlueNRG-LP development platform (STEVAL-IDB011V1). It demonstrates a Bluetooth LE device using the standard HID/HOGP Bluetooth low energy application profile. Keyboard and mouse demo examples are provided.

12.1 BLE HID/HOGP keyboard demonstration application

The BLE HID keyboard application implements a basic HID keyboard compliant with the standard HID/HOGP BLE application profile.
The HID keyboard device is named ‘STKeyboard’ in the central device list.
To use the HID keyboard, follow the steps below.
Step 1. Insert PIN (123456) to successfully complete the bonding and pairing procedure.
Step 2. Connect the Bluetooth LE development platform to a PC USB port.
Step 3. Open a HyperTerminal window (115200, 8, N,1).
Step 4. Put the cursor focus on the HyperTerminal window.
The keys sent to the central device using the HID/HOGP BLE application profile are also shown on the HyperTerminal window.

12.2 BLE HID/HOGP mouse demonstration application

The BLE HID mouse application implements a basic HID mouse with two buttons compliant with the standard HID/HOGP BLE application profile.
The left button is the ‘PUSH1’ button and the right button is the ‘PUSH2’ button.
The HID mouse device is named ‘STMouse’ in the central device list.
Mouse movements are provided by the 3D accelerometer and 3D gyroscope embedded in the BLE development platform.
13 BLE throughput demonstration application

The BLE throughput demonstration application provides some basic throughput demonstration applications to provide some reference figures regarding the achievable Bluetooth low energy data rate using the BlueNRG-LP device on the STEVAL-IDB011V1 development platform.

The Throughput Service contains two characteristics:

- the **TX** characteristic, with which the client can enable notifications; when the server has data to be sent, it sends notifications with the value of the **TX** characteristic.
- the **RX** characteristic is a writable characteristic; when the client has data to be sent to the server, it writes a value in this characteristic.

The device roles which can be selected are:
1. Server, which exposes the service with the **TX**, **RX** characteristics (Bluetooth LE peripheral device)
2. Client, which uses the service **TX**, **RX** characteristics (Bluetooth LE central device).

13.1 BLE throughput setup

To configure two BlueNRG-LP platforms with client and server roles, follow the procedure below.

**Step 1.** Program the client side on one BlueNRG-LP platform and reset it.
   The platform appears on the screen as a virtual COM port.

**Step 2.** Open the port in a serial terminal emulator.
   The required serial port baudrate is 921600.

**Step 3.** Program the server side on a second BlueNRG-LP platform and reset it.
   The platform appears on the screen as a virtual COM port.

**Step 4.** Open the port in a serial terminal emulator.
   The required serial port baudrate is 921600.
   The two platforms try to establish a connection. As soon as they get connected, the slave continuously sends notification of a characteristic to the client.

**Step 5.** Choose the desired link layer packet length: 27 (default), 100 or 251 (maximum allowed) bytes.

**Step 6.** Perform an ATT MTU exchange command on the server side to allow the server to increase the ATT_MTU size (247 bytes).

**Step 7.** Enable the bidirectional throughput on the client side (the client writes on the **RX** characteristic).

**Step 8.** Set the PHY to be used for the communication: 1 Mbps (default), 2 Mbps, etc.
13.2 BLE throughput server commands

After opening the Hyper Terminal (settings = 921600, 8, None,1, None), you can choose and press one of the following interactive options on the server side:

- **u** to send data len update request for 27 bytes
- **m** to send data len update request for 100 bytes
- **U** to send data len update request for 251 bytes
- **a** to send ATT_MTU exchange
- **l** to enable/disable L2CAP COS txing
- **z** to enable/disable slow throughput
- **1** to change L2C COS MTU value
- **2** to change L2C COS MPS value
- **c** to send connection parameter update request
- **f** to enable/disable flushable PDUs
- **e** to toggle notify
- **p** to print APP flags
- **s** to read LE PHY (TX, RX)
- **d** to set LE RX PHY to Coded
- **D** to set LE TX PHY to Coded
- **t** to set LE TX PHY to 1 Mbps
- **r** to set LE RX PHY to 1 Mpbs
- **T** to set LE TX PHY to 2 Mbps
- **R** to set LE RX PHY to 2 Mpbs
- **x** for system reset
- **?** to print help
13.3 BLE throughput client commands

After opening the Hyper Terminal (settings = 921600, 8, None, 1, None), you can choose and press one of the following interactive options on the client side:

- **u** to send data len update request for 27 bytes
- **m** to send data len update request for 100 bytes
- **U** to send data len update request for 251 bytes
- **a** to send ATT_MTU exchange
- **l** to enable/disable L2CAP COS txing
- **z** to enable/disable slow throughput
- **1** to change L2C COS MTU value
- **2** to change L2C COS MPS value
- **b** to switch bidirectional test on-off
- **n** to send notifications
- **i** to send indication
- **p** to print APP flags
- **s** to read LE PHY (TX, RX)
- **d** to set LE RX PHY to Coded
- **D** to set LE TX PHY to Coded
- **t** to set LE RX PHY to 1 Mbps
- **r** to set LE RX PHY to 1 Mbps
- **T** to set LE TX PHY to 2 Mbps
- **R** to set LE RX PHY to 2 Mbps
- **x** for System reset
- **?** to print help

*Note:* By default, the client and server configurations target a unidirectional throughput test: the server device sends characteristic notifications to the client device. The required serial port baud rate is 921600.
The BLE ANCS demonstration application configures a BlueNRG-LP device as a Bluetooth LE notification consumer, which facilitates Bluetooth accessory access to the many notifications generated on a notification provider.

After reset, the demo places the Bluetooth LE device in advertising with device name "ANCSdemo" and sets the device authentication requirements to enable bonding.

When the device is connected and bonded with a notification provider, the demo configures the BLE notification consumer device to discover the service and the characteristics of the notification provider. When the setup phase is complete, the Bluetooth LE device is configured as a notification consumer able to receive the notifications sent from the notification provider.

The BLE notification consumer demonstration application is supported by the BlueNRG-LP development platform (STEVAL-IDB011V1).
BLE security demonstration applications

The BLE Security demonstration applications are supported by the BlueNRG-LP development platform (STEVAL-IDB011V1). They provide some basic examples about how to configure two Bluetooth LE devices as Central and Peripheral, and set up a secure connection by performing a Bluetooth LE pairing procedure. Once paired, the two devices are also bonded.

The following pairing key generation methods are shown:
- PassKey entry with random pin
- PassKey entry with fixed pin
- Just works
- Numeric Comparison (new paring method supported only from the BlueNRG-LP BLE stack v3.x)

For each pairing key generation method, a specific project security configuration is provided for both Central and Peripheral devices as shown in the table below. Each Central and Peripheral device must be loaded with the application image targeting the proper security configuration to correctly demonstrate the associated Bluetooth LE security pairing functionality.

<table>
<thead>
<tr>
<th>Pairing key generation method</th>
<th>Central device security configuration</th>
<th>Peripheral device security configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>PassKey entry with random pin</td>
<td>Master_PassKey_Random</td>
<td>Slave_PassKey_Random</td>
</tr>
<tr>
<td>PassKey entry with fixed pin</td>
<td>Master_PassKey_Fixed</td>
<td>Slave_PassKey_Fixed</td>
</tr>
<tr>
<td>Just works</td>
<td>Master_JustWorks</td>
<td>Slave_JustWorks</td>
</tr>
<tr>
<td>Numeric Comparison</td>
<td>Master_NumericComp</td>
<td>Slave_NumericComp</td>
</tr>
</tbody>
</table>

15.1 Peripheral device

On reset, after initialization, the Peripheral device sets security IO capability and authentication requirements in order to address the selected pairing key generation method in combination with the related security settings of the Central device.

After initialization phase, the Peripheral device also defines a custom service with 2 proprietary characteristics (UUID 128 bits):
- TX characteristic: notification (CHAR_PROP_NOTIFY),
- RX characteristic with properties: read (CHAR_PROP_READ, GATT_NOTIFY_READ_REQ_AND_WAIT_FOR_APPL_RES (application is notified when a read request of any type is received for this attribute).

Based on the selected security configuration, the RX characteristic is defined with proper security permission (link must be "encrypted to read" on JustWorks method, link must be "encrypted to read and need authentication to read" on all other methods).

The Peripheral device enters Discovery mode with local name SlaveSec_Ax (x=0,1,2,3 depending on the selected security configuration).
Table 11. Peripheral device advertising local name parameter value

<table>
<thead>
<tr>
<th>Peripheral device configuration</th>
<th>Advertising local name</th>
<th>Pairing method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slave_JustWorks</td>
<td>SlaveSec_A0</td>
<td>Just works</td>
</tr>
<tr>
<td>Slave_PassKey_Fixed</td>
<td>SlaveSec_A1</td>
<td>PassKey entry with fixed pin</td>
</tr>
<tr>
<td>Slave_PassKey_Random</td>
<td>SlaveSec_A2</td>
<td>PassKey entry with random pin</td>
</tr>
<tr>
<td>Slave_NumericComp</td>
<td>SlaveSec_A3</td>
<td>Numeric Comparison</td>
</tr>
</tbody>
</table>

When a Central device starts the discovery procedure and detects the Peripheral device, the two devices connect.

After connection, the Peripheral device starts a slave security request to the Central device `aci_gap_slave_security_req()` and the Central device starts the pairing procedure.

Based on the pairing key generation method, the user may be prompted to perform certain actions (i.e., confirm the numeric value if the numeric comparison configuration is selected, add the key, displayed on Peripheral device, on Central hyper terminal, if the passkey entry with random pin configuration is selected).

After devices pair and are bonded, the Peripheral device displays the list of its bonded devices and adds the bonded Central device to its white list `aci_gap_configure_white_and_resolving_list()` API.

The Central device starts the service discovery procedure to identify the Peripheral service and characteristics and then enables the TX characteristic notification.

The Peripheral device starts TX characteristic notification to the Central device at periodic intervals and it provides the RX characteristic value to the Central device each time it reads it.

When connected, if user presses the Bluetooth LE platform button PUSH1, the Peripheral device disconnects and enters undirected connectable mode with advertising filter enabled (WHITE_LIST_FOR_ALL: Process scan and connection requests only from devices in the white list). This implies that the Peripheral device accepts connection requests only from devices on its white list: the Central device is still be able to connect to the Peripheral device; any other device connection requests are not accepted by the Peripheral device.

TX and RX characteristics length is 20 bytes and related values are defined as follow:

- TX characteristic value: `{S',L',A',V',E',_','S',E',C',U',R',I',T',Y','_','X','_','x1',x2}`; where x1, x2 are counter values

- RX characteristic value: `{S',L',A',V',E',_','S',E',C',U',R',I',T',Y','_','R','X','_','x1',x2}`; where x1, x2 are counter values

15.2 Central device

On reset, after initialization, the Central device uses the `Master_SecuritySet()` API for setting the security IO capability and authentication requirements to address the specific selected paring method in combination with the related security settings of the Central device. The Central device application uses the Central/Master library APIs and callbacks to perform the Central device Bluetooth LE operations (device discovery, connection, etc.).

The Central device starts a device discovery procedure (`Master_DeviceDiscovery()` API, looking for the associated Peripheral device `SlaveSec_Ax` (x= 0,1,2,3 : refer to Table 11. Peripheral device advertising local name parameter value).

When found, the Central connects to the Peripheral device. In order to start the pairing, the Central device waits for the Peripheral device to send a slave security request. Once the security request is received, the Central device starts the pairing procedure. Based on the pairing key generation method, the user may be asked to perform some actions (i.e. confirm the numeric value if the numeric comparison configuration is selected, add the key displayed on Peripheral device on Central hyper terminal if the passkey entry with random pin configuration is selected). Once the pairing and bonding procedure has completed, the Central device starts the service discovery procedure to determine the Peripheral TX & RX characteristics.

After Service Discovery, the Central device enables the TX characteristic notification. Then the Central device receives the TX characteristic notification value periodically from Peripheral device and reads the related RX characteristic value from the Peripheral device.
When connected, if the Bluetooth LE platform PUSH1 button is pressed, the Central device disconnects and reconnects with the Peripheral device, which enters undirected connectable mode with advertising filter enabled. Once connected to the Peripheral device, it enters the TX characteristic notification/RX characteristic read cycle again.

Note: When using a smart phone Central device that implements a random resolvable address, the Peripheral device is not able to accept connection or scan requests from it during the reconnection phase. This is due to the fact that, when disconnecting, the Peripheral device enters the undirected connectable mode with filtering enabled (WHITE_LIST_FOR_ALL: process scan and connection requests from the White List devices only). Therefore, it is only able to accept the smart phone scan or connection requests if the Privacy Controller is enabled on the Peripheral device.

A possible simple alternative is to replace the WHITE_LIST_FOR_ALL advertising filter policy on the Peripheral device with NO_WHITE_LIST_USE: the Peripheral device does not enable device filtering after reconnection and is able to accept connection or scan requests from a smart phone by using resolvable random addresses.
16 BLE Coded PHY demonstration application

This application demo shows how to control a remote device (like an actuator) using BlueNRG-LP and BLE specification v5.0 coded PHY feature.

The demo requires two BlueNRG-LP STEVAL-IDB011V1 boards configured, respectively, with client and server configurations.

The boards can be powered by USB cable or AAA batteries (after inserting the batteries, move jumper JP2 to BAT position).

The demonstration application defines two device roles:

- Client device role
- Server device role

16.1 Client and server demo application behavior

The server device starts by entering discoverable mode with a fixed address (0x0280E10000E1). LED2 (green) is blinking as the board is in advertising mode, using 1 Mbps PHY.

The client device (address = 0x0280E10000E0) starts by trying to connect to the server device. LED2 is blinking as the client is trying to connect using 1 Mbps PHY.

Once connected (LED2 always on, on both devices), the client writes into a characteristic to switch LED3 (blue) on and off on the server. The state of LED3 changes every 300 ms and on the client side when the write is sent to the server.

To measure the communication range, you have to place the two boards distant from each other until LED3 blinks on the server at a constant rate. When LED3 blinking rate is lower, several packets are retransmitted by the client and the boards are at the edge of a stable communication range.

While connected, you have to press PUSH1 button on one of the boards to change the used PHY to the coded PHY. When the procedure finishes (i.e. coded PHY activated), LED1 (red) is switched on.

You can measure the communication range when using the coded PHY by repeating the test above (placing the boards distant from each other until LED3 blinks on the server at a constant rate). An increased communication range (with respect to 1 Mbps PHY) should be visible.
17 BLE Controller Privacy demonstration application

This application provides a basic example of Bluetooth low energy controller privacy feature with Bluetooth LE master and slave devices.

17.1 Application scenario

The application scenario is based on two devices, master and slave, configured with `aci_gap_init(privacy flag = 0x02)`, which should perform the following macro steps:

1. Initially, master and slave devices have no info on their security database: the two devices should connect and make a pairing and bonding (fixed key: 123456).
2. Once the bonding is completed, the slave calls the `aci_gap_configure_white_and_resolving_list()` API to add its bonded device address to the controller’s white list.
3. The master device enables the slave characteristic notification. After the first connection and the pairing/bonding phase, devices disconnect.
4. The slave enters undirected connectable mode (`aci_gap_set_undirected_connectable()` API) with its own address type = resolvable address and white list = 0x03 as advertising filter policy.
5. The master device performs a direct connection to the detected slave device, which accepts the connection since the master address is on its white list: the two devices reconnect and the slave starts a notification cycle to the master.

Note: When the connection is established, if you press the BLE platform button PUSH1 on one of the two devices, it disconnects and the slave enters the undirected connectable mode with filtering enabled (`WHITE_LIST_FOR_ALL`). This implies that the slave device accepts connection requests only from devices on its white list: the master device is still able to connect to the slave device; any other device connection request is not accepted from the slave device.
18 BLE Multiple Connections demonstration application

This application provides a basic example of a multiple connection scenario. The Multiple Connections demonstration applications support three device roles:
1. Master_Slave device
2. Master/s device/s
3. Slave/s device/s

18.1 Application roles

The Master_Slave device role allows testing a multiple connection: simultaneous Master and Slave scenario. It configures the Master_Slave device as central and peripheral device and allows targeting a multiple connection scenario with a MasterSlave device (the collector) which can connect to a given number of peer devices as a master and to a given number of peer devices as a slave.

The Slave device role defines a Peripheral/GATT Server device which connects as slave to the Master_Slave device.

The Master device role defines a Central/GATT Client device which connects as master to the Master_Slave device.

18.2 How to run the application

The MasterSlave device automatically tries to connect to a set of known peer devices (the nodes), to which it is bonded.

Step 1. To bond with a new device, press PUSH1 button on the Slave device and PUSH1 button on the MasterSlave device. Once the two devices are connected, a bond is created: the MasterSlave tries to connect to the bonded device even after disconnection and the Slave device allows connection only from the bonded MasterSlave.

Step 2. To make the MasterSlave connectable as a slave to a master device (the inquirer), press PUSH2 button on the MasterSlave device.

Step 3. Press PUSH1 on the Master device to detect the MasterSlave device and connect to it.

Note: In this case, pairing is performed but no bond is created.

In this scenario, each Slave is a server and periodically sends data to the MasterSlave which acts as server and client. It periodically sends data to the Slaves as a client and to the Masters as a server. Each Master is a client and periodically sends data to the MasterSlave.

Step 4. Open a serial terminal on the associated COM port to show logs from the application.
The BLE power consumption demo application allows putting the selected BLE device in discovery mode: you can choose from a test menu which advertising interval to use (100 ms or 1000 ms). To measure the BlueNRG-LP current consumption, it is necessary to connect a DC power analyzer to the STEVAL-IDB011V1 kit platform JP1 connector (refer to Section 2.11 Current measurements). After that, you can set a connection up with another device configured as a master and measure the related power consumption.

The master role can be covered by another BlueNRG-LP kit platform configured with the DTM FW application (DTM_UART_WITH_UPDATER.hex) and running a specific script through the BlueNRG GUI or Script launcher PC applications available in the STSW-BNRGUI SW package.

In the BLE_Power_Consumption demo application project folder, two scripts are provided to configure the master device and create a connection with the BlueNRG-LP kit platform under test. The two scripts allow establishing a connection with 100 ms and 1000 ms as connection intervals, respectively.

The power consumption demo supports the following test commands:

- **f**: the device is in discoverable mode with a fast interval of 100 ms
- **s**: the device is in discoverable mode with a slow interval of 1000 ms
- **r**: to reset the BlueNRG-LP
- **?**: to display the help menu
20 BlueNRG-LP peripheral driver examples

The BlueNRG-LP DK SW package (STSW-BNRLP-DK) contains a set of peripheral driver examples demonstrating how to use the BlueNRG-LP device peripheral drivers (ADC, GPIOs, PC, RTC, SPI, Timers, UART, IWDG, etc.).

These examples are based on the BlueNRG-LP LL and HAL drivers available in the STSW-BNRLP-DK package folder: Drivers\Peripherals_Drivers.

Two peripheral driver frameworks are available:
1. LL (Low Level drivers)
2. HAL (Hardware Abstraction Layer)

The following section provides basic lists of the available BlueNRG-LP Peripheral drivers (LL and HAL). For further information on each demonstration application, refer to the related html documentation in the STSW-BNRLP-DK package in the Docs/BlueNRG-LP_Periph_LL_Examples and Docs/BlueNRG-LP_Periph_HAL_Examples folders.

Note: Specific examples for some IPs and device features (2.4 GHz radio proprietary solution, power save Modes, etc.) are provided in the Projects\Peripheral_Examples\Examples_MIX folder.

20.1 ADC examples

ADC battery sensor (LL/HAL) This example shows how to sample the supplied voltage using the ADC peripheral.

ADC temperature sensor (LL) This example shows how to sample the temperature using the internal temperature sensor.

ADC conversion sequence (LL) This example shows how to use the sequencer of the ADC peripheral to acquire samples from different sources.

ADC analog microphone (LL) This example shows how to use the ADC peripheral to interface an analog microphone.

20.2 BSP examples

BSP COM (LL) This example shows basic UART communication.

BSP LEDs, Buttons (LL) This example shows how to configure the EXTI and use GPIOs to toggle the user LEDs available on the board when a user button is pressed.

BSP Sensors (LL) This example shows how to configure the inertial and the pressure sensors.

20.3 CRC examples

CRC example (LL/HAL) This example shows how to configure the CRC calculation unit to compute a CRC code for a given data buffer, based on a fixed generator polynomial (default value 0x4C11DB7).

CRC User Defined Polynomial (LL/HAL) These examples show how to configure the CRC calculation based on a user-defined generating polynomial.

CRC Polynomial Update (MIX) This example shows how to use the CRC peripheral through the BLUENRG_LP CRC HAL and LL API (used for performance improvement).
20.4 DMA examples

DMA copy (LL/HAL)  This example shows how to use a DMA channel to transfer a word data buffer from an embedded SRAM to another one.

DMA RAM to RAM (MIX)  This example shows how to use a DMA to transfer a word data buffer from RAM to embedded SRAM through the BLUENRG_LP DMA HAL and LL API (used for performance improvement).

20.5 EXTI example

EXTI Toggle LED (LL)  This example shows how to configure the EXTI and use GPIOs to toggle the user LEDs.

20.6 Flash example

Erase Program (HAL)  This example shows how to use the basic Flash operations such as erase/write/read.

20.7 GPIO examples

GPIO Infinite LED Toggling (LL)  This example shows how to configure and use GPIOs to toggle the on-board user LEDs every 250 ms.

GPIO IO toggle (HAL)  This example shows how to configure and use GPIOs through the HAL API.

GPIO EXTI (HAL)  This example shows how to configure external interrupt lines.

20.8 HAL examples

HAL time base (HAL)  This example shows how to customize HAL using a general-purpose timer as main source of time base, in place of Systick.

HAL RTC Alarm (HAL)  This example shows how to customize HAL using RTC alarm as main source of time base, in place of Systick.

HAL TimeBase RTC WKUP (HAL)  This example shows how to customize HAL using RTC wakeup as main source of time base, in place of Systick.

20.9 I2C examples

I2C Two Boards DMA (LL/HAL)  This example shows how to handle I2C data buffer transmission/reception between two boards via DMA.

I2C Two Boards Polling (LL/HAL)  This example shows how to handle I2C data buffer transmission/reception between two boards in polling mode.

I2C Two Boards Restart Adv IT (HAL)  This example shows how to perform multiple I2C data buffer transmission/reception between two boards in interrupt mode and with restart condition.

I2C Two Boards Restart IT (LL/HAL)  This example shows how to handle single I2C data buffer transmission/reception between two boards in interrupt mode and with restart condition.
I2S examples

This example plays a pre-recorded audio frame, saved in the MCU Flash memory, via the ST350BW device soldered on the X-NUCLEO-CCA01M1 expansion board connected to a BlueNRG-LP evaluation kit.

This application is designed to configure the I2S peripheral on the master board to receive data from the slave board.

This application is designed to configure the I2S peripheral on the slave board to transmit data to the master board.

IWDG examples

This example shows how to configure the IWDG peripheral to ensure periodical counter update and generate an MCU IWDG reset when PUSH1 button is pressed.

This example shows how to periodically update the IWDG reload counter and simulate a software fault that generates an MCU IWDG reset after a preset amount of time.

This example shows how to handle the IWDG reload counter and simulate a software fault that generates an IWDG reset after a preset amount of time.

LPUART TX, RX (LL)

This example shows how to configure GPIO and LPUART peripherals to receive characters on LPUART RX. This example is based on the LPUART LL API. The peripheral is initialized using LL unitary service functions for optimization purposes (performance and size).

Micro MIX examples

This is an example for the basic 'BlueNRG-LP Hello World' application with crash info handling and device cut information. When you connect the BlueNRG-LP platform to a PC USB port and open a specific PC tool/program (like Tera Term), the "Hello World: BlueNRG-LP (x.x) is here!" message is displayed.

This test provides an example for the following BlueNRG-LP power save modes:

POWER_SAVE_MODE_SHUTDOWN

It turns the BlueNRG-LP device off. The only wakeup source is a low pulse on the RSTN pad (there is no smart power management as memory retention is not part of this power save level configuration).

POWER_SAVE_LEVEL_STOP_WITH_TIMER

It puts the BlueNRG-LP in deep sleep and the timer clock sources (LSI or LSE) remain running. Wakeup is possible from GPIOs (PA8 to PA11 and PB0 to PB7), RTC, IWDG, BlueCore and the HAL Virtual Timers.

POWER_SAVE_LEVEL_NOTIMER

It puts the BlueNRG-LP in deep sleep. All the peripherals and clock sources are turned off. Wakeup is possible only from GPIOs (PA8 to PA11 and PB0 to PB7).

POWER_SAVE_LEVEL_CPU_HALT

It puts the BlueNRG-LP in CPU halt. Only the CPU is halted. The rest of the chip continues running normally. The chip will wake up from any interrupt.
20.14  PDM demonstration application

PDM_DigitalMicrophone (LL)  This example shows how to use the PDM port to interface a digital MEMS microphone.

20.15  Public Key Accelerator (PKA) demonstration application

The BlueNRG-LP PKA demonstration application is supported by the BlueNRG-LP development platforms. It provides a basic example on how to use the available PKA driver APIs to perform a basic PKA processing and check the results.

The Public Key Accelerator (PKA) is a dedicated hardware block used for computation of cryptographic public key primitives related to elliptic curve cryptography (ECC).

Note: This peripheral is used by the BlueNRG-LP Bluetooth low energy stack during the security pairing procedure, so the user application must not use it in the meantime.

The PKA demonstration application performs the following steps:

1. Starting from the PKA known point on the PKS_SetData() ellipse with PKA_DATA_PCX, PKA_DATA_PCY and from a random generated key A, it performs a PKA process which generates a new point A on the ellipse.

2. The same process is repeated from a new generated random key B, leading to a new point B on the ellipse.

3. A new PKA process starts using the key A with the point B coordinates generating a new point C which is still on the same ellipse.

20.16  2.4 GHz radio proprietary MIX examples

The 2.4 GHz radio proprietary low level driver provides access to the BlueNRG-LP device 2.4 GHz radio to send and receive packets without using the Bluetooth link layer.

The available 2.4 GHz radio proprietary examples are:

AutomaticChMgm  TX example where the INC_CHAN Action Tag is used to automatically change the channel.

Beep  TX example where the device continuously sends a packet to three different channels.

BeepMultiState  TX example with multi-state functionality.

Serial port  Point-to-point communication generating a two-way Serial port.

Serial portEncrypt  As the previous example, but with the encryption enabled.

RemoteControl  A basic remote control scenario by pressing the PUSH1 button on the device makes toggle the LED1 on the receiver device.

Sleep  It demonstrates point-to-point communication with sleep management.

Sniffer  A sniffer application in a selected channel and a defined Network ID.

SnifferMultiState  A sniffer application with multi-state functionality.

StarNetwork  A star network example where a Master asks for packets to the slaves of the network.

Tx Rx  Point-to-point communication with computation of packet error rate (PER).

Throughput TX, RX  Throughput test example (unidirectional with one TX and one RX device, and bidirectional with two TX devices and one RX device)
OTA Client, Server  OTA firmware upgrade framework example based on 2.4 GHz radio proprietary low-level driver.

20.17 RCC examples

RCC Output System Clock on MCO (LL)  This example shows how to configure MCO pin (PA11) to output the system clock.

RCC HSE Startup test (MIX)  It shows how to perform HSE startup time measurement.

20.18 RNG examples

RNG Generate Random Numbers (LL)  This example shows how to configure RNG to generate 32-bit long random numbers. The peripheral initialization uses LL unitary service functions for optimization purposes (performance and size).

RNG Multi RNG (HAL)  This example shows how to configure the RNG using the HAL API. This example uses the RNG to generate 32-bit long random numbers.

20.19 RTC examples

RTC wakeup (LL)  This example shows how to configure GPIO, USART and RTC peripherals to set the power save level stop with timer.

RTC Alarm (LL/HAL)  Configuration of the RTC LL API to configure and generate an alarm using the RTC peripheral. In this example, the Time is set to 23:59:50 and the Alarm must be generated on 00:00:00.

RTC Autocalibration (HAL)  This example shows use of the LSE clock source auto-calibration to get a precise RTC clock.

RTC Calendar (LL/HAL)  This example shows the configuration of the calendar.

RTC Exit Standby with Wake-Up Timer (LL)  This example shows configuration of the RTC to wake up from Standby mode using the RTC Wakeup timer.

RTC Wakeup Calendar (LL)  This example shows configuration of the LL API to set the RTC calendar. When a reset occurs the RTC configuration is not lost.

20.20 SPI examples

SPI Half Duplex IT (LL)  This example shows how to configure GPIO and SPI peripherals to transmit bytes from an SPI Master device to an SPI Slave device in Interrupt mode.

SPI Half Duplex DMA (LL)  This example shows how to configure GPIO and SPI peripherals to transmit bytes from an SPI Master device to an SPI Slave device in DMA mode.

SPI Full Duplex IT Master/Slave (LL/HAL)  These examples show how to perform data buffer transmission and reception between two boards via SPI using Interrupt mode.

SPI Full Duplex DMA Master/Slave (LL/HAL)  These examples show how to perform data buffer transmission and reception between two boards via SPI using DMA.

SPI Full Duplex Polling Master/Slave (HAL)  These examples show how to perform data buffer transmission and reception between two boards via SPI using Polling mode.
20.21 TIM examples

TIM Break and Dead time Init (LL)  This example shows how to configure the TIM peripheral to generate three center-aligned PWM and complementary PWM signals, insert a defined dead time value, use the break feature and lock the break and dead-time configuration.

TIM_Frequency_Divider (LL/HAL)  The TIM1 peripheral is configured to generate an output signal with a frequency equal to the half of the input signal acquired.

TIM Input capture (LL/HAL/MIX)  This example shows how to use the TIM peripheral to measure a periodic signal frequency provided by an external signal generator.

TIM Output Compare Init (LL)  This example shows how to configure the TIM peripheral to generate an output waveform in different output compare modes.

TIM time base (LL)  This example shows how to configure the TIM peripheral to generate a time base.

TIM OC active (HAL)  This example shows how to use the TIM peripheral in Output Compare Active mode (when the counter matches the capture/compare register, the corresponding output pin is set to its active state).

TIM OC inactive (HAL)  This example shows how to use the TIM peripheral in Output Compare Inactive mode with the corresponding Interrupt requests for each channel.

TIM OC toggle (HAL)  This example shows how to use the TIM peripheral to generate four different signals at four different frequencies.

TIM PWM output (LL/HAL)  This example shows how to use the TIM peripheral in PWM (pulse width modulation) mode.

TIM PWM Train (LL)  Configuration of a timer to generate a fixed number of pulses in Output Compare mode.

TIM time base (HAL)  This example shows how to use the TIM peripheral to generate a time base of one second with the corresponding interrupt request.

TIM One Pulse (LL/HAL)  This example shows how to use the TIM peripheral to generate a single pulse when an external signal rising edge is received on the timer input pin.

TIM One Pulse TI2 Trigger (LL)  Configuration of a timer to generate a positive pulse in Output Compare mode with a length of tPULSE and after a delay of tDELAY.

20.22 UART examples

USART Rx IT (LL)  This example shows how to configure GPIO and USART peripherals to receive characters from a Hyper Terminal in Asynchronous mode using an interrupt.

USART Rx Cont IT (LL)  This example shows how to configure GPIO and USART peripheral to continuously receive characters from Hyper Terminal in Asynchronous mode using Interrupt mode.

USART Tx (LL)  This example shows how to configure GPIO and USART peripherals to send characters asynchronously to an Hyper Terminal in Polling mode. If the transfer cannot be completed within the allocated time, a timeout allows exiting from the sequence with a Timeout error code.
<table>
<thead>
<tr>
<th>Example Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USART Tx IT (LL)</td>
<td>This example shows how to configure GPIO and USART peripheral to send characters asynchronously to Hyper Terminal in Interrupt mode.</td>
</tr>
<tr>
<td>USART HyperTerminal DMA (HAL)</td>
<td>This example shows how to perform UART transmission (transmit/receive) in DMA mode between a board and a Hyper Terminal application.</td>
</tr>
<tr>
<td>USART HyperTerminal IT (HAL/MIX)</td>
<td>This example shows how to perform UART transmission (transmit/receive) in Interrupt mode between a board and a Hyper Terminal application.</td>
</tr>
<tr>
<td>USART printf (HAL)</td>
<td>This example shows how to perform re-routing of the C library printf function to the UART. The UART outputs a message on the Hyper Terminal.</td>
</tr>
<tr>
<td>USART Two Boards DMA (HAL)</td>
<td>This example shows how to perform UART transmission (transmit/receive) in DMA mode between two boards.</td>
</tr>
<tr>
<td>USART Two Boards IT (HAL)</td>
<td>This example shows how to perform UART transmission (transmit/receive) in Interrupt mode between two boards.</td>
</tr>
<tr>
<td>USART Two Boards Polling (HAL)</td>
<td>This example shows how to perform UART transmission (transmit/receive) in Polling mode between two boards.</td>
</tr>
<tr>
<td>USART HyperTerminal TxCB (MIX)</td>
<td>Use of a UART to transmit data (transmit/receive) between a board and the Hyper Terminal application both in Polling and Interrupt modes.</td>
</tr>
<tr>
<td>USART Tx Rx Hw Flow Control IT (LL/HAL)</td>
<td>This example shows how to configure GPIO and USART peripheral to send characters asynchronously to/from Hyper Terminal with hardware flow control mode in Interrupt mode.</td>
</tr>
</tbody>
</table>
Figure 23. STEVAL-IDB011V1 circuit schematic (1 of 3)
Figure 24. STEVAL-IDB011V1 circuit schematic (2 of 3)
Figure 25. STEVAL-IDB011V1 circuit schematic (3 of 3)
## Revision history

**Table 12. Document revision history**

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-Jul-2020</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>29-Jul-2020</td>
<td>2</td>
<td>Updated Figure 23. STEVAL-IDB011V1 circuit schematic (1 of 3).</td>
</tr>
</tbody>
</table>
Contents

1 Getting started ................................................................. 2
  1.1 Kit contents .................................................................. 2
  1.2 System requirements .................................................... 2
  1.3 BlueNRG-LP development kit setup ............................... 2

2 Hardware description ........................................................ 3
  2.1 STEVAL-IDB011V1 board overview ............................... 3
  2.2 BlueNRG-LP SoC connections ......................................... 4
  2.3 Power supply ................................................................ 5
  2.4 Jumpers ....................................................................... 6
  2.5 Sensors ........................................................................ 6
  2.6 Extension connector ..................................................... 6
  2.7 Push buttons .................................................................. 6
  2.8 LEDs ............................................................................ 6
  2.9 CMSIS-DAP and Virtual COM ........................................... 7
    2.9.1 Virtual COM port driver setup for Windows ................. 7
    2.9.2 System functionality checks ..................................... 7
    2.9.3 USB_CMSISDAP programming/debugging feature ...... 7
    2.9.4 USB_CMSISDAP firmware update .............................. 9
  2.10 BlueNRG-LP programming and debugging ....................... 10
  2.11 Current measurements .................................................. 10
  2.12 Hardware setup ............................................................. 11

3 BlueNRG-LP Navigator .......................................................... 12
  3.1 Demonstration applications ............................................. 12
  3.2 Basic examples ............................................................. 14
  3.3 BLE demonstration and test applications ......................... 15
  3.4 Peripheral driver examples ............................................. 16
  3.5 2.4 GHz radio proprietary examples ............................... 17
  3.6 Development kits .......................................................... 18
  3.7 Release Notes and License ............................................. 19
3.8 Documentation Index .......................................................... 20

4 BlueNRG-LP Radio Parameters Wizard ............................................ 21
  4.1 How to run the Radio Parameters Wizard ......................................... 21
  4.2 Main user interface window ..................................................... 21

5 Programming with the BlueNRG-LP system on chip ............................... 23
  5.1 Software directories ........................................................... 23

6 BLE beacon demonstration application ........................................... 24
  6.1 BLE Beacon application setup ................................................... 24
    6.1.1 Initialization ............................................................ 24
    6.1.2 Manufacturing data .................................................... 24
    6.1.3 Non-connectable mode .................................................. 24
    6.1.4 Extended advertising mode .......................................... 25
  6.2 BLE Beacon FreeRTOS example ................................................ 26

7 BLE Serial port demo application ................................................. 27
  7.1 Peripheral and central device setup ........................................... 27
    7.1.1 Initialization ............................................................ 27
    7.1.2 Add service and characteristics ...................................... 28
    7.1.3 Entering connectable mode ............................................ 28
    7.1.4 Connection with the central device .................................. 28

8 BLE Serial port master and slave demo application ................................ 30
  8.1 BLE Serial port master and slave roles ...................................... 30
    8.1.1 Initialization ............................................................ 30
    8.1.2 Add service and characteristics ...................................... 30
    8.1.3 Start discovery procedure ............................................ 31
    8.1.4 Enter connectable mode .............................................. 31
    8.1.5 Connection with serial port master and slave client device ......... 31

9 BLE remote control demo application ............................................. 33
  9.1 BLE remote control application setup ....................................... 33
    9.1.1 Initialization ............................................................ 33
    9.1.2 Define advertising data ............................................... 33
    9.1.3 Add service and characteristics .................................... 34
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.4 Connection with a Bluetooth LE Central device</td>
<td>34</td>
</tr>
<tr>
<td>10 BLE sensor profile demo</td>
<td>35</td>
</tr>
<tr>
<td>10.1 BlueNRG app for smartphones</td>
<td>35</td>
</tr>
<tr>
<td>10.2 BLE sensor profile demo: connection with a central device</td>
<td>36</td>
</tr>
<tr>
<td>10.2.1 Initialization</td>
<td>36</td>
</tr>
<tr>
<td>10.2.2 Add service and characteristics</td>
<td>36</td>
</tr>
<tr>
<td>10.2.3 Enter connectable mode</td>
<td>37</td>
</tr>
<tr>
<td>10.2.4 Connection with central device</td>
<td>37</td>
</tr>
<tr>
<td>11 BLE sensor profile center demo</td>
<td>38</td>
</tr>
<tr>
<td>12 BLE HID/HOGP demonstration applications</td>
<td>39</td>
</tr>
<tr>
<td>12.1 BLE HID/HOGP keyboard demonstration application</td>
<td>39</td>
</tr>
<tr>
<td>12.2 BLE HID/HOGP mouse demonstration application</td>
<td>39</td>
</tr>
<tr>
<td>13 BLE throughput demonstration application</td>
<td>40</td>
</tr>
<tr>
<td>13.1 BLE throughput setup</td>
<td>40</td>
</tr>
<tr>
<td>13.2 BLE throughput server commands</td>
<td>41</td>
</tr>
<tr>
<td>13.3 BLE throughput client commands</td>
<td>42</td>
</tr>
<tr>
<td>14 BLE notification consumer demonstration application</td>
<td>43</td>
</tr>
<tr>
<td>15 BLE security demonstration applications</td>
<td>44</td>
</tr>
<tr>
<td>15.1 Peripheral device</td>
<td>44</td>
</tr>
<tr>
<td>15.2 Central device</td>
<td>45</td>
</tr>
<tr>
<td>16 BLE Coded PHY demonstration application</td>
<td>47</td>
</tr>
<tr>
<td>16.1 Client and server demo application behavior</td>
<td>47</td>
</tr>
<tr>
<td>17 BLE Controller Privacy demonstration application</td>
<td>48</td>
</tr>
<tr>
<td>17.1 Application scenario</td>
<td>48</td>
</tr>
<tr>
<td>18 BLE Multiple Connections demonstration application</td>
<td>49</td>
</tr>
<tr>
<td>18.1 Application roles</td>
<td>49</td>
</tr>
<tr>
<td>18.2 How to run the application</td>
<td>49</td>
</tr>
<tr>
<td>19 BLE power consumption demo application</td>
<td>50</td>
</tr>
<tr>
<td>20 BlueNRG-LP peripheral driver examples</td>
<td>51</td>
</tr>
<tr>
<td>20.1 ADC examples</td>
<td>51</td>
</tr>
</tbody>
</table>
### List of figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>STEVAL-IDB011V1 development platform based on BlueNRG-LP.</td>
<td>1</td>
</tr>
<tr>
<td>Figure 2</td>
<td>STEVAL-IDB011V1 board components</td>
<td>3</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Windows Device Manager - CMSIS-DAP</td>
<td>7</td>
</tr>
<tr>
<td>Figure 4</td>
<td>ST IDB011VX mass storage device</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5</td>
<td>IAR EWARM project - Debugger option</td>
<td>8</td>
</tr>
<tr>
<td>Figure 6</td>
<td>KEIL µVision project - Debugger option</td>
<td>9</td>
</tr>
<tr>
<td>Figure 7</td>
<td>USB_CMSISDAP firmware - MAINTENANCE mass storage device</td>
<td>9</td>
</tr>
<tr>
<td>Figure 8</td>
<td>BlueNRG-LP Navigator</td>
<td>12</td>
</tr>
<tr>
<td>Figure 9</td>
<td>BlueNRG-LP Navigator - BLE Beacon application</td>
<td>13</td>
</tr>
<tr>
<td>Figure 10</td>
<td>BlueNRG-LP Navigator - BLE Beacon Flash programming</td>
<td>14</td>
</tr>
<tr>
<td>Figure 11</td>
<td>BLE Beacon documentation</td>
<td>14</td>
</tr>
<tr>
<td>Figure 12</td>
<td>BlueNRG-LP Navigator - BlueNRG-LP basic examples</td>
<td>15</td>
</tr>
<tr>
<td>Figure 13</td>
<td>BlueNRG-LP Navigator - BLE demonstration and test applications</td>
<td>16</td>
</tr>
<tr>
<td>Figure 14</td>
<td>BlueNRG-LP Navigator - Peripherals LL driver examples</td>
<td>17</td>
</tr>
<tr>
<td>Figure 15</td>
<td>BlueNRG-LP Navigator - 2.4 GHz radio proprietary examples</td>
<td>18</td>
</tr>
<tr>
<td>Figure 16</td>
<td>BlueNRG-LP Navigator - development kit components</td>
<td>19</td>
</tr>
<tr>
<td>Figure 17</td>
<td>BlueNRG-LP Navigator - development kit 3D view</td>
<td>19</td>
</tr>
<tr>
<td>Figure 18</td>
<td>BlueNRG-LP Radio Parameters Wizard - general configuration</td>
<td>21</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Bluetooth LE Serial port client</td>
<td>29</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Bluetooth LE Serial port server</td>
<td>29</td>
</tr>
<tr>
<td>Figure 21</td>
<td>BLE sensor demo GATT database</td>
<td>35</td>
</tr>
<tr>
<td>Figure 22</td>
<td>BlueNRG sensor app</td>
<td>36</td>
</tr>
<tr>
<td>Figure 23</td>
<td>STEVAL-IDB011V1 circuit schematic (1 of 3)</td>
<td>58</td>
</tr>
<tr>
<td>Figure 24</td>
<td>STEVAL-IDB011V1 circuit schematic (2 of 3)</td>
<td>59</td>
</tr>
<tr>
<td>Figure 25</td>
<td>STEVAL-IDB011V1 circuit schematic (3 of 3)</td>
<td>60</td>
</tr>
</tbody>
</table>
## List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>STEVAL-IDB011V1 component descriptions</td>
<td>4</td>
</tr>
<tr>
<td>Table 2</td>
<td>BlueNRG-LP pin description with board functions</td>
<td>4</td>
</tr>
<tr>
<td>Table 3</td>
<td>STEVAL-IDB011V1 kit platform power supply modes</td>
<td>6</td>
</tr>
<tr>
<td>Table 4</td>
<td>BlueNRG-LP kit platform jumpers</td>
<td>6</td>
</tr>
<tr>
<td>Table 5</td>
<td>External SWD and STEVAL-IDB011V1 pin connections</td>
<td>10</td>
</tr>
<tr>
<td>Table 6</td>
<td>BlueNRG-LP kit platform and user board pin connection</td>
<td>10</td>
</tr>
<tr>
<td>Table 7</td>
<td>BLE beacon application - manufacturing data advertising</td>
<td>24</td>
</tr>
<tr>
<td>Table 8</td>
<td>Serial port configuration</td>
<td>27</td>
</tr>
<tr>
<td>Table 9</td>
<td>Bluetooth LE remote advertising data</td>
<td>33</td>
</tr>
<tr>
<td>Table 10</td>
<td>BLE security demonstration applications - security configuration combinations</td>
<td>44</td>
</tr>
<tr>
<td>Table 11</td>
<td>Peripheral device advertising local name parameter value</td>
<td>45</td>
</tr>
<tr>
<td>Table 12</td>
<td>Document revision history</td>
<td>61</td>
</tr>
</tbody>
</table>