AN4820

Application note

BlueNRG-1 and BlueNRG-2 low power modes

Introduction

The BlueNRG-1 and BlueNRG-2 are very low power Bluetooth low energy (BLE) single-mode systems-on-chip, compliant with Bluetooth® specification. The architecture core is a Cortex-M0 32-bit.

This application note explains the low power modes for the BlueNRG-1 and BlueNRG-2 devices.

Note:

The contents of this document are valid for both BlueNRG-1 and BlueNRG-2 devices. Therefore, any reference to the BlueNRG-1 device is also valid for the BlueNRG-2 device. Specific differences have been highlighted whenever needed.

1 BlueNRG-1 and BlueNRG-2 HW low power modes

Three low power modes are provided by the BlueNRG-1 and BlueNRG-2 hardware to achieve the best compromise between low power consumption, short startup time and available wakeup sources:

- CPU-Halt mode
  - In this mode only the CPU is stopped. All the device peripherals continue to operate and they can wake the CPU up when an interrupt/event occurs. This is the lowest power save mode.
- Sleep mode
  - In this mode the CPU is stopped and all the peripherals are disabled. Only the low speed oscillator block and the external wakeup source block are running.
  - The wakeup source are the *wakeup timer* or the *IO9*, *IO10*, *IO11*, *IO12* and *IO13*.
  - When the wakeup is triggered by previous listed sources, the system reverts to the run mode with all the peripherals on. Exiting from the sleep mode, the application needs to wait until the high speed oscillator is stable.
- Standby mode
  - In this mode the CPU is stopped and all the peripherals are disabled. The only wakeup source are *IO9*, *IO10*, *IO11*, *IO12* and *IO13*.
  - Exiting from the standby mode, the application needs to wait until the high speed and low speed oscillators are stable. This mode is the highest power save mode.

Refer to BlueNRG-1 and BlueNRG-2 datasheets for current consumption related to all the low power modes.

2 BlueNRG-1 and BlueNRG-2 low power mode support

The BlueNRG-1_2 DK software package provides software to support all the BlueNRG-1 and BlueNRG-2 hardware low power modes.

The low power software combines the low power requests coming from the application with the radio operating mode, choosing the best low power mode applicable in the current scenario.

This negotiation between the radio module and the application requests prevents data loss and is performed by the low power software.

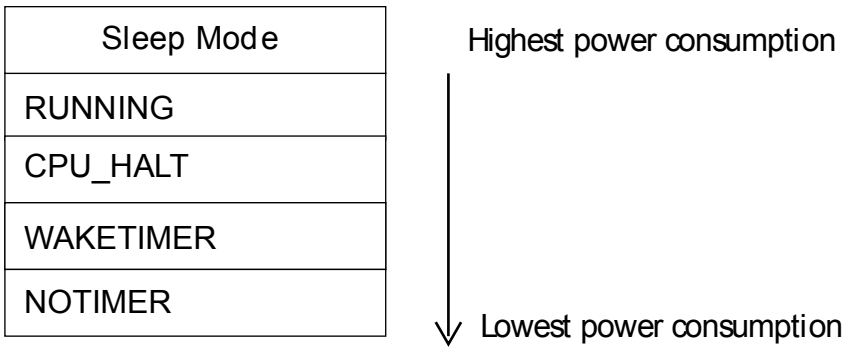
When the BlueNRG-1 and BlueNRG-2 exit from any low power mode a reset occurs: all the peripheral configuration and the application context are lost.

The low power software implements a mechanism to save and restore all the peripheral configuration and the application context when a power save procedure is called. So, from the application point of view, the exit from a low power procedure is fully transparent: when a wakeup from low power occurs, the CPU executes the next instruction after the low power function call.

The low power software implements the following power save modes:

- **SLEEPMODE_RUNNING**
  - In this low power mode, everything is active and running. In practice this mode is not used, but it is defined for completeness of information: it is not a real power save mode.
- **SLEEPMODE_CPU_HALT**
  - This power save mode implements the *HW CPU-Halt* low power mode.
- **SLEEPMODE_WAKETIMER**
  - This power save mode implements the *HW Sleep* low power mode.
- **SLEEPMODE_NOTIMER**
  - This power save mode implements the *HW Standby* low power mode.

Figure 1. BlueNRG-1 and BlueNRG-2 power save modes



To enable any low power mode the application calls the function “BlueNRG_Sleep()”.

```
uint8_t BlueNRG_Sleep(SleepModes sleepMode,
                      uint8_t gpioWakeBitMask,
                      uint8_t gpioWakeLevelMask);
```

Where:

- **sleepMode** is the low power mode to enable:
  - SLEEPMODE_RUNNING
  - SLEEPMODE_CPU_HALT
  - SLEEPMODE_WAKETIMER
  - SLEEPMODE_NOTIMER
- **gpioWakeBitMask** is a bit mask of the IOs that can wake the chip from low power mode:
  - WAKEUP_IO9
  - WAKEUP_IO10
  - WAKEUP_IO11
  - WAKEUP_IO12
  - WAKEUP_IO13

  If this field is zero the **gpioWakeLevelMask** parameter is ignored.
- **gpioWakeLevelMask** is a bit mask used to set up the active wakeup level for each wakeup source:
  - WAKEUP_IOx_LOW, the system wakes up when IO is low
  - WAKEUP_IOx_HIGH, the system wakes up when IO is high (to be not used for IO9,IO10, IO11)

Note: IO9, IO10 and IO11 are sensitive only to low level as they have an internal pull-up (activated by default). On the BlueNRG-2, user can disable the IO internal pull-up, nevertheless at each sleep sequence (enter sleep and wakeup from sleep), the internal GPIO pull-up is temporarily activated beyond the selected configuration, leading to a temporary spike. As a consequence on both BlueNRG-1 and BlueNRG-2 devices, it is strongly recommended to use these IOs as wake-up sources set at low level (system wakes up when IO is low).

The function returns the status:

- **BLUENRG_SLEEP_CONFIGURATION_ERROR**
- **SUCCESS**

The low power software exports other functions useful for the application:

- **BlueNRG_WakeupSource**
  - This function returns the last wakeup source from the low power mode.
- **App_SleepMode_Check()**
  - This function allows the application to set its desired sleep mode based on the application power management policy. When the user calls `BlueNRG_Sleep()`, a negotiation occurs to define the sleep mode and to get specific inputs from the application `App_SleepMode_Check()` allows overriding the sleep mode parameter passed by `BlueNRG_Sleep()`.

    This function is executed with interrupt masks (using `PRIMASK`): no interrupt handler can execute and change the application state and, potentially, its desired sleep mode. Therefore, when this function is called, the application software state is frozen.

    This function could be used in the example below:

```
/* Step 1. Application computes its desired sleep mode */
        sleepMode = computeSleepMode();
       /* Step 2. Application calls BlueNRG_Sleep with its desired sleep mode */
       BlueNRG_Sleep(sleepMode,...);
```

    If an interrupt occurs between step 1 and step 2, it changes the software state and `computeSleepMode` returns a different value, there is no way for the application to change its power management policy.

    If the application repeats the check done in step 1 inside the `App_SleepMode_Check()`, any change between step 1 and step 2 is taken into account.

    The key difference is that the second check is performed inside `BlueNRG_Sleep()` with interrupt masks and no change in the application software state occurs before entering the computed sleep mode.

    Any interrupt occurring after the interrupt mask instruction inside the `BlueNRG_Sleep()` and the call to WFI instruction will cause WFI behaving as NOP and the system will skip low power mode activation, thus serving the interrupt once re-enabled.

If the application wants to use the sleep mode (SLEEPMODE_WAKETIMER or SLEEPMODE_NOTIMER) and starts the sleep procedure with a wakeup source already active, the system does not switch the power management off and remains blocked in the WFI instruction until an interrupt occurs. In this context, if other wakeup sources, set in the `BlueNRG_Sleep()` API, occur, they will be ignored. The only way to restart the code execution is through an interrupt occurring from any peripheral.

In addition to user specific needs, it is mandatory to enable, at NVIC level and GPIO level, the interrupt associated with the GPIO wakeup sources as follows:

- If wakeup source level is LOW, the associated IRQ level must be on falling edge (IRQ_ON_FALLING_EDGE)
- If wakeup source level is HIGH, the associated IRQ level must be on rising edge (IRQ_ON_RISING_EDGE)

The GPIO handler should also handle the interrupt associated with the edges, by clearing the interrupt cause.

If the application needs also GPIO interrupt on the other edge, it could enable both edges and discard the edge not needed in the GPIO handler.

Example: let us assume the wake-up source is IO13 with low level and no specific interrupt is enabled and associated to this IO at application level.

User must enable, at NVIC level, the interrupt on IO13 with IRQ level on falling edge. A typical source code is as follows:

```
while (1) {

            BTLE_StackTick();
            wakeup_source = WAKEUP_IO13;
            wakeup_level = (WAKEUP_IO_LOW << WAKEUP_IO13);
            <enable IO13 interrupt with IRQ level on falling edge since wakeup
            level is LOW(IRQ_ON_FALLING_EDGE)>
            BlueNRG_Sleep(SLEEPMODE_NOTIMER, wakeup_source,wakeup_level);
        }
```

3 BlueNRG-1 and BlueNRG-2 low power mode examples

Some low power modes application use cases are provided in the following sections.

3.1 Low power SLEEPMODE_RUNNING

In this low power mode, everything is active and running: it is not a real low power mode. A typical source code is:

```
 while(1) /*main loop*/
{
 /* BLE Stack Tick */
 BTLE_StackTick();
 /* Application Tick */
 APP_Tick();
 /* Power Save management */
BlueNRG_Sleep(SLEEPMODE_RUNNING, 0, 0);
}
```

This low power mode does not need wakeup sources.

3.2 Low power SLEEPMODE_CPU_HALT

In this mode only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt occurs. A typical source code is:

```
while(1) /*main loop*/
{
 /* BLE Stack Tick */
 BTLE_StackTick();
 /* Application Tick */
 APP_Tick();
 /* Power Save management */
BlueNRG_Sleep(SLEEPMODE_CPU_HALT, 0, 0);
 }
```

In this low power mode the only wakeup sources are the peripheral interrupts.

If the radio operating mode does not allow this low power request, as it is executing a non-stoppable operation, the low power mode is converted automatically inside the low power software in SLEEPMODE_RUNNING.

3.3 Low power SLEEPMODE_WAKETIMER

In this mode the CPU is stopped and all the peripherals are disabled. Only the low speed oscillator block and the external wakeup source block are running.

A typical source code is:

```
/* Starts the VTimer 0 with timeout 2 sec. */
HAL_VTimerStart_ms(0, 2000);
while(1) /*main loop*/
{
 /* BLE Stack Tick */
 BTLE_StackTick();
 /* Application Tick */
 APP_Tick();
 /* Power Save management */
 BlueNRG_Sleep(SLEEPMODE_WAKETIMER,
               WAKEUP_IO13,
               WAKEUP_IOx_LOW<< WAKEUP_IO13_SHIFT_MASK);
}
```

The wakeup source is the IO13 with low level sensitive; the application sets a virtual timer to wake up the system when the 2 sec. timeout occurs.

In this scenario, the application enables the low power mode *WAKETIMER*. If the radio operating mode is in connection or advertising state, the stack accepts the low power mode proposed by the application, but, if necessary, the system can be woken up before the application timeout to follow the connection interval time profile or the advertising interval time profile.

3.4 Low power SLEEPMODE_NOTIMER

In this low power mode the CPU is stopped and all the peripherals are disabled. Only the external wakeup source block is running. A typical source code is:

```
while(1) /*main loop*/
{
 /* BLE Stack Tick */
 BTLE_StackTick();
 /* Application Tick */
 APP_Tick();
 /* Power Save management */
BlueNRG_Sleep(SLEEPMODE_NOTIMER,
              WAKEUP_IO13,
              WAKEUP_IOx_HIGH << WAKEUP_IO13_SHIFT_MASK);
}
```

The wakeup source is the IO13 with high level sensitive.

In this scenario the application enables the low power mode *NOTIMER*. If the radio module is in connection state, after the negotiation with the radio stack, the low power software changes the low power mode into *WAKETIMER* to follow the connection time profile. Instead, if the radio module is in an idle state, the radio stack accepts the low power mode *NOTIMER* requested from the application and the low power software does not change it.

4 Current consumption measurement test scenarios during BLE

This section shows typical current consumption measurements during BLE advertising and connection times.

To take these measurements the following configuration has been used:

- **Hardware**: STEVAL-IDB007V1
- **Firmware**: BLE_Power_Consumption test application released in the BlueNRG-1_2 DK software package
- **Power supply**: 3.3 V
- **Tx Power**: -2 dBm
- **Retention**: full RAM retention 24 KB
- **Crystal Startup time**: 642 µs
- **Master SCA:** 100 ppm
- **Slave SCA:** 100 ppm
- **Advertising interval**: 100 ms and 1000 ms, with 28 byte packets length
- **Connection interval**: 100 ms and 1000 ms, with empty packets

The BLE_Power_Consumption test application configures the BlueNRG-1 as a Bluetooth Low Energy (BLE) peripheral.

It is included in the BlueNRG-1_2 DK software package (STSW-BLUENRG1-DK), in Project, BLE_Examples folder.

The BLE master role is covered by another BlueNRG-1 device configured with the DTM firmware application available in the same folder and running some master scripts provided in the BLE_Power_Consumption test application folder.

The scripts allow configuring a BlueNRG-1 device as a BLE master and create a connection with the BlueNRG-1 under test. The scripts run using the BlueNRG-GUI available in the BlueNRG GUI SW package (STSW-BNRGUI).

The current consumption measured in these scenarios is related only to the BlueNRG-1. The values are measured connecting a DC power analyzer to the STEVAL-IDB007V1 JP4 probe.

Table 1. BlueNRG-1 current consumption values

BLE scenario	Real current consumption	Estimated current consumption
Advertising interval 100 ms	136.931 µA	181.8 µA
Advertising interval 1000 ms	15.342 µA	18.99 µA
Connection interval 100 ms	50.423 µA	48.81 µA
Connection interval 1000 ms	7.059 µA	6.95 µA

The third column values are calculated using the BlueNRG Current Consumption Estimation Tool (STSW-BNRG001).

The following pictures show the advertising procedure snapshots for both intervals (the packet length is 28 bytes).

Figure 2. BlueNRG-1 advertising procedure: interval 100 ms

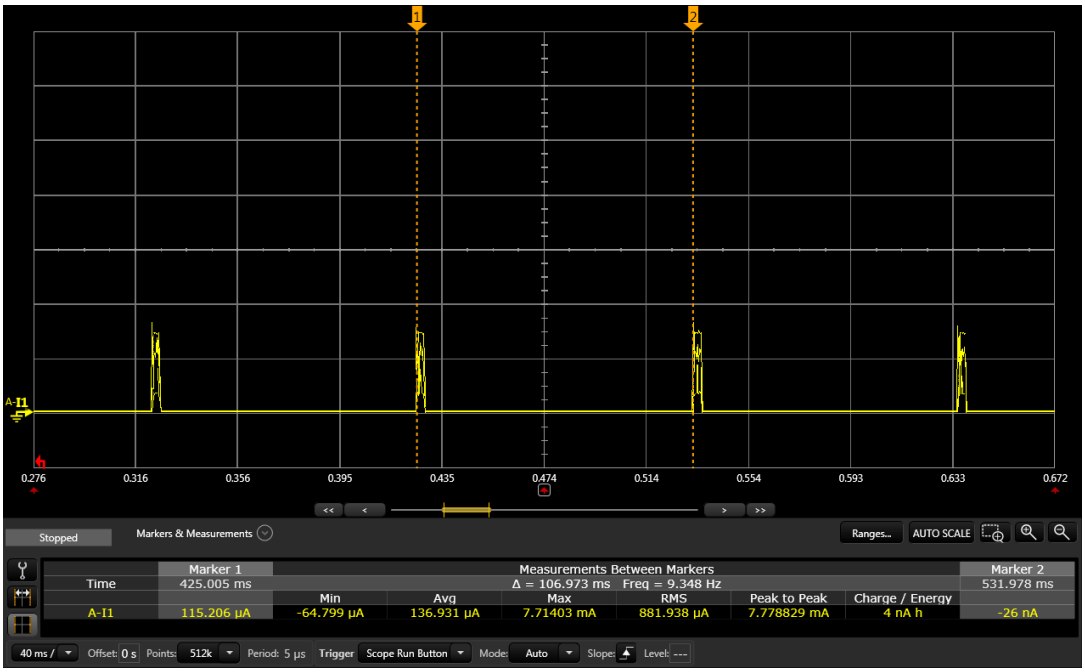
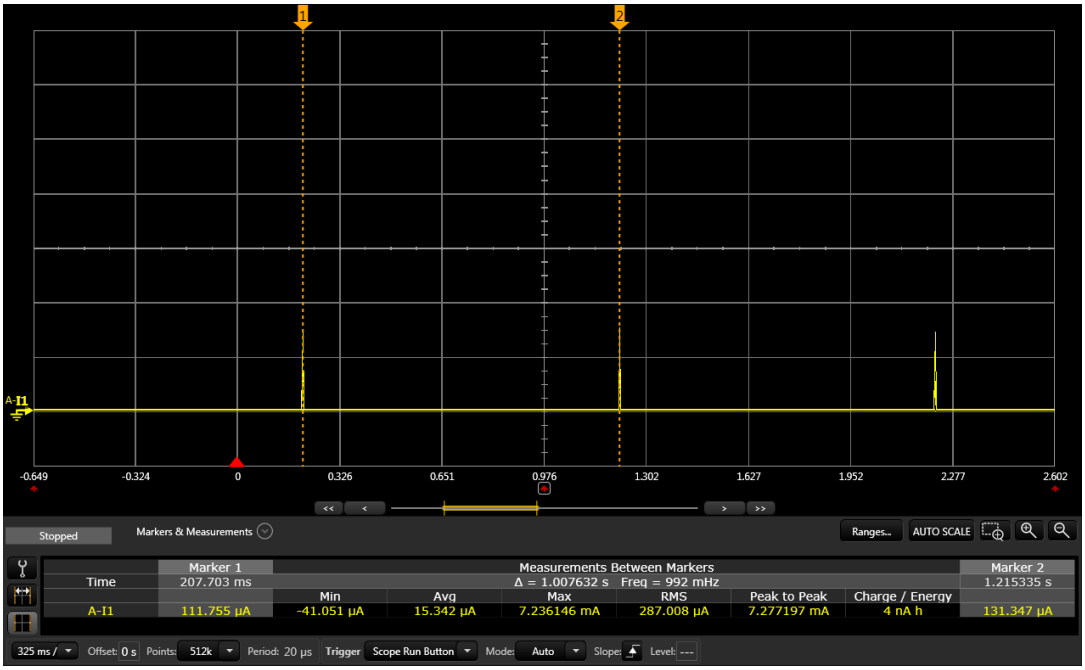


Figure 3. BlueNRG-1 advertising procedure: interval 1000 ms



The following pictures show the connection procedure snapshots for both intervals (with empty packet).

Figure 4. **BlueNRG-1 connection procedure: interval 100 ms**

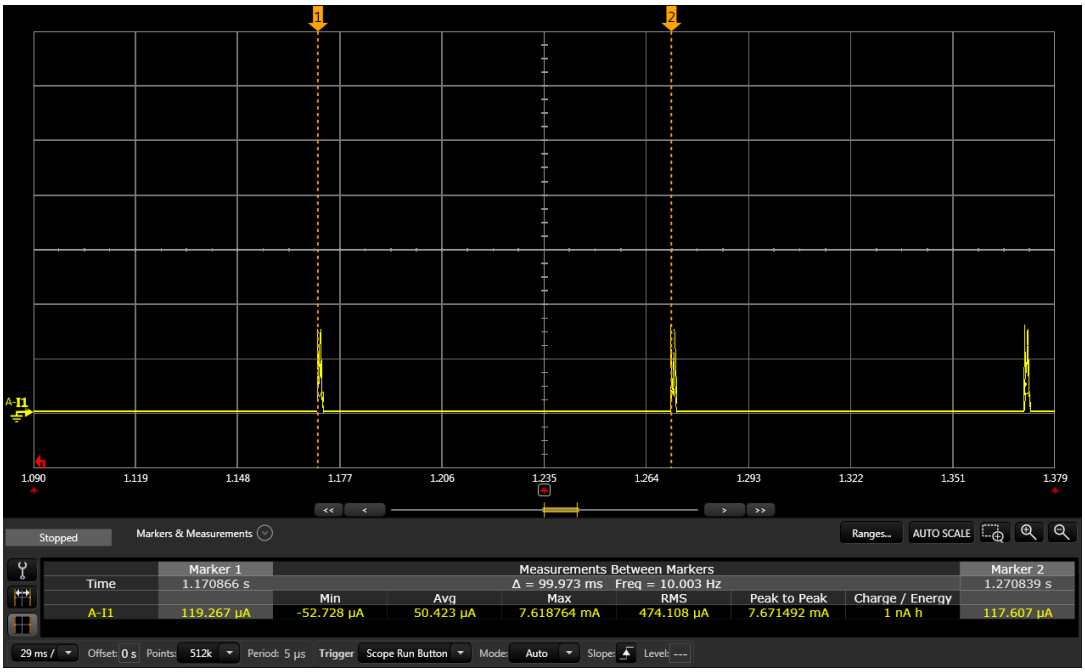
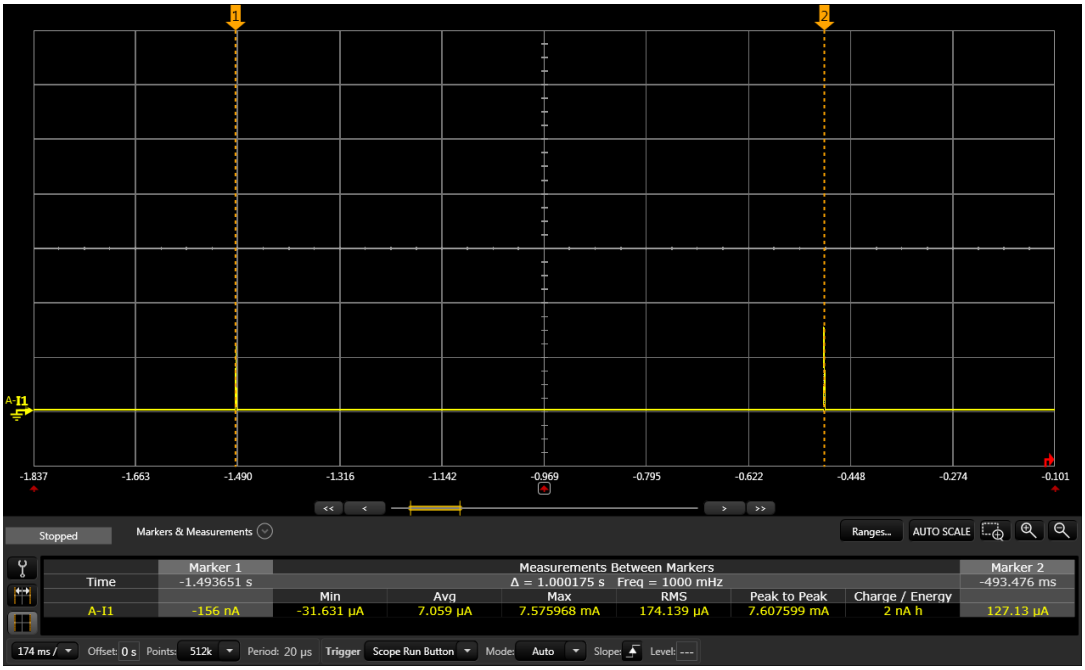


Figure 5. **BlueNRG-1 connection procedure: interval 1000 ms**



The following pictures compare the the DC power analyzer current waveform measured during the advertising procedure and the estimated waveform calculated via the BlueNRG Current Consumption Estimation Tool (1000 ms advertising interval and 28 bytes packet length).

Figure 6. BlueNRG Current Consumption Estimation Tool: advertising scenario power estimation

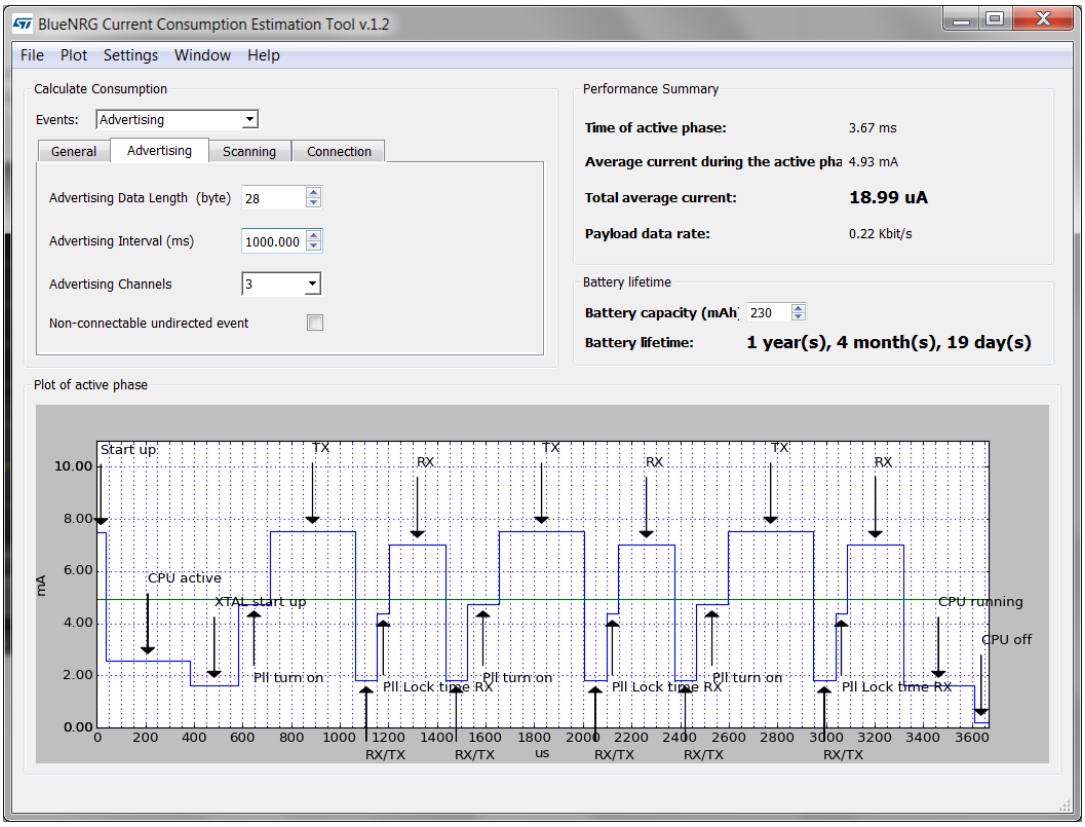
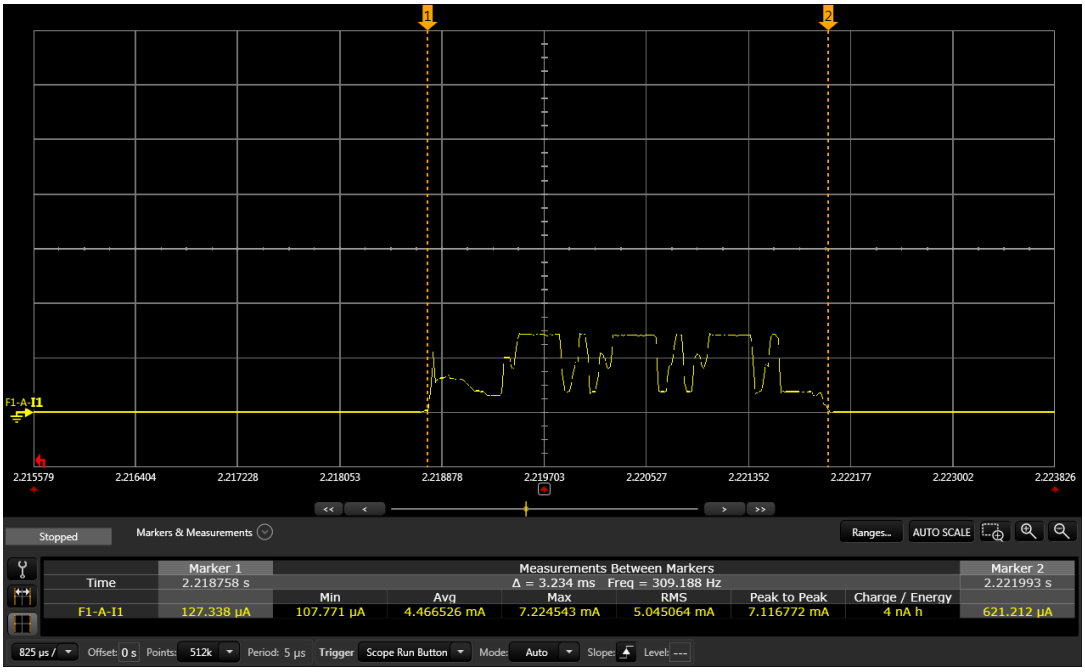


Figure 7. BlueNRG-1 advertising scenario power consumption



Revision history

Table 2. **Document revision history**

Date	Version	Changes
29-Jun-2016	1	Initial release.
12-Jul-2017	2	Added references to BlueNRG-2 throughout text. Updated: Introduction, Section 2: "BlueNRG-1 and BlueNRG-2 low power mode support", Section 3.1: "Low power SLEEPMODE_RUNNING", Section 3.2: "Low power SLEEPMODE_CPU_HALT", Section 3.3: "Low power SLEEPMODE_WAKETIMER" and Section 3.4: "Low power SLEEPMODE_NOTIMER". Added: Section 4: "Current consumption measurement test scenarios during BLE"
17-Dec-2018	3	Updated Section 2 BlueNRG-1 and BlueNRG-2 low power mode support.
20-Nov-2019	4	Updated Section 2 BlueNRG-1 and BlueNRG-2 low power mode support.

Contents