life.augmented

# AN5466

Application note

# BlueNRG-LP power save modes

## Introduction

The BlueNRG-LP is a very low power Bluetooth Low Energy (LE) single-mode system-on-chip, compliant with Bluetooth® specification. The architecture core is a Cortex-M0 32-bit.

This application note explains the power save modes for the BlueNRG-LP device.

# 1 The BlueNRG-LP HW power save modes

Two main power save modes are provided by the BlueNRG-LP hardware to achieve the best compromise between low power consumption, short start-up time and available wake-up sources:

- Deepstop mode
  - The system and the bus clocks are stopped
  - Only the essential digital power domain is ON and supplied at 1.0 V
  - The bank RAM0 is kept in retention
  - The other banks of RAM can be in retention or not, depending on the software configuration
  - The low speed clock can run or stop, depending on the software configuration
    - ON or OFF
    - Sourced by LSE or by LSI
  - The RTC and the IWDG stay active if enabled and the low speed clock is ON
  - The radio wake-up block including its timer stay active (if enabled and the low speed clock is ON)
  - Wakeup is possible from GPIOs only (PA0 to PA15 and PB0 to PB11) if the low speed clock is OFF, plus the RTC, IWDG, radio and Hal Virtual Timers if the low speed clock is ON
  - When the wakeup is triggered by previous listed sources, the system reverts to the run mode with all the peripherals on. Exiting from the deepstop mode, the application needs to wait until the high speed oscillator is stable
- Shutdown mode
  - Shutdown mode is the least power consuming mode. In shutdown mode, the BlueNRG-LP is in ultra-low power consumption: all voltage regulators, clocks and the RF interface are not powered.
  - The BlueNRG-LP can enter shutdown mode by internal software sequence. The only way to exit shutdown mode is by asserting and de-asserting the RESET pin

Refer to BlueNRG-LP datasheet for current consumption related to all the power save modes.

# 2 The BlueNRG-LP power save mode support

The BlueNRG-LP DK software package provides software to support all the BlueNRG-LP hardware power save modes.

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

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

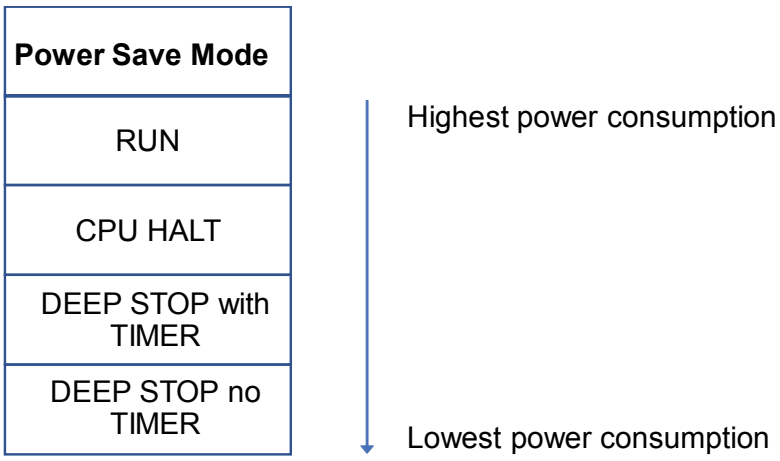
When the BlueNRG-LP exits from any power save mode, a reset occurs: all the peripheral configurations and the application contexts are lost.

The power save software implements a mechanism to save and restore all the peripheral configurations and the application contexts 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 power save occurs, the CPU executes the next instruction after the power save function call.

The power save software implements the following power save modes:

- POWER_SAVE_LEVEL_RUNNING
  - In this power save mode, everything is active and running. This mode is not used, but it is defined to give all information: it is not a real power save mode.
- POWER_SAVE_LEVEL_CPU_HALT
  - This power save mode implements the HW CPU-Halt power save mode (only the CPU is halted)
- POWER_SAVE_LEVEL_STOP_WITH_TIMER
  - This power save mode implements the HW deepstop mode with the low speed clock ON. The device is in deep sleep and the timer clock sources (LSI or LSE) remain running. Wakeup is possible from GPIOs (PA0 to PA15 and PB0 to PB11), RTC, IWDG, Radio and the Hal Virtual Timers.
- POWER_SAVE_LEVEL_NOTIMER
  - This power save mode implements the HW deepstop mode with the low speed clock OFF
  - The device is in deep sleep. All the peripherals and clock sources are turned off
  - Wakeup is possible only from GPIOs (PA0 to PA15 and PB0 to PB11)

**Figure 1. BlueNRG-LP power save modes**




To enable any power save mode the application calls the function `HAL_PWR_MNGR_Request()`:

```
uint8_t HAL_PWR_MNGR_Request(PowerSaveLevels level,
                             WakeupSourceConfig_TypeDef wsConfig,
                             PowerSaveLevels *negotiatedLevel)
```

where

- **level** is the power save mode to enable:
  - POWER_SAVE_LEVEL_RUNNING
  - POWER_SAVE_LEVEL_CPU_HALT
  - POWER_SAVE_LEVEL_STOP_WITH_TIMER
  - POWER_SAVE_LEVEL_STOP_NOTIMER
- **wsConfig** is the Wakeup Source Configuration
  - it specifies whether RTC is ON or OFF
  - it specifies which IOs are configured to wake up and exit from STOP level with low polarity (if not used, the parameter must be initialized to zero)
  - it specifies which IOs are configured to wake up and exit from STOP level with high polarity (if not used, parameter must be initialized to zero)
- **negotiatedLevel** returns the negotiated power save level really applied:
  - POWER_SAVE_LEVEL_RUNNING
  - POWER_SAVE_LEVEL_CPU_HALT
  - POWER_SAVE_LEVEL_STOP_WITH_TIMER
  - POWER_SAVE_LEVEL_STOP_NOTIMER

The function returns the status:

- **ERROR code**
- **SUCCESS**

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

- **HAL_PWR_MNGR_WakeupSource**
  - This function returns the last wake-up source from the power save mode (the return value can be a combination of the following values: PA0 - PA15, PB0 - PB11, RTC, radio, TIMER)

- **App_PowerSaveLevel_Check(PowerSaveLevels level))**
  - This function allows the application to set its desired sleep mode based on the application power management policy. When the user calls `HAL_PWR_MNGR_Request()`, a negotiation occurs to define the power save mode and to obtain specific inputs from the application. `App_PowerSaveLevel_Check()` allows the power save mode parameter passed by `HAL_PWR_MNGR_Request()` to be overriden.

    This function is executed with interrupts disabled: no interrupt handler can execute and change the application state and, potentially, its desired power save 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 power save mode */
powerSaveMode = computePowerSaveMode();
/* Step 2. Application calls HAL_PWR_MNGR_Request with its desired sleep mode */
HAL_PWR_MNGR_Request (powerSaveMode,...);
```

    If an interrupt occurs between step 1 and step 2, it changes the software state and `computePowerSaveMode` 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_PowerSaveLevel_Check()`, any change between step 1 and step 2 is taken into account.

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

    Any interrupt occurring after the interrupt mask instruction inside the `HAL_PWR_MNGR_Request()` and the call to WFI instruction causes WFI to behave as NOP and the system skips power save mode activation, thus serving the interrupt once re-enabled.
- **HAL_PWR_MNGR_ShutdownRequest(uint8_t BOR_enabled)**
  - This function allows the BlueNRG-LP to be put into shutdown mode: the only wake-up source is a low pulse on the RSTN pad
  - The conditions to enter shutdown mode are:
    - the radio is sleeping
    - the CPU is sleeping (WFI with SLEEPDEEP information active)
    - A shutdown exit is similar to a POR startup of the board. The associated reset reason is the PORRSTF flag
    - The BOR feature may be enabled or disabled during shutdown mode

## 2.1 GPIO interrupt and wake-up source handling

When in deepstop mode, the BlueNRG-LP GPIO peripherals are disabled.

So, if a signal variation happens in an IO wake-up pin, the device is awakened but the associated interrupt is lost since the peripheral is not clocked and configured before the context restore.

In order to properly serve the interrupt, the application must also define the `HAL_PWR_MNGR_WakeupIOCallback()` callback with the user specific code associated to the interrupt.

Example: let us assume the wake-up source is PA10 and the user application requires to perform specific operations on associated IRQ function (`GPIOA_IRQHandler()`).

On the main application the user enables the power save mode by setting PA10 as wake-up source:

```
static WakeupSourceConfig_TypeDef wakeupIO = {0,0, WAKEUP_PA10};
```

On the related PA10 IRQ handler (`GPIOA_IRQHandler(void)`), the user adds the application specific code allowing the interrupt to be handled:

```
/* GPIOA_IRQHandler */
void GPIOA_IRQHandler(void)
{
    <if PA10 pending bit is set>
    {
        <clear pending bit>

        /* Run the set of user actions associated to the interrupt */
        <DO_USER_SPECIFIC_ACTION>
    }
}
```

Since PA10 is also a wake-up source with associated interrupt, in order to properly handle the case when interrupt occurs during the device deepstop mode, the user must also define the following callback to permit the device to correctly perform the requested user specific actions associated with the interrupt:

```
/* User callback to be called if an interrupt is associated to a wakeup source */
void HAL_PWR_MNGR_WakeupIOCallback(uint16_t source)
{
  if (source & WAKEUP_PA10)
  {
    /* Run the set of user actions associated to the interrupt */
    <DO_USER_SPECIFIC_ACTION>
   }
 }
```

# 3 BlueNRG-LP power save mode examples

Some power save mode application use cases are provided in the following sections.

## 3.1 Power save POWER_SAVE_LEVEL_RUNNING

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

```
WakeupSourceConfig_TypeDef wakeupIO;
PowerSaveLevels stopLevel;

/* No Wakeup Source needed */
wakeupIO.IO_Mask_High_polarity = 0;
wakeupIO.IO_Mask_Low_polarity = 0;
wakeupIO.RTC_enable = 0;
while(1) /*main loop*/
{
   /* Modules ticks */
   ModulesTick();
   /* Application Tick */ APP_Tick();
   /* Power Save management */
   HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_RUNNING, wakeupIO, &stopLevel);
}
```

This power save mode does not need wake-up sources.

## 3.2 Power save POWER_SAVE_LEVEL_CPU_HALT

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

```
WakeupSourceConfig_TypeDef wakeupIO;
PowerSaveLevels stopLevel;

/* No Wakeup Source needed */
wakeupIO.IO_Mask_High_polarity = 0;
wakeupIO.IO_Mask_Low_polarity = 0;
wakeupIO.RTC_enable = 0;
while(1) /*main loop*/
{
   /* Modules ticks */
   ModulesTick();
   /* Application Tick */ APP_Tick();
   /* Power Save management */
   HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_CPU_HALT, wakeupIO, &stopLevel);
}
```

In this power save mode the only wake-up 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 power save mode is converted automatically inside the power save software in POWER_SAVE_LEVEL_CPU_HALT.

## 3.3 POWER_SAVE_LEVEL_STOP_WITH_TIMER

In this low power mode the CPU is stopped and all the peripherals are disabled. Only the timer clock sources (LSI or LSE) and the external wake-up source block are running. Wakeup is possible from external wake-up sources, RTC, IWDG, radio and the Hal Virtual Timers. A typical source code is:

```
VTIMER_HandleType TimerHandle;
WakeupSourceConfig_TypeDef wakeupIO = {0, WAKEUP_PA8, 0};
PowerSaveLevels stopLevel
/* Starts the virtual timer  with timeout 2 sec. */
HAL_VTimerStart_ms(TimerHandle, 2000);
while(1) /*main loop*/
{
   /* Modules ticks */
   ModulesTick();
   /* Application Tick */ APP_Tick();
   /* Power Save management */
   HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_STOP_WITH_TIMER, wakeupIO, &stopLevel);
}
```

The wake-up source is the PA8 with low polarity level (detection of wakeup event on falling edge); the application sets a virtual timer to wake up the system when the 2 sec. timeout occurs.

In this scenario, the application enables the power save mode *POWER_SAVE_LEVEL_STOP_WITH_TIMER*. If the radio operating mode is in connection or advertising state, the stack accepts the power save 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 POWER_SAVE_LEVEL_STOP_NO_TIMER

In this power save mode the CPU is stopped and all the peripherals are disabled. Only the external wake-up source block runs.

A typical source code is:

```
WakeupSourceConfig_TypeDef wakeupIO = {0, 0, WAKEUP_PA10};
PowerSaveLevels stopLevel;
while(1) /*main loop*/
{
   /* Modules ticks */
   ModulesTick();
   /* Application Tick */
   APP_Tick();
   /* Power Save management */
   HAL_PWR_MNGR_Request(POWER_SAVE_LEVEL_STOP_NO_TIMER, wakeupIO, &stopLevel);
}
```

The wake-up source is the PA10 with low polarity level (detection of wakeup event on falling edge).

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

# 4 Current consumption measurement test scenarios

This section describes a power consumption demonstration application allowing typical current consumption measurements to be shown during Bluetooth LE advertising and connection phases.

To take these measurements the following configuration has been used:

- **Hardware**: STEVAL-IDB0011V1
- **Firmware**: BLE_Power_Consumption test application released in the STSW-BRNGLP-DK, BlueNRG-LP software package
- **Power supply**: 3.3 V
- **Tx Power**: 0 dBm
- **Retention**: full RAM retention 64 kB
- **Crystal Startup time**: 780 µs
- **System clock** : direct HSE configuration
- **BLE clock**: 16 MHz
- **Advertising interval**: 100 ms and 1000 ms, with 28 byte packet length
- **Connection interval**: 100 ms and 1000 ms, with empty packets

The BLE_Power_Consumption test application configures the BlueNRG-LP as a Bluetooth Low Energy (LE) peripheral device.

It is included in the BlueNRG-LP DK software package, in Projects, BLE_Examples folder.

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

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

The current consumption values could be measured connecting a DC power analyzer to the STEVAL-IDB001V1 probe.

**Table 1. BlueNRG-LP current consumption values**

| Bluetooth LE scenario | Real current consumption |
|---|---|
| Advertising interval 100 ms | 137.898 µA |
| Advertising interval 1000 ms | 15.412 µA |
| Connection interval 100 ms | 69.546 µA |
| Connection interval 1000 ms | 8.816 µA |

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

Figure 2. **BlueNRG-LP advertising procedure: interval 100 ms**

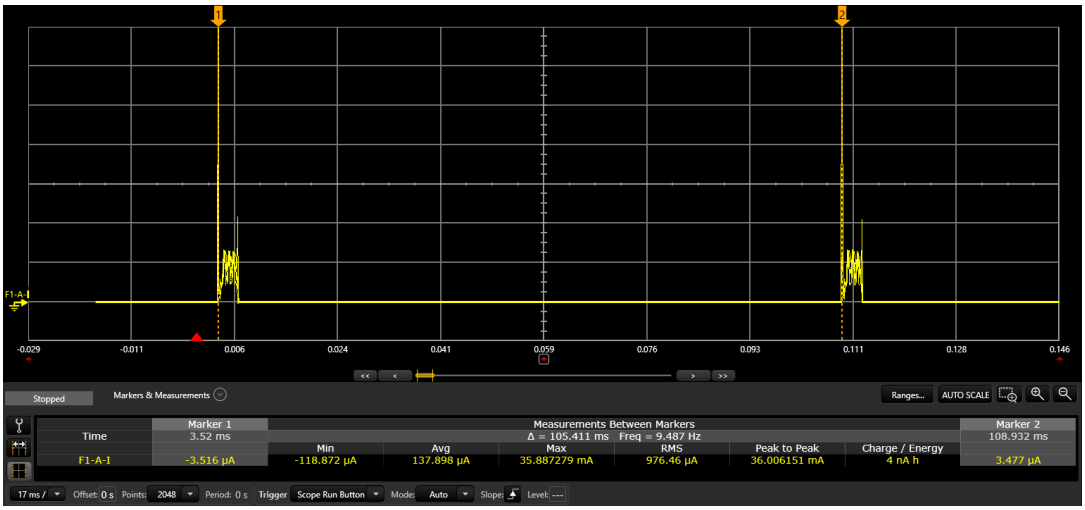
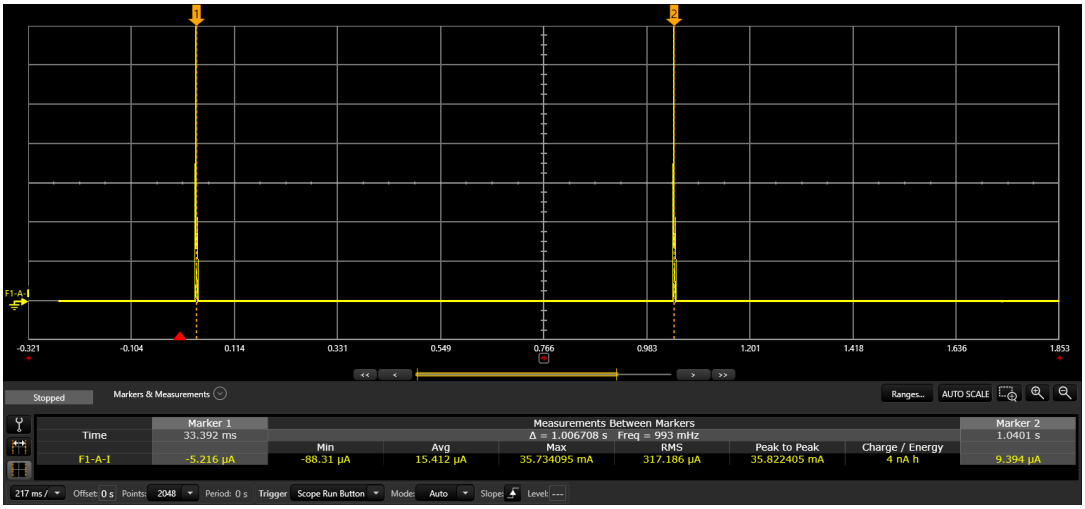



Figure 3. **BlueNRG-LP advertising procedure: interval 1000 ms**




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

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

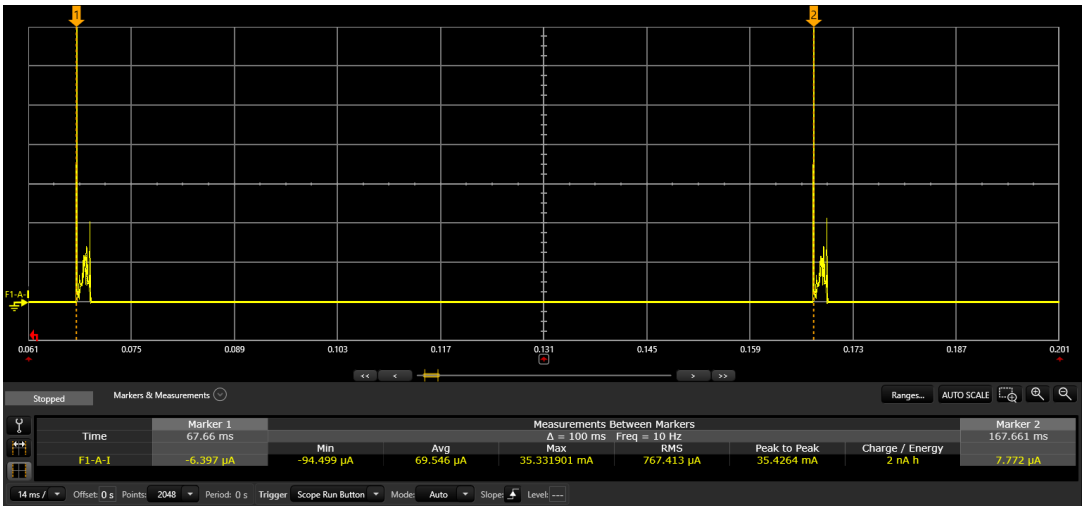
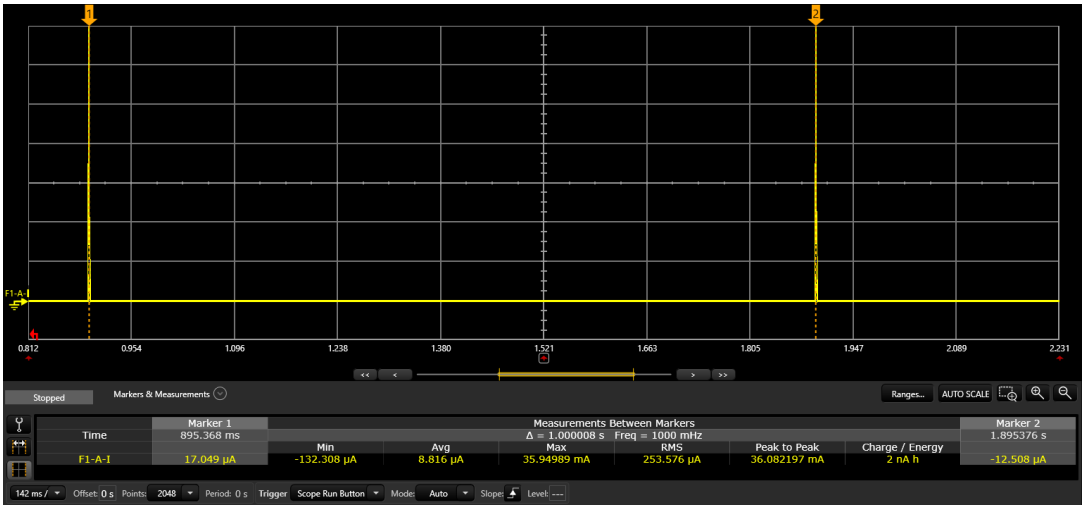



Figure 5. **BlueNRG-LP connection procedure: interval 1000 ms**



# Revision history

Table 2. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 13-Jul-2020 | 1 | Initial release. |

# Contents

# List of tables

# List of figures