AN5302

Application note

Power profiling of the 6LoWPAN applications of X-CUBE-SUBG1 v3.1.0 expansion software for STM32Cube

Introduction

This document shows how to perform power measurements on X-CUBE-SUBG1 (version v3.1.0 or higher) expansion software for STM32Cube and check its power improvement over previous versions, with respect to 6LoWPAN applications.

Low power optimization has been added from version 3.1.0 to better support battery-operated devices.

Although X-CUBE-SUBG1 supports different hardware platforms, only the NUCLEO-L152RE development board equipped with the X-NUCLEO-S2868A1 expansion board has been taken into consideration for this study.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
API	Application programming interface
UDP	User datagram protocol
6LoWPAN	IPv6 over low power wireless personal area networks
RPL	Routing protocol for low power and lossy networks
MCU	Microcontroller unit
LP	Low power
DC	Duty cycle
SDK	Software development kit
MSI	Multi-speed internal

2 Hardware and software requirements

To perform power measurement on X-CUBE-SUBG1 (v3.1.0 or higher), you need the following resources:

- Hardware
  - two NUCLEO-L152RE development boards
  - two X-NUCLEO-S2868A1 Sub-GHz expansion boards
  - two X-NUCLEO-LPM01A expansion boards for power consumption measurement (one for MCU and one for radio of a single node under test)
- Software
  - X-CUBE-SUBG1 v3.1.0 or above
- Tools
  - IAR v8.20.2/Keil v5.24.2/AC6 System Workbench v1.16.0 (to rebuild the projects by changing some parameters)
  - ST-LINK, ST-LINK/V2, ST-LINK/V2-1 USB driver signed for Windows7, Windows8, Windows10
  - STM32CubeMonPwr

To set up the system, two NUCLEO-L152RE plus X-NUCLEO-S2868A1 stacks are required to create on node running the *Udp_sender* firmware and another node running the *Udp_receiver* firmware.

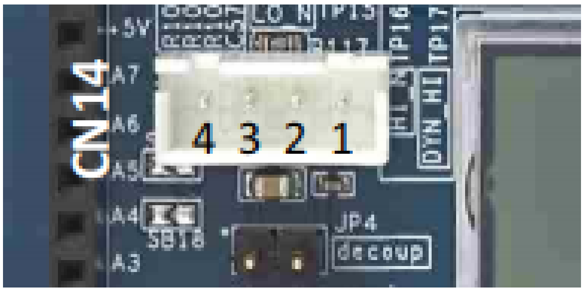
Only one node at a time can be tested, by connecting it to the two X-NUCLEO-PLM01A expansion boards to retrieve current consumption measures for MCU and radio.

3 Hardware configuration

Step 1. Configure one of the NUCLEO-L152RE development boards as follows:

- **Step 1a.** Open JP6 (IDD).
- **Step 1b.** With a wire, connect its second pin to the third pin of CN14 of one of the X-NUCLEO-LPM01A expansion boards (red line on the right in Figure 2. Hardware configuration for the node under test).

Figure 1. **X-NUCLEO-LPM01A expansion board: CN14 pin numbering**



Important:
Refer to UM2243, freely available at ww.st.com, for pin numbering (from right to left) and functionality of CN14 connector.

Step 2. Configure the X-NUCLEO-S2868A1 expansion board as follows:

- **Step 2a.** Open JP1.
- **Step 2b.** With a wire, connect its first pin to the third pin of CN14 of one of the X-NUCLEO-LPM01A (red line on the left in the figure below).

Step 3. Configure the two X-NUCLEO-LPM01A expansion boards as follows:

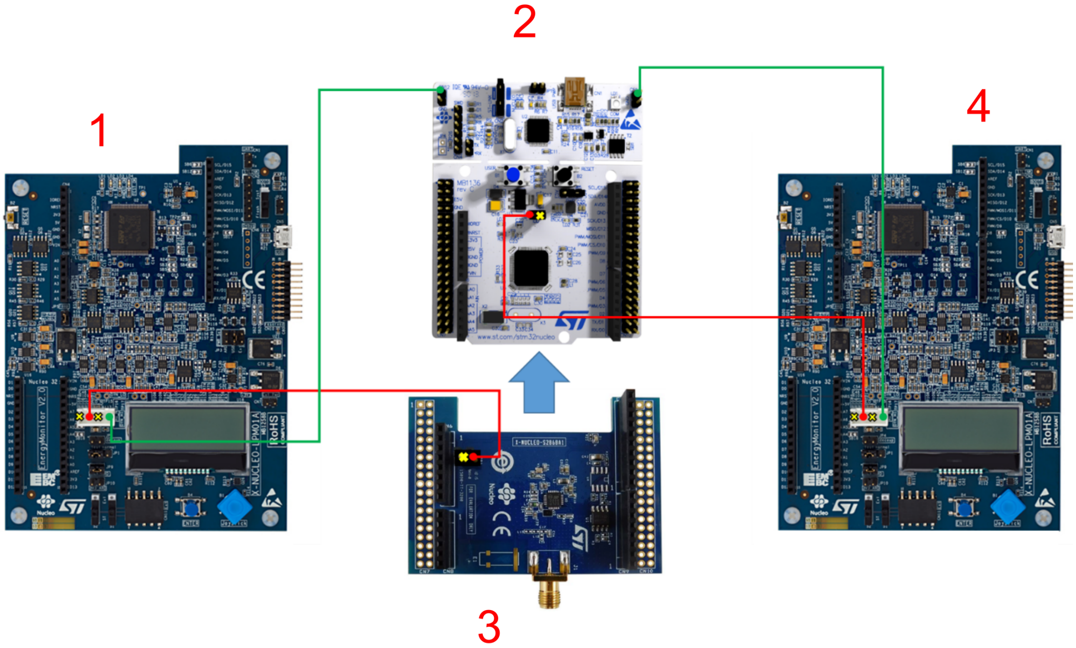
- **Step 3a.** For each of them, with a wire, connect the first pin of CN14 to a GND pin on the NUCLEO-L152RE (green lines in the figure below).

Step 4. Stack the X-NUCLEO-S2868A1 on top of the NUCLEO-L152RE.

Step 5. Connect the NUCLEO-L152RE and the two X-NUCLEO-LPM01A power shield boards to a PC USB port.

Figure 2. Hardware configuration for the node under test

1 - X-NUCLEO-LPM01A to measure radio consumption, USB connected to a PC for STM32CubeMonPwr tool
2 - NUCLEO-L152RE, USB connected to a PC for terminal output
3 - X-NUCLEO-S2868A1 S2-LP expansion board to be plugged on the STM32 Nucleo
4 - X-NUCLEO-LPM01A to measure MCU consumption, USB connected to a PC for STM32CubeMonPwr tool



4 How to flash the STM32 Nucleo board and set up the STM32CubeMonPwr tool

Step 1. Download the X-CUBE-SUBG1 firmware.

Step 2. Program the STM32 Nucleo board, that is, drag and drop the pre-built binary: *STM32CubeExpansion_SUBG1_V3.1.0\Projects\Multi\Applications\Contiki\Udp-receiver\Binary \S2LP_STM32L152RE_Nucleo\RX_S2868A1_STM32L152RE_Nucleo.bin*.

Step 3. Launch two instances of the STM32CubeMonPwr tool and for each of them:

- **Step 3a.** Select one of the COM ports from the [**SELECT X-NUCLEO-LPM01A**] drop down menu.
- **Step 3b.** Press the [**TAKE CONTROL**] button.
- **Step 3c.** Select [**Show Report**] in the plot top right area.
- **Step 3d.** Adjust [**Sampling Frequency**] and [**Acquisition Time**] (100000 Hz and ∞ s are used here).
- **Step 3e.** Press the [**START ACQUISITION**] button to start retrieving the current consumption.

5 Data with no low power optimization

The behavior of the *Udp_receiver* firmware has been tested. The node hardware running the UDP receiver sample application consists of a NUCLEO-L152RE development board with an X-NUCLEO-S2868A1 S2-LP radio expansion board.

The consumption is shown for idle case, that is without any other node sending data.

These consumption profiles are common to almost any other firmware running on the same hardware if no low power feature is enabled.

The following figures highlight how the current profile is flat, since both MCU and radio are always active in the same power domain.

The aggregate consumption (MCU plus radio) is in the order of 20 mA.

Figure 3. Udp receiver application: MCU current consumption without low power optimization

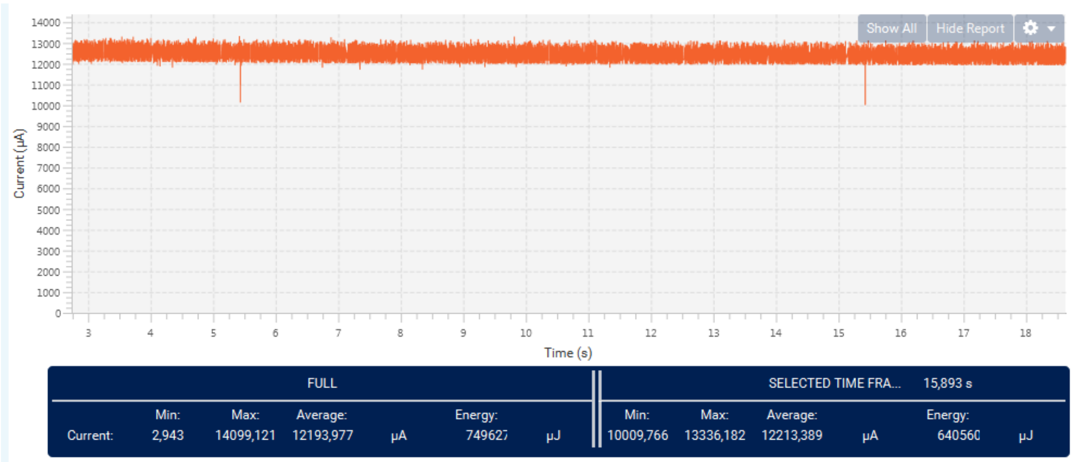
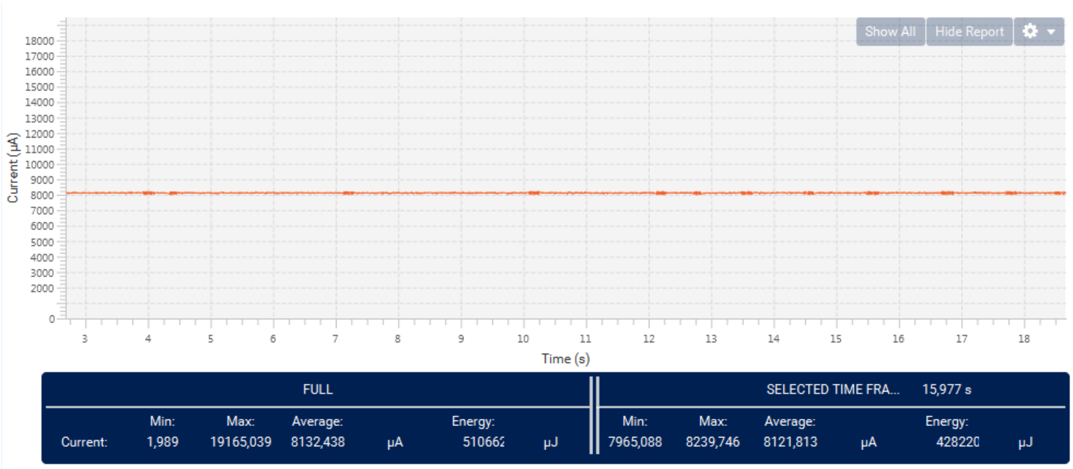


Figure 4. Udp receiver application: radio current consumption without low power optimization



6 Low power optimization

In the X-CUBE-SUBG1 package, starting from version 3.1.0, low power features are available when using NUCLEO-L152RE and X-NUCLEO-S2868A1 boards, for both the MCU and the radio.

By default, they are enabled in the project settings (for *Udp_sender* and *Udp_receiver* applications) by the definitions:

- `MCU_LOW_POWER=1`
- `RADIO_LOW_POWER=1`

The low power strategy consists of a set of system wide settings:

- overall system clock frequency is set to 4 MHz instead of the default 32 MHz using the `LP_sysclk_config()` API
- by calling the `LP_enter_sleep_mode()`/`LP_exit_sleep_mode()` APIs, the MCU is put into sleep mode during the idle loop, that is, when there is no process to be scheduled by the Contiki OS. During the sleep phase, the MCU clock is furtherly reduced to 65 KHz
- the S2-LP radio uses the duty cycle sniff mode

It also consists of two application level primitives, `LP_enter_stop_mode()` and `LP_exit_stop_mode()`, that, together with the radio driver API, in particular `subGHz_radio.on()`, enable the implementation of the application level duty cycle, sending the whole system into a very low power consumption for a given period.

When the sleep period ends, it is possible to wake up both MCU and radio at the same time to periodically send data, or wake up the MCU only to check if there are any data to send (so, in a driven-event context).

6.1 Low power parameters

6.1.1 Radio sniff mode

The sniff mode is one of the radio duty cycle capabilities natively supported by the S2-LP radio, together with the LDC mode.

The sniff mode has been chosen for this study as it does not require any synchronization between the node acting as a sender and the node acting as a receiver.

The sniff mode settings are the same as the STSW-S2LP-DK default:

- in *radio-driver.h*:

```
#define DATARATE 38400
#define PREAMBLE_BYTE(v) (4*v)
#define PREAMBLE_LENGTH PREAMBLE_BYTE(64)
#define MIN_PERIOD_WAKEUP_MS ((8000*((PREAMBLE_LENGTH/4)-2))/DATARATE
#define RX_TIMEOUT_MS 30
```

- in *radio-driver.c*:

```
S2LPTimerSetWakeUpTimerUs(1000*MIN_PERIOD_WAKEUP_MS); //12 ms
/* set the rx timeout */
S2LPTimerSetRxTimerUs(1000*RX_TIMEOUT_MS); //30 ms
#ifdef CSMA_ENABLE
/* use SLEEP_B mode (mandatory for CSMA) */
S2LPTimerSleepB(S_ENABLE);
#else /*!CSMA_ENABLE*/
/* use SLEEP_A mode (default) */
S2LPTimerSleepB(S_DISABLE);
#endif /*CSMA_ENABLE*/
 /* enable LDC mode, FAST RX TERM and start Rx */
S2LPTimerLdcrMode(S_ENABLE);
/* enable the fast rx timer */
S2LpTimerFastRxTermTimer(S_ENABLE);
```

By choosing a longer preamble (modifying `PREAMBLE_LENGTH` macro), it is possible to further reduce the radio consumption by having a longer period between two consecutive channel probes.

6.1.2 MCU settings

The low power APIs for the MCU behavior are implemented in the *low-power.c* file, at application level.

For the MCU, to enter and exit from sleep mode two API functions are used:

- `void LP_enter_sleep_mode(void);`
- `void LP_exit_sleep_mode (void);`

For the MCU low power settings, the main API is `void LP_sysclk_config (unsigned int clk_freq);`, used to set up:

- the overall (run mode) system clock frequency (`USER_CLOCK_FREQUENCY_HZ`), that, by default, is set to 4 MHz instead of the nominal 32 MHz (called by `main()`)
- the reduced system clock frequency (`SLEEP_CLOCK_FREQUENCY_HZ`) for the sleep mode (called by `LP_enter_sleep_mode()`), and obviously to restore the original settings (in `LP_exit_sleep_mode()`).

These clock frequency values can be changed in the *low-power.h* file.

Since going into sleep mode with reduced frequency changes the behavior of the clock, an offset of ticks must be added to guarantee the correct timing of the system, once it goes back to run mode. This value depends on the selected system clock frequency (`USER_CLOCK_FREQUENCY`), on the clock frequency used in sleep mode (`SLEEP_CLOCK_FREQUENCY_HZ`) and on the compiler in use, since the implementation of the `__WFI()` instruction depends on the latter.

If you change the value of the `SLEEP_CLOCK_FREQUENCY_HZ` macro to something different from 65 KHz, the number of compensation ticks also must be changed.

In the *low-power.h* file, some predefined values (`USER_CLOCK_FREQUENCY` = 32 MHz or 4 MHz, `SLEEP_CLOCK_FREQUENCY_HZ` = 65 KHz) are available, in the macro `COMPENSATE_TICKS`, but this value should be validated by the final application developer.

6.1.2.1 *Modifying MCU frequency settings*

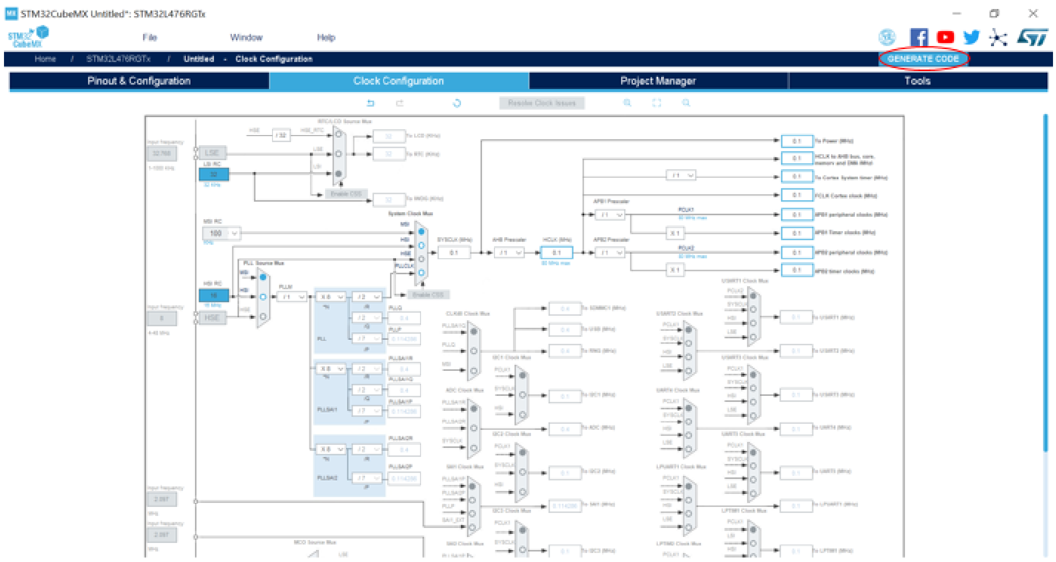
The STM32CubeMX tool can be used to generate the desired clock configuration and to implement sleep mode (that is with low power consumption through the reducing of the clock frequency) for different settings or for another MCU family. This tool allows configuring the MCU via a graphical interface and automatically generating the corresponding code.

The figure below shows the clock configuration of STM32L476RG using STM32CubeMX. As mentioned in the related reference manual (RM0432 freely available at www.st.com), the STM32L476RG multi-speed internal (MSI) clock includes twelve frequency ranges: 100 - 200 - 400 - 800 kHz, 1 - 2 - 4 - 8 - 16 - 24 - 32 and 48 MHz.

The MSI is selected as a system clock mux with MSI RC set to 100 kHz: the system clock frequency is equal to 100 kHz, thus, reducing the clock frequency results in decreasing the MCU energy consumption.

Clicking the [**GENERATE CODE**] button on the window top right (circled in red), the tool generates a project containing the `SystemClock_Config()` function in the *main.c* file with the chosen configuration. In this manner, you can easily obtain the code of the desired system clock frequency.

Figure 5. STM32L476RG clock configuration using STM32CubeMX (example for clock frequency=100 kHz)



7 Consumption data with low power optimization enabled

7.1 Udp_receiver

By enabling the system wide settings, you can check the behavior of the `Udp_receiver` firmware in idle mode, and compare it to the results shown in Section 5 Data with no low power optimization.

Figure 6. Udp_receiver: MCU current consumption with low power (idle mode)

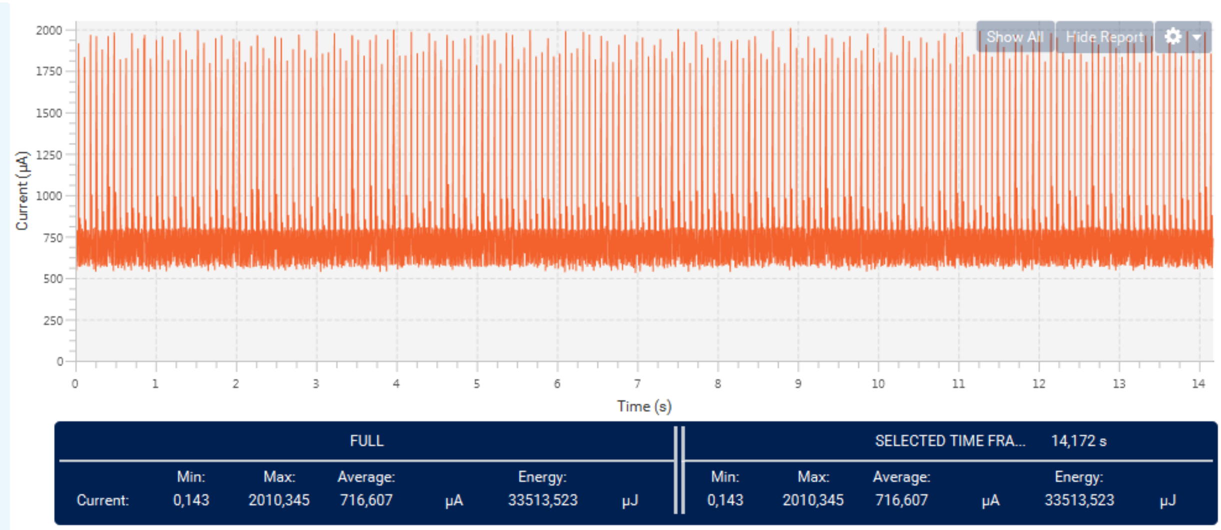
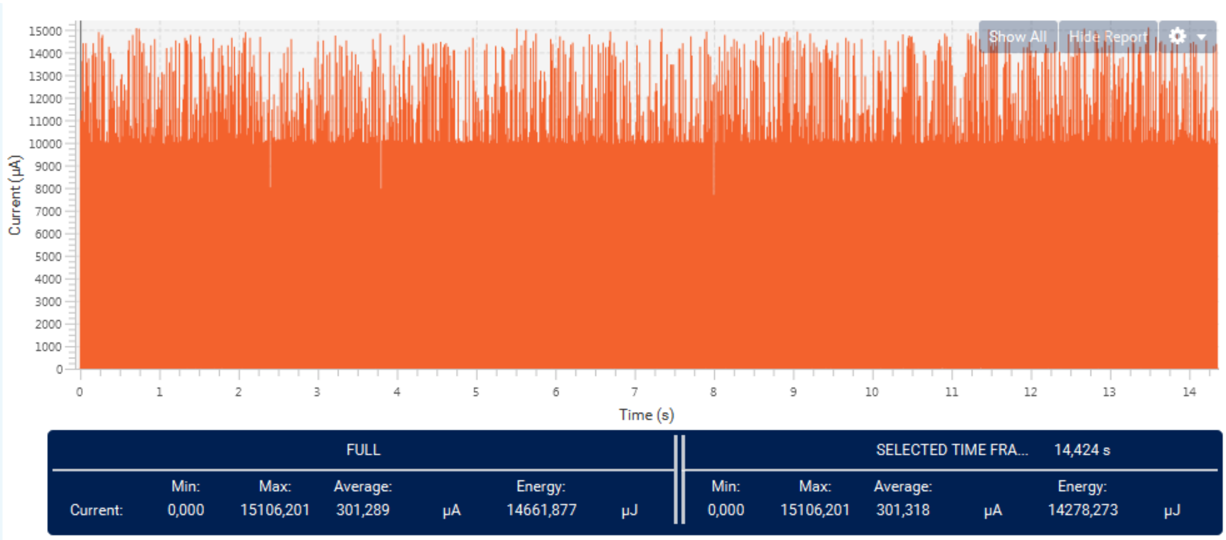


Figure 7. Udp_receiver: radio current consumption with low power



The current consumption profiles are not flat anymore because both MCU and radio switch into or out of low power consumption states. In this case, the idle aggregate consumption is below 1.1 mA, a reduction of almost 95% of the starting value, related to no low power mode implementation.

When zooming, with reference to a 1 second timeframe, you can notice the MCU being awaken (running at 4 MHz) every 73 ms (around), and then going back to sleep mode (running at 65 KHz) for the idle loop (see Figure 8. Udp_receiver: MCU current consumption with low power (zoom to 1 s period)).

Regarding the radio, the channel becomes active every 12 ms due to the sniff mode selected LDCR wake up timer (see Figure 9. Udp_receiver: radio current consumption with low power (zoom to 1 s period)).

Figure 8. **Udp_receiver: MCU current consumption with low power (zoom to 1 s period)**

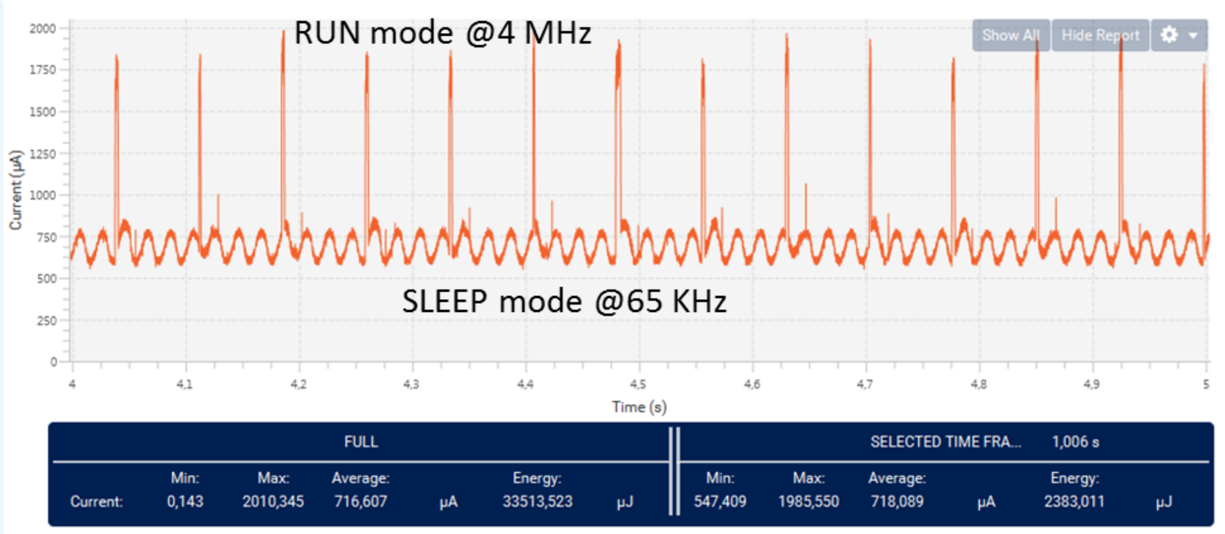
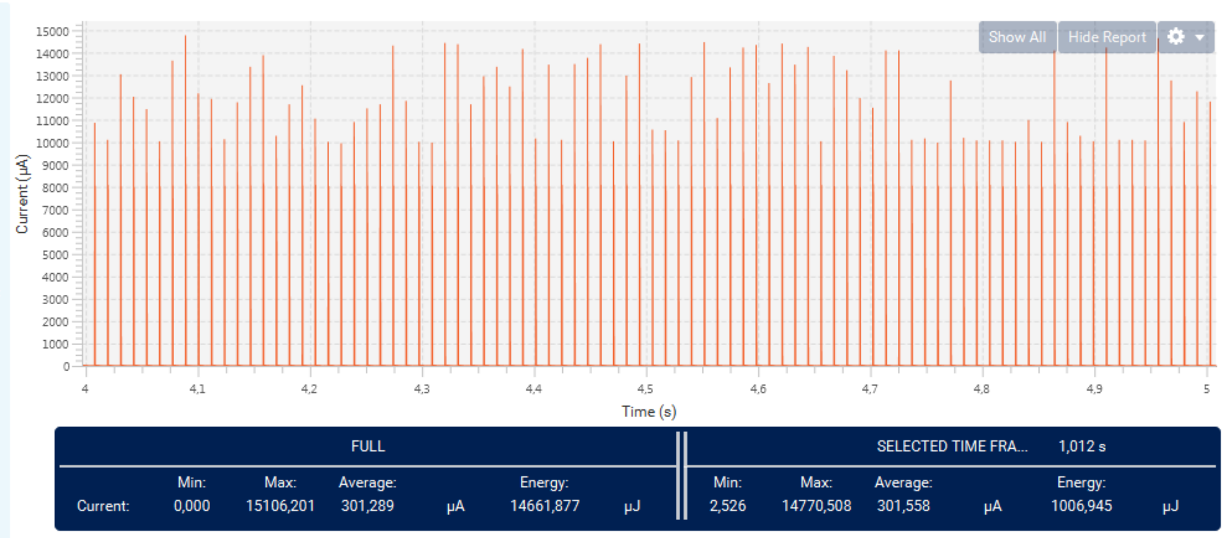


Figure 9. **Udp_receiver: radio current consumption with low power (zoom to 1 s period)**



When programming the second node (NUCLEO-L152RE plus X-NUCLEO-S2868A1) with a `Udp_sender` firmware (pre-built binary available in *STM32CubeExpansion_SUBG1_V3.1.0\Projects\Multi\Applications\Contiki\Udp-sender\Binary\S2LP_STM32L152RE_Nucleo\TX_S2868A1_STM32L152RE_Nucleo.bin*) and switching it on by connecting it to a PC via USB, you can see the behavior of the `Udp_receiver` (that is currently the node under test) when it starts receiving packets (every 3 seconds, approximately).

Figure 10. Udp_receiver: MCU current consumption with low power (with UDP traffic)

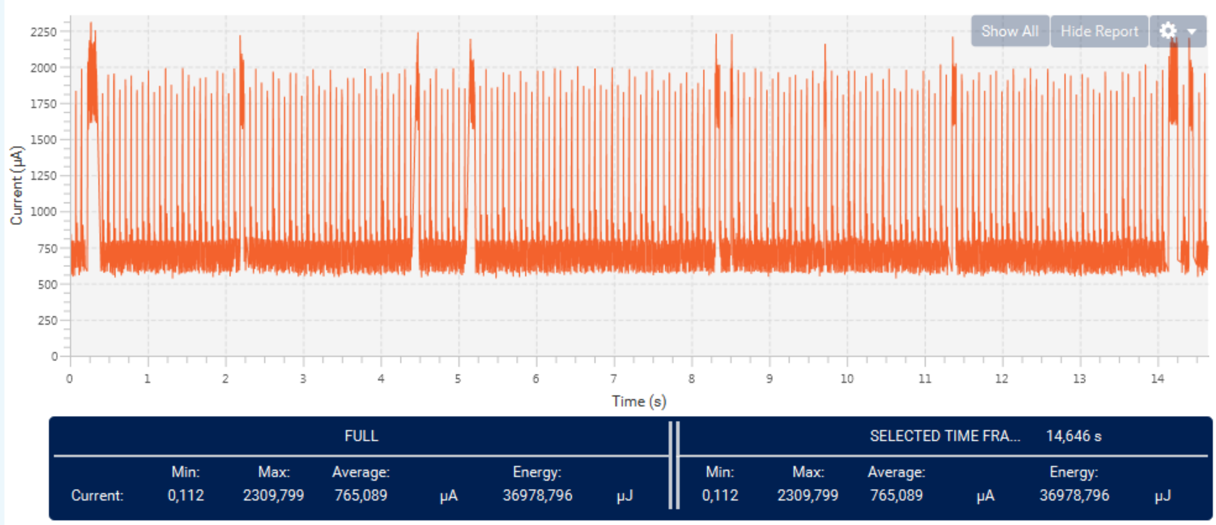
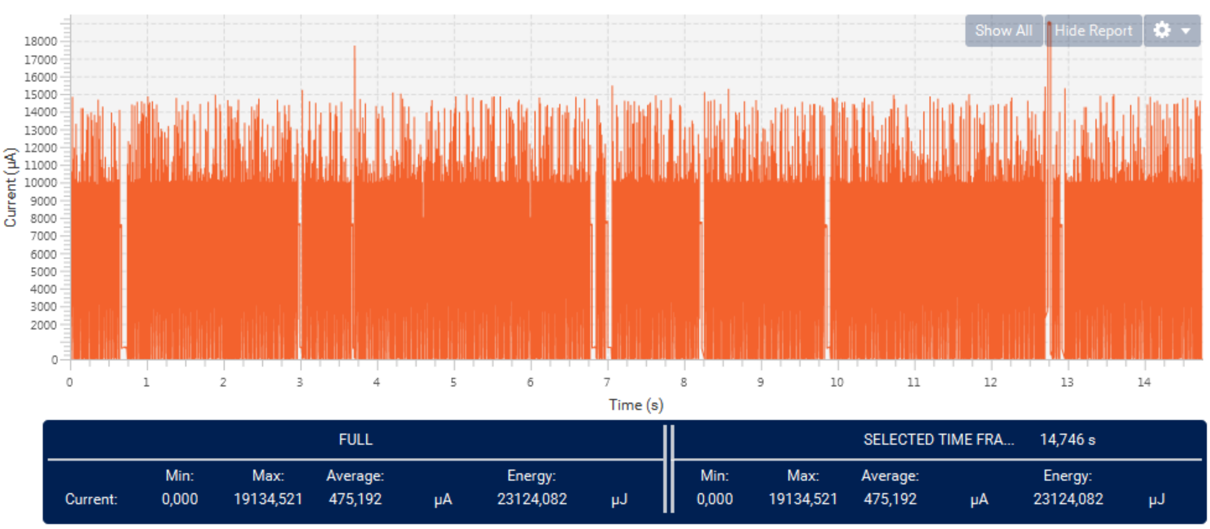


Figure 11. Udp_receiver: radio current consumption with low power (with UDP traffic)



The slight delay (represented by the holes in the graphs shown in the two figures above) for packet reception is due to the very low frequencies (run and sleep modes), compared to the sytem configuration nominal values.

7.2 Udp_sender

When analyzing the behavior of the `Udp_sender` application in terms of current consumption (that is when flashing the node under test with this firmware, and the other node with the `Udp_receiver` firmware), you can notice a similar profile, the idle behavior and the peaks due to the packet transmission (UDP every about 3 seconds in addition to some RPL traffic).

Figure 12. **Udp_sender: MCU current consumption with low power (with UDP traffic)**

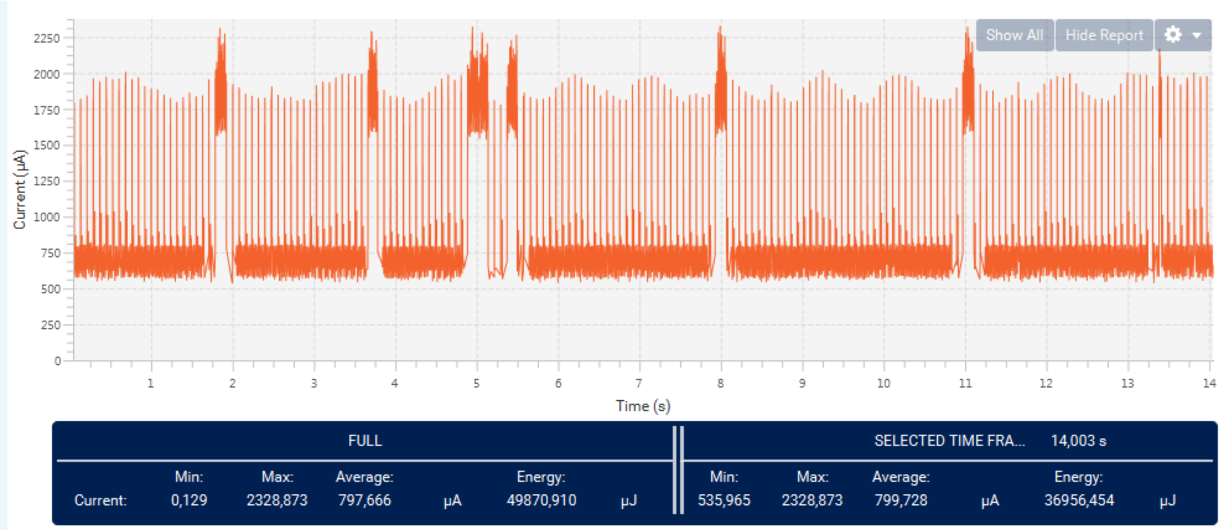
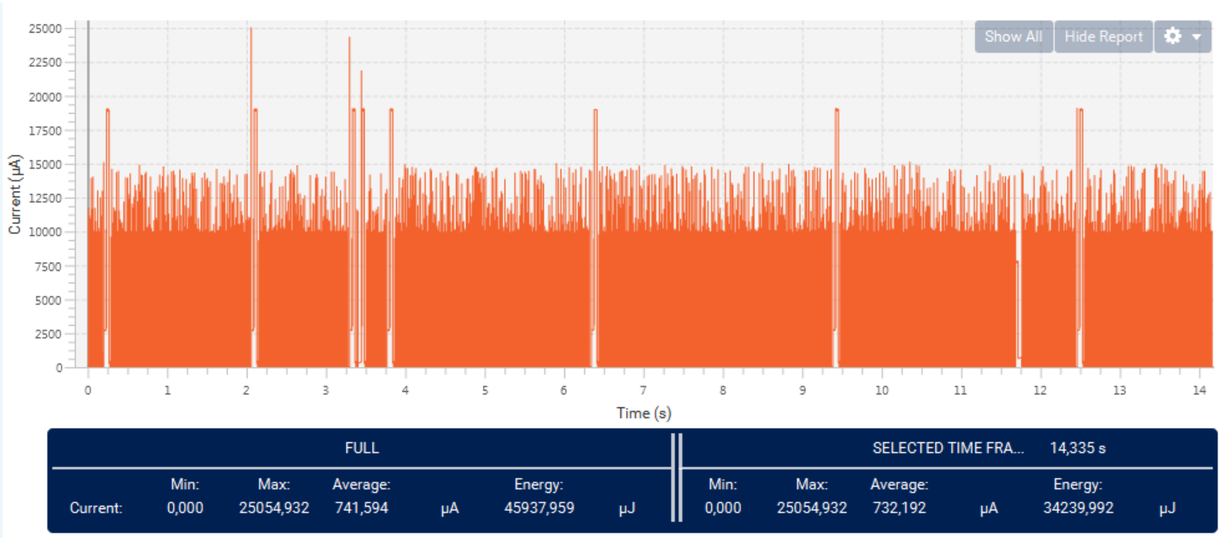


Figure 13. **Udp_sender: radio current consumption with low power (with UDP traffic)**



When using the `Udp_sender` firmware, you can enable the application layer duty cycle functions referred to in Section 6 Low power optimization.

By pressing the user button, the `Udp_sender` goes through three different operative modes:

1. `DUTY_CYCLE_NONE`: this is the normal behavior, MCU and radio are always on, according to the low power settings (that is the MCU sleeps in idle and the radio is in sniff mode). UDP packets are sent every `APP_DUTY_CYCLE_SLOT = 3` seconds by default.
2. `DUTY_CYCLE_SIMPLE`: simplest case of application layer duty cycle, that can address a periodic use case (that is send temperature value every x seconds); we have a stop slot of `APP_DUTY_CYCLE_OFF = 10` seconds in which both the MCU and the radio are off, the node is not reachable and cannot perform any computation. Then, both MCU and radio are switched on for two time slots of `APP_DUTY_CYCLE_SLOT = 3` seconds, in the middle of this time frame, and a UDP packet is always sent. As a result, UDP packets are sent every `(APP_DUTY_CYCLE_OFF + 2*APP_DUTY_CYCLE_SLOT) = 16` seconds by default. This functional mode can be implemented by calling `LP_exit_stop_mode(wake_up_radio)` with argument=1 (which equals "true") value.
3. `DUTY_CYCLE_PROBING`: a more complex application layer duty cycle, that can address an event driven use case (that is, to probe if there is smoke in the room and send the alarm); we have a stop slot of

`APP_DUTY_CYCLE_OFF = 10` seconds in which both the MCU and the radio are off, the node is not reachable and cannot perform any computation. Then only the MCU is switched on while the radio remains off, so the node can perform computation, but it is not reachable. A probe function can be called (the firmware implements a simple demo only). If the application needs to send data, then it switches the radio on and sends the packet. Probing is called every `APP_DUTY_CYCLE_OFF = 10` seconds and keeps MCU on for `APP_DUTY_CYCLE_SLOT = 3` seconds. In this mode, there should be no fixed periodic UDP sending interval, but, actually, the `probe()` function returns positive results every two calls, so UDP packets are sent every `(2*APP_DUTY_CYCLE_OFF + 3*APP_DUTY_CYCLE_SLOT) = 29` seconds by default. This functional mode can be implemented by calling `LP_exit_stop_mode(wake_up_radio)` with argument=0 (which equals "false") value, and implementing the `probe()` function.

The figure below shows the current consumption profiles for the probing duty cycle, for both MCU (at the top) and radio (at the bottom).

Figure 14. Udp_sender: probing duty cycle

The two current consumption plots are synchronized to highlight the two different behaviors when the `probe()` call returns false value (when the MCU only stays on), and when it returns true (when the radio is also turned on to send the packet).

The provided example is just a demo: the `probe()` function reports some data to be sent every two calls. The aggregate consumption of the system when in off state (actually, stop mode for the MCU and standby mode for the radio) is below 2 µA.

The application layer duty cycle, as shown in the `Udp_sender` firmware, can be used in a star topology when the node that is entering stop mode is acting as a leaf of the RPL tree (that is, it does not act as a parent of other nodes).

Note: *To allow all the nodes in the Mesh network to be turned off for a given period, an additional synchronization function is required, which has not been taken into account in this work.*

8 Power consumption gain

The table below shows the obtained gain in power consumption, highlighting the contribution of MCU and radio, comparing the results for the idle state, when the firmware is running without any packet transmission or reception.

It also shows the consumption values for off mode for the low power scenario.

Table 2. Power gain comparison for idle state

Low power optimization	Idle				STOP (µA)		
	MCU (µA)	Radio (µA)	TOT (µA)	Gain (%)	MCU (µA)	Radio (µA)	TOT (µA)
No	11860	8115	19975	-	-	-	-
Yes	745	300	1045	94.8%	1.4	0.4	1.8

When taking into account the packet transmission, you can evaluate the actual average current consumption of a node running the `Udp_sender` firmware and the gain that we can have by using the low power settings together with some application layer duty cycle strategy (see, for example, the demo in which the node is off for only 10 seconds).

Using the STM32CubeMonPwr tool, other measurements have been performed for current consumption:

- Low power:
  – UDP packet transmission for MCU: ~1875 µA (1.875 mA) for 0.14 s
  – On/off transitions for MCU: ~1750 µA (1.75 mA) for 0.09 s
  – UDP packet transmission for radio: ~9525 µA (9.525 mA) for 0.08 s
- Normal case with MCU running at 32 MHz:
  – UDP packet transmission for MCU: ~12300 µA (12.3 mA) for 0.038 s
  – UDP packet transmission for radio: ~13900 µA (13.9 mA) for 0.03 s

The `Udp_sender` firmware is configured to send data every 16 seconds for all the cases:

- normal case, no low power, MCU and radio are always on, system frequency of 32 MHz;
- low power system wide settings, system frequency of 4 MHz, the MCU goes to sleep in idle loop (at 65 KHz) and the radio is in sniff mode;
- low power settings as the previous case and simple duty cycle from application (`Udp_sender`).

The table below shows the results: the proposed low power case (B in the table) provides a considerable gain of about 94.5% compared to the normal case (A in the table).

Adding application layer duty cycle functions, the gain obviously depends on the duration of the off period.

Further improvements can be obtained by implementing, in the eligible scenarios, the probing duty cycle.

Table 3. Power gain comparison for Udp_sender case

Case	Description	MCU idle (µA)	MCU with TX (µA)	MCU with TX and on/off (µA)	Radio Idle (µA)	Radio with TX (µA)	TOT (µA)	Gain (%)
A	no LP, no DC	11860	11861	-	8115	8126	19987	-
B	LP, no DC	745	755	-	300	345	1100	94.5% (over A)
C	LP and DC	280	294	302	113	160	462	97.7% (over A) 58% (over B)

9 Conclusions

In this application note, a power profiling of the X-CUBE-SUBG1 v3.1.0 software package has been outlined, with a detailed observation of the results of power optimization in 6LoWPAN applications.

The power optimization has been performed for a specific subset of the supported platforms: the NUCLEO-L152RE development board, equipped with the X-NUCLEO-S2868A1 Radio (S2-LP) expansion board. Therefore, this application note does not consider the other hardware platforms supported by the X-CUBE-SUBG1 v3.1.0 software package.

The X-NUCLEO-LPM01A power shield board and the companion STM32CubeMonPwr tool have been used to profile the firmware.

In summary, depending on the use case, gains in power consumption have been measured up to almost 95% for the idle case, and even more for the use cases that allow the implementation of application layer duty cycle strategies.

Revision history

Table 4. **Document revision history**

Date	Version	Changes
01-Mar-2019	1	Initial release.

Contents

List of tables

List of figures