Getting started with the X-CUBE-SFXS2LP1 software expansion for STM32Cube

Introduction

The X-CUBE-SFXS2LP1 expansion software package for STM32Cube runs on the STM32 and includes the drivers for S2-LP and the library for the Sigfox™ proprietary protocol.

This software together with the suggested combination of STM32 and S2-LP device can be used, for example, to develop applications for smart home/building and smart cities, agriculture, parking, lighting, etc.

The expansion is built on STM32Cube software technology to ease portability across different STM32 microcontrollers.

The software comes with a sample implementation of the drivers running on the X-NUCLEO-S2868A1 expansion board connected to a NUCLEO-L053R8, NUCLEO-L152RE or NUCLEO-L476RG development board.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
BSP	Board support package
CLI	Command line interface
CMSIS	Cortex® microcontroller software interface standard
GUI	Graphical user interface
HAL	Hardware abstraction layer
ID	Unique device ID
PAC	Port authorization code
SPI	Serial peripheral interface

2 What is STM32Cube?

STM32Cube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

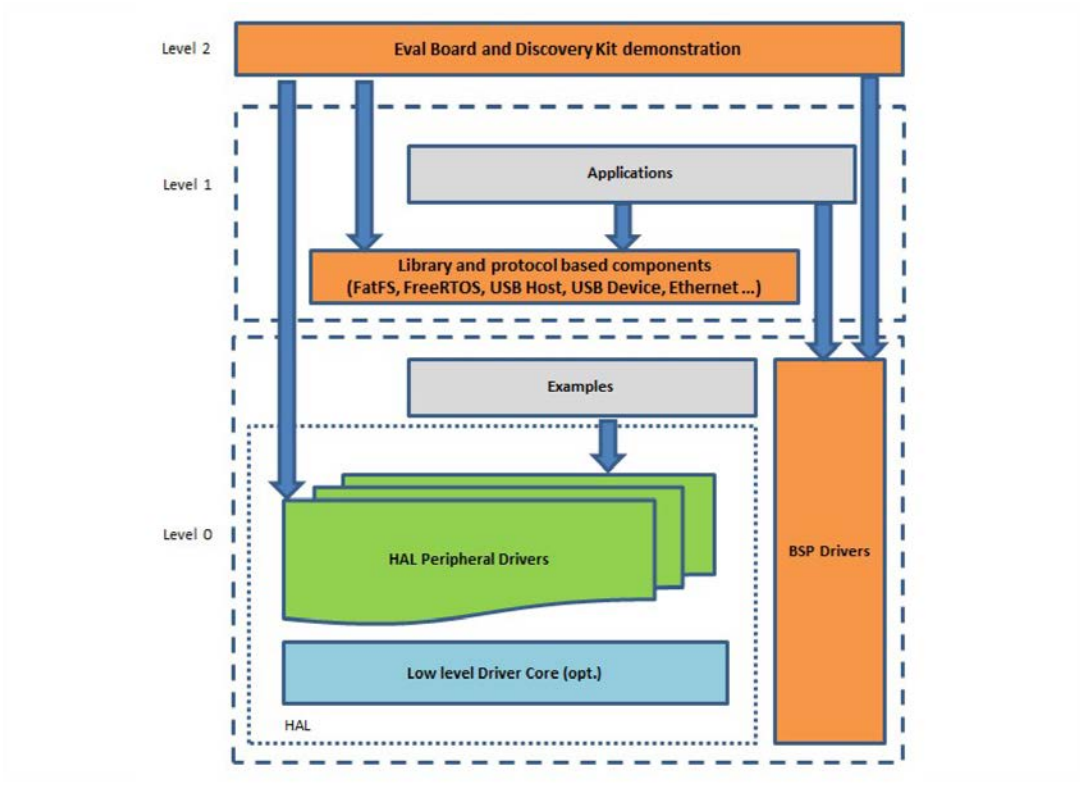
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32CubeF4 for the STM32F4 series), which includes:
  - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
  - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
  - all embedded software utilities with a full set of examples

2.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below.

Figure 1. Firmware architecture


Level 2
Eval Board and Discovery Kit demonstration
Level 1
Applications
Library and protocol based components (FatFS, FreeRTOS, USB Host, USB Device, Ethernet ...)
Examples
Level 0
HAL Peripheral Drivers
Low level Driver Core (opt.)
HAL
BSP Drivers

Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc…); it is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines. It is composed of two parts:

  - Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
  - BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().
- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I²C, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

3 X-CUBE-SFXS2LP1 software expansion for STM32Cube

3.1 Overview

The X-CUBE-SFXS2LP1 software package key features are:

- Complete software to build applications using Sigfox™ long range wireless area network running on the S2-LP high performance ultra-low power RF transceiver
- S2-LP Sigfox™ library with a complete set of APIs to develop embedded applications
- Compatible with the STSW-S2LP-SFX-DK graphical user interface (GUI) to register end-device to Sigfox™ network and get ID (Unique Device ID)/PAC (Port Authorization code) /Key from the pool assigned to ST devices
- GUI PC application available as interactive interface to transmit messages to the Sigfox™ network
- Sample implementation available on the X-NUCLEO-S2868A1 expansion board connected to a NUCLEO-L053R8, NUCLEO-L152RE or NUCLEO-L476RG development board
- ID/PAC/Key stored in internal MCU flash or external EEPROM
- Easy portability across different MCU families, thanks to STM32Cube
- Free, user-friendly license terms

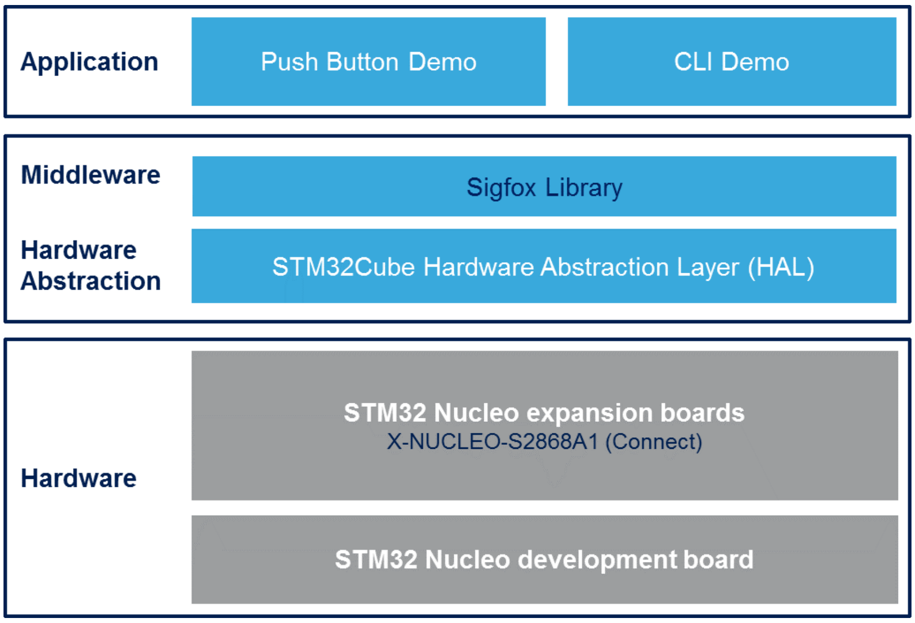
3.2 Architecture

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller and extends STM32Cube by providing the middleware library for the Sigfox application using the S2-LP expansion board and ready-to-use samples to show this library usage.

The software layers used by the application software to access and use the board are:

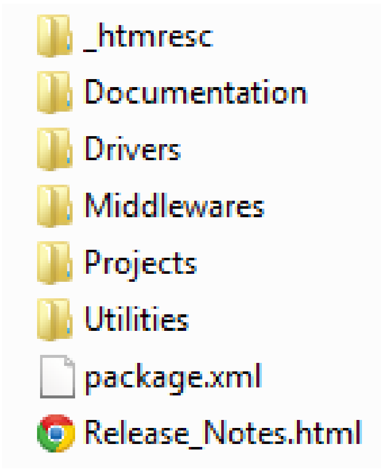
- **STM32Cube HAL layer**: provides a generic, multi-instance set of simple APIs (application programming interfaces) to interact with the upper layers (application, libraries and stacks). It is composed of generic and extension APIs. It is directly built around a generic architecture and allows the layers that are built upon, such as the middleware layer, to implement their functionalities without dependencies on the specific hardware configuration for a given Microcontroller Unit (MCU). This structure improves the library code reusability and guarantees easy portability across devices.
- **Board support package (BSP) layer**: supports the peripherals on the STM32 Nucleo board, except for the MCU. This limited set of APIs provides a programming interface for certain board specific peripherals (like the user button, the reset button, etc.) and helps in identifying the specific board version.
- **Middleware**: includes the Sigfox Libraries in lib(.a) form with a complete set of APIs to develop embedded applications.

Figure 2. X-CUBE-SFXS2LP1 architecture


Application
Push Button Demo
CLI Demo
Middleware
Sigfox Library
Hardware Abstraction
STM32Cube Hardware Abstraction Layer (HAL)
Hardware
STM32 Nucleo expansion boards
X-NUCLEO-S2868A1 (Connect)
STM32 Nucleo development board

3.3 Folder structure

Figure 3. X-CUBE-SFXS2LP1 package folder structure



The software is packaged in the following main folders:

- **Documentation**: with a compiled HTML file generated from the source code that details the software components and APIs.
- **Drivers**: contains the HAL drivers, the board specific drivers for each supported board or hardware platform, including the on-board components ones and the CMSIS layer which is a vendor-independent hardware abstraction layer for the ARM Cortex-M processor series.
- **Middlewares**: contains the Sigfox library and Sigfox interface for STM32 as well as the libraries for ETSI, ARIB,FCC and ALL specification.
- **Projects**: provides sample applications for the push button demo and CLI demo.

  In the push button demo, the Sigfox message is transmitted by simply pressing the user button on the STM32 Nucleo boards. The message is shown on the Sigfox web application.

  In the CLI demo, CLI (command line interface) mode provides the connectivity with Sigfox GUI. The Sigfox message is transmitted by clicking “Tx” in the GUI. The projects are provided for the NUCLEO-L053R8, NUCLEO-L152RE and NUCLEO-L476RG platforms with three development environments (IAR Embedded

Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM) and System Workbench for STM32 (SW4STM32)).

- **Utilities**: contains the files to interpret the commands sent via PC using the GUI or any terminal utility. It also contains the Sigfox GUI.

3.4 APIs

Detailed function and parameter descriptions for the user-APIs are compiled in an HTML file in the software package Documentation folder.

3.5 Sample application description

The Sigfox over S2-LP library features:

- complete set of the APIs to develop the embedded application
- graphical user interface (GUI) PC application to provide an interactive interface to transmit messages to the Sigfox network
- ready-to-use projects available for IAR, Keil and SW4STM32
- easy portability across different MCU families, thanks to STM32Cube

The user application features:

- initialization of the Sigfox and S2-LP stack
- user functions required for the application
- application handling

3.5.1 RF_LIB library

The RF_LIB library configures and implements the S2-LP modulation scheme.

The library drives the S2-LP according to the Sigfox modulation protocol:

- DBPSK for uplink (14 dBm at 100 bps for RCZ1/3, 22 dBm at 600 bps for RCZ2/4)
- 2GFSK, BT=1 for downlink

The channel frequency, data rate and other relevant parameters depend on the applicable radio control zone (RCZ).

The library is compiled for devices equipped with certain ARM® Cortex® cores. Separate versions are supplied in the Projects/Middlewares/Sigfox/CMx folder for specific radio control zones:

- RCZ1: ST_RF_LIB_ETSIv_CMx_C.c
- RCZ2 and 4: ST_RF_LIB_FCCv_CMx_C.a
- RCZ3: ST_RF_LIB_ARIBv_CMx_C.a
- All RCZ: ST_RF_LIB_ALLv_CMx_C.a

Note: *In the above naming convention:*

- *v = version number of the library (ex. v = 250 for V2.50)*
- *x = the CMx core class (CM0, CM3, CM4 or CM7)*
- *C = compiler (IAR, Keil, GCC)*

User applications call generic APIs not linked to any specific hardware whereas the RF_LIB library calls low level APIs to provide the necessary hardware platform support.

3.5.2 MCU_API module

The framework can be easily ported to another board equipped with a microprocessor of the same type but with a different pinout by simply re-implementing the MCU_API module.

Table 2. **MCU_API module details**

Name	Argument	Description
MCU_API_malloc	size: the number of bytes to be allocated pointer: pointer to the new allocated block of memory	Allocates memory for library usage (memory usage = size in bytes) This function is called only once at the Sigfox library opening.
MCU_API_free	pointer: pointer to the memory of free up	Free memory allocated to the library.
MCU_API_get_voltage_temperature	voltage_idle: pointer to the variable where voltage in idle should be stored voltage_tx: pointer to the variable where voltage in TX should be stored temperature: pointer to the variable where temperature should be stored	Gets voltage and temperature for out of band frames. The value must respect the units for backend compatibility.
MCU_API_delay	delay_ms: number of ms to wait	Blocking function for delay_ms milliseconds.
MCU_API_aes_128_cbc_encrypt	encrypted_data: pointer to the destination buffer where the encrypted data must be stored data_to_encrypt: pointer to the source buffer where the data to encrypt are stored data_len: length of the data buffer to encrypt	This function is in charge of encrypting the data transmitted by the Sigfox library.
MCU_API_get_nv_mem	read_data: pointer to the array where the NVM content should be stored.	This function is used to read data from the NVM used by the Sigfox library.
MCU_API_set_nv_mem	data_to_write: pointer to the array containing the data to be stored in the NVM.	This function is used to write data to the NVM used by the Sigfox library.
MCU_API_timer_start_carrier_sense	timer_duration_ms: the total duration of the timer in milliseconds. This function starts a timer without blocking the application.	This timer is used for the carrier sense (in RCZ3 only).
MCU_API_timer_start	timer_duration_ms: the total duration of the timer in milliseconds	This function starts a timer without blocking the application. You should use an RTC to let the µC enter low power mode.
MCU_API_timer_stop	None	This function stops the timer started by the MCU_API_timer_start.
MCU_API_timer_stop_carrier_sense	None	This function stops the timer started by the MCU_API_timer_start_carrier_sense
MCU_API_timer_wait_for_end	None	Blocking function to wait for interrupt indicating timer elapsed. This function is used only for the 20 seconds in downlink.
MCU_API_report_test_result	status: 1 - passed 0 - failed rssi: RSSI of the received frame	Reports the result of RX test for each valid message received/validated by the library.
MCU_API_get_version	pointer to the array variable and size	This function gets current version
MCU_API_get_device_id_and_payload_encryption_flag	pointer to the uint8_t array variable where the returned ID should be stored.	Gets the device ID
MCU_API_get_initial_pac	pointer to the uint8_t array variable where the returned PAC should be stored.	Gets the device initial PAC
ST_MCU_API_SpiRaw	number: number of elements of the total SPI transaction value_ptr_in: pointer to the input buffer (µC memory where the SPI data to write are stored) value_ptr_out: pointer to the output buffer (the µC memory where the data from SPI must be stored) can_return_bef_tx: if this flag is 1, the function can be non-blocking, returning immediately.	Performs a raw SPI operation with the passed input buffer and stores the returned SPI bytes in the output buffer.

Name	Argument	Description
ST_MCU_API_GpioIRQ	pin: the GPIO pin of the S2-LP (integer from 0 to 3) new_state: enables or disables the EXTI trigger_flag: 1: rising edge, 0: falling edge	Enables or disables the external interrupt on the µC side. The interrupt must be set on the rising or falling edge of the input signal according to the trigger_flag. The pin number represents the S2-LP GPIO number.
ST_MCU_API_Shutdown	sdn_flag: 1 - enter shutdown 0 - exit shutdown	Sets the S2-LP on or off via GPIO
ST_MCU_API_LowPower	low_power_flag: 1 - enter in low power mode 0 - do not use low power	Instructs the firmware to use the low power when blocking procedures are called.
ST_MCU_API_WaitForInterrupt	None	The µC waits for an interrupt function. This can be a null implementation or can activate µC low power mode.
ST_MCU_API_SetSysClock	None	Sets the system clock. This function is used after waking up from low power.
ST_MCU_API_TimerCalibration	None	RTC calibration routine.
ST_MCU_API_SetEncryptionPayload	ePayload: 1 -Enable encryption 0 - Disable encryption	Enables the encryption payload flag.

This module has to call the callbacks listed below and implemented by the ST Sigfox library.

Table 3. **Callbacks exported by the RF_LIB module**

Name	Arguments	Description
ST_RF_API_S2LP_IRQ_CB	None	The RF_LIB module configures the S2-LP to raise interrupts and to notify them on a GPIO. This function must be called in case of GPIO interrupt.
ST_RF_API_Timer_CB	None	It must be called when the timer started by the MCU_API_timer_start expires.
ST_RF_API_Timer_Channel_Clear_CB	None	It must be called when the timer started by the MCU_API_timer_start_Carrier_sense expires.

The applicative callback `void Appli_Exti_CB(uint16_t GPIO_Pin)` can be implemented to demand application management of all the ETXI interrupts different from the `ST_RF_API_S2LP_IRQ_CB`.

3.5.3 Sigfox data retriever

The `MCU_API_aes_128_cbc_encrypt` function encrypts an input buffer using AES128-CBC encryption.

While the CBC algorithm IV vector should be set to 0, the encryption key is provided by Sigfox and is associated with each node.

This key must be stored and used in the MCU_API_aes_128_cbc_encrypt routine.

In the ST reference design, the key is stored in the board during the registration phase.

ST provides a compiled ID_KEY_RETRIEVERv_CMx_C.a library that exports the functions listed below.

Table 4. **ID_KEY_RETRIEVER_CMx.a functions**

Name	Arguments	Description
enc_utils_encrypt	encrypted_data: pointer to the destination buffer where the encrypted data must be stored data_to_encrypt: pointer to the source buffer where the data to encrypt are stored data_len: length of the data buffer to encrypt	Perform the AES128-CBC encryption using the AES KEY associated with the board.
enc_utils_retrieve_data	id_ptr: pointer to the 32bits word variable where the ID of the board must be stored. pac_ptr: pointer to the 8bytes array where the PAC of the board must be stored. rcz_ptr: pointer to the byte where the RCZ number of this board must be stored.	Retrieve the ID, PAC and RCZ number of the board and returns it to the caller. The ID should be used when opening the library. The PAC is used to register the node on the backend.
enc_utils_set_public_key	en: if 1 switch to the public key, if 0 (default config) use the one associated to the board.	Set the public key for encryption (used for test purposes).
enc_utils_get_id	pointer to the ID uint8_t array	Gets the ID stored in the EEPROM.
enc_utils_get_initial_pac	pointer to the PAC uint8_t array	Gets the PAC stored into the EEPROM.
enc_utils_set_test_key	en: if 1 switch to test key; if 0 reset to default	Sets the RSA test key: 0x0123456789ABCDEF0123456789ABCDEF
enc_utils_set_test_id	en: if 1 switch to test ID; if 0 reset to default	Sets the RSA test key: 0xFEDCBA98
enc_utils_set_credentials_data	Sigfox settings	Overrides credentials and RCZ.

3.5.4 Application level Sigfox API

Table 5. **Sigfox API details**

Name	Arguments	Description
SIGFOX_API_open	rcz: pointer to sfx_rc_t type representing the RCZ number (1, 2, 3 or 4).	This function opens the library initializing all state machine parameters. It does not involve the radio configuration.
SIGFOX_API_send_frame	cust_data: pointer to the data to transmit cust_data_size: size in bytes of the data to transmit (max. 12) cust_response: pointer to the buffer where to store the received payload (only if initiate_downlink_flag=1) tx_repetition: number of repetitions initiate_downlink_flag: wait for a response after transmitting	Sends the frame.
SIGFOX_API_close	None	Closes the Sigfox library, resetting its state.
SIGFOX_API_set_std_config	config_words_ptr: 3-config-word array to select the FCC channels to use. default_sigfox_channel: default channel to be used among those selected by the config_words.	Sets the standard configuration.
SIGFOX_API_get_version	version_ptr: pointer to the array where to store the lib version. version_size_ptr: size of the written version array type (MCU, RF, etc.)	Returns the library version.
SIGFOX_API_get_info	info: array containing info	Gets info

Name	Arguments	Description
SIGFOX_API_send_outofband	oob_type: type of the OOB frame to send	Sends an out-of-band frame, that is a test frame used to monitor the node parameters (voltage, temperature)
SIGFOX_API_send_bit	bit_value: bit value to send cust_response: pointer to the buffer where to store the received payload (only if initiate_downlink_flag=1) tx_repetition: number of repetitions (only if initiate_downlink_flag=1) initiate_downlink_flag: wait for a response after transmitting	This function is used to send a single bit mainly when the node seeks downlink data (not to transmit).
SIGFOX_API_start_continuous_transmission	frequency: frequency at which the signal has to be generated type: type of modulation to use in continuous mode	Executes a continuous wave or modulation depending on the parameter type
SIGFOX_API_stop_continuous_transmission	None	Stops the current continuous transmission
SIGFOX_API_send_test_frame	frequency: frequency at which the wave is generated customer_data: data to transmit customer_data_length: data length in bytes initiate_downlink_flag: flag to initiate a downlink response	This function builds a Sigfox frame with the customer payload and sends it at a specific frequency
SIGFOX_API_receive_test_frame	frequency: frequency at which the wave is generated mode: mode (AUTHENTICATION_ON or AUTHENTICATION_OFF) buffer: depends on the authentication mode: • if AUTHENTICATION_OFF : buffer is used as input to check the bit stream of the received frame • if AUTHENTICATION_ON : buffer is used as output to get the received payload timeout: timeout for the reception of a valid downlink frame rssi: RSSI of the received frame	This function waits for a valid downlink frame during timeout time and returns the data received in customer_data.
SIGFOX_API_get_device_id	dev_id: pointer where to write the device ID	This function copies the device ID to the pointer given as parameter.
SIGFOX_API_get_initial_pac	initial_pac: pointer to initial PAC	Sets initial pac
SIGFOX_API_switch_public_key	use_public_key: switch to public key if SFX_TRUE, private key else	Switches device to public or private key
SIGFOX_API_set_rc_sync_period	rc_sync_period: transmission period of the RC Sync frame (in number of 'normal' frames)	Sets the period for transmission of RC Sync frame

3.5.5 ST_RF_API

The application calls a set of functions to instruct the RF_LIB to configure the radio properly. These functions are exported by the ST_RF_API header (st_rf_api.h) and are implemented in the RF_LIB module.

Table 6. **ST_RF_API functions**

Name	Arguments	Description
ST_RF_API_set_xtal_freq	An integer with the XTAL value in Hz.	Sets the XTAL frequency of the S2-LP in Hertz (default is 50 MHz).
ST_RF_API_set_freq_offset	An integer with the RF offset value in Hz.	Sets the RF frequency offset in Hertz (default is 0 Hz).
ST_RF_API_set_tcxo	A boolean value (0 or 1).	Instructs the library to configure the S2- LP for a TCXO or for a XTAL. This is needed to configure the S2-LP oscillator registers.
ST_RF_API_set_rssi_offset	An integer with the RSSI offset value in dB.	Sets an RSSI offset for the RSSI. Very useful if the RF frontend has an LNA or to calibrate the RSSI measurement
ST_RF_API_get_rssi_offset	A pointer to the variable where the RSSI value should be stored.	Gets the RSSI offset for the RSSI
ST_RF_API_gpio_irq_pin	An integer representing the number of the GPIO to be set as an interrupt source.	Configures one of the S2-LP pin to be an IRQ pin.
ST_RF_API_gpio_tx_rx_pin	An integer representing the number of the GPIO to be set as a TX or RX state indication. 0xFF to configure no one of the S2-LP GPIO with this function.	Configures one of the S2-LP pin to be to be configured as (RX or TX) signal
ST_RF_API_gpio_rx_pin	An integer representing the number of the GPIO to be set as a RX state indication. 0xFF to configure no one of the S2-LP GPIO with this function.	Configures one of the S2-LP pin to be configured as RX signal
ST_RF_API_gpio_tx_pin	An integer representing the number of the GPIO to be set as a TX state indication. 0xFF to configure no one of the S2-LP GPIO with this function.	Configures one of the S2-LP pin to be configured as TX signal.
ST_RF_API_reduce_output_power	Power reduction in half dB	Reduces the output power of the transmitted signal by a factor (reduction*0.5 dB against the actual value)
ST_RF_API_smps	SMPS voltage word	Instructs the library to configure the S2-LP with a user defined smps frequency
ST_RF_API_set_pa	A boolean value (0 or 1).	Instructs the library to configure the S2- LP for a external PA (Power Amplifier).
ST_RF_API_get_ramp_duration	None	Returns the duration of the initial (or final) ramp in ms
ST_RF_API_Get_Transmission_State	None	Returns information about the TX state of the MCU API

3.5.6 Opening the Sigfox library

`SIGFOX_API_open` must be called to initialize the library before performing any other operation.

This API requires pointer to the Radio Configuration zone structure to be used.

Uplink frequencies are:

- RCZ1 - 868.13 MHz
- RCZ2 - 902.2 MHz
- RCZ3 - 923.3 MHz
- RCZ4 - 920.8 MHz

Note: As frequency hopping is implemented, the transmission frequency is not fixed.

Downlink frequencies are:

- RCZ1 - 869.525 MHz
- RCZ2 - 905.2 MHz
- RCZ3 - 922.2 MHz
- RCZ4 - 922.3 MHz

3.5.7 Sending the frames

`SIGFOX_API_send_frame` is the core Sigfox library function; this blocking function handles message exchange between the node and the base-stations.

3.5.8 Setting the standard configuration

This function has different purposes according to the RC mode used for the serial port opening.

FCC allows the transmitters to choose certain macro channels to implement a frequency hopping pattern allowed by the standard.

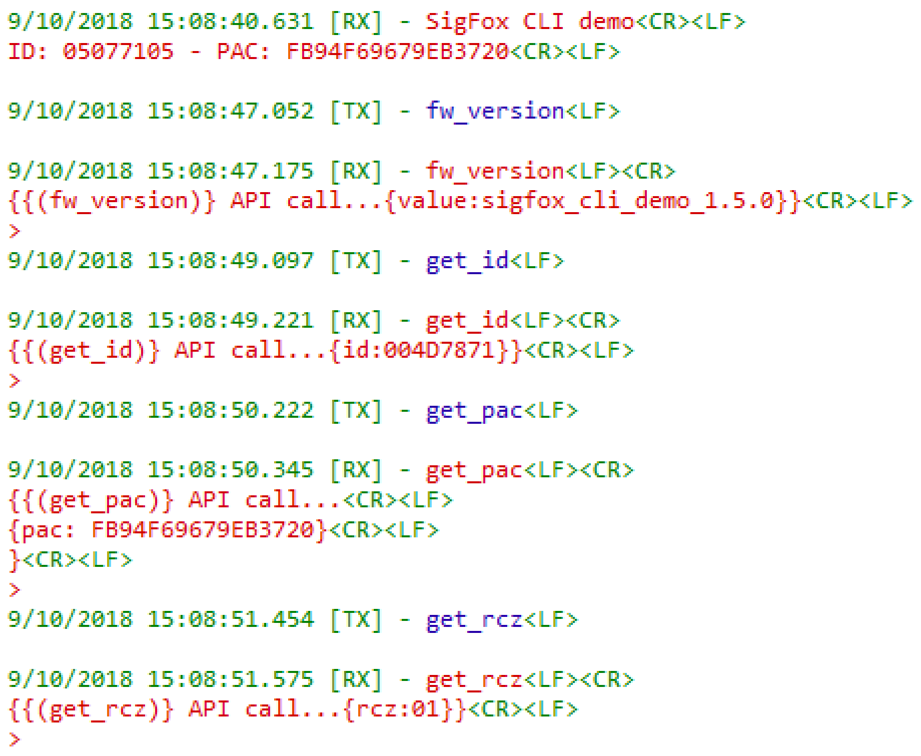
The channel map is specified in the first argument of `SIGFOX_API_set_std_config`, which consists of an array of three 32-bit configuration words.

3.5.9 Running CLI demo commands via terminal

For the CLI demo, some commands can be run via any terminal utility.

The command list can be retrieved using the help command in the terminal window (for the complete command list, refer to UM2169, Section 4.3.1, on www.st.com).

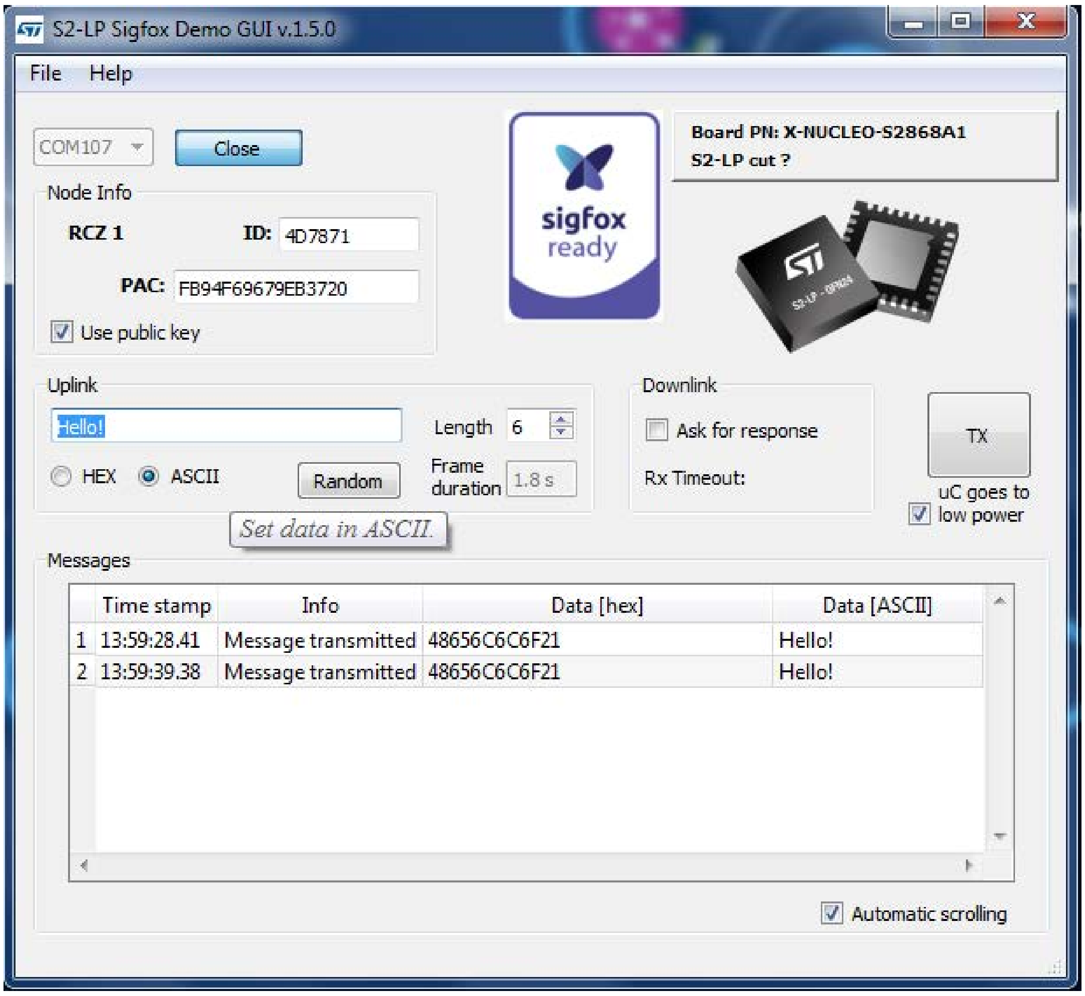
Figure 4. **SigFox CLI demo: running commands via terminal**



```
9/10/2018 15:08:40.631 [RX] - SigFox CLI demo<CR><LF>
ID: 05077105 - PAC: FB94F69679EB3720<CR><LF>

9/10/2018 15:08:47.052 [TX] - fw_version<LF>

9/10/2018 15:08:47.175 [RX] - fw_version<LF><CR>
{{(fw_version)} API call...{value:sigfox_cli_demo_1.5.0}}<CR><LF>
>
9/10/2018 15:08:49.097 [TX] - get_id<LF>

9/10/2018 15:08:49.221 [RX] - get_id<LF><CR>
{{(get_id)} API call...{id:004D7871}}<CR><LF>
>
9/10/2018 15:08:50.222 [TX] - get_pac<LF>

9/10/2018 15:08:50.345 [RX] - get_pac<LF><CR>
{{(get_pac)} API call...<CR><LF>
{pac: FB94F69679EB3720}<CR><LF>
}<CR><LF>
>
9/10/2018 15:08:51.454 [TX] - get_rcz<LF>

9/10/2018 15:08:51.575 [RX] - get_rcz<LF><CR>
{{(get_rcz)} API call...{rcz:01}}<CR><LF>
>
```

Once the board is programmed using the CLI mode demo, it can be connected to the GUI.

Figure 5. **Running the CLI demo via GUI**



S2-LP Sigfox Demo GUI v.1.5.0

File Help

COM107 Close

Board PN: X-NUCLEO-S2868A1
S2-LP cut ?

Node Info

RCZ 1 ID: 4D7871

PAC: FB94F69679EB3720

☑ Use public key

Uplink

Hello! Length 6

○ HEX ● ASCII Random Frame duration 1.8 s

Set data in ASCII.

Downlink

☐ Ask for response

Rx Timeout:

TX

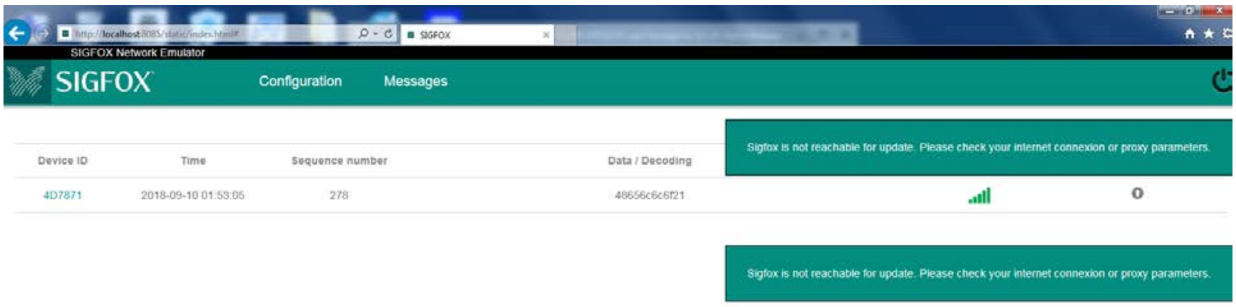
☑ uC goes to low power

Messages

	Time stamp	Info	Data [hex]	Data [ASCII]
1	13:59:28.41	Message transmitted	48656C6C6F21	Hello!
2	13:59:39.38	Message transmitted	48656C6C6F21	Hello!

☑ Automatic scrolling

Figure 6. **Sigfox message captured**



http://localhost:8085/static/index.html#

SIGFOX Network Emulator

SIGFOX Configuration Messages

Device ID	Time	Sequence number	Data / Decoding
4D7871	2018-09-10 01:53:05	278	48656c6c6f21

Sigfox is not reachable for update. Please check your internet connexion or proxy parameters.

Sigfox is not reachable for update. Please check your internet connexion or proxy parameters.

4 System setup guide

4.1 Hardware description

4.1.1 STM32 Nucleo platform

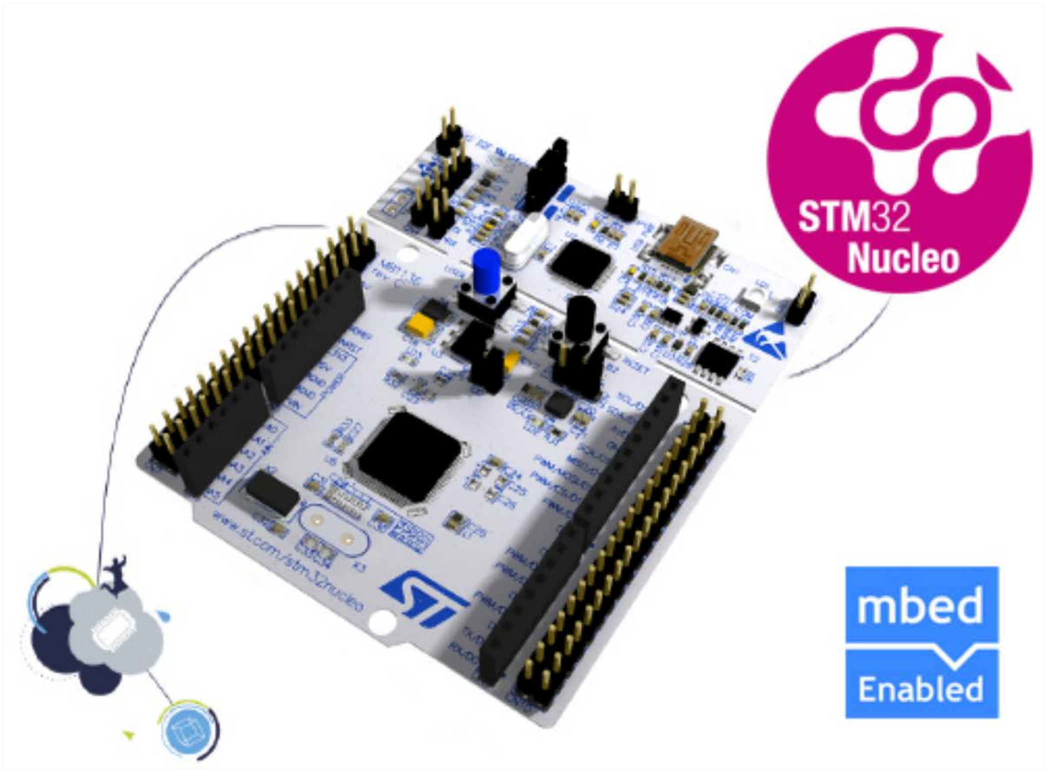
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

The Arduino™ connectivity support and ST morpho connectors make it easy to expand the functionality of the STM32 Nucleo open development platform with a wide range of specialized expansion boards to choose from.

The STM32 Nucleo board does not require separate probes as it integrates the ST-LINK/V2-1 debugger/ programmer.

The STM32 Nucleo board comes with the comprehensive STM32 software HAL library together with various packaged software examples.

Figure 7. **STM32 Nucleo board**



Information regarding the STM32 Nucleo board is available at www.st.com/stm32nucleo

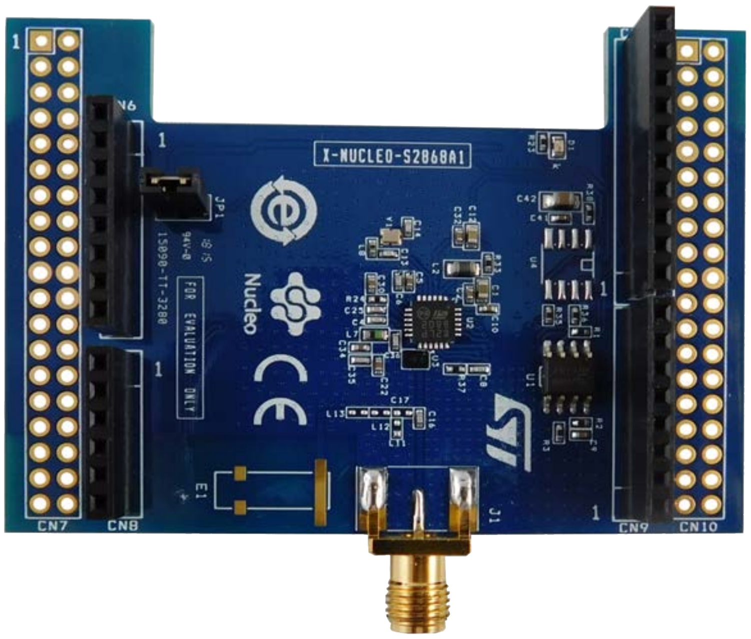
4.1.2 X-NUCLEO-S2868A1 expansion board

The X-NUCLEO-S2868A1 expansion board is based on the S2-LP radio and operates in the 868 MHz ISM frequency band.

The expansion board is compatible with ST morpho and Arduino UNO R3 connectors.

The X-NUCLEO-S2868A1 interfaces with the STM32 Nucleo microcontroller via SPI connections and GPIO pins. You can change some of the GPIOs by mounting or removing the resistors.

Figure 8. X-NUCLEO-S2868A1 expansion board



4.2 Hardware setup

The following hardware components are needed:

1. One STM32 Nucleo development platform (NUCLEO-L053R8,NUCLEO-L152RE or NUCLEO-L476RG)
2. One S2-LP expansion board (order code: X-NUCLEO-S2868A1)
3. One USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

4.2.1 STM32 Nucleo and X-NUCLEO-S2868A1 expansion board setup

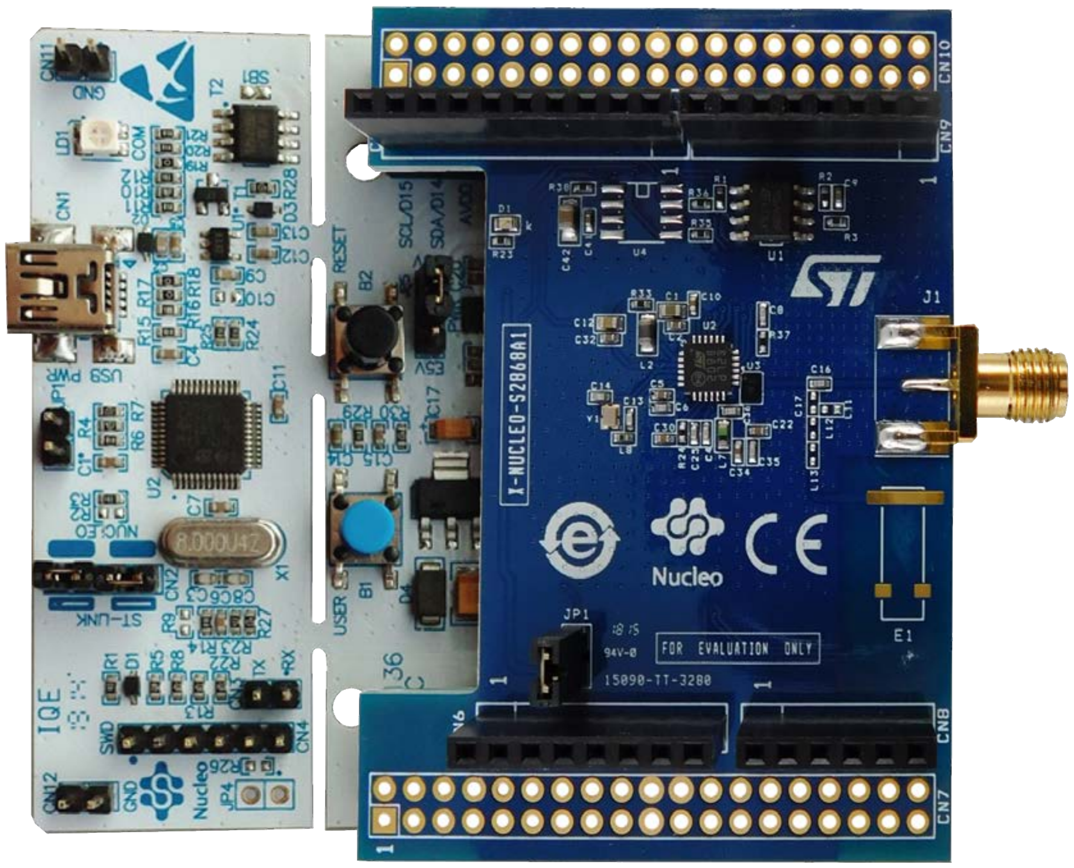
The STM32 Nucleo board integrates the ST-LINK/V2-1 debugger/programmer.

The developer can download the ST-LINK/V2-1 USB driver by looking for the STSW-LINK009 software on www.st.com.

The S2-LP X-NUCLEO-S2868A1 expansion board can interface the external STM32 microcontroller on the STM32 Nucleo using SPI.

It also can be easily connected to the STM32 Nucleo through the Arduino UNO R3 extension connector as shown below.

Figure 9. **X-NUCLEO-S2868A1 expansion board connected to the STM32 Nucleo board**



4.3 Software setup

The following software components are needed for a suitable development environment for creating applications for the STM32 Nucleo equipped with the S2-LP X-NUCLEO-S2868A1 expansion board:

- X-CUBE-SFXS2LP1 expansion for STM32Cube dedicated to Sigfox applications development.
- One of the following development tool-chain and compilers:
  - Keil RealView Microcontroller Development Kit (MDK-ARM-STM32) + ST-LINK
  - IAR Embedded Workbench for ARM (IAR-EWARM) + ST-LINK
  - OpenSTM32 System Workbench for STM32 (SW4STM32) + ST-LINK

Revision history

Table 7. **Document revision history**

Date	Version	Changes
09-Oct-2018	1	Initial release.

Contents

1 Acronyms and abbreviations 2

2 What is STM32Cube? 3

2.1 STM32Cube architecture 3

3 X-CUBE-SFXS2LP1 software expansion for STM32Cube 5

3.1 Overview 5

3.2 Architecture 5

3.3 Folder structure 6

3.4 APIs 7

3.5 Sample application description 7

3.5.1 RF_LIB library 7

3.5.2 MCU_API module 7

3.5.3 Sigfox data retriever 9

3.5.4 Application level Sigfox API 10

3.5.5 ST_RF_API 11

3.5.6 Opening the Sigfox library 12

3.5.7 Sending the frames 13

3.5.8 Setting the standard configuration 13

3.5.9 Running CLI demo commands via terminal 13

4 System setup guide 15

4.1 Hardware description 15

4.1.1 STM32 Nucleo platform 15

4.1.2 X-NUCLEO-S2868A1 expansion board 15

4.2 Hardware setup 16

4.2.1 STM32 Nucleo and X-NUCLEO-S2868A1 expansion board setup 16

4.3 Software setup 17

Revision history 18

List of tables

Table 1. List of acronyms 2
Table 2. MCU_API module details 8
Table 3. Callbacks exported by the RF_LIB module 9
Table 4. ID_KEY_RETRIEVER_CMx.a functions 10
Table 5. Sigfox API details. 10
Table 6. ST_RF_API functions 12
Table 7. Document revision history 18

List of figures

Figure 1. Firmware architecture 3
Figure 2. X-CUBE-SFXS2LP1 architecture 6
Figure 3. X-CUBE-SFXS2LP1 package folder structure 6
Figure 4. SigFox CLI demo: running commands via terminal 13
Figure 5. Running the CLI demo via GUI 14
Figure 6. Sigfox message captured. 14
Figure 7. STM32 Nucleo board 15
Figure 8. X-NUCLEO-S2868A1 expansion board 16
Figure 9. X-NUCLEO-S2868A1 expansion board connected to the STM32 Nucleo board 17