# UM2245
# User manual

## Sigfox™ software expansion for STM32Cube

## Introduction

This user manual describes the Sigfox™ embedded expansion software implementation on the CMWX1ZZABZ-091 or CMWX1ZZABZ-078 Murata modules. X-CUBE-SFOX is the name of the corresponding STM32Cube Expansion Package.

This user manual also explains how to interface with the Sigfox™ API from an application point of view.

Sigfox™ is a type of wireless telecommunication network designed to allow long-range communication at very low bit rates and enable the use of long-life battery-operated sensors. The Sigfox Stack™ library from Sigfox™ manages the channel access and security protocol that ensures interoperability with the Sigfox™ network.

The Sigfox™ software expansion for CMWX1ZZABZ modules is Sigfox Verified™.

X-CUBE-SFOX main features are:

- Application integration ready
- Easy add-on of the low-power LoRa® solution
- Extremely low CPU load
- No latency requirements
- Small STM32 memory footprint
- Low-power timing services provided

The X-CUBE-SFOX software is based on the STM32Cube HAL drivers (refer to *Chapter 3: X-CUBE-SFOX stack description*).

This user manual targets the following tool:

- B-L072-LRAWAN1 STM32 Discovery kit embedding the CMWX1ZZABZ-091 module from Murata. CMWX1ZZABZ-091 embeds the STM32L072 microcontroller.

# Contents

# List of tables

# List of figures

# 1 Overview

The X-CUBE-SFOX STM32Cube Expansion Package runs on STM32 32-bit microcontrollers based on the Arm®(a) Cortex®-M processor.

## 1.1 Acronyms and abbreviations

**Table 1. List of acronyms and abbreviations**

| Term | Definition |
|---|---|
| DC | Duty cycle |
| FH | Frequency hopping |
| LBT | Listen before talk |
| NA | Not applicable |
| RC | Regional configuration |
| RSSI | Received signal strength indication |
| Rx | Reception |
| SDR | Software defined radio |
| Tx | Transmission |

## 1.2 Reference

1. B-L072Z-LRWAN1 Discovery kit technical documentation such as the *Discovery kit for LoRaWAN™ and LPWAN protocols with STM32L0* data brief (DB3090) available on the www.st.com web site
2. Sigfox™ *Radio Signal Analyzer User guide* available on https://resources.sigfox.com

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

# 2 Sigfox™ standard overview

## 2.1 Overview

This section provides a general overview of Sigfox™ focusing in particular Sigfox™ end-device which is the core subject of this user manual.

Sigfox™ is a wireless telecommunication Network operator designed to allow long range communication at a low bit-rate enabling long-life battery operated sensors. The Sigfox™ software expansion includes the Sigfox Stack™ library delivered by Sigfox™. The Sigfox™ software expansion named X-CUBE-SFOX loaded on B-L072Z-LRWAN1 is Sigfox Verified™.

Sigfox™ limits the use of its network to 144 messages per day and per device. Each message can be from 1 bit up to 12 bytes.

## 2.2 End-device hardware architecture

The end-device is made of an RF transceiver (also known as radio) and a host STM32L072. The RF transceiver is composed of a modem and an RF up-converter. The built-in modulator does not support Sigfox™ BPSK modulation; therefore it is implemented inside the STM32L072.

The STM32L072 implements the radio driver, the Sigfox™ modulator, the Sigfox Stack™ libraries and the application layer.

## 2.3 Regional radio resource

The European, North American and Asian markets have different spectrum allocations and regulatory requirements. Sigfox™ has split requirements in Region Configurations (RC) as listed in *Table 2*.

**Table 2. Region country list**

| Region configuration | Country |
|---|---|
| RC1 | Europe countries, Oman, Lebanon, South Africa, Kenya |
| RC2 | USA, Canada, Mexico |
| RC3c | Japan |
| RC4 | Brazil, Colombia, Peru, New–Zealand, Australia, and Singapore |

*Table 3* provides an overview of the regulatory requirements for the region configurations.

**Table 3. Region configuration RF parameters**

| Parameters | RC1 | RC2 | RC3c | RC4 |
|---|---|---|---|---|
| Frequency band downlink (MHz) | 869.525 | 905.2 | 922.2 | 922.3 |
| Frequency band uplink (MHz) | 868.130 | 902.2 | 923.2 | 920,8 |
| Uplink modulation | DBPSK | DBPSK | DBPSK | DBPSK |

**Table 3. Region configuration RF parameters (continued)**

| Parameters | RC1 | RC2 | RC3c | RC4 |
|---|---|---|---|---|
| Downlink modulation | GFSK | GFSK | GFSK | GFSK |
| Uplink data-rate | 100 | 600 | 100 | 600 |
| Downlink data-rate | 600 | 600 | 600 | 600 |
| Max output power (dBm) | 14 | 22 | 13 | 22 |
| Medium access | Duty cycle 1% | Frequency hopping<br>Max On time 400 ms / 20 s | Carrier sense | Frequency hopping<br>Max On time 400 ms / 20 s |
| CS center frequency (MHz) | NA | NA | 923.2 | NA |
| CS bandwidth (kHz) | NA | NA | 200 | NA |

## 2.4 Tx/Rx radio time diagram

The end-device transmits data to the network in an asynchronous manner. This is due to the fact that transmission data is only sent per device-report event. *Figure 1* and *Figure 2* depict the timing sequences with and without a downlink.

*Note: The presence of a downlink only depends on device configuration.*

The three transmissions Tx1, Tx2 and Tx3 contain the same payload information. These consecutive transmissions only maximize the probability of a correct reception by the network. When the device observes good link quality to the network, it may decide to send only Tx1 to save power consumption only if downlink frame is requested. The API to select preferred scheme is described in *Section 4.1.3: Sending frames/bits on page 17*.

**Figure 1. Timing diagram for uplink only**

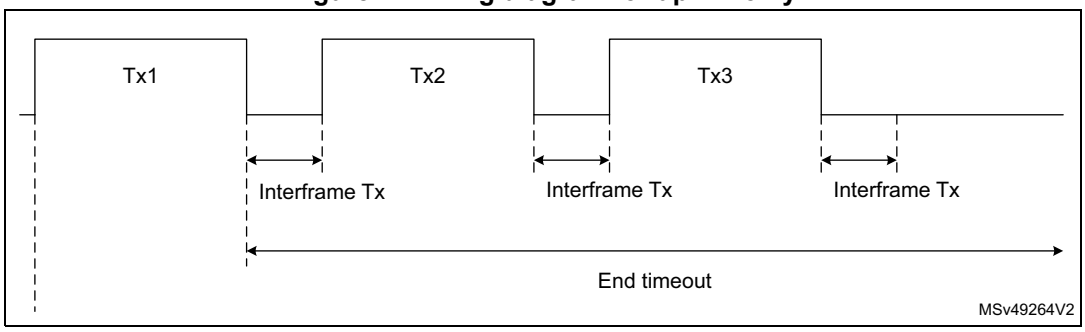
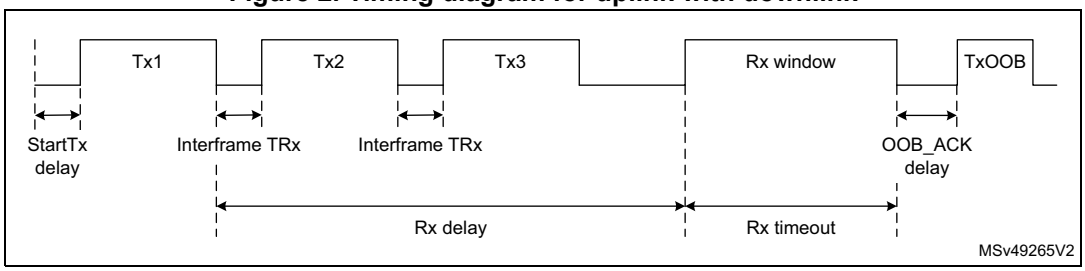



**Figure 2. Timing diagram for uplink with downlink**



The timings shown in *Figure 1* and *Figure 2* are given in *Table 4* for the various regional configurations.

**Table 4. Timings**

| | StartTx delay | Interframe Tx/TRx | Rx delay | Rx timeout | OOB_ACK delay | End timeout |
|---|---|---|---|---|---|---|
| RC1 | 0 s | 500 ms | 20 s | 25 s | 1.4 s | NA |
| RC2 | 0 s | 500 ms | 20 s | 25 s | 1.4 s | 10 s |
| RC3c | 100 ms max (LBT) | 500 ms + LBT | 19 s | 34 s | 1.4 s | NA |
| RC4 | 0 s | 500 ms | 20 s | 25 s | 1.4 s | 10 s |

The Tx periods depend on the number of bytes sent and on the RC zone:

- It takes 10 ms to send a bit in RC1 and RC3c.
- It takes 1.66 ms to send a bit in RC2 and RC4.

A message can be 26-byte long at the most (including sync word, header, and payload data). Therefore, for RC1, a Tx period can be maximum 26 x 8 x 10 ms = 2.08 s.

## 2.5 Listen before talk (LBT)

In RC3c, LBT is mandatory before any transmission. The device must listen and check if the channel is free. The channel is considered as free if the power within a 200 kHz bandwidth stays below -80 dBm for 5 ms.

When the channel is free, the device starts a transmission. The transmission is not started otherwise.

# 3 X-CUBE-SFOX stack description

## 3.1 Overview

The X-CUBE-SFOX package contains:

- The Sigfox™ middleware
- The board support package composed of:
  - B-L072-LRAWAN1 Discovery board drivers
  - CMWX1ZZABZ Murata module drivers (MLM32L07X01)
  - sx1276 radio drivers
  - Sensor ST drivers
- The CMSIS and STM32L0xx HAL drivers
- Two Sigfox™ application examples

The Sigfox™ middleware for STM32 microcontrollers is split into several modules:

- Sigfox™ Core library
- Sigfox™ Addon Library (for Sigfox™ testing purpose)
- Sigfox™ Cmac Library, which manages frame security
- Sigfox™ utilities module (such as timer server, power management and scheduler)

The application layer, which hosts the two examples *Sgfx_ModemAT* and *SGfx_push_button*, includes a specific common directory for the CMWX1ZZABZ Murata module with:

- The Sigfox™ modulation library for the sx1276 radio
- The Credentials library, which manages Sigfox™ credentials for the module

The Sigfox™ Core library implements a Sigfox™ medium access controller from Sigfox™. It interfaces with the Cmac library encrypting uplink payload and verifying downlink payload. The Cmac library interfaces with the Credentials library holding the cryptographic functions. It also interfaces with the lower layer Sigfox™ modem designed to drive the sx1276 radio. The Sigfox™ library, cryptographic library and modem library modules are provided as compiled objects.

*Figure 7 on page 27* provides an overview of the structure of the project files.

## 3.2 Sigfox™ certification

The *Sigfox Addon Verified* library, as well as the *RF_API* and *MCU_API*, have been verified by Sigfox™ Test Lab. Nevertheless, the end product based on the CMWX1ZZABZ module must pass the Sigfox Ready™ certification before its commercialization.

## 3.3 Features

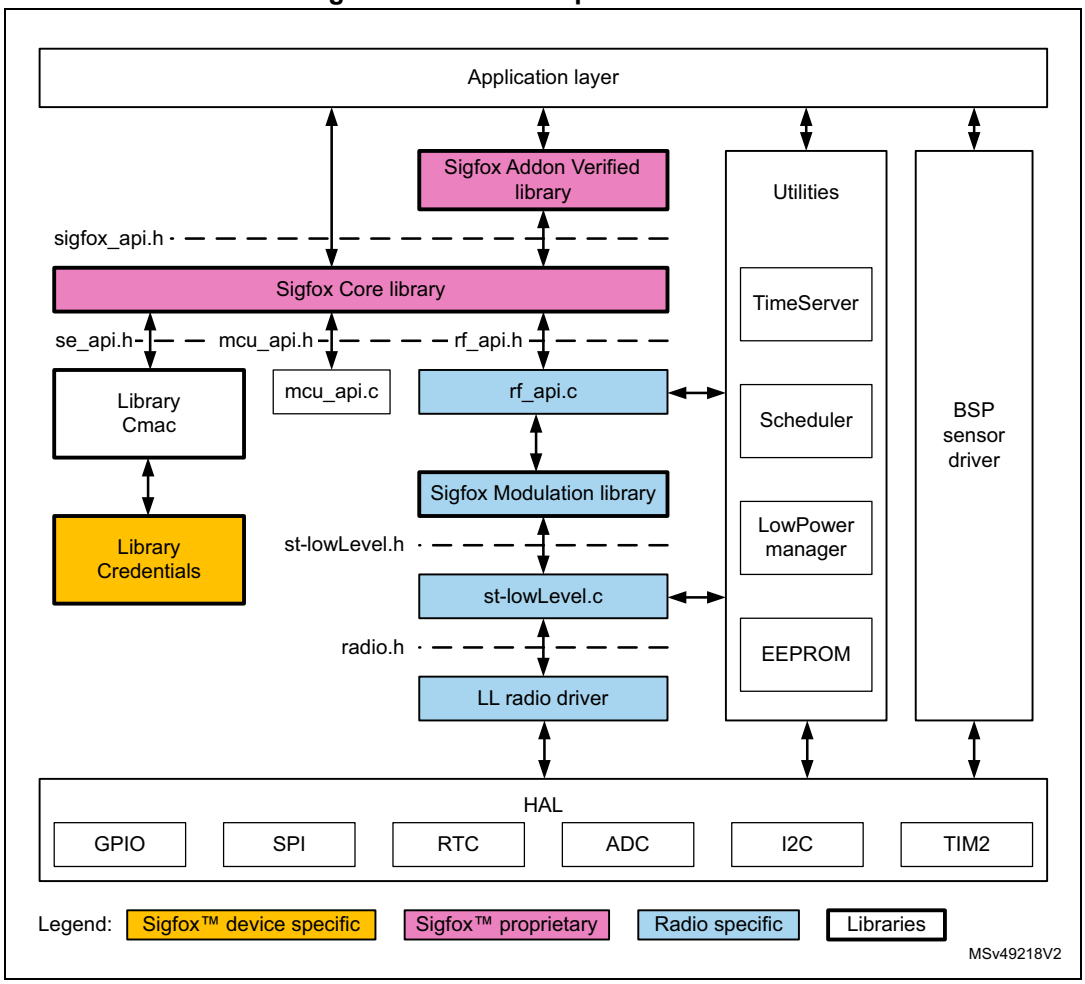
The X-CUBE-SFOX package main features are:

- Sigfox Verified™
- RC1, RC2, RC3c, and RC4 support
- Low-power optimization
- Low real-time constraints even during modulation
- Support of the CMWX1ZZABZ-091 module

## 3.4 Architecture

*Figure 3* depicts the top-level firmware structure of the X-CUBE-SFOX application.

**Figure 3. Firmware top-level structure**




The HAL uses cube API to drive the STM32L072 hardware required by the application.

The RTC provides a centralized time unit that continues to run even in the low-power Stop mode. The RTC alarm is used to wake up the system at specific times managed by the timer server.

The radio driver uses the SPI and the GPIO HW (as shown in *Figure 3*) to control the radio, it also provides a set of API to be used by higher level software. Only the low level radio driver is used in X-CUBE-SFOX. The radio driver is split in two parts:

- The *\BSP\Components\sx1276\sx1276.c* file contains all functions which are radio component dependent only.
- The *\BSP\MLM32L07X01\mlm32l07x01.c* file contain all the radio module dependent functions.

*Note: The sx1276.c and the mlm32l07x01.c files are the same as in the I-CUBE-LRWAN package.*

The Common\cwmx1zzabz\Radio_Sigfox_Driver*st-lowlevel.c* file implements the interface between library Sigfox™ modulation *SgfxSTModemSx1276V123_CM0_IDE_O3.lib* and the sx1276 radio driver. It also interfaces interrupt to the handler function.

*mcu_api.c* interfaces with non-volatile memory to save data in EEPROM, and implements the timer management and energy-efficient wait function.

The Sigfox™ modulation library is called by *rf_api.c*. It starts and stops the radio using the proper Tx and Rx configuration. It implements the modulation using the double buffer DMA to relax timing constraints on the modulation. This library is radio dependent. More information is provided in *Section 4.2: Library Sigfox™ modulation on page 19*.

The Sigfox™ Core library embeds the medium access controller (MAC). More information is provided in *Section 4.1: Sigfox™ Core library on page 15*.

The application is built around an infinite loop including a scheduler. The scheduler process tasks and events. When nothing remains to be done, the scheduler transitions to idle state and calls the low-power manager.

A typical application can be for instance:

- an AT layer to interface with external Host (refer to *Section 3.4: Architecture*)
- any application reading and sending sensor data upon an action (refer to *Section 5.4: Push-button application on page 35*)

## 3.5 Required STM32 peripherals to drive the radio

### 3.5.1 Radio reset

One MCU GPIO f is used to reset the radio. A reset of the radio is performed before any radio activity.

### 3.5.2 Radio SPI

The sx1276 radio registers are accessed through the SPI bus at 8 Mbps.

Radio NSS is controlled in two ways: during Tx modulation periods, the NSS pin is set as an alternate function and mapped on Timer2 output compare; the rest of the time, it is set by software through a GPIO.

### 3.5.3 RTC

The RTC calendar is used as 32-bit counter running in all power modes from the 32 kHz external oscillator. By default, it is programed to provide 1024 ticks (sub-seconds) per second. It is programed once at the initialization of the hardware when the STM32L072

starts for the first time. Its output is limited to a 32-bit timer which corresponds to about a 48-day period.

**Caution:** When changing the tick duration, the user shall keep it below 1ms.

### 3.5.4 TIMER2

It is used during the modulation (Tx mode). It is used as NSS source and triggers the DMA to transfer data to the SPI peripheral. This ensures phase synchronization between the SPI NSS state and SPI data, and allows the Arm® core to enter the Sleep mode while modulation data is sent to the radio.

### 3.5.5 Interrupt

In the Tx mode, the DMA interrupt has the highest priority. All other interrupts must be lower-level interrupts to ensure proper Sigfox™ transmit mask.

*Note: In the Tx mode, no interrupt line is needed.*

In the Rx mode, two radio interrupt lines are dedicated to receive the interrupts from the radio.

- DIO0 is used to signal that the LoRa® radio has successfully received a radio packet (RxDone)
- DIO4 is used to signal that the radio has successfully detected a synchronization, and therefore that the STM32 can read RSSI.

The priority levels are defined in file *hw_conf.h*.

# 4 X-CUBE-SFOX middleware programming guidelines

## 4.1 Sigfox™ Core library

Embedded applications using the Sigfox™ Core library call SIGFOX_APIs to manage communication. The content of SIGFOX_APIs is described in *Table 5*.

**Table 5. Application level Sigfox™ APIs**

| Function | Description |
|---|---|
| `sfx_error_t SIGFOX_API_get_device_id (sfx_u8 *dev_id);` | Copies the ID of the device to the pointer given in parameter.<br>The ID is 4-byte long and in binary format. |
| `sfx_error_t SIGFOX_API_get_initial_pac (sfx_u8 *initial_pac);` | Gets the value of the PAC stored in the device. This value is used when the device is registered for the first time on the backend.<br>The PAC is 8-byte long. |
| `sfx_error_t SIGFOX_API_open (sfx_rc_t *rc);` | Initializes the library. It saves the input parameters once (cannot be changed until `SIGFOX_API_close()` is called)<br>– `rc`: pointer on the radio configuration zone. It is mandatory to use already existing defined RCs. |
| `sfx_error_t SIGFOX_API_close (void);` | Closes the library and stops the RF. |
| `sfx_error_t SIGFOX_API_send_frame (sfx_u8 *customer_data, sfx_u8 customer_data_length, sfx_u8 *customer_response, sfx_u8 tx_repeat, sfx_bool initiate_downlink_flag);` | Sends a standard Sigfox™ frame with customer payload.<br>– `customer_data`: payload cannot exceed 12 bytes.<br>– `customer_data_length`: length in bytes.<br>– `customer_response`: received response.<br>– `tx_repeat`: when 0, sends one Tx, when 1, sends three Tx.<br>– `initiate_downlink_flag`: if set, the frame sent is followed by a receive downlink frame and an out-of-band Tx frame (voltage, temperature and RSSI). |

**Table 5. Application level Sigfox™ APIs (continued)**

| Function | Description |
|---|---|
| `sfx_error_t SIGFOX_API_send_bit (sfx_bool bit_value, sfx_u8 *customer_response, sfx_u8 tx_repeat, sfx_bool initiate_downlink_flag);` | Sends a standard Sigfox™ frame with null customer payload. This frame is the shortest frame that Sigfox™ library can generate.<br>– `bit_value`: bit sent.<br>– `customer_response`: received response.<br>– `tx_repeat`: when 0, sends one Tx, when 1, sends three Tx.<br>– `initiate_downlink_flag`: if set, the frame sent is followed by a receive downlink frame and an out-of-band Tx frame (voltage, temperature and RSSI). |
| `sfx_error_t SIGFOX_API_set_std_config (sfx_u32 config_words[3], sfx_bool timer_enable);` | Configures specific variables for standard.<br>Parameters have different meanings whether in FH or LBT mode.<br>*Note: this function has no influence in DC.*<br>*See Section 5.3.17: ATS400 to configure specific variables for standard on page 35 for details.* |

Secondary APIs are described in *sigfox_api.h*. The library can be found in directory *Middlewares\Third_Party\SigfoxLib*.

### 4.1.1 Opening the library

`ST_SIGFOX_API_open` must be called to initialize the library before any other operation is performed.

This API requires the RC argument number representing the radio configuration zone. Refer to *Section 2.3: Regional radio resource on page 8* for details.

For radio control zones 2 and 4, FCC requires frequency hopping so the transmission frequency won't be fixed. Refer to *Section 4.1.4: Set standard configuration for FH* for the mapping of the macro channels.

### 4.1.2 *Sigfox Addon Verified* library

This library is used to test the device for Sigfox Verified™ certification. Ultimately, this library can be removed from the build once certified. The content of the *Sigfox Addon Verified* library is described in *Table 6*.

**Table 6. *Sigfox Addon Verified* library**

| Function | Description |
|---|---|
| `sfx_error_t ADDON_SIGFOX_VERIFIED_API_test_mode(sfx_rc_enum_t rc_enum, sfx_test_mode_t test_mode);` | Executes the test modes needed for the Sigfox Verified™ certification:<br>– `rc_enum`: rc at which the test mode is run.<br>– `test_mode`: test mode to run. |

This library is located in *Middlewares\Third_Party\Sgfx\SigfoxLibTest*.

### 4.1.3 Sending frames/bits

ST_SIGFOX_API_send_frame is the main Sigfox™ library function. This blocking function handles message exchange between the end node and the base stations. An important parameter of this function is the initiate_downlink_flag which selects different transmission behaviors:

- When the initiate_downlink_flag is 0, the library requests only uplink frame. The sent frame is transmitted once if tx_repeat equal to 0, or three times if tx_repeat is different from 0 with a 500 ms pause.(refer to *Figure 1 on page 9*).The transmit payload can be maximum 12-byte long.
- When the initiate_downlink_flag is 1, the frame to be sent is transmitted three times with a 500 ms pause. A 25 s Rx window then opens 20 s after the end of the first repetition (*Figure 2 on page 9* illustrates such a sequence).

If the reception is successful, the received 8-byte downlink frame is stored in the buffer location indicated by the customer_response buffer.

### 4.1.4 Set standard configuration for FH

The FCC allows the transmitters to choose certain macro channels to implement a frequency hopping pattern authorized 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.

A macro-channel consists of 6 micro channels centered about the center frequency of the macro channel and separated by 25 kHz. For example, in the 902.2 MHz macro channel, the 6 micro channels are 902.1375 MHz, 902.1625 MHz, 902.1875 MHz, 902.2125 MHz, 902.2375 MHz, and 902.2625 MHz.

A typical Sigfox™ frame lasts between 200 ms and 350 ms at 600 bps, and FCC mandates a max dwell time of 400 ms. A transmitter cannot return to a given channel before 20 s. Therefore, at least 20 / 0.4 = 50 channels must be used for continuous transmission.

Actually, a device only transmits a few frames per day (144 messages maximum). Enabling one macro channel only and inserting 10-s delays between 2 groups of 3 repeated frames (1 frame per micro channel means 6 micro channels) pass the regulation limits.

Each bit of the config_words[0,1,2] array represents a macro channel according to the mapping described in *Table 7*.

**Table 7. Macro channel mapping**

| Bit | config_words[0] Frequency mapping (MHz) | config_words[1] Frequency mapping (MHz) | config_words[2] Frequency mapping (MHz) |
|---|---|---|---|
| 0 | 902.2 | 911.8 | 921.4 |
| 1 | 902.5 | 912.1 | 921.7 |
| 2 | 902.8 | 912.4 | 922 |
| 3 | 903.1 | 912.7 | 922.3 |
| 4 | 903.4 | 913 | 922.6 |
| 5 | 903.7 | 913.3 | 922.9 |
| 6 | 904 | 913.6 | 923.2 |

Table 7. Macro channel mapping (continued)

| Bit | config_words[0] Frequency mapping (MHz) | config_words[1] Frequency mapping (MHz) | config_words[2] Frequency mapping (MHz) |
|---|---|---|---|
| 7 | 904.3 | 913.9 | 923.5 |
| 8 | 904.6 | 914.2 | 923.8 |
| 9 | 904.9 | 914.5 | 924.1 |
| 10 | 905.2 | 914.8 | 924.4 |
| 11 | 905.5 | 915.1 | 924.7 |
| 12 | 905.8 | 915.4 | 925 |
| 13 | 906.1 | 915.7 | 925.3 |
| 14 | 906.4 | 916 | 925.6 |
| 15 | 906.7 | 916.3 | 925.9 |
| 16 | 907 | 916.6 | 926.2 |
| 17 | 907.3 | 916.9 | 926.5 |
| 18 | 907.6 | 917.2 | 926.8 |
| 19 | 907.9 | 917.5 | 927.1 |
| 20 | 908.2 | 917.8 | 927.4 |
| 21 | 908.5 | 918.1 | 927.7 |
| 22 | 908.8 | 918.4 | 928 |
| 23 | 909.1 | 918.7 | 928.3 |
| 24 | 909.4 | 919 | 928.6 |
| 25 | 909.7 | 919.3 | 928.9 |
| 26 | 910 | 919.6 | 929.2 |
| 27 | 910.3 | 919.9 | 929.5 |
| 28 | 910.6 | 920.2 | 929.8 |
| 29 | 910.9 | 920.5 | 930.1 |
| 30 | 911.2 | 920.8 | 930.4 |
| 31 | 911.5 | 921.1 | 930.7 |

A macro channel is only enabled when the corresponding config_words[] bit is set to 1. For example, bit 0 of config word[0] corresponds to channel 1 while bit 30 of config_word[1] corresponds to channel 63. At least nine macro channels must be enabled to meet the FCC specifications.

Long message configuration example:

- config_words[0] = [0x0000 01FF]
- config_words[1] = [0x0000 0000]
- config_words[2] = [0x0000 0000]

In this example, channels 1 to 9 are enabled with frequencies ranging from 902.2 MHz to 904.6 MHz.

By default the X-CUBE-SFOX sets one macro channel with timer_enable set to 1. Macro channel 1 in RC2 (operational frequency is 902.2 MHz) and macro channel 63 in RC4 (operational frequency is 920.8 MHz). This is the short message configuration operational for Sigfox™ (see defined value `RCx_SM_CONFIG` in *sigfox_api.h*).

A delay (timer_enable) is implemented to avoid one micro channel to be re used with an interval lower than 20 s. When using one macro channel only (6 micro channels) performing 3 repetitions, this delay corresponds to 10 s. When using two macro channels (12 micro channels) the delay automatically becomes 5 s.

For certification test purposes, timer_enable may be set to 0, but shall be set to 1 otherwise. The default settings can nevertheless be modified using the ATS400 command (refer to *Section 5.3.17: ATS400 to configure specific variables for standard on page 35*) to speed up the certification process.

## 4.2 Library Sigfox™ modulation

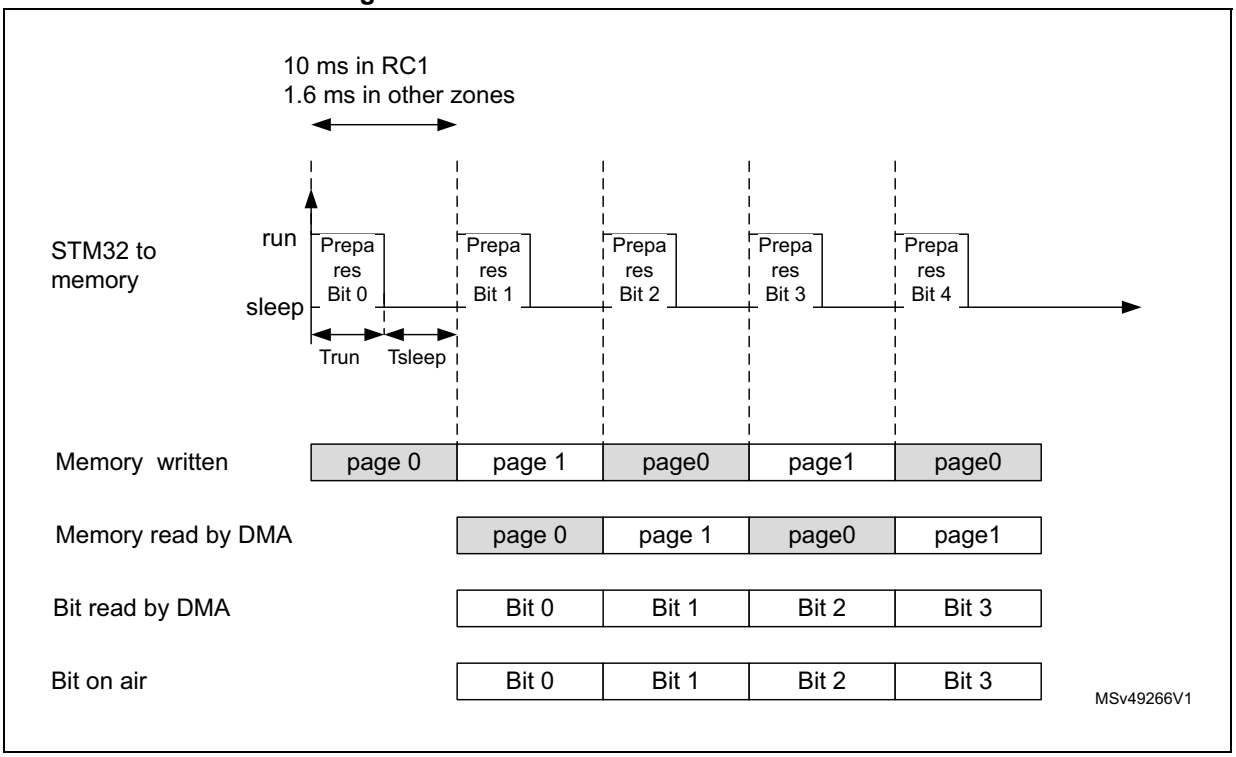
The library is used to configure the radio in the transmit or receive mode. It also provides the interface to the STM32 hardware such as time, interrupts and the sx1276 radio.

This library is radio specific. It can therefore only be used with sx1276 Semtech Radio.

The DMA is used to let the real time modulation run with minimum real time constraint. Indeed, a bit in RC1 lasts 10 ms and a bit in RC2 and RC4 lasts 1.66 ms. The modulator needs to send around 800 samples through to the radio to respect the Sigfox™ spectrum specification and claim for certification. Therefore, the worst case requirement in RC2 is 2 µs per sample. To decrease real time constraints, the STM32L0 prepares bit N+1 (write samples in memory) while the DMA sends samples for bit N to the SPI peripheral.

By using a double buffer technique, bit N+1 can be prepared in memory page (N+1)%2 while bit N is being sent from memory page (N)%2, and so forth. This is illustrated in *Figure 4*.

**Figure 4. Dual buffer modulation with STM32**




The Trun duration is approximately 430 µs. Each bit preparation can be delayed by Tsleep in the worst case. During the Tsleep duration, the STM32 is in the Sleep mode and the Arm® clock is switched off.

Timer 2 is used to:

- Trigger the DMA transfer from the memory to the SPI data register
- Generate the NSS from output compare (OC)

In summary, the microcontroller is only active during 430 µs for each bit sent. It can sleep or serve other routines the rest of the time.

The modulation library is located in directory *Projects\B-L072Z-LRWAN1\Applications\Sgfx\Common\cwmx1zzabz\Radio_Sigfox_Driver*.

## 4.3 Cmac library

The Cmac library encrypts uplink payloads and authenticates downlink payloads.

**Table 8. Cmac primary APIs**

| Function | Description |
|---|---|
| `sfx_u8 SE_API_get_device_id (sfx_u8 dev_id[ID_LENGTH]);` | This function copies the device ID in `dev_id`. |
| `sfx_u8 SE_API_get_initial_pac (sfx_u8 *initial_pac);` | Get the initial PAC. |
| `sfx_u8 SE_API_secure_uplink_message( sfx_u8 *customer_data, sfx_u8 customer_data_length, sfx_bool initiate_downlink_frame, sfx_se_frame_type_t frame_type, sfx_bool *send_rcsync, sfx_u8 *frame_ptr, sfx_u8 *frame_length);` | Generation of an uplink frame bitstream. |
| `sfx_u8 SE_API_verify_downlink_message (sfx_u8 *frame_ptr, sfx_bool *valid);` | Authentication (and payload decryption) of a received message. |

*Note 1: This library interfaces the 'se_nvm' functions to store/retrieve `SFX_SE_NVMEM_BLOCK_SIZE` bytes from the non-volatile memory.*

*Note 2: "se_api.h" is the interface to the Sigfox™ secure element.The secure element can be either a physical secure element, or emulated by firmware with the Cmac library and the Credentials library.*

The Cmac library is located in directory *\Middlewares\Third_Party\Sgfx\Crypto*.

## 4.4 Credentials library

The Credentials library can access the keys, the PAC and the IDs. It can also encrypt data with the Sigfox™ key.

**Table 9. Credentials primary APIs**

| Function | Description |
|---|---|
| `void CREDENTIALS_get_dev_id(uint8_t* dev_id);` | Gets the device ID. |
| `void CREDENTIALS_get_initial_pac (uint8_t* pac);` | Gets the device initial PAC. |
| `sfx_bool CREDENTIALS_get_payload_encryption_flag(void);` | Gets the encryption flag. Sets to false by default. |
| `sfx_error_t CREDENTIALS_aes_128_cbc_encrypt(uint8_t* encrypted_data, uint8_t* data_to_encrypt, uint8_t block_len);` | Encrypts data with the secret key. The secret key can be set to `CMAC_KEY_PRIVATE` or `CMAC_KEY_PUBLIC` (refer to *Section 5.3.9: ATS410 to set AES key on page 31*). |

The library is specific to the CMWX1ZZABZ. For other targets, the library can be replaced by *sgfx_credentials_template.c* and *aes_template.c,* and customized based on other specific hardware.

It is located in *Projects\B-L072Z-LRWAN1\Applications\Sgfx\Common\cwmx1zzabz\Credentials_libs*.

## 4.5 Utilities

Utilities is located in directory *Middlewares\Third_Party\Sgfx\Utilities*.

The main APIs in Utilities are described in:

- *Section 4.5.1: Timer server*
- *Section 4.5.2: Low-power manager*
- *Section 4.5.3: Scheduler*

Secondary APIs and additional information are provided in the header files related to the drivers.

### 4.5.1 Timer server

A timer server is provided so that the user can request timed tasks to be executed. As the hardware timer is based on the RTC, the time is always counted, even in the low-power modes.

The timer server provides a reliable clock for the user and the stack. The user can request as many timers as the application requires. Care must be taken that the timer callback duration is less than 1 ms.

Four main APIs are provided as described in *Table 10*.

**Table 10. Timer server APIs**

| Function | Description |
|---|---|
| `void TimerInit(TimerEvent_t *obj, void (*callback)(void));` | Initializes the timer and associates a callback function executed when timer elapses. |
| `void TimerSetValue (TimerEvent_t *obj, uint32_t value);` | Sets the timer with a timeout value in milliseconds. |
| `void TimerStart(TimerEvent_t *obj);` | Starts the timer. |
| `void TimerStop(TimerEvent_t *obj);` | Stops the timer. |

### 4.5.2 Low-power manager

The APIs described in *Table 11* allow to manage the low-power mode of the STM32L072 core.

**Table 11. Low-power APIs**

| Function | Description |
|---|---|
| `void LPM_SetStopMode(LPM_Id_t id, LPM_SetMode_t mode)` | Used to set the Stop mode:<br>– id: process ID<br>– mode requested LPM_Enable or LPM_Disable |
| `void LPM_SetOffMode(LPM_Id_t id, LPM_SetMode_t mode)` | id: process ID<br>mode requested LPM_Enable or LPM_Disable |
| `void LPM_EnterLowPower(void);` | Cortex®-M deep-sleep management |

The low-power mode by default is the Off mode. The Off mode on STM32L0 is Standby.

If at least one process disabled the Stop mode, and low-power is entered (by calling `LPM_EnterLowPower`), the Sleep mode is entered next.

If at least one process disabled the Off mode, but no process disabled the Stop mode, when low-power is entered (by calling `LPM_EnterLowPower`), the Stop mode is entered next.

If no process disabled the Off mode and no process disabled the Stop mode, when low-power is entered (by calling `LPM_EnterLowPower`), the Off mode is entered next (the Off mode is the Standby mode on STM32L0).

**Table 12. Low-power modes**

| Low-power mode | LPM_SetStopMode | LPM_SetOffMode |
|---|---|---|
| Low-power Sleep | Disable | X(1) |
| Low-power Stop | Enable | Disable |
| Low-power Off (standby) | Enable | Enable |

1. Not considered.

As an example, during a transmission, the DMA transfers data from the memory to the SPI data register. Since the DMA needs the system clock to run, the system shall enter neither Stop mode nor Standby mode; Stop mode is therefore disabled using the `LPM_SetStopMode(LPM_LIB_Id, LPM_Disable)` function.

The user is free to add his own IDs in the in `LPM_Id_t` structure depending on his system requirements (`LPM_Id_t` is defined in *\Projects\B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_ModemAT\inc\utilities_conf.h*).

### 4.5.3 Scheduler

The scheduler provides a robust and easy framework to execute tasks in the background and enter low power mode when there is no more activity. It implements mechanism to prevent race conditions.

In addition, it provides a major feature (Wait Event) for easily saving MIPS and power in any application that implements the "run to completion" command.

The *utilities_conf.h* located at *\Projects\B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_ModemAT\inc* is used to configure the task and event Ids. The ones already listed must not be removed.

The APIs described in *Table 13*allow to manage the low-power mode of the STM32L072 core.

**Table 13. Scheduler APIs**

| Function | Description |
|---|---|
| `void SCH_Run(void)` | Requests the scheduler to execute all pending tasks. When no task are pending, calls SCH_Idle(); |
| `void SCH_Idle(void)` | Idle task. It must be implemented in the main. This is where the low power handler must be called (within critical section). |
| `void SCH_RegTask (uint32_t task_id, void (*task)(void));` | Registers a task in the scheduler to be executed when task_id is set.<br>– task_id: the ID of the task. It must be lower than 32. |
| `void SCH_SetTask (uint32_t task_id);` | Requests task task_id to be executed. |
| `void SCH_WaitEvt (uint32_t evt_id);` | Waits for a specific event to be set. The scheduler loops until the event is set.<br>Note: A task may be executed while waiting for an event to be set.<br>Note: when called recursively, the scheduler waits for the last event requested to be set even though one of the already requested event has been set.<br>– evt_id: the ID of the task. It must be lower than 32. |
| `void SCH_SetEvt (uint32_t evt_id);` | Sets an event that is a waited with SCH_WaitEvt(). |
| `void SCH_ClrEvt (uint32_t evt_id);` | Clears an event from the event wait list SCH_WaitEvt(). |

## 4.6 EEPROM driver

The EEPROM data are placed in variable `E2pData`, the structure member of which can be freely defined in file *hw_eeprom_conf.h*. This configuration file also contains the factory settings.

The EEPROM APIs are described in *Table 14*.

**Table 14. EEPROM APIs**

| Function | Description |
|---|---|
| `HW_EEPROM_WRITE(VAR, Data)` | Writes a data in EEPROM.<br>– For instance:<br>`HW_EEPROM_WRITE(E2pData.TxPower, 14);` |

**Table 14. EEPROM APIs (continued)**

| Function | Description |
|---|---|
| `void HW_EEPROM_Init(void);` | DEFAULT_FACTORY_SETTINGS is written in `E2pData` when EEPROM is empty. |
| `void HW_EEPROM_RestoreFs(void);` | DEFAULT_FACTORY_SETTINGS is written in `E2pData` whenever called. |

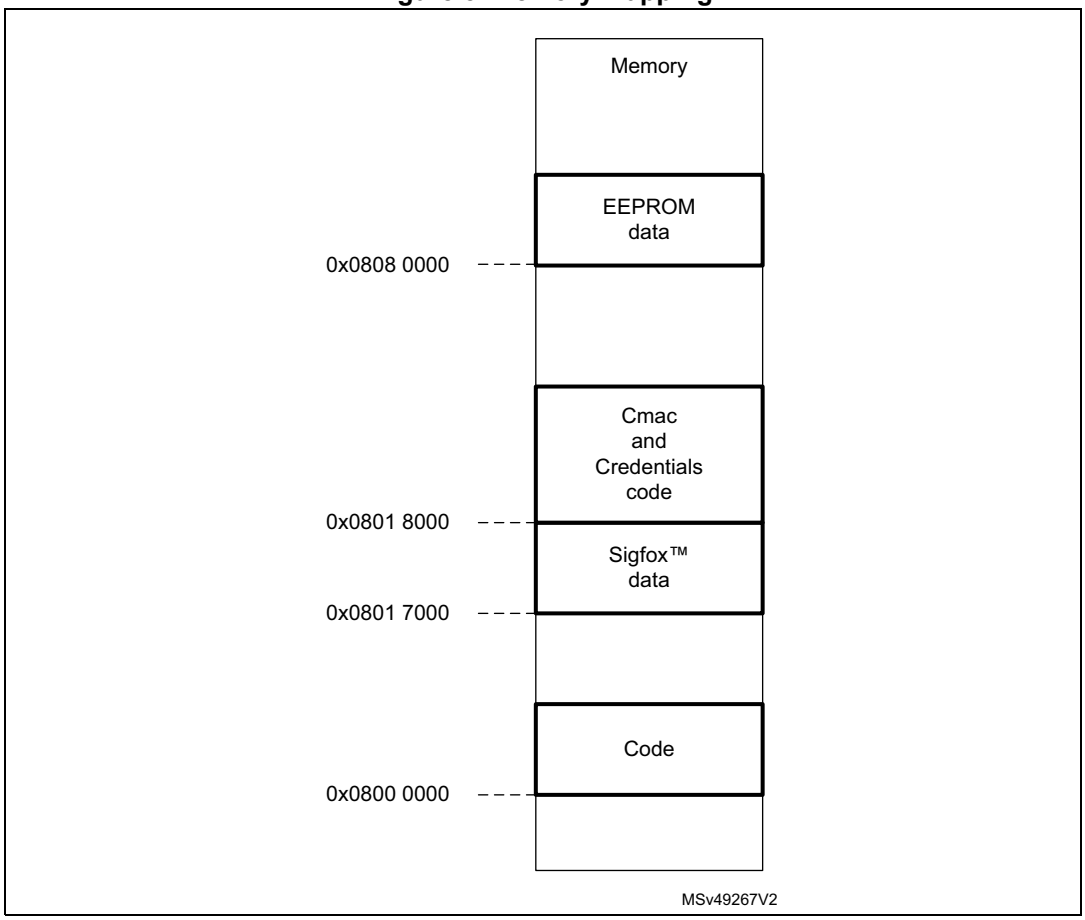
## 4.7 Memory section

The memory maps are defined in the scatter files.

The code is placed at 0x0800 0000. The sigfox_data is placed at 0x0801 7000. As well EEPROM embeded in the STM32L0 is used to store data that must be retained even if power supply is lost. EEPROM is based at 0x0808 0000.

The memory mapping is depicted in *Figure 5*.

**Figure 5. Memory mapping**



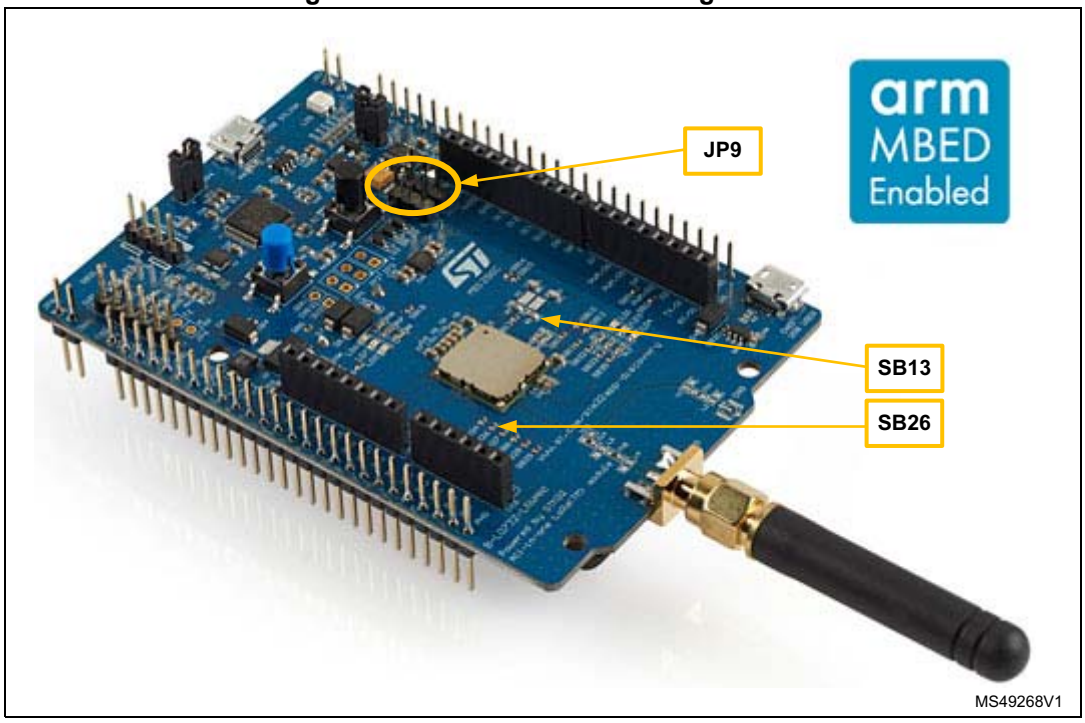

The scatter files can be adapted to the application.

# 5 Application description

## 5.1 Hardware

The applications are based on a B-L072-LRAWAN1 discovery board platform. The Discovery kit (B-L072Z-LRWAN1) is presented in *Figure 6*. It is an RF Discovery board featuring the CMWX1ZZABZ-091 LoRa® module from Murata. This module incorporates the SX1276 low-power transceiver which features a LoRa® long-range modem. This module supports high-performance and OOK/FSK modulations. The transceiver is controlled by an STM32L072CZY6 microcontroller embedded in the module. More information is available in *[1]*.

**Figure 6. B-L072-LRAWAN1 configuration**



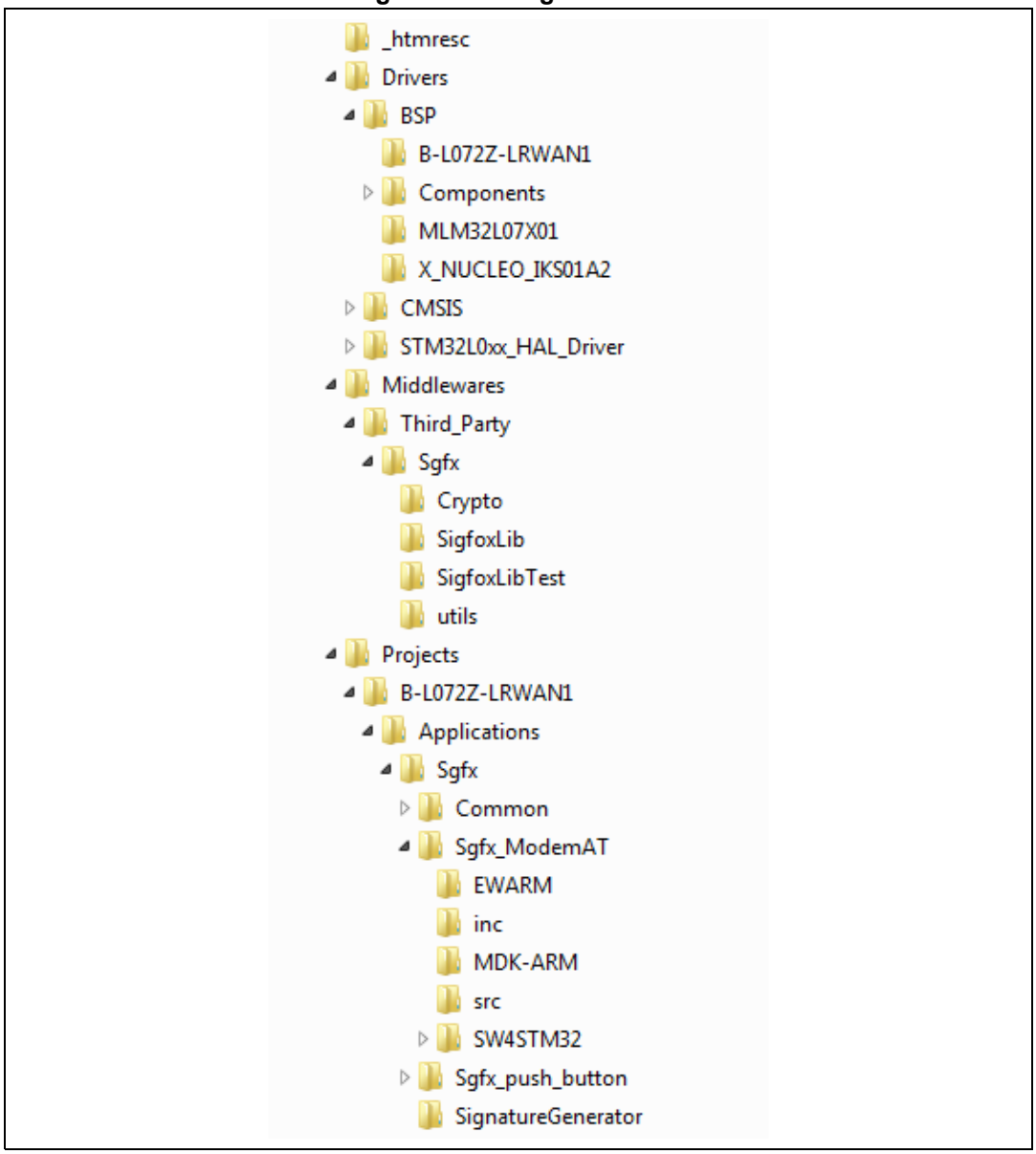

The following hardware setting is required:

- Solder bridge SB13 must be closed.
- For RevC/D discovery boards, JP9 must have pins 1 and 2 connected (TCXO controlled by MCU).
- Solder bridge SB26 must be closed.

## 5.2 Package description

When the user unzips the X-CUBE-SFOX package, the internal package structure is as shown in *Figure 7*.

**Figure 7. Package overview**




The X-CUBE-SFOX package contains three applications:

- Sgfx_ModemAT
- Sgfx_push_button
- SignatureGenerator

For the ModemAT and PushButton applications, three toolchains are available:

- MDK-ARM
- IAR™
- SW4STM32.

The SignatureGenerator application is provided in binary format and can be directly uploaded into the microcontroller.

## 5.3 AT modem Application

The purpose of this application is to implement a Sigfox™ modem controlled though AT command interface over UART by an external host. The external host can be a host-microcontroller embedding the application and the AT driver or simply a computer executing a terminal. The Sgfx_ModemAT application implements the Sigfox Stack™ driving the built-in sx1276 Semtech radio. The stack is controlled through AT command interface over UART. The modem is always in Stop mode unless it processes an AT command from the external host.

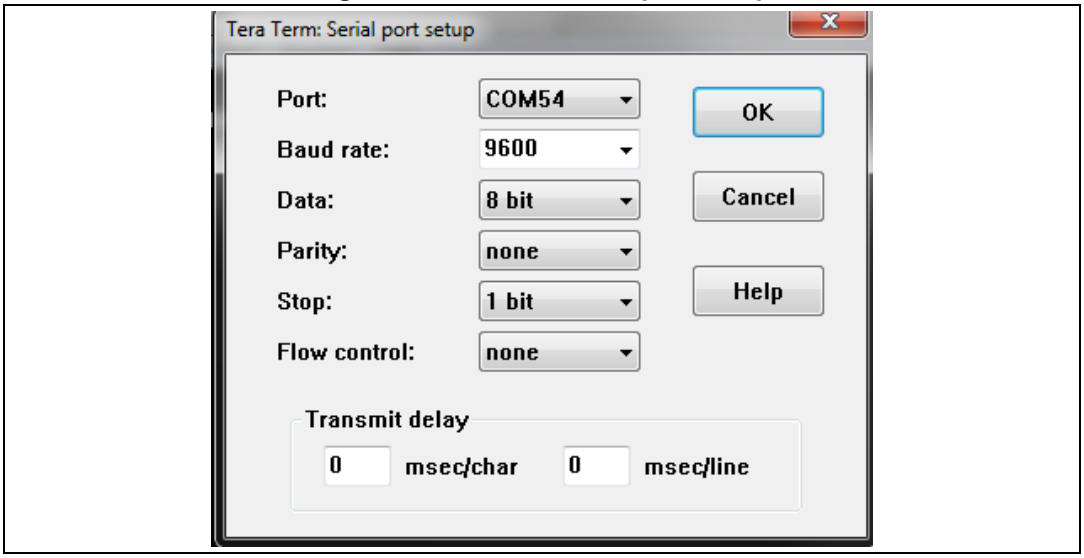
In order to launch the Sgfx_ModemAT project, the user must go to the *\Projects\ B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_ModemAT* folder and choose his favorite toolchain folder.

### 5.3.1 UART interface

In this example the LPUART is used at 9600 baud. The device can receive a character while in stop mode consuming 1.1 µA ready to receive.

Tera Term is used as terminal to control the Sigfox™ modem. The following settings are used:

**Figure 8. Tera Term serial port setup**




The available commands are given in sections *5.3.3* to *5.3.17* with the following format:

- All commands setting parameters are in the form *ATXX=Y<CR>*
- All commands getting parameters are in the form *ATXX?<CR>*

### 5.3.2 Default parameters

The default parameters when the program starts for the first time (EEPROM empty) are:

- RC1 default values for the region configuration
- 14 dBm output power
- Default key to private

These default values can be changed by modifying DEFAULT_FACTORY_SETTINGS in the *hw_eeprom_config.h* configuration file.

The default private key and private ID are the test keys described in the Sigfox™ test specification. They are stored in the *sigfox_data.h* file.

### 5.3.3 AT? to list the available commands

This command is used to get the list of all supported AT commands.

### 5.3.4 AT$RFS to restore factory settings

**Cmd**: `AT$RFS<CR>`
Restores factory settings.

**Resp**: `OK<CR><LF>`

Factory settings are defined in file *hw_eeprom_conf.h*.

### 5.3.5 AT$ID to get the device ID

**Cmd**: `AT$ID?<CR>`
Gets the device ID on 32 bits.

**Resp**: `0000 1234 <CR><LF>`

The ID is provided on 8 hexadecimal characters from MSB to LSB.

*Note: If the credentials file (sigfox_data.h) is not replaced, the returned value is not valid for activation (see Section 5.7: Personalization and activation on page 37).*

### 5.3.6 AT$PAC to get the device PAC

**Cmd**: `AT$PAC?<CR>`
Gets the device PAC on 8 bytes. The default response with the deployed package is 0x0. This value changes once the *sigfox_data* is downloaded.

**Resp**: `1122 3344 5566 7788 <CR><LF>`
(8-byte format)

*Note: If the credentials file (sigfox_data.h) is not replaced, the returned value is not valid for activation (see Section 5.7: Personalization and activation on page 37).*

### 5.3.7 AT$SB to send a bit status

**Cmd:** `AT$SB=<status value>{,<Optional ResponseWaited>}{,<Optional txrepeat>}<CR>`
Sends a bit status (0 or 1).

**Resp**: `OK<CR><LF>`

**Parameters**:

- <status_value> = 0/1
- <Optional ResponseWaited>
  - 0: no response awaited (default)
  - 1: response awaited
- <Optional txrepeat>
  - 0: one Tx frame sent
  - 1: three Tx frames sent (default)

**Examples**:

AT$SB = 1: Sends bit 1 with no response awaited.

AT$SB = 0,1: Sends bit 0 with a response awaited.

AT$SB = 0,1,1: Sends bit 0 with a response awaited, with 3 Tx frames sent.

The <Optional ResponseWaited> parameter is optional when 0.

*Note: This command is blocking and cannot be aborted (refer to Section 2.4: Tx/Rx radio time diagram).*

### 5.3.8 AT$SF to send the payload in bytes

**Cmd**: AT$SF=<payload data>{,<Optional ResponseWaited>}{,<Optional txrepeat>}<CR>
Sends the payload.

**Resp**: OK<CR><LF>

**Parameters**:

- <payload data> maximum 12 bytes
- <Optional ResponseWaited>
  - 0: no response awaited (default)
  - 1: response awaited
- <Optional txrepeat>
  - 0: one Tx frame sent
  - 1: three Tx frames sent (default)

**Examples**:

AT$SF=313245: sends 0x31 0x32 0x45 (ASCII string 12E) payload with no response awaited.

AT$SF=010205,1: sends 0x01 0x02 0x05 (ASCII string !”%) payload with a response awaited.

*Note: This command is blocking and cannot be aborted (refer to Section 2.4: Tx/Rx radio time diagram). OK is returned only after command completion. Refer to the timings presented in Section 2.4: Tx/Rx radio time diagram on page 9.*
*When the radio is transmitting, LED4 is on.*

### 5.3.9 ATS410 to set AES key

**Cmd**: `ATS410=<KEY_USAGE>`
By default the test key is used.

**Resp**: `OK<CR><LF>`

**Parameters**:

- `<KEY_USAGE>`
  - 0: the private key is used.
  - 1: the public key is used. This key must be used in conjunction with the Sigfox™ Network Emulator (USB SDR dongle).

*Note: This value is stored in EEPROM.*

### 5.3.10 ATS300 to send an out-of-band message once

**Cmd**: `ATS300<CR>`
Sends one keep-alive out-of-band message.

**Resp**: `OK<CR>(<LF: optional>)`

*Note: Out-of-band messages are messages which have Sigfox™ network well known format. They can be sent every 24 hours.*

### 5.3.11 ATS302 to set the radio output power

**Cmd**: `ATS302=<power> <CR>`

**Resp**: `OK<CR><LF>`

**Parameter**:

- `<power>`: in dBm. The allowed range is 10 dBm to 20 dBm.

*Note: The default power is 14 dBm in RC1, 20 dBm in RC2 and RC4, and 13 dBm in RC3c.*

*This command is mandatory for CE certification of the device.*

*The RF output power value is saved in EEPROM. When the RC zone is changed, the RF output power is restored to its default value.*

### 5.3.12 AT$CW continuous wave

**Cmd**: `AT$CW=<frequency><CR>`

**Resp**: `OK<CR><LF>`

**Parameters**:

- `<frequency>`: center frequency for carrier
  - 0: stops the CW.
  - Not 0: starts continuous unmodulated wave at frequency parameter (frequency in hertz or megahertz).

**Examples**:

`AT$CW=868000000:` starts a CW emission at 868 MHz.

`AT$CW=868:` starts a CW emission at 868 MHz.

`AT$CW=0:` stops the CW emission.

*Note: The output power can be modified with ATS302 (refer to Section 5.3.11).*

*Before starting a CW test again, it must be stopped.*

### 5.3.13 AT$PN PRBS9 continuous Tx wave

**Cmd**: `AT$PN=<frequency>,<bitrate><CR>`

**Resp**: `OK<CR><LF>`

**Parameters**:

- `<frequency>`: center frequency for carrier
  - 0: stops the PRBS9 continuous wave.
  - Not 0: starts continuous unmodulated wave at frequency parameter (frequency in hertz or megahertz).
- `<bitrate>`: 100 or 600 bits per second

**Examples**:

`AT$PN=868000000,100:` starts a 100-bps modulated wave at 868 MHz.

`AT$PN=0:` stops the PRBS9 continuous wave emission.

*Note: The current PN test must be stopped before another PN test can be started.*

### 5.3.14 AT$TM Sigfox™ test mode

The TST modem must implement the command below. It is possible to use this test mode in front of Sigfox™ Radio Signal Analyzer (RSA) and SDR dongle *[2]*. This command is for test-mode purposes only and cannot be used to connect to the Sigfox™ network.

**Cmd**: `AT$TM=<rc>,<mode><CR>`
Activates the test mode.

**Resp**: `OK<CR><LF>`

**Parameters**: the test mode values are:

- `rc =` 1, 2, 3c, or 4

  The region configuration at which the test shall run.
- `mode` = SFX_TEST_MODE_TX_BPSK = 0

  Sends only BPSK 26-byte packets including synchro bit and PRBS synchro frame at the `Tx_frequency` uplink frequency defined in *Table 3: Region configuration RF parameters on page 8*. The uplink frequency is RC dependent.
- `mode` = SFX_TEST_MODE_TX_PROTOCOL = 1

  Full protocol with internal Sigfox™ key: sends all Sigfox™ protocol frames with all possible lengths available with hopping, meaning that it sends bits with downlink flag set and unset, sends out-of -band frames, sends frames with downlink flag set and unset with all possible payload lengths from 1 to 12 bytes.

  **Caution:** This test lasts several minutes. No other AT command must be sent before the completion of the test.
- `mode` = SFX_TEST_MODE_RX_PROTOCOL = 2,

  This test consists in calling functions to test the complete protocol in downlink only with full protocol using Sigfox™ key.
  The `Rx_frequency` downlink frequency is defined in *Table 3: Region configuration RF parameters on page 8*. The downlink frequency is RC dependent.
- `mode` = SFX_TEST_MODE_RX_GFSK = 3

  This test is used to measure the approximate sensitivity of device.

  Rx mode in GFSK with expected pattern =
  AA AA B2 27 1F 20 41 84 32 68 C5 BA 53 AE 79 E7 F6 DD 9B sent at the `Rx_frequency` downlink frequency defined in *Table 3: Region configuration RF parameters on page 8*. The downlink frequency is RC dependent. The test lasts 30 seconds.
- `mode` = SFX_TEST_MODE_RX_SENSI = 4

  This test is used to measure the real sensitivity of the device. It requests one uplink and one downlink frame with Sigfox™ key with specific timings.
- `mode` = SFX_TEST_MODE_TX_SYNTH = 5

  Does 1 uplink frame on each Sigfox™ channel frequency.

  This test takes a couple of minutes.

### 5.3.15 AT$RSSICAL to set and get the RSSI calibration

This command is used to set the Rx RSSI offset of the device for calibration purpose. It allows end-product makers adapting the RSSI level according to their loss environment. It is also applicable to the LBT threshold.

**Cmd**: `AT$RSSICAL=<rssical><CR>`
Sets the RSSI calibration in dB.

**Resp**: `OK<CR>`

**Parameters**:

- `<rssical>`: integer

**Example**:

`AT$RSSICAL?<CR>`: gets the current RSSI calibration value in dB.

*Note: The value is stored in EEPROM. It is maintained after a change of the RC zone. The value is 0 dB by default.*

### 5.3.16 AT$RC to set/get the zones

**Cmd**: `AT$RC=<rc><CR>`
Sets the current zone.

**Resp**: `OK<CR>`

**Parameter**:

- `<rc>` = 1, 2, 3c, or 4

**Examples**:

`AT$RC?<CR>` to get the current zone.

`1, 2, 3c, or 4.`

*Note: This value is stored in EEPROM.*
*When requesting RC1, the RF output power is automatically set to 14 dBm.*
*When requesting RC2 or RC4, the RF output power is automatically set to 20 dBm.*
*When requesting RC3c, the RF output power is automatically set to 13 dBm.*
*The new output power is saved in EEPROM.*

### 5.3.17 ATS400 to configure specific variables for standard

**Cmd**: `ATS400=<8_digit_word0><8_digit_word1><8_digit_word2>, <timer_enable>`

**Resp**: `OK<CR>(<LF: optional>)`

This command has different meanings depending on the medium access mode.

#### FH (RC2 & RC4):

The command configures the enabled channels for FCC (0 for disable or 1 for enable). Channels are computed from the Tx frequency (Refer to *Section 4.1.3: Sending frames/bits on page 17*) with the following formula:

channel = tx_frequency + 300 kHz x macro_channel_id

The default value in RC2 is <0000 0001><0000 0000><0000 0000>,1.

The default value in RC4 is <0000 0000><4000 0000><0000 0000>,1.

**Example**: ATS400=000001FF0000000000000000,1: the timer between consecutive Tx frames is enabled and the following macro channels are enabled: 902.2 MHz | 902.5 MHz | 902.8 MHz | 903.1 MHz | 903.4 MHz | 903.7 MHz | 904.0 MHz | 904.3 MHz | 904.6 MHz.

#### LBT (RC3c):

The command configures the Carrier Sense feature for the first frame:

- `config_word[0]`: number of attempts to send the first frame (must be greater than or equal to 1)
- `config_word[1]`: maximum Carrier Sense sliding window (in ms) (must be greater than 6 ms, which is `CS_MIN_DURATION_IN_MS` + 1)
- `config_word[2]`:
  - bit 8: set the value to 1 to indicate that the device uses the full operational radio band (192 kHz)
  - bits 7-3: number of Carrier Sense attempts
- `timer_enable`: unused

*Note: This command is reserved for certification purpose.*

*For details about configuration word and timer_enable, refer to Section 4.1.4: Set standard configuration for FH on page 17.*

*Only configuration word is stored in EEPROM.*

*This is not applicable in the RC1 zone.*

## 5.4 Push-button application

The push-button application is a stand-alone example.

On user push-button event, this application reads the temperature, humidity and atmospheric pressure from the sensors through $I^2C$. These three data (temperature, humidity, atmospheric pressure) are sent to the Sigfox™ network. In order to launch the Sigfox™ push-button project, the user must reach the *\Projects\ B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_push_button* folder and choose his favorite toolchain folder.

This application reads the temperature, humidity and atmospheric pressure from the sensors through $I^2C$ when expansion board X-NUCLEO-IKS01A2 is plugged and `#SENSOR_ENABLED` is set to 1.

The application enters the Stop mode by default while waiting for a button to be pressed. Commenting out the `#define STDBY_ON` preprocessor switch, and removing solder bridge SB31, makes the device enter the Standby mode instead. When a positive pulse is applied on PA0 (using a wire bridge to VDD_MCU), the device wakes-up, transmits a packet, and enters the Standby mode again. By default, the application runs in RC1; This can be changed in *main.c* by redefining the following line:

```
#define APPLI_RC    ST_RC1
```

*Note: When the X-NUCLEO-IKS01A2 shield is plugged on the B-L072Z- LRWAN1 board, the SB25 solder bridge of X-NUCLEO-IKS01A2 must be removed.*

## 5.5 Signature generator

This is a binary file. The binary generates a 128 bit-signature on two UART pins (PA2 and PA9) with the settings described in *Section 5.3.1: UART interface on page 28*. This signature is used to personalize the device. The process is described in *Section 5.7: Personalization and activation*.

## 5.6 Static switches

Static Defines are used to switch optional features such as debug, trace, and disable low-power.

The user must edit the *\Projects\B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_ModemAT\inc\hw_conf.h* file to modify the static switches.

### 5.6.1 Debug switch

*#define DEBUG*: the debug mode enables the DBG_GPIO_SET and the DBG_GPIO_RST macros as well as the debugger mode even when the MCU enters low-power.

*#define TRACE*: the trace mode enables the DBG_PRINTF macro.

*Note: In order to enable true low-power, both above-mentioned "#define" must be commented out.*

### 5.6.2 Sensor switch

When no sensor expansion board is plugged on the set-up, *#define SENSOR_ENBALED* must be commented out.

### 5.6.3 Disable low-power switch

To force the STM32 to remain in Run mode, LOW_POWER_DISABLE must be defined.

## 5.7 Personalization and activation

When compiling and loading the firmware using the default *sigfox_data.h* file, the default Sigfox™ credentials are loaded into the device. This only allows to test the Sigfox™ Murata device locally in the laboratory.

For the Sigfox™ Murata device to send data to the Sigfox™ backend server, two steps are required

- Personalization. Each Sigfox™ device must be loaded with unique dedicated credentials. These credentials are known as ID, PAC and private key. They are necessary to activate the device so that it can send data to the Sigfox™ data server. The personalization step is a process for a Murata module user to obtain the unique credentials.
- Activation. Once the device is personalized, the device needs to be recorded by the Sigfox™ back-end server. This step requires to log-on the Sigfox™ back-end server.

*Figure 9* depicts the sequence of personalization and activation steps.

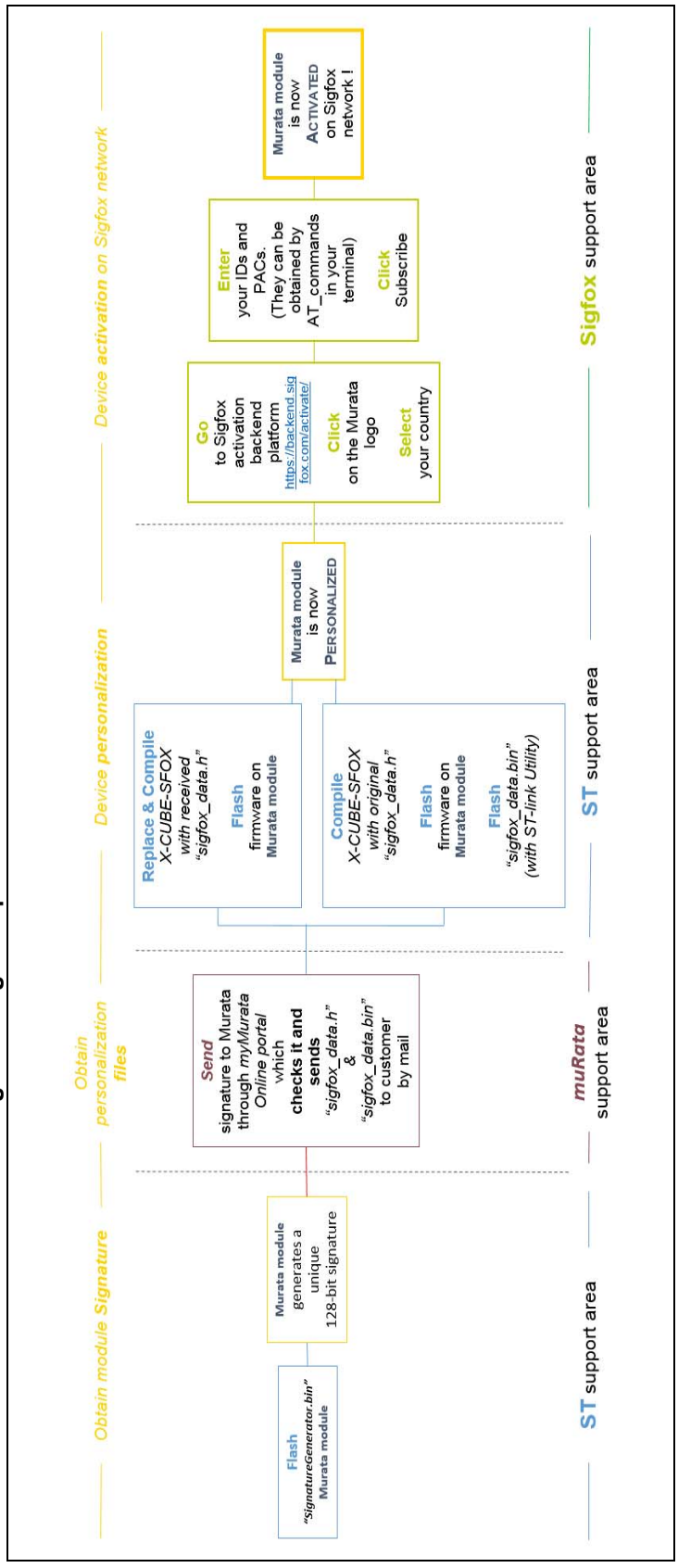



**Figure 9. Sigfox™ personalization and activation overview**

### 5.7.1 Personalization

#### Generating the signature

*SignatureGenerator.bin* is a binary file. It generates a 128-bit signature on UART pins PA2 and PA9 with the settings described in *Section 5.3.1: UART interface on page 28*.

*SignatureGenerator.bin* located in *Projects\B-L072Z-LRWAN1\Applications\Sgfx\SignatureGenerator* must be loaded in the device. Loading can either be done by using ST-Link Utility or by dragging and dropping *SignatureGenerator.bin* into the DIS_L072 drive when using the B-L072Z-LRWAN1 discovery board.

If the discovery kit board is used, a COM port is emulated over the USB where a terminal can be connected.

Otherwise, an FTDI cable can be used, the UART Rx pin of which is connected to PA2 or PA9. In the latter case, the terminal must be connected to the FTDI COM port.

After a hardware reset, the signature is sent on the COM ports. The signature is displayed as follows (note that the signature is device dependent):

signature=
D33AA579E7B9D7209EAB354801574D15

#### Getting credentials from the Murata server

Log on the B-L072-LRAWAN1 related page of myMurata.com on-line portal and copy/paste the signature. A device-dependent *sigfox_data.h* (together with its binary-format equivalent *sigfox_data.bin*) is sent by mail if the signature is authenticated.

#### Loading user credentials into the Murata module

They are two ways to load the credentials.

1. First method: replace the default sigfox_data.h located in *Projects\B-L072Z-LRWAN1\Applications\Sgfx\Sgfx_ModemAT\inc* by the one received from Murata. After project compilation and load, the device is personalized.
2. Second method: overwrite the already loaded sigfox_data (refer to *Section 4.7*) located at 0x0801 7000. After the application with default *sigfox_data* is loaded into the device, the data can be erased and directly replaced by the data from *sigfox_data.bin*.
   In order to do so, the following ST-Link Utility command line must be executed:
   ```
   ST-LINK_CLI.exe -c swd ur -SE 736 -P sigfox_data.bin 0x08017000
   -hardrst -v
   ```

*Note: Option '-SE 736' erases sector 736 (@ 0x0801 7000).*

Once ST-LINK command is executed, the device is personalized.

**Caution:** The credentials received from myMurata are device specific. The received credentials **cannot** be used on any other device.

### 5.7.2 Activation

1. In the terminal, use AT commands `AT$ID?<CR>` and `AT$PAC?<CR>` from the ATmodem application in order to get Sigfox™ ID & PAC.
2. Go to https://backend.sigfox.com/activate/
3. Click on the muRata logo
4. Select the country
5. Enter ID & PAC
6. Enter account details and click on *Subscribe*
7. The kit is now activated on the Sigfox™ network.

# 6 System performance

## 6.1 Memory footprint

The values in *Table 15* have been measured for the following configuration of the Keil® compiler (MDK-ARM compiler 5.05):

- Optimization: optimized for size level 3
- Debug option: off
- Trace option: off
- Target: B-L072Z-LRWAN1 with CMWX1ZZABZ

**Table 15. Memory footprint measured values**

| Module | Flash (bytes) | RAM (bytes) | Description |
|---|---|---|---|
| Application layer and stack | 11206 | 1035 | Application + stack size. |
| Sigfox Core library | 5774 | 8 | Library Sigfox™. |
| Cmac library | 1408 | 30 | Encrypts and authenticates uplink and downlink frames respectively. |
| Modem library | 1446 | 3247 | Modulation library. |
| Credentials library | 4994 | 274 | Cryptography library. |
| Sigfox Addon Verified library | 1954 | 36 | Test library. |
| Radio | 5958 | 696 | Radio driver. |
| HAL and low-level drivers | 9270 | 1032 | Hardware abstraction layer and low-level drivers. |
| Microlib | 1574 | 0 | All microlib. |
| Total application (user) | 43584 | 6358 | Memory footprint for the overall modem AT application. |

## 6.2 Real-time constraints

Real time constraints apply when the transmitter is running. The SPI modulation is critical. The DMA is driving the SPI, but the STM32 MCU must fill in the memory with modulation data. The process is described in *Section 4.2: Library Sigfox™ modulation on page 19*.

## 6.3 Power consumption

The power consumption presented in this section has been measured on the ST discovery B-L072Z-LRWAN1.

The measurement setup is:

- No DEBUG
- No TRACE
- No SENSOR_ENABLED

In these conditions, the typical MCU consumption (STM32L0xx) in Stop mode is 1.1 µA while the consumption in Run mode is 8.0 mA.

Futhermore, power consumptions while receiving and transmitting are:

- When the radio is receiving, the MCU is in Stop mode but the radio consumes 10 mA. This includes the consumption of the TCXO.
- When the radio is transmitting at 14 dBm, the MCU is in Run mode for 430 µs and in Sleep mode for the rest of the bit. The radio consumes 30 mA. This does not include the consumption of the TCXO.
- When the radio is transmitting at 20 dBm, the MCU is in Stop mode but the radio consumes 120 mA. This does not include the consumption of the TCXO.

# 7 Revision history

Table 16. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 19-Dec-2017 | 1 | Initial release. |
| 21-May-2018 | 2 | Updated timings in *Figure 1: Timing diagram for uplink only*, *Figure 2: Timing diagram for uplink with downlink*, and *Table 4: Timings*. Updated document with RC3c regional configuration. Updated *Figure 3: Firmware top-level structure*. Updated *Figure 5: Memory mapping*. Updated *Figure 7: Package overview*. Updated *Table 8: Cmac primary APIs*. Updated *Table 15: Memory footprint measured values*. Updated *Section 5.4: Push-button application*. Added *Section 2.5: Listen before talk (LBT)*. Added *Section 3.2: Sigfox™ certification*. Added *Section 4.1.2: Sigfox Addon Verified library*. Added *Section 4.4: Credentials library*. Added *Section 5.3.13: AT$PN PRBS9 continuous Tx wave*. |