UM2173

User manual

Getting started with the S2-LP Sigfox firmware

Introduction

The S2-LP Sigfox firmware framework provided by ST allows you to develop embedded applications on the STEVAL-FKI868V2 (and STEVAL-FKI868V1), STEVAL-FKI915V1 and the X-NUCLEO-S2868A1 platforms.

The package also includes the support for the STEVAL-IDB007V2 and STEVAL-IDB008V2 kits to be used with the shields included in the above mentioned kits. This enables the support for BlueNRG-1 and BlueNRG-2 System-on Chip to the STM32 microcontrollers.

You should first read user manual UM2169 Getting started with the Sigfox S2-LP kit, which explains how to prepare the board with a Sigfox ID/PAC/KEY and to register the node on your backend account.

1 Hardware requirements

A Windows® PC with:

- 2 USB ports
- 125 MB free hard disk space

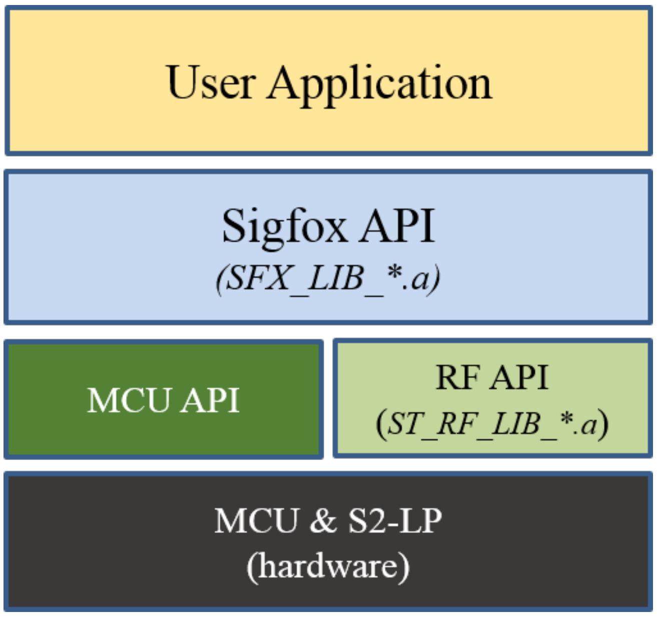
At least one of the following ST evaluation kits:

- STEVAL-FKI868V2 (and STEVAL-FKI868V1) (for RCZ1 and RCZ3) kit with:
  1. STEVAL-FKI868V2 (and STEVAL-FKI868V1) evaluation board featuring the S2-LP sub-1 GHz (860-940 MHz) ultra-low power low data-rate transceiver
  2. STM32 Nucleo-64 development board or STEVAL-IDB007V2/IDB008V2 board
- STEVAL-FKI915V1 (for RCZ2 and RCZ4)
  1. STEVAL-FKI915V1 evaluation board based on the S2-LP sub-1 GHz (860-940 MHz) ultra-low power low data-rate transceiver, with external power amplifier
  2. STM32 Nucleo-64 development board or STEVAL-IDB007V2/IDB008V2 board

2 Firmware architecture

The firmware consists of stacked modules in a framework where each module demands the implementation of lower level functions from the module beneath it.

Figure 1. **Sigfox firmware application**



2.1 RF_LIB library

The RF_LIB library is responsible of the S2-LP configuration and implementation of the modulation scheme.

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

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

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

The library is compiled for devices equipped with certain ARM® Cortex® cores (see the release notes of the STSW-S2LP-SFX-DK package), separate versions are supplied in the Projects/Middlewares/Sigfox/CMx folder for specific radio control zones:

1. RCZ1: RF_LIB_ETSI_CMx.a
2. RCZ2 and 4: RF_LIB_FCC_CMx.a
3. RCZ3: RF_LIB_ARIB_CMx.a
4. All RCZ: RF_LIB_ALL_CMx.a

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

2.2 MCU_API

If the platform changes, but the processor type remains the same, you only need to re-implement the MCU_API module. This means that the framework can easily be ported to another board equipped with a microprocessor of the same type but with a different pinout by simply re-implementing this module.

Table 1. **MCU_API**

Name	Argument	Description
`MCU_API_malloc`	**size:** the number of bytes to be allocated **pointer:** pointer to the new allocated block of memory	Allocate memory for library usage (Memory usage = size (Bytes)) This function is only called once at the opening of the Sigfox Library
`MCU_API_free`	**pointer:** pointer to the memory o free up	Free memory allocated to 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	Get voltage and temperature for Out of band frames Value must respect the units bellow 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 passed 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 ley 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 only used for the 20 seconds wait in downlink
`MCU_API_report_test_result`	**status:** 1 - passed 0 - **failed rssi:** RSSI of the received frame	Report the result of Rx test for each valid message received/ validated by library.
`MCU_API_get_version`	pointer to the array variable and size	
`MCU_API_get_device_id_and_payload_encryption_flag`	pointer to the uint8_t array variable where the returned ID should be stored.	Get the device ID of the device

Name	Argument	Description
`MCU_API_get_initial_pac`	pointer to the uint8_t array variable where the returned PAC should be stored.	Get the initial PAC of the device
`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, it means that 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.
`ST_MCU_API_GpioIRQ`	**pin:** the GPIO pin of the S2-LP (integer from 0 to 3) **new_state:** enable or disable 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 passed represents the GPIO number of the S2-LP.
`ST_MCU_API_Shutdown`	**sdn_flag:** 1 - enter shutdown 0 - exit shutdown	Set the S2-LP On or OFF via GPIO
`ST_MCU_API_LowPower`	**low_power_flag:** 1 - enter in low power mode 0 - don't 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`	**duration_ms**: duration of the calibration process in ms	RTC calibration routine.
`ST_MCU_API_SetEncryptionPayload`	**ePayload:** 1 -Enable encryption 0 - Disable encryption	Enables the encryption payload flag

The following callbacks must be called by this module and are implemented by the ST-Sigfox library:

Table 2. **MCU_API_CB**

Name	Arguments	Description
`ST_RF_API_S2LP_IRQ_CB`	None	This callback is exported by the RF_LIB module. The RF_LIB module configures the S2-LP to raise interrupts and to notify them on a GPIO. When the interrupt of this GPIO is raised, this function must be called.
`void ST_RF_API_Timer_CB`	**state**: 0 for timer start, 1 for timer stop	This callback is exported by the RF_LIB module. It must be called when the timer started by the MCU_API_timer_start expires.

Name	Arguments	Description
`void ST_RF_API_Timer_Channel_Clear_CB`	None	This callback is exported by the RF_LIB module. It must be called when the timer started by the MCU_API_timer_start_Carrier_sense expires.

Finally, an applicative callback: `void Appli_Exti_CB(uint16_t GPIO_Pin)` can be implemented to demand application management of all the ETXI interrupts apart from the `ST_RF_API_S2LP_IRQ_CB`.

2.3 Sigfox data retriever

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

While the IV vector of the CBC algorithm 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, this key is stored on the board during the registration phase.

ST provides a compiled ID_KEY_RETRIEVER_CMx.a library that exports the functions in Table 3. Retriever API.

Table 3. **Retriever API**

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 to 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 into the EEPROM
`enc_utils_get_initial_pac`	pointer to the PAC uint8_t array	gets the PAC stored into the EEPROM

Name	Arguments	Description
`enc_utils_set_test_key`	**en:** if 1 switch to test key; if 0 reset to default	Set the RSA test key: 0x0123456789ABCDEF0123456789ABCDEF
`enc_utils_set_test_id`	**en:** if 1 switch to test ID; if 0 reset to default	Set the RSA test key: 0xFEDCBA98
`enc_utils_retrieve_data_from_flash`	**id_ptr**: pointer to the 32-bit word variable where the board ID must be stored **pac_ptr**: pointer to the 8-byte array where the board PAC must be stored **rcz_ptr**: pointer to the byte where the board RCZ number must be stored **freqOffset_ptr**: pointer to frequency offset **rssiOffset_ptr:** pointer to RSSI offset	Retrieve the ID, PAC and RCZ number of the board stored in the Flash memory 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.

3 Application development

Embedded applications using the Sigfox framework call SIGFOX_APIs to manage communication.

Table 4. **Application level Sigfox APIs**

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 the state machine parameters. This function 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, see below) **tx_repetition**: number of repetitions **initiate_downlink_flag**: wait for a response after transmitting.	Section 3.2 Sending frames
`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. **sfx_bool timer_enable**: enable timer feature for FH	Section 3.3 node_set_std_config command description
`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:** The type of version (MCU, RF, ...)	Returns the library version.
`SIGFOX_API_get_info`	**info**: array containing info	
`SIGFOX_API_send_outofband`	**oob_type:** Type of the OOB frame to send	Sends an out-of-band frame. These are test frames used to monitor the node parameters (voltage, temperature).

Name	Arguments	Description
`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, see below) **tx_mode**: tx_mode shall be set to 2 **initiate_downlink_flag**: wait for a response after transmitting.	This function is used to send a single bit. It is mainly used when the node seeks downlink data (and 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	Stop 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 send 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 of 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 return in customer_data the data received.
`SIGFOX_API_get_device_id`	**dev_id**: Pointer where to write the device ID	This function copies the ID of the device to the pointer given in parameter.
`SIGFOX_API_get_initial_pac`	**initial_pac**: Pointer to initial PAC	
`SIGFOX_API_switch_public_key`	**use_public_key**: Switch to public key if SFX_TRUE, private key else	Switch device on 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)	Set the period for transmission of RC Sync frame

The application should call a set of functions in order to instruct the RF_LIB to configure the radio in the proper way.

These functions are exported by the ST_RF_API header (st_rf_api.h) and are implemented into the RF_LIB module.

Table 5. **ST_RF_API**

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 50MHz).
`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.	Set 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.	Get 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 facor (reduction*0.5dB 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_Continuous_TX_Flag`	None	Returns information about the TX state of the MCU API

3.1 Opening the 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 struct to be used.

Uplink frequencies are:
- For RCZ1 it is 868.13MHz
- For RCZ2 it is 902.2MHz
- For RCZ3 it is 923.3MHz
- For RCZ4 it is 920.8MHz

Note: As frequency hopping is implemented, the transmission frequency won't be fixed.

For radio control zones 2 and 4 (FCC), refer to Section 3.3 node_set_std_config command description for how to map the macro channels.

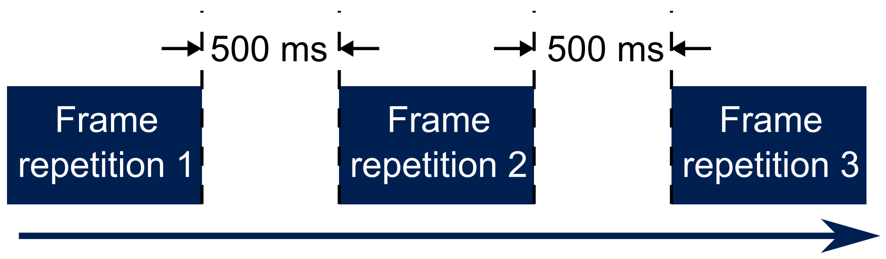
Downlink frequencies are:
- For RCZ1 it is 869.525MHz
- For RCZ2 it is 905.2MHz
- For RCZ3 it is 922.2MHz
- For RCZ4 it is 922.3MHz

3.2 Sending frames

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

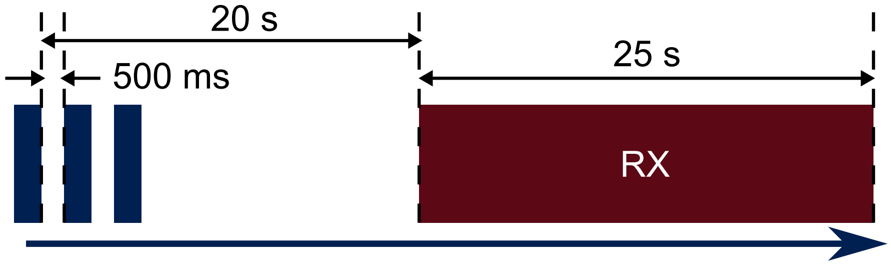
An important parameter of this function is initiate_downlink_flag:
- When the **initiate_downlink_flag** is **0**, the library only reads the first two parameters. The send frame is transmitted three times with a 500 ms pause.

Figure 2. **TX Frame timing**



- When the **initiate_downlink_flag** is **1**, the send frame is transmitted **tx_repetition** times + 1 (max. 3) with a 500 ms pause. A 25 s RX window then opens 20 s after the end of the first repetition.

Figure 3. **TX/RX timings**



If the reception is successful, the received 8-byte downlink frame is stored in the buffer location indicated by the **cust_response** input parameter.

The content of the frame can be set in the backend or by registering a callback to a website.

3.3 node_set_std_config command description

This function has different purposes according to the RC mode at which the serial port is open. 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.

Note: This API and its arguments are not applicable to RCZ1.

Each bit of the `config_words[0,1,2]` array represents a macro channel according to the following mapping:

Table 6. **Macro channel mapping - config_words[0]**

Macro Ch.	1	2	3	4	5	6	7	…	32
Frequency (MHz)	902.2	902.5	902.8	903.1	903.4	903.7	904.0	…	911.5
config_words[0] bit	0	1	2	3	4	5	6	…	31

Table 7. **Macro channel mapping - config_words[1]**

Macro Ch.	33	34	35	36	37	38	39	…	64
Frequency (MHz)	911.8	912.1	912.4	912.7	913.0	913.3	913.6	…	921.1
config_words[1] bit	0	1	2	3	4	5	6	…	31

Table 8. **Macro channel mapping - config_words[2]**

Macro Ch.	65	66	67	68	69	70	71	…	86
Frequency (MHz)	921.4	921.7	922.0	922.3	922.6	922.9	923.2	…	927.7
config_words[2] bit	0	1	2	3	4	5	6	…	21

A macro channel is only enabled when the corresponding `config_words[]` bit is set to 1. At least 9 macro channels must be enabled to meet the FCC specifications.

The second argument is a boolean indicating whether to use a timer feature in RCZ2 or 4 to be sure fulfilling the FCC duty cycle requirements.

In RCZ3 the function is used to configure the LBT mode.

`config_word[0]`: number of attempts to send the first frame (has to be greater or equal to 1)

`config_word[1]`: maximum carrier sense sliding window (in ms) (greater than 6 ms)

`config_word[2]`: bit 8: set the value to 1 to indicate that the device will use the full operational radio band (192 kHz). bit 7-3: number of carrier sense attempts. bit 2-0: number of frames sent.

`timer_enable`: unused

The delay between several attempts of carrier sense for the first frame is set by `SFX_DLY_CS_SLEEP`

This setting only affects the uplink and should be called whenever `SIGFOX_API_open` is called to open the library.

3.4 Duty cycle

3.4.1 RCZ1

The European regulation governing the 868 MHz band enforces a transmission duty cycle of 1%. Since each message can last up to 6 seconds, it is possible to send up to 6 messages per hour.

Your application must take this duty cycle into account in order to comply with the ETSI regulation.

3.4.2 RCZ2/4

According to FCC 15.247, each device should ensure that continuous transmission never exceeds 400 ms and that a given frequency channel is not reused inside 20 seconds. These limits are ensured by defining at least 9 macro-channels via `SIGFOX_API_set_std_config`.

Each macro-channel is a group of 6 25 kHz bandwidth channels with which the library performs random frequency hopping. There must be at least 9 macro-channels to satisfy the minimum 50 channel limit set in FCC 15.247.

When the `SIGFOX_API_send_frame` is called:

1. the device transmits 3 repetitions on 3 different channels of the default macro-channel hopping list
2. the second transmission takes place over the other 3 channels available on the same macro-channel
3. the third transmission hops to another listed macro-channel and uses 3 channels from that group
4. hopping and transmission continues according to the same logic

Since the transmission lasts between 200 and 350 ms and a minimum delay of 500 ms occurs between frames, the device never returns to a given channel inside 20 seconds.

The Sigfox base stations installed in RCZ2 are currently enabled to receive in the macro channel 1. The ones in the RCZ4 are enabled to receive on the 63rd channel.

Any transmission performed over a different macro channel from the default is lost.

3.5 Listen before talk

3.5.1 RCZ3

In RCZ3, the transmitter should sense the channel before starting to transmit.

For this reason, each TX phase into the figures 3 and 4 is preceded by a RX phase of variable duration.

The duration of the listen before talk is called carrier sense time.

This time can be in the range [*cs_min, cs_max*].

The timings *cs_min* is 5ms, *cs_max* is defined by the user into the *config_word[1]* argument of the *SIGFOX_API_set_std_config.*

The behavior of the LBT is the following:

- Enter in RX
- If no power >-80dBm is detected within the *cs_min* window, the LBT exits with success allowing a new transmission
- If a power > -80dBm is detected and no *cs_min* time has elapsed with the power <-80dBm, the LBT exits with error, preventing a new transmission.

Case no power detected:

Figure 4. **Behavior of the LBT above -80dBm is detected within the *cs_min* window**

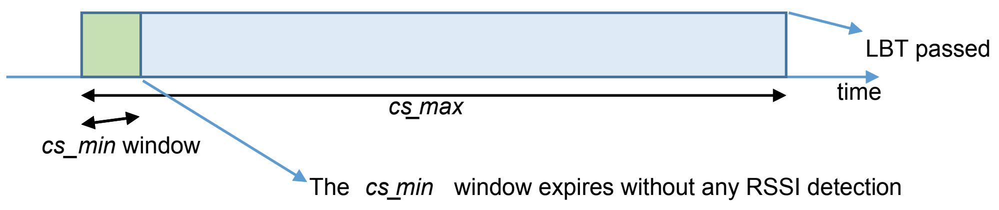
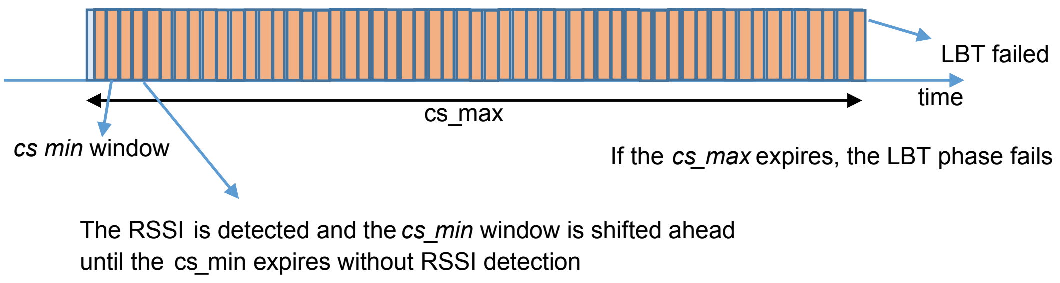


Figure 5. **Behavior of the LBT -80dBm is detected and no *cs_min* time has elapsed with the power below -80dBm**


LBT failed
time
cs_max
cs min window
If the *cs_max* expires, the LBT phase fails
The RSSI is detected and the *cs_min* window is shifted ahead until the cs_min expires without RSSI detection

If the channel is found busy for the *cs_max* time, the transmission will fail with error code 0x7D.

The LBT is implemented by the ST *RF_LIB* in the following way.

When a LBT cycle must be done, the lib will call the function *MCU_API_timer_stop_carrier_sense*.

This timer will start to count the *cs_max* time.

Then, the library will call the *RF_API_wait_for_clear_channel* function. In its implementation, the MCU will program the S2-LP as follows:

- Set the RSSI_THRESHOLD=-80dBm (this is an argument of the function)
- Register the 2 interrupts RX_TIMEOUT and RSSI_ABOVE THRESHOLD.
- Program the RX TIMEOUT to the *cs_min* (passed to the function as an argument)
- Set the device in RX and go into low power mode.
- If the RX timeout arises, it means that one *cs_min* window has elapsed without any RSSI_ABOVE_THRESHOLD (no RSSI detected) for the current *cs_min* window. Return without error after having disabled all the IRQs on the S2-LP.

If the RSSI_THRESHOLD is raised, stop the RX and restart it again. If the channel becomes free at some point within the *cs_max* and it is kept free for a time equal to *cs_min*, the RX_TIMEOUT interrupt will be raised and the function will return with no error. Otherwise the radio will be continuously stopped until the *cs_max* timeout (started by the *MCU_API_timer_stop_carrier_sense*) occurs, in this case the function returns with error.

3.6 Application example

To develop an application on the supported platforms, the application should:

1. Initialize the hardware:
   – System clock
   – S2-LP SPI initialization
   – S2-LP SDN pin initialization
2. Retrieve the crystal frequency and the carrier offset from the manufacturer data E2PROM, via the function `S2LPManagementIdentificationRFBoard()`.
3. Retrieve the Sigfox ID, PAC and RCZ via the function `enc_utils_retrieve_data` or `enc_utils_retrieve_data_from_flash`. Assume that this function will return the ID of the board in the variable uint32_t id.

The procedure for opening and setting the library differs slightly for each radio control zone.

- For **RCZ1**, the application should call the following APIs:
  – `SIGFOX_API_open (&(sfx_rc_t)RC1);`
- For the **RCZ2**, the application should call the following APIs:
  – `SIGFOX_API_open (&(sfx_rc_t)RC2);`
  – `uint32_t config_words[3]={0x1FF,0,0};`
  – `SIGFOX_API_set_std_config(config_words,0);`
- For the **RCZ3**, the application should call the following API:

  - `SIGFOX_API_open(&(sfx_rc_t)RC3C);`
  - `uint32_t config_words[3]={2,100,0};`
  - `SIGFOX_API_set_std_config(config_words,0);`
- For the **RCZ4**, the application should call the following APIs:
  - `SIGFOX_API_open (&(sfx_rc_t)RC4);`
  - `uint32_t c onfig_words[3]={0,0xF0000000,0x1F};`
  - `SIGFOX_API_set_std_config(config_words,0);`
  -

Assuming you have the 4-byte customer_data buffer to send and no downlink request:

- `SIGFOX_API_send_frame(customer_data,4,customer_resp,0,0);`

With donwnlink request:

- `SIGFOX_API_send_frame(customer_data,4,customer_resp,0,1);`

The function will return after about 50 seconds and, in the absence of errors (error code = 0), the customer_resp buffer will be filled with an 8-byte response.

For further details, refer to the SigFox_PushButton_Project example code and corresponding doxygen documentation in the ST-Sigfox package.

3.7 Test mode

The test modes are available in the separate library addon_sigfox_verified_api.

Beside this testing library, the test application needs to link both the SIGFOX_LIBRARY and the ST_RF_LIB

In order to access the test modes, it is necessary to close the Sigfox library.

The test mode API is the following:

sfx_error_t ADDON_SIGFOX_VERIFIED_API_test_mode (sfx_rc_enum_t rc_enum, sfx_test_mode_t test_mode)

where:

- rc_enum is a number representing the number of RCZ.

- test_mode is a number representing the test_mode.

Please see the doxygen documentation included in the package STSW-S2LP-SFX-DK.

The test mode should be used to perform the Sigfox verified certification using the instrumentation provided by Sigfox.

4 Current consumption on ST reference design

4.1 Current consumption for the STEVAL-FKI868V1/V2 (for RCZ1/3)

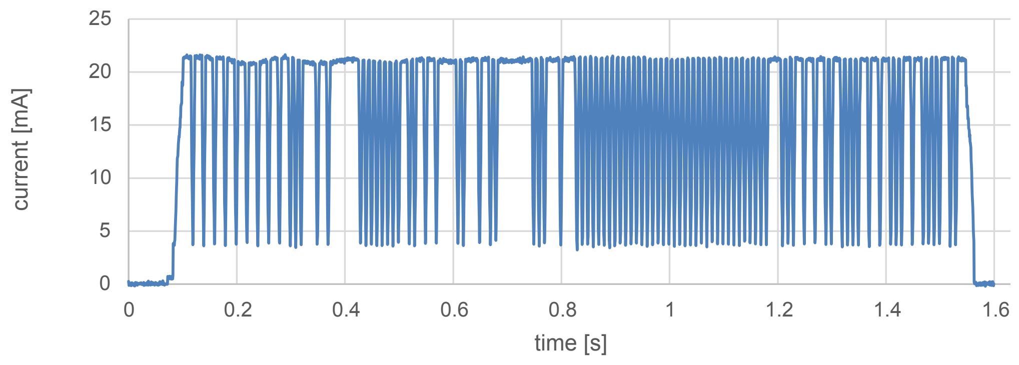
The current consumption of the S2-LP and of the STM32L152 is given below for both TX and RX phases.

In the non-active phases, the S2-LP is maintained in shutdown mode with negligible current consumption.

4.1.1 Transmission

During transmission, the averaged current on the S2-LP on the STEVAL-FKI868V2 (and STEVAL-FKI868V1) is about 17 mA.

Figure 6. S2-LP TX current profile for STEVAL-FKI868V1/V2



The current on the STM32 side is much lower because the microcontroller is set in low power mode for most of the transmission phase: it is only woken during transmission when the TX FIFO must be filled with new data.

The resulting current profile shows some 7 mA spikes every 10 ms, with troughs in between representing the sleep current below 10 µA.

Figure 7. microcontroller TX current profile for STEVAL-FKI868V1/V2

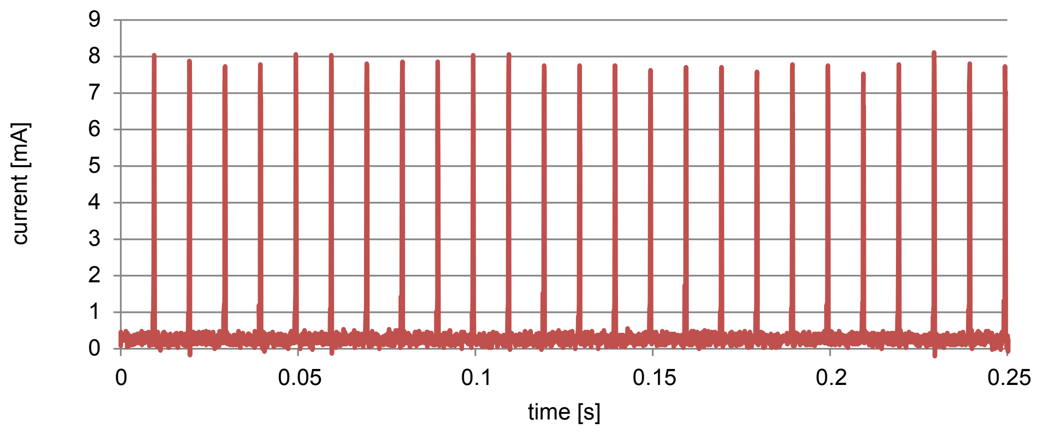
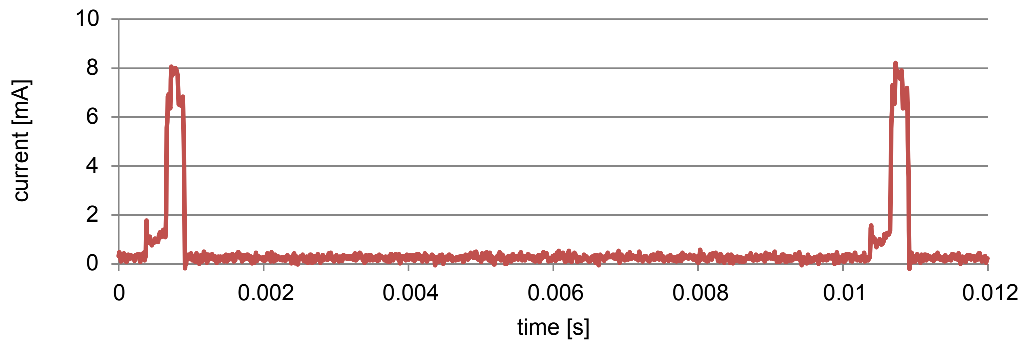


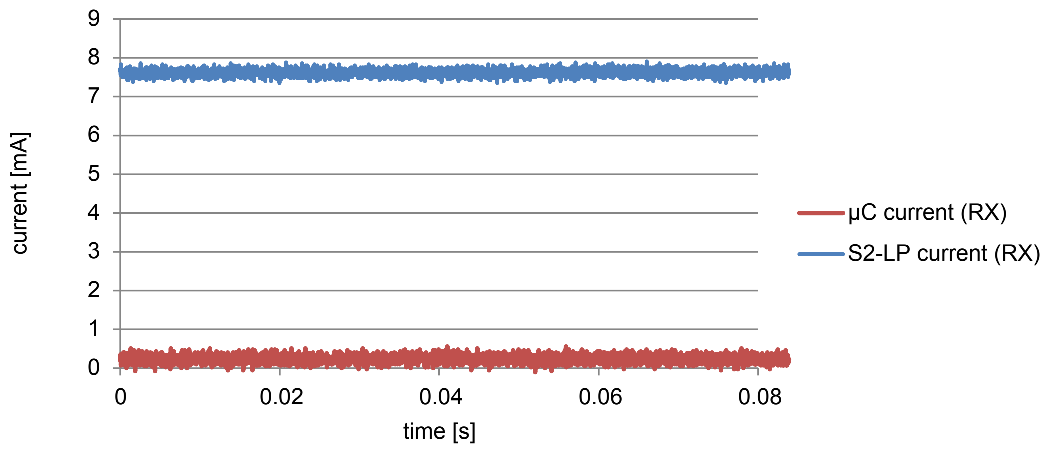
Figure 8. microcontroller TX current profile - zoom



4.1.2 Reception

According to the Sigfox protocol, reception lasts a maximum of 25 seconds. During this time, the S2-LP current consumption remains around 7.5 mA and the STM32 is maintained in stop mode with a current below 10 µA.

Figure 9. S2-LP and microcontroller RX current profile for STEVAL-FKI868V1



4.2 Current consumption for the STEVAL-FKI915V1 (for RCZ2/4)

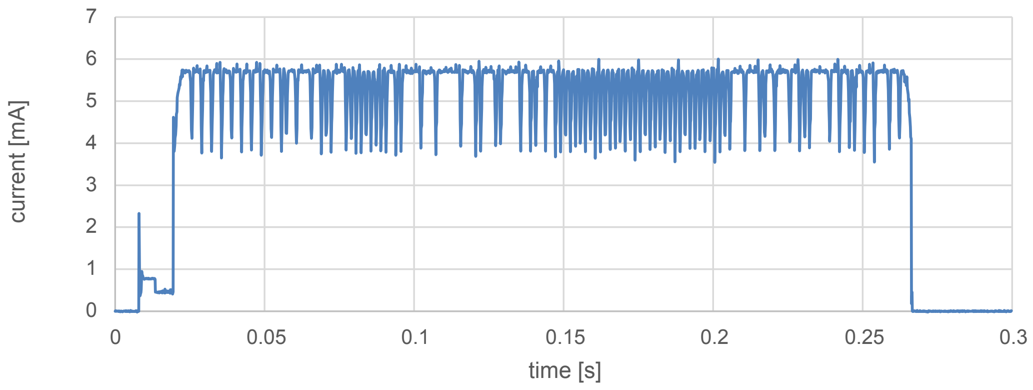
For the STEVAL-FKI915V1, we need only consider the current on the S2-LP and on the FEM (Skyworks-SE2435L) as the current consumption of the STM32 is the same as for the STEVAL-FKI868V2 (and STEVAL-FKI868V1).

In the non-active phases both the S2-LP and FEM are kept under shutdown and their current consumption is negligible.

4.2.1 Transmission

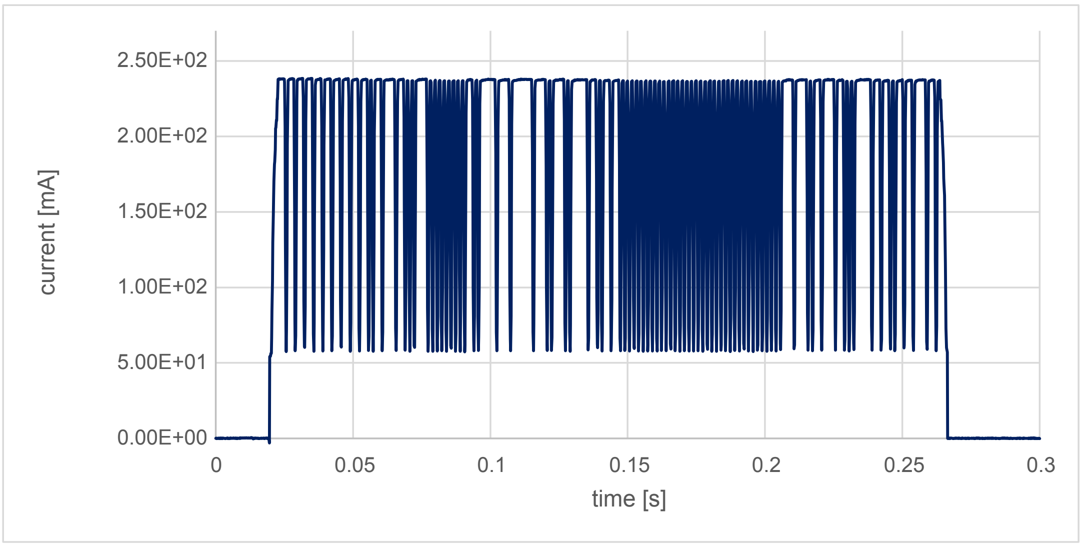
The average current consumption of the S2-LP during frame transmission is about 5 mA.

Figure 10. **S2-LP TX current profile for FKI915V1**



The 22 dBm power is provided by the FEM, with the following current shown below.

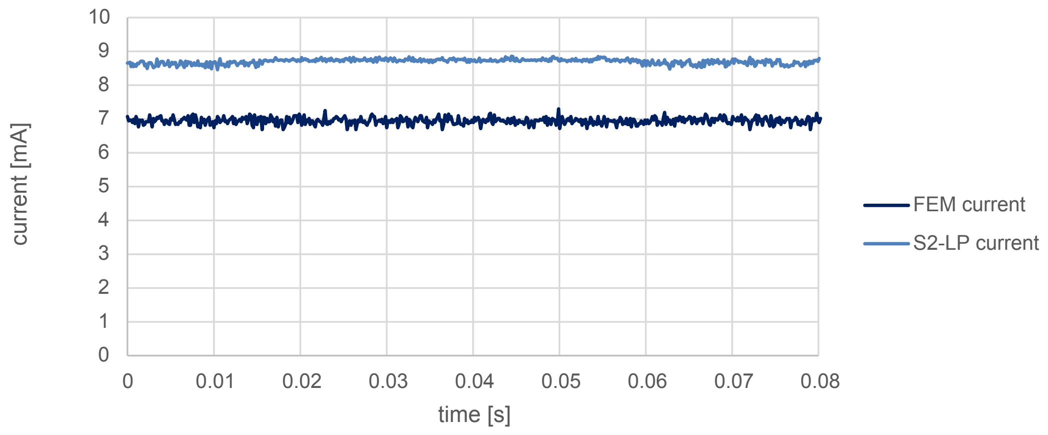
Figure 11. **FEM TX current profile for FKI915V1**



4.2.2 Reception

According to the Sigfox protocol, the reception lasts a maximum of 25 seconds. During this time, the current consumption of the S2-LP is constant, at about 8.7 mA, while the FEM has a consumption of 7 mA.

Figure 12. S2-LP and FEM RX current profile for STEVAL-FKI915V1



Revision history

Table 9. **Document revision history**

Date	Version	Changes
08-Feb-2017	1	Initial release.
01-Jun-2017	2	Minor text edits Updated *Table 1: MCU API* Updated *Table 3: Retriever API*
01-Dec-2017	3	Updated: *Figure1, Section 1, Section 2.1, section 2.2, Section 3.1 and Section 3.4.2* Added: *Test mode section.* Minor text edits.
22-May-2018	4	Updated API description. Minor text edits.
10-Sep-2018	5	Updated Introduction, Section 2.2 MCU_API, Section 2.3 Sigfox data retriever, Section 3 Application development and Section 3.6 Application example.

Contents

List of tables

List of figures