Application Note:

Long-Range FHSS Demo

Disclaimer

Long Range-Frequency Hopping Spread Spectrum (LR-FHSS) is a high link-budget, high-performance technology combining the benefits of a modulation employing low energy per bit and advanced frequency hopping schemes to achieve improved coexistence, spectral efficiency, and sensitivity. Semtech Corp. holds patents directed to aspects of the LR-FHSS technology.

Your use of LR-FHSS software made available by Semtech Corp. or its affiliates does not grant any rights to their patents for LR-FHSS technology. Rights under Semtech patents may be available via various mechanisms, including by purchasing Semtech SX1261, SX1262, or LR11xx semiconductor devices, or their authorized counterparts from Semtech or its affiliates, or their respective licensees.

Table of Contents

List of Figures

List of Tables

1 Introduction

The SX1261/2 and LR11xx transceivers have frequency hopping capabilities that make it possible for them to transmit data using the Long Range Frequency Hopping Spread Spectrum (LR-FHSS) modulation.

1.1 Purpose of This Document

This document presents how to use the SX1261/2 and LR11xx radio drivers to transmit data using LR-FHSS.

1.2 Scope of This Document

It is recommended to read this document in conjunction with the following resources:

- LR-FHSS demo package software with bundled SX126x and LR11xx drivers
    - https://github.com/Lora-net/SWDM001, v1.2 or later
- Standalone drivers
    - SX126x: https://github.com/Lora-net/sx126x_driver, v2.0.1 or later
    - LR11xx: https://github.com/Lora-net/SWDR001, v2.1.1 or later
- API reference provided in the doc directory of the LR-FHSS demo package

2 SX126x Driver

The SX126x driver provides LR-FHSS transmission capabilities, making it possible, for example, to communicate with an LR-FHSS-enabled LoRaWAN® gateway. This driver does not implement LR-FHSS reception.

The SX126x driver was mainly designed to execute simple radio commands as described in the transceiver specification. Although the SX126x hardware is capable of transmitting LR-FHSS packets, it needs help from host-MCU software to do so, in particular when it is necessary to prepare a proper LR-FHSS payload and hop to the next frequency.

Figure 1 shows the general sequence of API calls that are necessary to transmit LR-FHSS packets. Certain details do not appear in this figure for the sake of simplicity. See the API reference provided with the source code for more information.

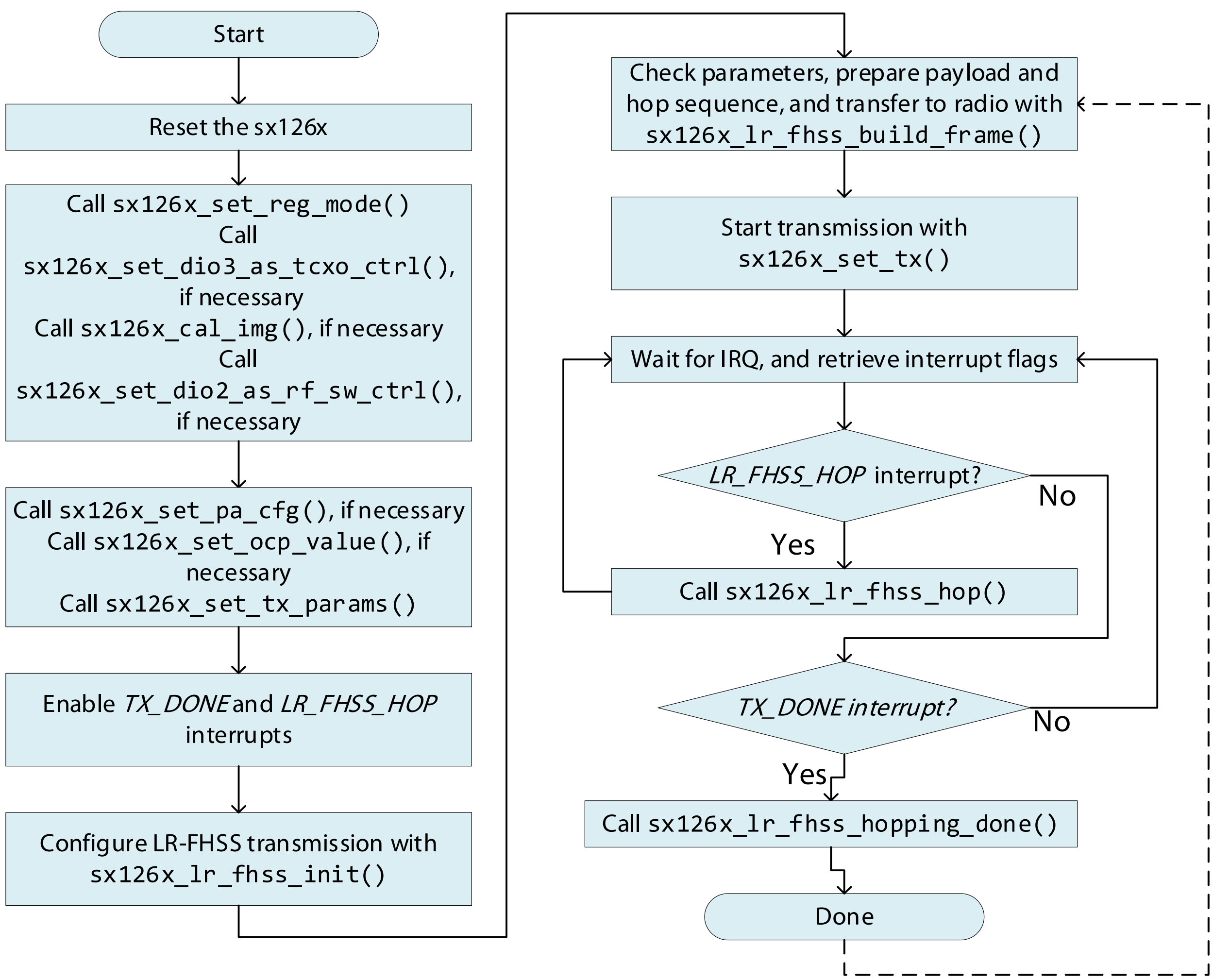

Figure 1: LR-FHSS Transmission

2.1 SX126x Driver LR-FHSS API Functions

The driver code provides the following principal LR-FHSS API functions:

Table 1: SX126x LR FHSS API Functions

Function	Description
`sx126x_lr_fhss_init()`	Initializes the modulation parameters for LR-FHSS, required after reset, after using a different transceiver modulation, or after using sleep mode without retention
`sx126x_lr_fhss_build_frame()`	Validates transmission parameters, prepares physical payload, and transfers partial hop sequence and payload to radio
`sx126x_lr_fhss_handle_hop()`	Responds to an LR_FHSS_HOP interrupt by programming the next hop frequency
`sx126x_lr_fhss_handle_tx done()`	Responds to a TX_DONE interrupt by disabling frequency hopping
`sx126x_lr_fhss_get_time_on_air_in_ms()`	Gets the time on air, in ms, for LR-FHSS transmission
`sx126x_lr_fhss_get_hop_sequence_count()`	Returns the number of valid hop sequences (512 or 384)

Most of the work performed by the driver is in the function `sx126x_lr_fhss_build_frame()`. The principal operation of this function is as follows:

- Call `sx126x_lr_fhss_process_parameters()` to check the parameters, and in case of success, generate the digest summary which contains size information that is necessary for building and sending the physical payload.
- Call `lr_fhss_build_frame()` to prepare the physical payload.
- Call `sx126x_lr_fhss_write_payload()` to transfer the physical payload to the radio.
- Call `sx126x_lr_fhss_write_hop_sequence_head()` to transfer the beginning of the hop sequence to the radio.

Only two of the above functions communicate with the radio: `sx126x_lr_fhss_write_hop_sequence_head()` and `sx126x_lr_fhss_write_payload()`. See the API documentation provided with the source code for more details.

Once `sx126x_lr_fhss_build_frame()` has been called, the SX126x driver function `sx126x_set_tx()` can be called to initiate packet transmission.

The SX126x driver provides many other API functions which implement other radio capabilities as described in the radio specification. Since they are not directly related to LR-FHSS, they are not described here in detail. Some of them, however, are needed for basic radio setup, as shown in Figure 1.

2.2 SX126x Driver LR-FHSS API Requirements

To use the driver, it is necessary to implement the API functions declared in `sx126x_hal.h`.

These functions must be implemented to provide SPI communication, reset, and wakeup functions that are necessary to use the SX126x. Many SX126x driver functions take an opaque "*void* context*" argument. The driver does not use this argument in any way, it passes it directly to the board support read and write functions which communicate with the radio. The board support package defines if and how the context argument is used. Generally, for a board with several transceivers, this argument could be used to distinguish between different transceiver instances. This demo code assumes that a single transceiver is used.

The demo application provides example implementations of these API functions that will need to be adapted to application requirements.

2.3 Library Organization

The library is organized as follows:

Table 2: SX126x Library Directories Contents

Directory	Contents
`doc\sx126x`	Driver package API documentation (open `doc\sx126x\html\index.html` with a web browser)
`lib\sx126x_driver`	Version of SX126x radio driver with LR-FHSS support
`lib\sxlib2`	Generic platform support libraries Example implementations of `sx126x_hal.h` API functions
`lib\sx_comp`	API for some common driver-related operations
`lib\sx126x_comp`	Implementation of sx_comp API functions for SX126x
`project\keil_stm32l_polling`	Keil® projects for building this demo
`platform`	Board, shield, HAL, BSP, and system files
`src\demos\sx126x_lr_fhss_ping`	sx126x_lr_fhss_ping example project
`lib\stm32cubel4subset`	Components adapted from ST Microelectronics' STM32Cube™ HAL

2.4 Driver Library Usage

1. Add the following directories to your C or C++ compiler include path:

   ```
   lib\sx126x_driver\src
   ```

2. Include the following in your application source code files:

   ```
   #include "sx126x_lr_fhss.h"
   ```

3. Include the following header file in the source code of your HAL implementation, and implement all functions declared therein:

   ```
   #include "sx126x_hal.h"
   ```

See the demo project for more information.

3 LR11xx Driver

The LR11xx transceiver provides complete LR-FHSS transmission capabilities in the firmware, making it possible, for example, to communicate with an LR-FHSS-enabled LoRaWAN gateway. This driver does not implement LR-FHSS reception.

The LR11xx LR-FHSS firmware support means that the host MCU does not need to manage hopping in real time. Also, the physical payload preparation is done by the LR11xx firmware, greatly simplifying the host MCU driver.

Figure 2 shows the general sequence of API calls that are necessary to transmit LR-FHSS packets. Certain details do not appear in this figure for the sake of simplicity. See the API reference provided with the source code for more information.

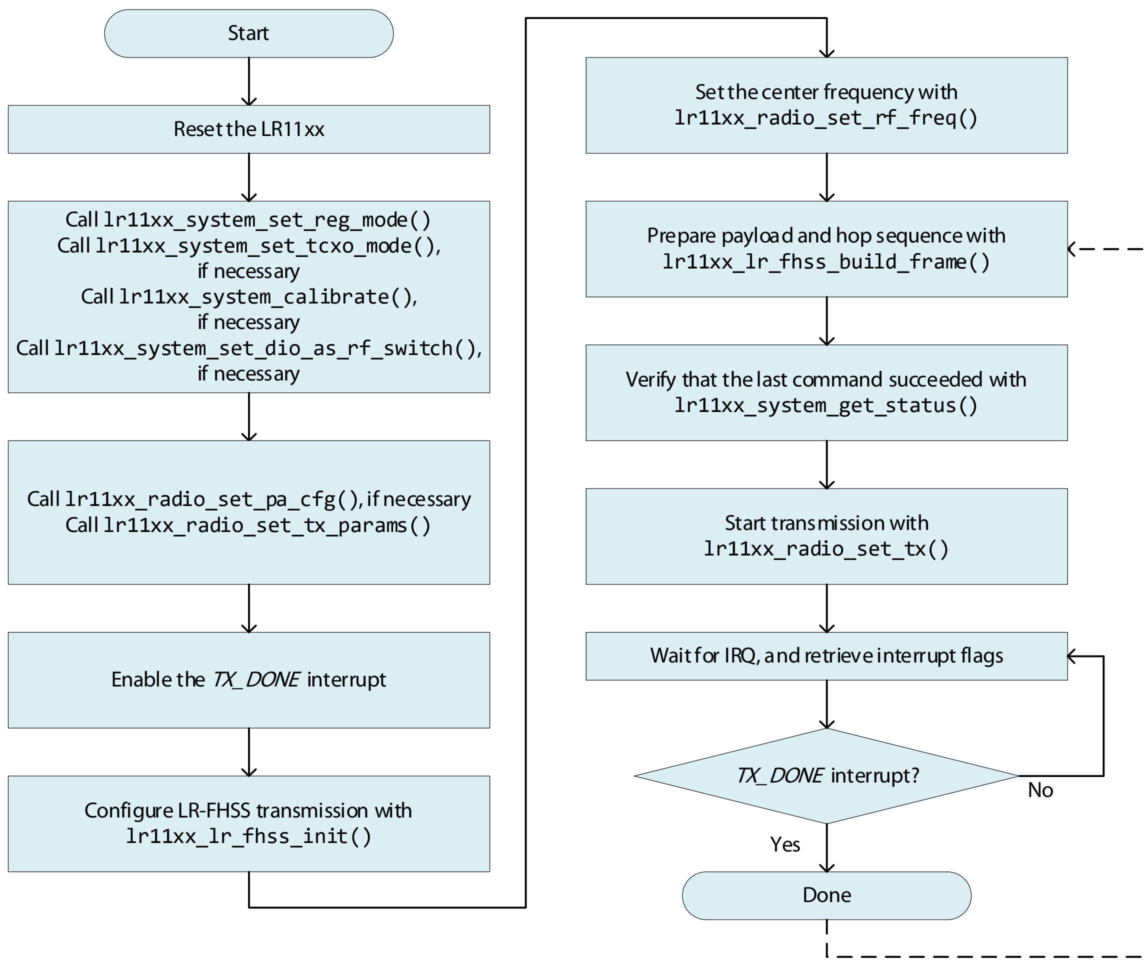

Figure 2: LR11xx LR-FHSS Transmission

3.1 LR11xx Driver LR-FHSS API Functions

The driver code provides the following principal LR-FHSS API functions:

Table 3: LR11xx LR FHSS API Functions

Function	Description
`lr11xx_lr_fhss_init()`	Initializes the modulation parameters for LR-FHSS, required after reset, after using a different transceiver modulation, or after using sleep mode without retention
`lr11xx_radio_set_rf_freq()`	Sets the LR-FHSS center frequency
`lr11xx_lr_fhss_build_frame()`	Validates transmission parameters, prepares physical payload, and transfers partial hop sequence and payload to radio
`lr11xx_radio_set_tx()`	Initiates the LR-FHSS transmission
`lr11xx_lr_fhss_get_time_on_air_in_ms()`	Gets the time on air, in ms, for LR-FHSS transmission
`lr11xx_lr_fhss_get_hop_sequence_count()`	Returns the number of valid hop sequences (512 or 384)

The LR11xx driver provides many other API functions which implement the other radio capabilities as described in the radio specification. Since they are not directly related to LR-FHSS, they are not described here in detail. Some of them, however, are needed for basic radio setup, as shown in Figure 2.

3.2 LR11xx Driver LR-FHSS API Requirements

To use the driver, it is necessary to implement the API functions declared in `lr11xx_hal.h`.

These functions must be implemented to provide SPI communication, reset, and wakeup functions that are necessary to use the LR11xx. Many LR11xx driver functions take an opaque "void* context" argument. The driver does not use this argument in any way, except that it passes it directly to the board support read and write functions that communicate with the radio. It is up to the board support package to define if and how the context argument is used. Generally, for a board with several transceivers, this argument could be used to distinguish between different transceiver instances. This demo code assumes that a single transceiver is used.

The demo application provides example implementations of these API functions that will need to be adapted to application requirements.

3.3 Library Organization

The library is organized as follows:

Table 4: LR11xx Library Directories Contents

Directory	Contents
`doc\lr11xx`	Driver package API documentation (open `doc\lr11xx\html\index.html` with a web browser)
`lib\lr11xx_driver`	Version of LR11xx radio driver with LR-FHSS support
`lib\sxlib2`	Generic platform support libraries Example implementations of `lr11xx_hal.h` API functions
`lib\sx_comp`	API for some common driver-related operations
`lib\lr11xx_comp`	Implementation of sx_comp API functions for LR11xx
`project\keil_stm32l_polling`	Keil projects for building this demo
`platform`	Board, shield, HAL, BSP, and system files
`src\demos\lr11xx_lr_fhss_ping`	lr11xx_lr_fhss_ping example project
`lib\stm32cubel4subset`	Components adapted from ST Microelectronics' STM32Cube HAL

3.4 Driver Library Usage

1. Add the following directories to your C or C++ compiler include path (replacing xx with your device number):

   `lib\lr11xx_driver\src`

2. In your application source code files, you must include:

   `#include "lr11xx_lr_fhss.h"`

   `#include "lr11xx_system.h"`

   `#include "lr11xx_radio.h"`

3. The source code of your HAL implementation must include the following header file, and implement all functions declared therein:

   `#include "lr11xx_hal.h"`

See the demo project for more information.

4 SX126x/LR11xx LR-FHSS Demo

The SX126x and LR11xx demos are part of a Keil project that compiles and runs on STM32L476 Nucleo boards, and periodically sends LR-FHSS packets of different lengths as shown in Figure 3.

In what follows, the prefix "xxxxxx" refers to either "sx126x" or "lr11xx", depending on the desired demo.

In certain cases described below, there are differences between the transceivers. In this case, the actual transceiver is referred to by name.

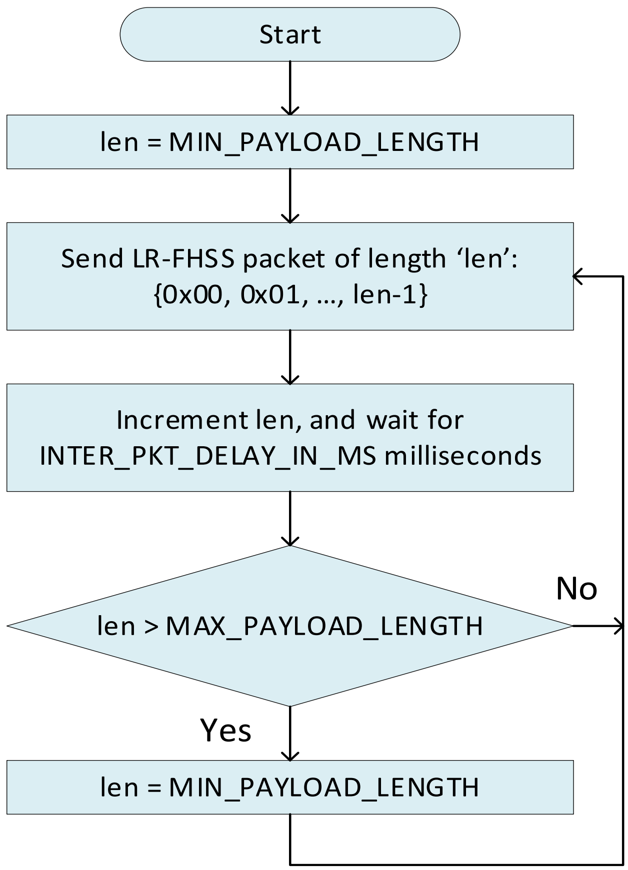

Figure 3: xxxxxx_lr_fhss_ping Demo Application Program Flow

4.1 Application Startup

The `main()` function does the following:

- Initialize the board-related hardware by calling `smtc_board_init()`.
- Initialize the shield-related hardware by calling `smtc_shield_init()`.
  Note that this initializes a global_radio object, whose address is passed to the transceiver driver API and sx_comp API as "*void* context*".
- Initialize the timing facilities.
- Call `main_()`.

The `main_()` function does the following:

- Initialize the system time.
- Call `sxlib_Radio_identify_radio()`, which identifies the radio shield and binds `xxxxxx_comp` function definitions to the radio function pointer *(void* context)* so that sx_comp API calls point to the proper functions for the specific transceiver that was identified, with any necessary information like TCXO configuration held in the sx_comp_t object.
- Initialize the radio event-handling mechanism.
- Call `sx_comp_radio_reset_and_init()`, which uses the binding described above to indirectly call `xxxxxx_comp_radio_reset_and_init_base()`. The latter function calls the transceiver and sx_comp API functions that reset the radio, set the RegMode, configure automatic antenna switch control, and perform basic calibration.
- Start the Figure 3 demo by calling `sxlib_demo_launcher()`, which is found in the `xxxxxx_lr_fhss_ping_start.c` file. This file, and all the other application logic related to this flowchart, can be found in the files of the Keil project group named `demo_xxxxxx_lr_fhss_ping`.

4.2 UART Diagnostics

The demo performs serial logging if this C preprocessor symbol is set globally:

SXLIB_LOG_LEVEL=1000

Connect to the Nucleo debug Virtual COM port at 115000 baud, N-8-1.

Note that it is possible to reduce the code size by setting SXLIB_LOG_LEVEL to 0, which disables logging.

4.3 Configurability

Project configuration changes can be achieved through both link-time file selection, and parameter and function modification. Certain C pre-processor symbols can be added or removed to the global configuration to change behavior, via the **Project → Options For Target → C/C++ configuration** variables.

See Table 5, Table 6, and Section 4.6 for more information.

Table 5: Configurable Parameters

Parameter	File	Comment
GPIO parameters related to shield	`smtc_shield.c`	Adapts software to work with different shields
SPI parameters related to shield	`smtc_shield.c/smtc_board.c`	Adapts software to work with different shields
INTER_PKT_DELAY_IN_MS	`xxxxxx_lr_fhss_ping_start.c`	Simple inter-packet delay (must be 20000 or more to satisfy FCC requirements)
MAX_PAYLOAD_LENGTH	`xxxxxx_lr_fhss_ping.c`	Lower limit for the payload length sweep
MIN_PAYLOAD_LENGTH	`xxxxxx_lr_fhss_ping.c`	Upper limit for the payload length sweep
device_offset	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.bw (bandwidth)	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.cr (coding rate)	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.enable_hopping	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.grid	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.sync_word	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
lr_fhss_params.header_count	`xxxxxx_lr_fhss_ping.c`	see `lr_fhss_send_packet()`
CONFIG_ALLOW_SMTC_RADIO_SLEEP	`xxxxxx_lr_fhss_ping.c`	Can be defined in global pre-processor symbols to allow radio to sleep between transmissions
RF_FREQUENCY	`xxxxxx_lr_fhss_ping.c`	LR-FHSS center frequency (must be set according to region)
cal_img_freq1_in_mhz, cal_img_freq2_in_mhz	`identify_xxxxxx.c`	Calibration frequency range used by `sx_comp_radio_reset_and_init()`. These must be set according to region, for lr11xx. For sx126x, these are auto-detected.
POWER_IN_DBM	`xxxxxx_lr_fhss_ping.c`	`Output power ()` (must be set according to region)

Table 6: Configuration through Function Modification

Function	File	Comment
smtc_shield_init()	shield_sx126x/smtc_shield.c for SX126x shield_lr11xx/smtc_shield.c for LR11xx	Configure shield, from main(), in main_polling.c or main_rtx5.c.
smtc_board_init()	smtc_board.c	Configure board, from main(), in main_polling.c or main_rtx5.c.
main_single_interface_init()	prepare_interface.c	Select LEDs to be used by the application, from main(), in main_single_radio.c.
sxlib_Radio_identify_radio()	identify_sx126x.c/identify_lr1110_lr1120_tcxo.c/identify_lr1121_xtal.c	If necessary, auto-identify the transceiver being used by reading GPIO configuration bridges. Define TCXO parameters and calibration frequencies. This also binds xxxxxx_comp function definitions to the radio function pointer (context) so that sx_comp API calls point to the proper functions for the identified transceiver. Called from main_(), in main_single_radio.c.

4.4 Hardware Requirements

4.4.1 MCU Board

An STM32L476 Nucleo board is required.

4.4.2 SX126x Shield

For the following Semtech part numbers, the calibration frequency, clock source, and SX126x part number should be properly identified and configured by the function `sxlib_Radio_identify_radio()`:

- SX1261MB1BAS
- SX1261MB2BAS
- SX1262MB1CAS
- SX1262MB1PAS
- SX1262MB2CAS

Manual setting of region-specific transmission power and center frequency are required as described in Section 4.3.

It may be possible to get other Semtech SX126x boards to work through manual modification of `smtc_board_init()`, `smtc_shield_init()`, or `sxlib_Radio_identify_radio()`.

4.4.3 LR11xx Shield

This demo software does not perform active identification of the LR11xx shield. The following shields have been tested with the lr1110_lr1120_tcxo target:

- LR1110MB1DIS
- LR1110MB1DJS
- LR1120MB1DIS
- LR1120MB1DJS

The following shield has been tested with the lr1121_xtal target:

- LR1121MB1DIS

Manual setting of region-specific transmission power, calibration frequency, and center frequency are also required, as described in Section 4.3.

It may be possible to get other Semtech LR11xx boards to work through manual modification of `smtc_board_init()`, `smtc_shield_init()`, or `sxlib_Radio_identify_radio()`. Shields using a TCXO are likely to work with the lr1110_lr1120_tcxo target. Shields using a crystal oscillator are likely to work with the lr1121_xtal target. For LR1110, firmware version 0x0307 or later is required. Find recent LR11xx firmware updates here: https://github.com/Lora-net/radio_firmware_images/

4.5 Software Requirements

The free 32k-limited lite version of Keil® MDK (https://www2.keil.com/mdk5) is all you need to compile, debug, and run the LR-FHSS demo application. If compiler optimization is disabled, this may no longer be possible.

4.6 Compile & Run Demo

4.6.1 Asynchronous Bare-Metal Example

1. Open Arm® Keil® µVision®5 and start the Pack Installer from the toolbar. From the Devices tab, select the **MCU STMicroelectronics/STM32L4/STM32L476RGTx**. In the right pane, make sure that the Keil::STM32L4xx_DFP Keil and ARM::CMSIS packages are installed and up-to-date.
2. With Keil MDK, open the `project\keil_stm32l_polling\STM32L476.uvprojx` file.
3. Make sure that the desired transceiver, **sx126x, lr1110_lr1120_tcxo**, or **lr1121_xtal** is selected in the Keil 'Select Target' list.
4. Select **Project** -> **Build Target**.
5. Connect the appropriate hardware as described in Section 4.4.
6. (Optional) Connect a terminal emulator to the debug UART, as described in Section 4.2.
7. In Keil MDK, select **Project** -> **Options for Target** -> **Debug**, and select **ST-Link Debugger**.
8. For the **Use:** option, select **ST-Link Debugger** and select **Settings**.
9. Activate the **Download Options**: **Verify Code Download** and **Download to Flash**.
10. Select the '**under Reset**' Connect Options, and Port '**SW**'.
11. Press **OK**.
12. Press **OK**.
13. (Optional) Start an LR-FHSS gateway if you wish to test packet reception.
14. Select **Debug** -> **Start Debug Session**, then **Debug** -> **Run**.

Figure 4 shows an example of the UART log with an SX126x transmitter, and Figure 5 shows an example of successful packet reception by the gateway.

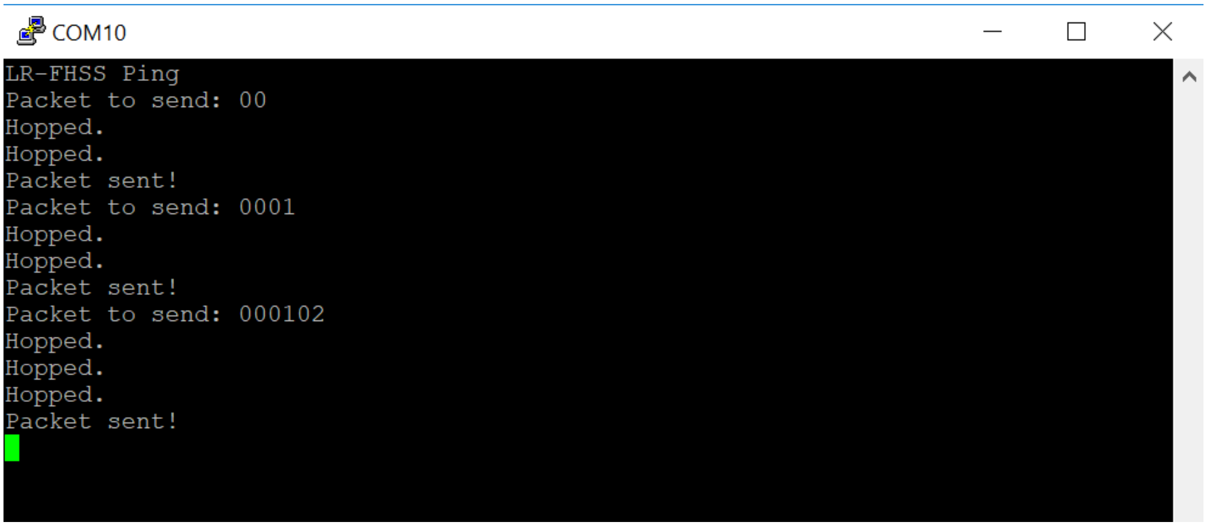

Figure 4: Debug Log

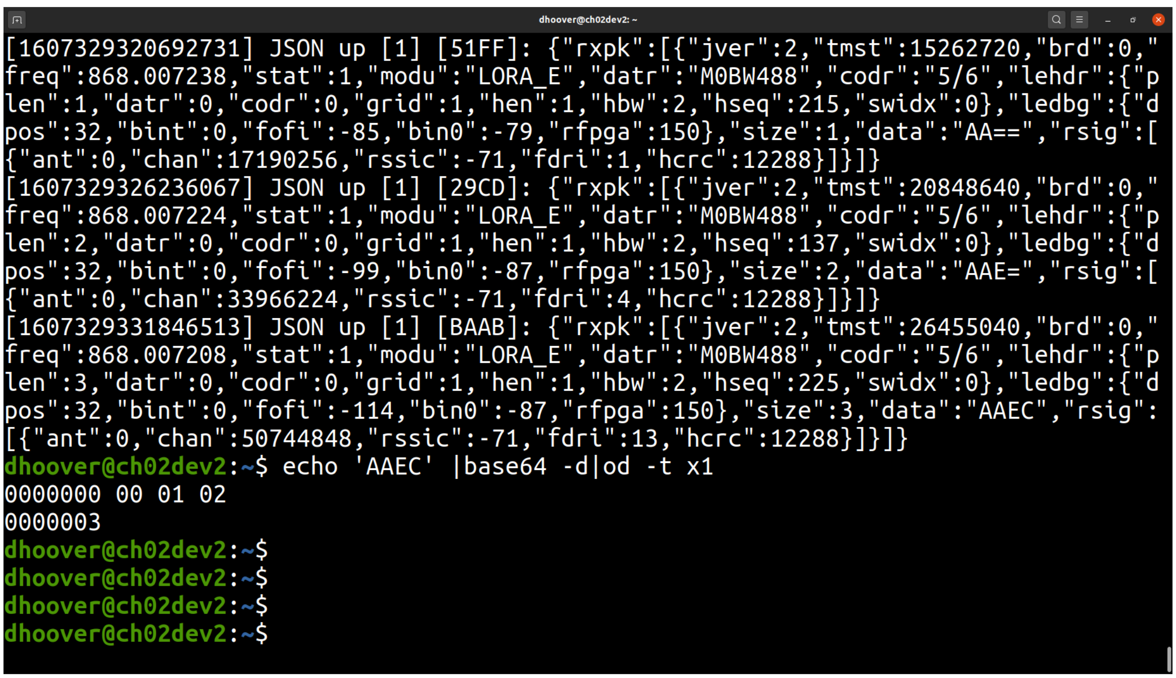

Figure 5: Gateway Log And Payload Decoding

4.6.2 Synchronous RTX5 Example

An RTX5-based synchronous variant of the demo exists.

The principal difference is in the use of the `xxxxxx_lr_fhss_ping_sync.c` main loop instead of the event-based code in `xxxxxx_lr_fhss_ping_async.c`.

5 Application Debugging

5.1 Observe LR-FHSS Hopping Using SX126x Demo

As described in Section 4.2, you can observe if frequency hopping is occurring by activating logging and connecting a terminal emulator to the UART.

You can see the approximate timing of each hop by activating a strobe on a GPIO output for each hop interrupt and setting the GPIO line high during the application sleep phase.

5.1.1 Demo Parameters

In this section, we configure the sx126x_lr_fhss_ping demo to transmit 15-byte user payloads at 868MHz, with a 136.7kHz bandwidth and a 3.9kHz grid.

In the `sx126x_lr_fhss_ping.c` file, make sure that the bandwidth and grid parameters are properly configured, and set header_count to 2. Set *MIN_PAYLOAD_LENGTH* and *MAX_PAYLOAD_LENGTH* to 15.

In `sx126x_lr_fhss_ping_start.c`, set *INTER_PKT_DELAY_IN_MILLI* to a small value, like 10.

5.1.2 Activate Strobe on Each Hop

To activate a strobe on each hop, a global LED device named **global_gpio_led_debug** has been defined in the shield files. It is active by default on the STM32 PC10 GPIO line (Nucleo CN7, pin 1).

1. To access this global variable, near the top of `sx126x_lr_fhss_ping.c` add:

   ```
   #include "smtc_shield.h"
   ```

2. To activate GPIO toggling, in `sx126x_lr_fhss_ping.c` at the very top of both `sx126x_lr_fhss_ping_handle_hop()` and `sx126x_lr_fhss_ping_handle_tx_done()` functions, add:

   ```
   sxlib_Gpio_Led_on( &global_gpio_led_debug );
   ```

3. In `sx126x_lr_fhss_ping.c` at the very bottom of `sx126x_lr_fhss_ping_handle_hop()` function, and at the top of the `sx126x_lr_fhss_ping_launch()` function, add:

   ```
   sxlib_Gpio_Led_off( &global_gpio_led_debug );
   ```

4. Set the *MIN_PAYLOAD_LENGTH* and the *MAX_PAYLOAD_LENGTH* definitions to 15, for example, so that every transmitted packet has the same length.

5. Rebuild the project and start the application.

5.1.3 Determine Theoretical Number of Hops and Their Durations

Add a breakpoint to the line of `sx126x_lr_fhss_ping.c` that calls `sx126x_lr_fhss_build_frame()` and restart the application. When the breakpoint is reached, step **into** the `sx126x_lr_fhss_build_frame()` function, and then step **over** `sx126x_lr_fhss_process_parameters()`. After returning without error, the latter function should have determined the number of hops and the physical payload size in the `state->digest` structure. With the unmodified LR-FHSS parameters, you can observe state->digest.nb_bytes=51, state->digest.nb_bits=407, and state->digest.nb_hops=6. Of these 6 blocks that are transmitted:

- The first two are header blocks composed of 114 bits, with a duration of 114 / 488.28125 = 233 ms.
- The next three blocks are complete data blocks composed of 50 bits, with a duration of 50 / 488.28125 = 102 ms.
- The last data block is partial, length 407 – 114 – 114 – 3 * 50 = 29 bits, with a duration of 29 / 488.28125 = 59 ms.

5.1.4 Observe Time between Hop Strobes

The timing markers in Figure 6 show the amount of time between hop strobes. This, of course, does not correspond exactly with the theoretical values (above) because the debug line strobes are not synchronized with the operations taking place within the transceiver. If you closely observe the first interval, you see that the strobe occurs before the SetTx call. Using the transceiver BUSY line as a reference results in a better measurement of the first interval, because it indicates at what point the transceiver is almost ready to transmit.

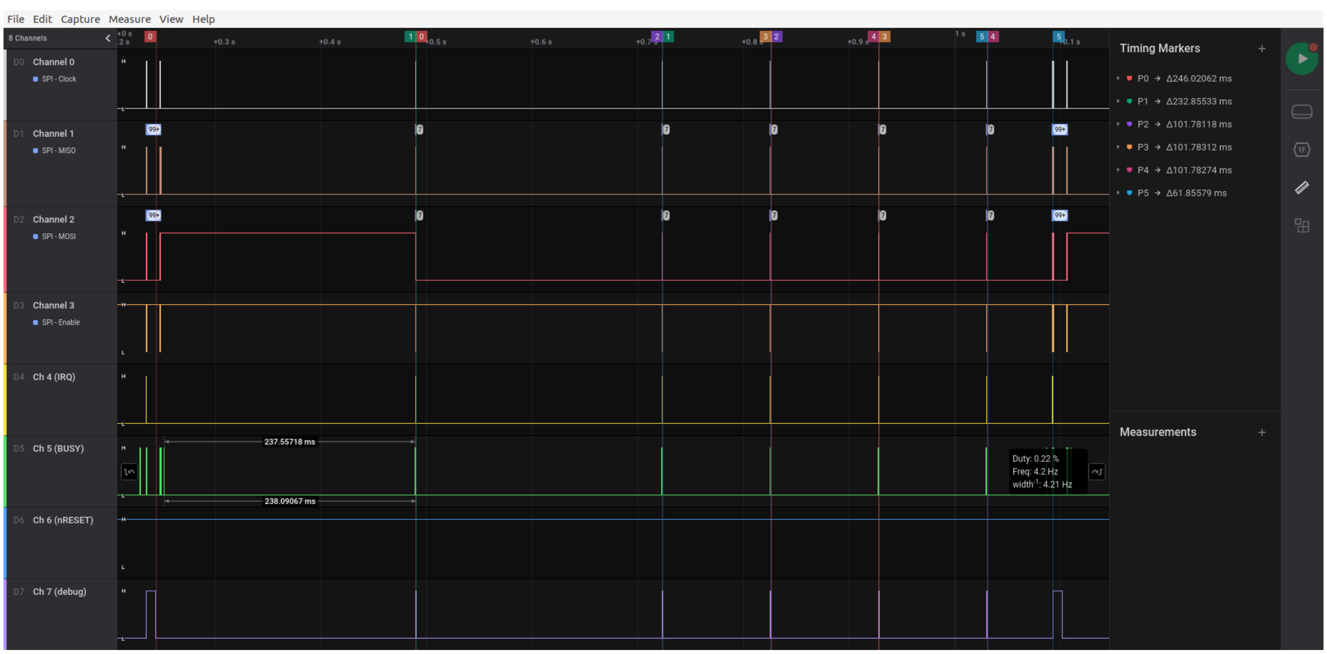

Figure 6: Approximate Hop Durations

5.2 LR-FHSS Signal Spectrum

This example demonstrates how to observe LR-FHSS hopping on a spectrum analyzer.

5.2.1 Demo Parameters

In this section, we configure the xxxxxx_lr_fhss_ping demo to transmit 15-byte user payloads at 868MHz, with a 136.7kHz bandwidth and a 3.9kHz grid.

In the `xxxxxx_lr_fhss_ping.c` file, make sure that the bandwidth and grid parameters are properly configured, and set header_count to 2. Also, set *MIN_PAYLOAD_LENGTH* and *MAX_PAYLOAD_LENGTH* to 15.

In `xxxxxx_lr_fhss_ping_start.c`, set *INTER_PKT_DELAY_IN_MILLI* to a small value, like 10.

5.2.2 Single Sweep Spectrum

After setting the center frequency to 868MHz, the span to 250kHz, and the resolution bandwidth to 500Hz, you can see a single sweep in Figure 7.

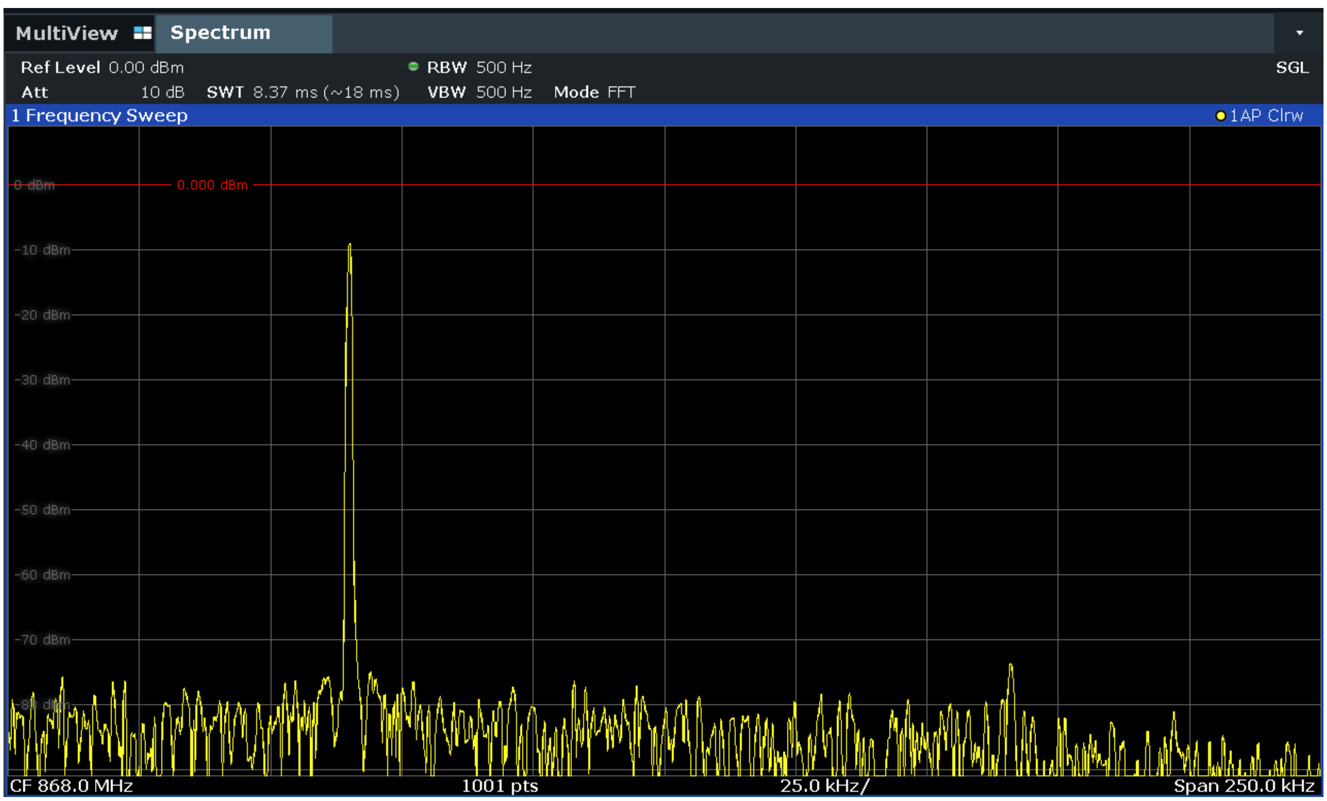

Figure 7: Single Hop

5.2.3 Max Hold Spectrum over Several Sweeps

According to the specification, Ngrid=35 for this configuration. If the spectrum analyzer trace is set to 'Max Hold', then after a few minutes you should observe a trace similar to Figure 8. 'Max Hold' preserves the maximum amplitude that was attained over the entire trace execution. As in Section 5.1, there are two header blocks and four data blocks. Therefore, each complete transmission occupies 6 different frequencies. Over the course of many transmissions, however, all 35 frequencies will be used. Note that the difference between the markers corresponds to the 136.7kHz LR-FHSS bandwidth.

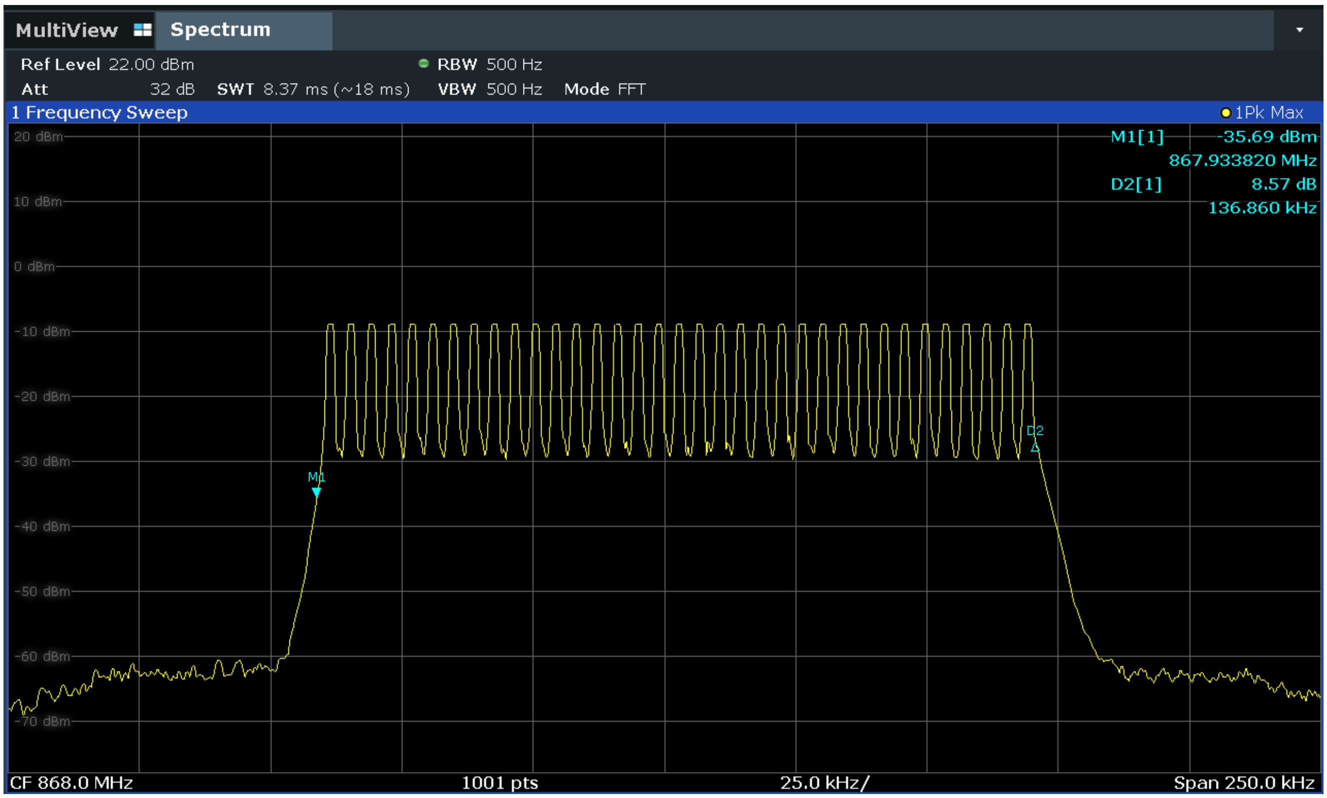

Figure 8: Hopping Observed With Max Hold Trace Settings

5.2.4 Spectrogram

During a single transmission, two sync header blocks and four data blocks are transmitted, each at a different frequency. This is clearly visible in Figure 9, where you also observe that the last data block is shorter than the three others.

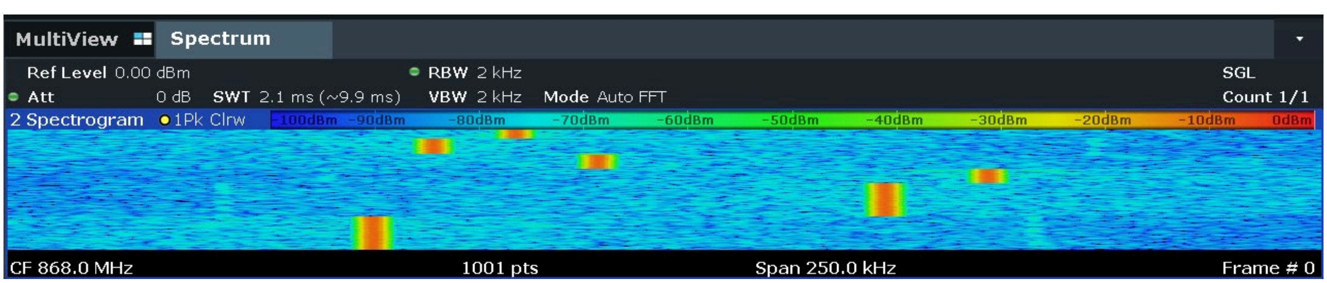

Figure 9: Spectrogram over a Single Transmission

6 Conclusion

This application note has shown how to transmit Long-Range Frequency Hopping Spread Spectrum packets using an STM32 Nucleo board equipped with a Semtech SX1261/2 or LR11xx radio shield.

7 Glossary

Term	Description
API	Application Programming Interface
BW	Bandwidth
CR	Coding Rate
GPIO	General Purpose Input Output
LoRa®	Long Range Communication The LoRa® Mark is a registered trademark of the Semtech Corporation
LoRaWAN®	Long Range Low Power, Wide Area (LPWA) Networking Standard
LR-FHSS	Long-Range Frequency Hopping Spread Spectrum
MCU	Micro-Controller Unit
SPI	Serial Peripheral Interface
TCXO	Temperature Compensated Crystal Oscillator
UART	Universal Asynchronous Receiver/Transmitter

8 Revision History

Version	ECO	Date	Modifications
1.0	-	Jul 2021	First Release
1.1	060546	Feb 2022	Public Release
1.2	064753	Jan 2023	Added support for LR1120 and LR1121 transceivers

Contact Information

Semtech Corporation
Wireless & Sensing Products
200 Flynn Road, Camarillo, CA 93012
E-mail: sales@semtech.com
Phone: (805) 498-2111, Fax: (805) 498-3804