UM2073
User manual

STM32 LoRa™ software expansion for STM32Cube

Introduction

This application note describes the LoRa™ embedded expansion software implementation on the STM32Lx Series; this software is called I-CUBE-LRWAN. This document also explains how to interface with the LoRaWAN™ to manage the LoRa™ wireless link.

LoRa™ is a type of wireless telecommunication network designed to allow long range communications at a very low bit-rate and enabling long-life battery operated sensors. LoRaWAN™ defines the communication and security protocol that ensures the interoperability with the LoRa™ network.

The LoRa™ software expansion is compliant with the LoRa™ Alliance specification protocol named LoRaWAN™ (version V1.0 January 2015).

The I-CUBE-LRWAN main features are the following:

- 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 I-CUBE- LRWAN software is based on the STM32Cube HAL drivers (see *Section 2*).

This user manual provides customer application examples on NUCLEO-L053R8 , NUCLEO-L152RE and NUCLEOL476RG using Semtech expansion board. The following Semtech radios are supported by the mentioned boards: SX1276MB1MAS, SX1276MB1LAS and SX1272MB2DAS.

This document uses the NUCLEO-L053R8 board as an example, but note that this document is applicable to all STM32 devices and that the I-CUBE-LRWAN can be ported to other STM32 series boards.

Contents

1 **Overview** 7

1.1 Acronyms and abbreviations 7

1.2 References 7

2 **LoRa™ standard overview** 8

2.1 Overview 8

2.2 Network architecture 8

2.2.1 End-device architecture 9

2.2.2 End-device classes 9

2.2.3 End-device activation (joining) 10

2.2.4 Regional spectrum allocation 11

2.3 Network layer 11

2.3.1 Physical layer (PHY) 12

2.3.2 MAC sublayer 12

2.4 Message flow 12

2.4.1 End-device activation (joining) 12

2.4.2 End-device data communication (class A) 13

2.5 Data flow 15

3 **LRWAN middleware description** 16

3.1 Overview 16

3.2 Features 18

3.3 Architecture 18

3.4 Hardware related components 19

3.4.1 Radio reset 19

3.4.2 SPI 19

3.4.3 RTC 19

3.4.4 Interrupt lines 20

4 **LRWAN middleware programming guidelines** 21

4.1 Middleware initialization 21

4.2 Middleware MAC layer functions 21

4.2.1 MCPS services 21

4.2.2 MLME services . . . 22
4.2.3 MIB services . . . 22
4.3 Middleware MAC layer callbacks . . . 22
4.3.1 MCPS . . . 22
4.3.2 MLME . . . 23
4.3.3 MIB . . . 23
4.3.4 Battery level . . . 23
4.4 Middleware utilities functions . . . 23
4.4.1 Timer server . . . 23
4.5 Middleware utilities callbacks . . . 24
4.5.1 Delay Rx window . . . 24
4.5.2 Delay for MAC layer state checking . . . 24
4.5.3 Delay for Tx frame transmission . . . 24
4.5.4 Delay for Rx frame . . . 24
4.6 Middleware low-power functions . . . 25
4.7 Middleware class A application function . . . 25
4.7.1 LoRa™ class A initialization . . . 27
4.7.2 LoRa™ class A FSM entry point . . . 27
4.8 LIB class A application callbacks . . . 28
4.8.1 Current battery level . . . 28
4.8.2 Board unique ID . . . 28
4.8.3 Board random seed . . . 28
4.8.4 Make Tx frame . . . 28
4.8.5 Make Rx frame . . . 28

5 Getting started . . . 29
5.1 Hardware description . . . 29
5.1.1 STM32 Nucleo boards . . . 29
5.1.2 ST X-NUCLEO IKS01A1 expansion board . . . 30
5.1.3 LoRa™ radio expansion board . . . 30
5.1.4 Hardware mapping . . . 31
5.1.5 Hardware modifications . . . 32
5.2 Software description . . . 32
5.2.1 Compilation switches . . . 33
5.3 PingPong application description . . . 34

6 System performances 36
6.1 Memory footprints 36
6.2 Real-time constraints 36
6.3 Power consumption 37

7 Revision history 39

List of tables

Table 1. List of acronyms and abbreviations . . . 7
Table 2. LoRa™ classes intended usage . . . 8
Table 3. LoraWAN™ regional spectrum allocation . . . 11
Table 4. Middleware initialization function . . . 21
Table 5. MCPS services function . . . 21
Table 6. MLME services function . . . 22
Table 7. MIB services functions . . . 22
Table 8. MCPS primitives . . . 22
Table 9. MLME primitive . . . 23
Table 10. Battery level function . . . 23
Table 11. Timer server functions . . . 23
Table 12. Delay Rx functions . . . 24
Table 13. Delay for MAC layer state checking function . . . 24
Table 14. Delay for Tx frame transmission function . . . 24
Table 15. delay for Rx frame function . . . 24
Table 16. Middleware low-power functions . . . 25
Table 17. LoRa™ class A initialization function . . . 27
Table 18. LoRa™ class A FSM entry point function . . . 27
Table 19. Current battery level function . . . 28
Table 20. Board unique ID function . . . 28
Table 21. Board random seed function . . . 28
Table 22. make Tx frame function . . . 28
Table 23. Make Rx frame . . . 28
Table 24. LoRa™ radio expansion boards characteristics . . . 30
Table 25. Hardware mappings: used I/Os . . . 31
Table 26. STM32L0xx IRQ priorities . . . 32
Table 27. Switch options for the application's configuration . . . 34
Table 28. Memory footprint measured values . . . 36
Table 29. Document revision history . . . 39

List of figures

Figure 1. Network diagram . . . 9
Figure 2. TX/Rx time diagram (class A) . . . 9
Figure 3. Tx/Rx time diagram (class B) . . . 10
Figure 4. Tx/Rx time diagram (class C) . . . 10
Figure 5. LoRaWAN™ layers . . . 12
Figure 6. Message sequence chart for joining (MLME primitives) . . . 13
Figure 7. Message sequence chart for confirmed-data (MCPS primitives) . . . 14
Figure 8. Message sequence chart for unconfirmed-data (MCPS primitives) . . . 14
Figure 9. Data flow . . . 15
Figure 10. Project files structure . . . 17
Figure 11. Firmware's main design . . . 18
Figure 12. Operation model . . . 26
Figure 13. LoRa™ state machine . . . 27
Figure 14. Hardware setup . . . 29
Figure 15. ST X-NUCLEO IKS01A1 expansion board . . . 30
Figure 16. SX1276MB1LAS board . . . 31
Figure 17. I-CUBE-LRWAN structure . . . 33
Figure 18. PingPong setup . . . 35
Figure 19. Rx/Tx time diagram . . . 36
Figure 20. STM32L0 current consumption against time . . . 37
Figure 21. STM32L0 current consumption against time: zoom on run mode period . . . 38

1 Overview

1.1 Acronyms and abbreviations

Table 1. List of acronyms and abbreviations

Term	Definition
ABP	Activation by personalization
APP	Application
API	Application programming interface
BSP	Board support package
FSM	Finite state machine
HAL	Hardware abstraction layer
IOT	Internet of things
LoRa™	Long range radio technology
LoRaWan™	LoRa wide-area network
LPWAN	Low-power, wide-area network
MAC	Media access control
MCPS	MAC common part sublayer
MIB	MAC information base
MLME	MAC sublayer management entity
MPDU	MAC protocol data unit
OTAA	Over-this-air activation
PLME	Physical sublayer management entity
PPDU	Physical protocol data unit
SAP	Service access point

1.2 References

LoRa™ Alliance Specification Protocol named LoRaWanTM version V1.0 - 2015, January - Released

IEEE Std 802.15.4TM - 2011. Low-Rate Wireless Personal Area Networks (LR-WPANs)

2 LoRa™ standard overview

2.1 Overview

This chapter is intended to provide a general overview of the LoRa™ and LoRaWAN™ recommendations. It focuses in particular on the LoRa™ end device which is the core subject of this user manual.

LoRa™ is a type of wireless telecommunication network designed to allow long range communication at a very low bit-rate and enabling long-life battery operated sensors. LoRaWAN™ defines the communication and security protocol ensuring the interoperability with the LoRa™ Network.

The LoRa™ software expansion is compliant with the LoRa™ Alliance specification protocol named LoRaWAN™ (version V1.0 January 2015).

Table 2 shows the LoRa™ classes usage definition. Refer to *Section 2.2.2* for further details on these classes.

Table 2. LoRa™ classes intended usage

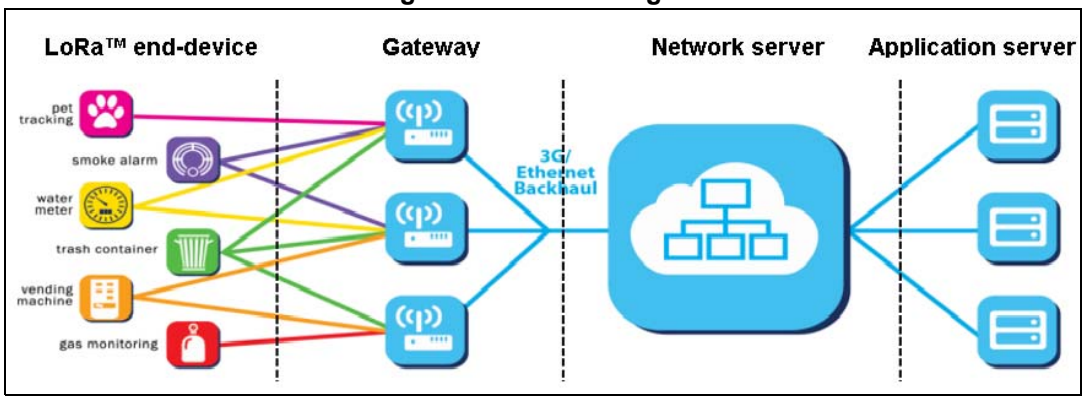
Class name	Intended usage
A - All	Battery powered sensors or actuators with no latency constraint.Most energy efficient communication class.Must be supported by all devices.
B - Beacon	Battery powered actuators. Energy efficient communication class for latency controlled downlink. Based on slotted communication synchronized with a network beacon.
C - Continuous	Main powered actuators. Devices which can afford to listen continuously. No latency for downlink communication.

Note: While the physical layer of LoRa™ is proprietary, the rest of the protocol stack (LoRaWAN™) is kept open and its development is carried out by the LoRa™ Alliance.

2.2 Network architecture

LoRaWAN™ network is structured in a star of stars topology, where the end devices are connected via a single LoRa™ link to one gateway as shown in *Figure 1*.

Figure 1. Network diagram



2.2.1 End-device architecture

The end device is made up of an RF transceiver (also know as radio) and a host STM32 MCU. The RF transceiver is composed of a modem and an RF up-converter. The MCU implements the radio driver, the LoRaWan™ stack and optionally the sensor drivers.

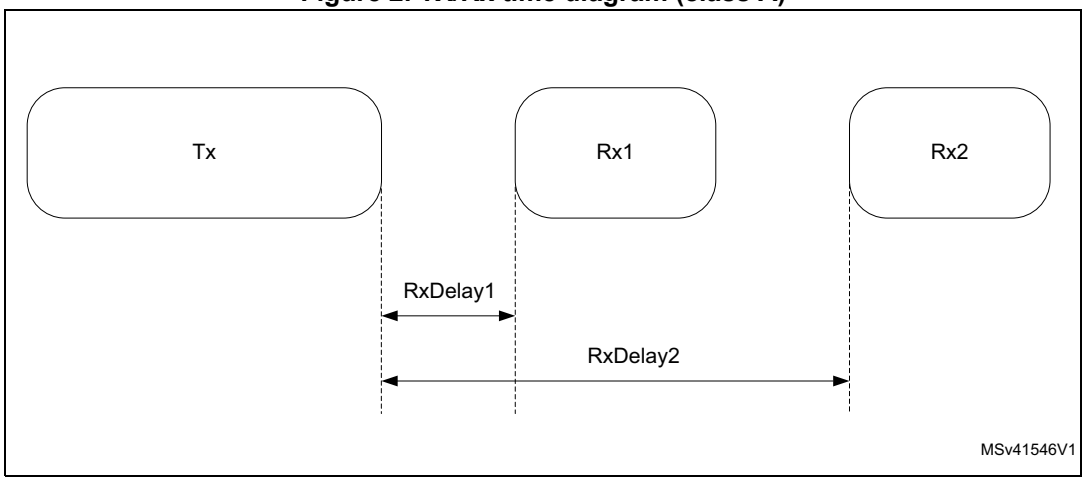
2.2.2 End-device classes

The LoRaWAN™ has several different classes of end-point devices, this is done in order to address the different needs reflected in the wide range of applications.

Bi-directional end-devices - class A - (all devices)

- Class A operation is the lowest power end-device system
- Each end-device's uplink transmission is followed by two short downlink receive windows
- Downlink communication from the server shortly after the end-device has sent an uplink transmission (see *Figure 2*)
- Transmission slot is based on own communication needs of the end-device (ALOHA-type of protocol).

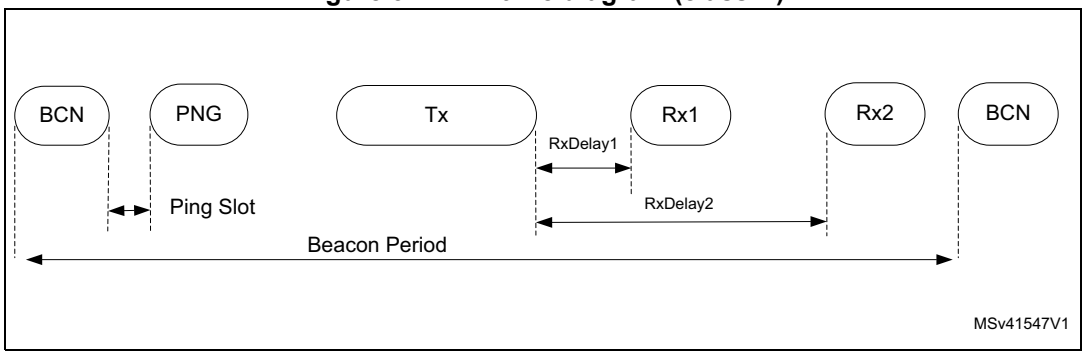
Figure 2. TX/Rx time diagram (class A)



Bi-directional end-devices with scheduled receive slots - class B - (beacon)

- Mid power consumption
- Class B devices open extra receive windows at scheduled times (see *Figure 2*)
- In order for the end-device to open the receive window at the scheduled time, it receives a time-synchronized beacon from the gateway.

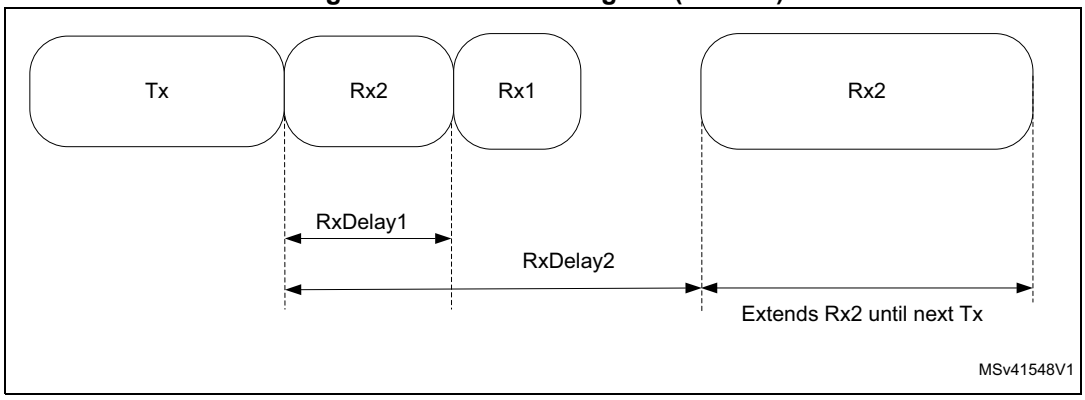
Figure 3. Tx/Rx time diagram (class B)



Bi-directional end-devices with maximal receive slots - class C - (continuous)

- Large power consumption
- End-devices of Class C have nearly continuously open receive windows, only closed when transmitting (see *Figure 4*).

Figure 4. Tx/Rx time diagram (class C)



2.2.3 End-device activation (joining)

Over-the-air activation (OTAA)

The OTAA (over-the-air activation) is a joining procedure for the LoRa™ end-device to participate in a LoRa™ network. Both the LoRa™ end-device and the application server share the same secret key known as the AppKey. During a joining procedure, the LoRa™ end-device and the application server exchange inputs to generate session keys. They generate both a network session key (NwkSKey) and an application session Key (AppSKey), their function is to encrypt MAC commands and application data respectively.

Activation by personalization (ABP)

In this case, the ABP are the NwkSkey and AppSkey, and they are already stored in the LoRa™ end-device that sends the data directly to the LoRa™ network.

2.2.4 Regional spectrum allocation

The LoRaWAN™ specification varies slightly from region to region. The European, North American and Asian markets have different spectrum allocations and regulatory requirements respectively. See *Table 3* for more details below:

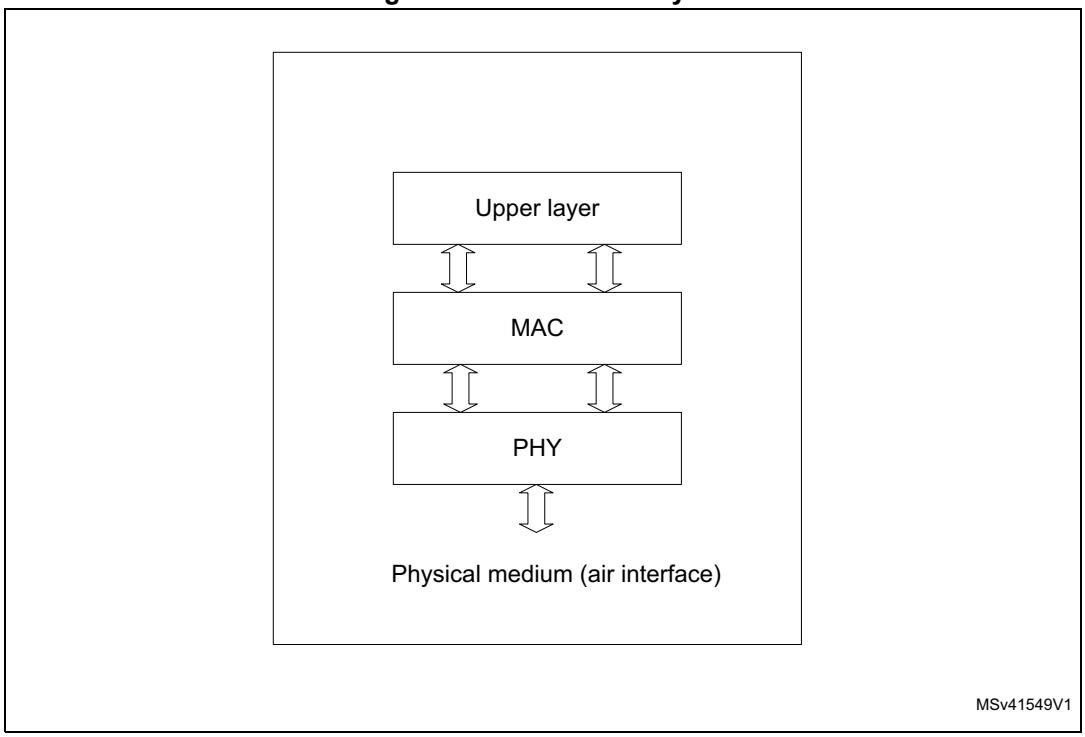
Table 3. LoraWAN™ regional spectrum allocation

Region	Supported	Band [MHz]	Duty cycle	Output power
EU	Y	868	<1%	+14dbm
EU	Y	433	<1%	+10dbm
US	Y	915	<2% (BW<250kHz) Or <4% (BW>=250kHz) Transmission slot < 0.4s	+20dbm
CN	N	779	<0.1%	+10dbm

2.3 Network layer

The LoRaWAN™ architecture is defined in terms of blocks, also called “layers”. Each layer is responsible for one part of the standard and it offers services to higher layers. The end-device is made up at least of one PHY, which embeds the radio frequency transceiver, and a MAC sublayer that provides access to the physical channel (see *Figure 5*).

Figure 5. LoRaWAN™ layers



2.3.1 Physical layer (PHY)

The physical layer provides two services: the PHY data service and the PHY management service.

The PHY data service enables the Tx/Rx of physical protocol data units (PPDUs) while the PHY management service enables the personal area network information base (PIB) management.

2.3.2 MAC sublayer

The MAC sublayer provides two services: the MAC data service and the MAC management service.

The MAC data service enables the transmission and reception of MAC protocol data units (MPDU) across the physical layer while the MAC sublayer management enables the PIB management.

2.4 Message flow

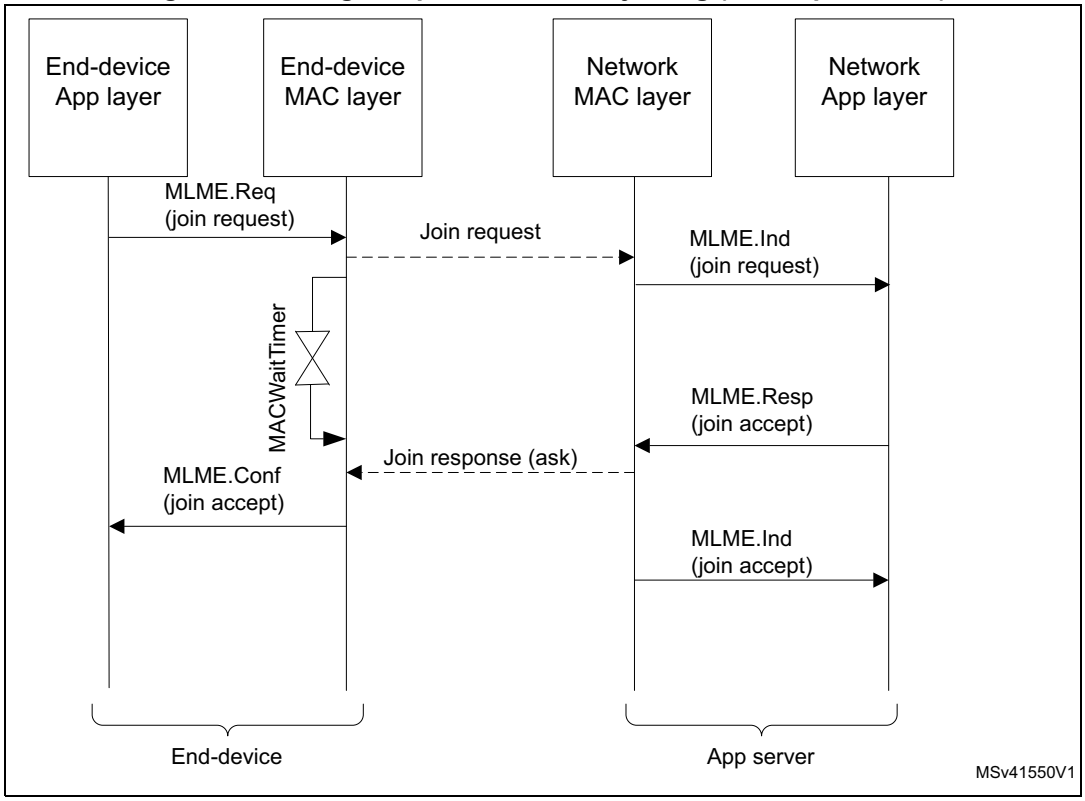
This chapter describes the information flow between the N-user and the N-layer. The request to a service is done through a service primitive.

2.4.1 End-device activation (joining)

Before the end-device can communicate on the LoRaWAN™ network, it has to be associated or activated following one of the two activation methods described in *Section 2.2.3: End-device activation (joining)*.

The following message sequence chart (MSC) in *Figure 6* shows the OTAA activation method.

Figure 6. Message sequence chart for joining (MLME primitives)



2.4.2 End-device data communication (class A)

When the end-device wants to transmit data, it can do it by following two different methods. Either it transmits through a confirmed-data message method (see *Figure 7*) or through an unconfirmed-data message (see *Figure 8*). In the first method, the end-device requires an "Ack" (acknowledgment) to be done by the receiver while in the second method, the "Ack" is not required.

When an end-device sends data with an "Ackreq" (acknowledgment request), it should wait while an acknowledgment duration (“AckWaitDuration") to receive the acknowledgment frame (refer to *Section 4.3.1: MCPS on page 22*).

If the acknowledgment frame is received, then the transmission is successful, else the transmission has failed.

Figure 7. Message sequence chart for confirmed-data (MCPS primitives)

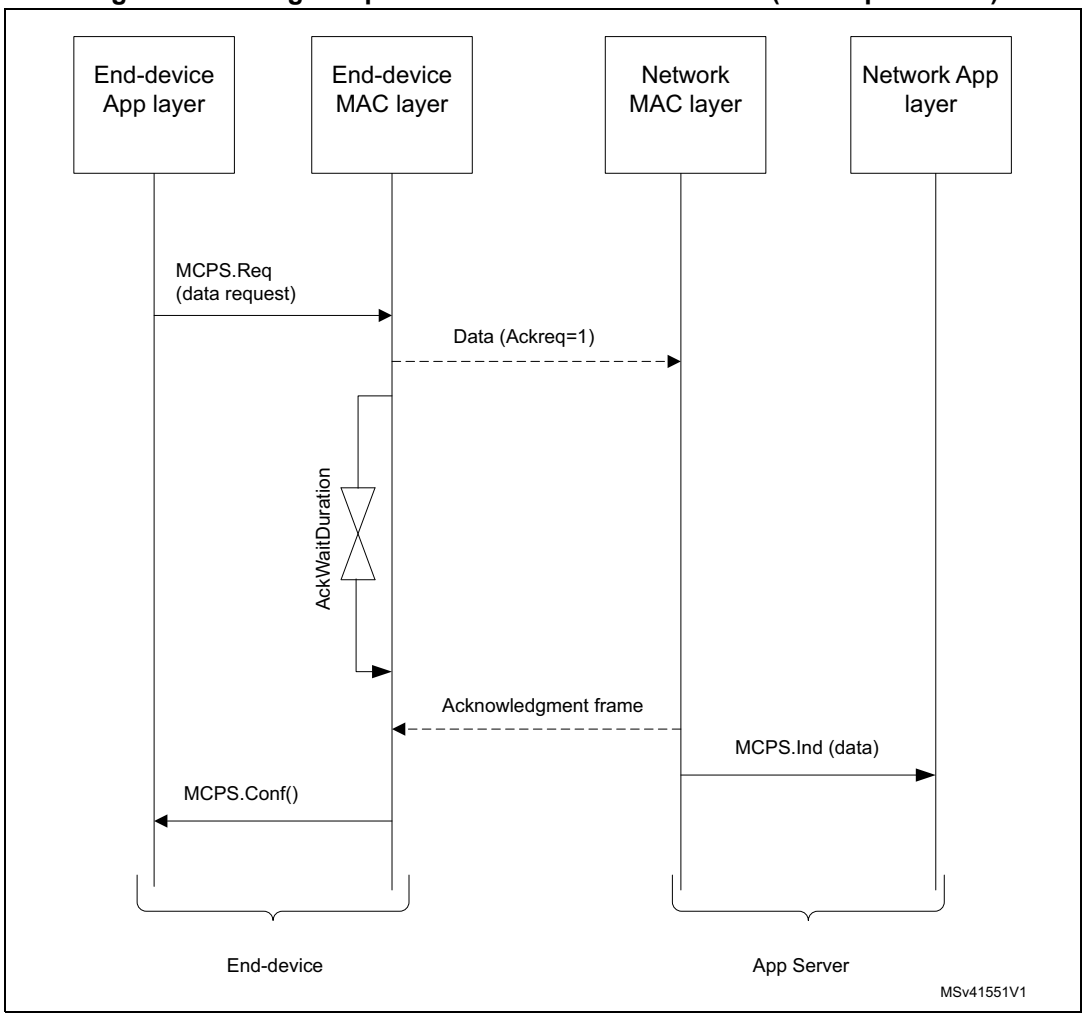
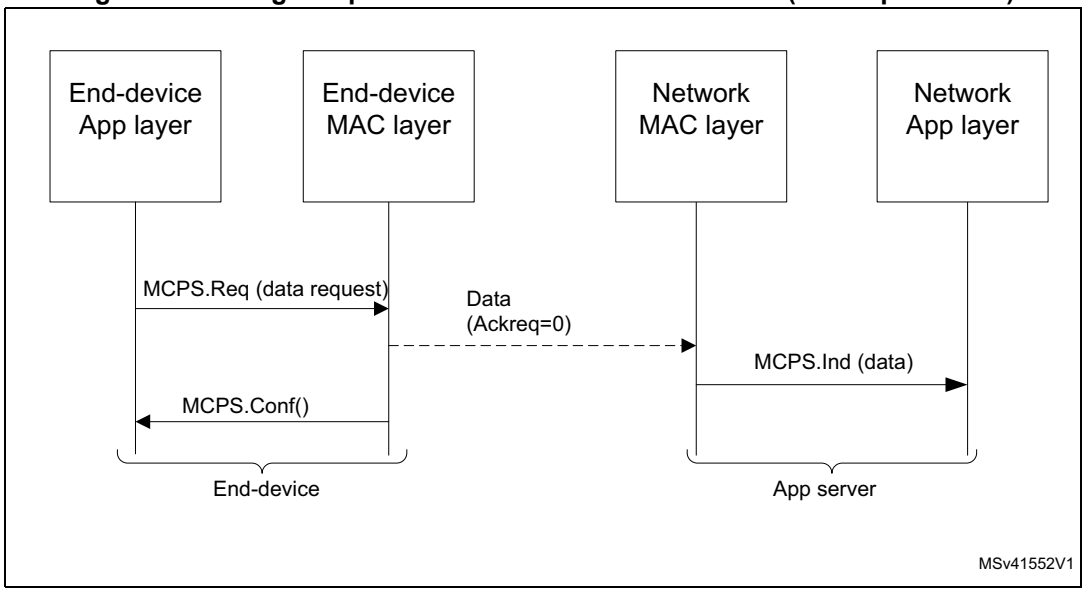


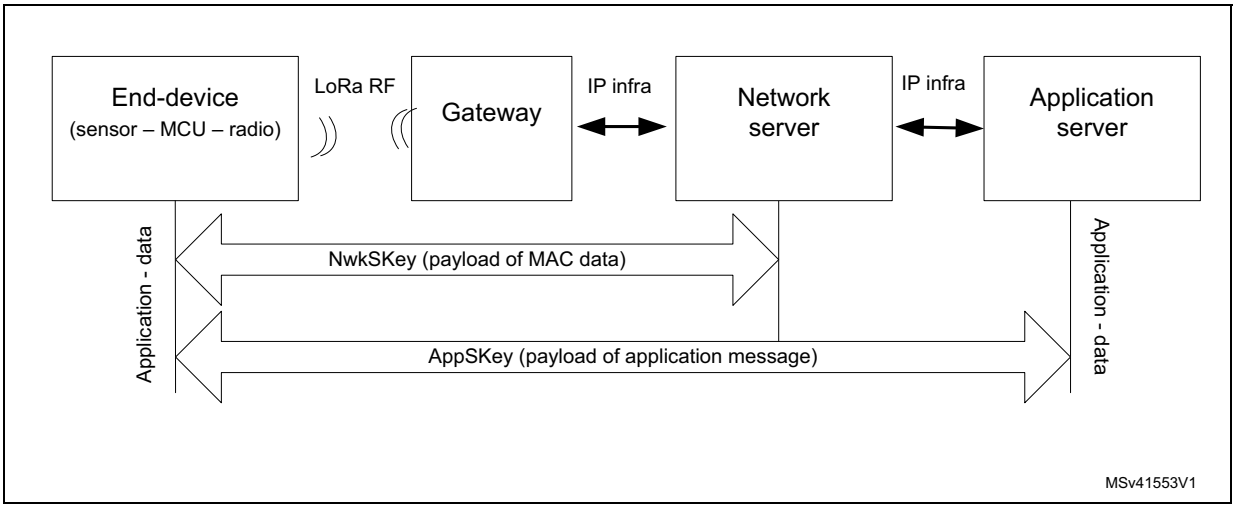
Figure 8. Message sequence chart for unconfirmed-data (MCPS primitives)



2.5 Data flow

The data integrity is ensured by the network session key (NwkSKey) and the application session key (AppSKey). The NwkSKey is used to encrypt and decrypt the MAC payload data and the AppSKey is used to encrypt and decrypt the application payload data.

Figure 9. Data flow



The NwkSKey is shared between the end-device and the network server. It provides message integrity for the communication and it provides security for the end-device towards the network server communication.

The AppSKey is shared between the end-device and the application server. It is used to encrypt/decrypt the application data. In other words, it provides security for the application's payload. In this way, the application data sent by an end-device can not be interpreted by the network server.

3 LRWAN middleware description

3.1 Overview

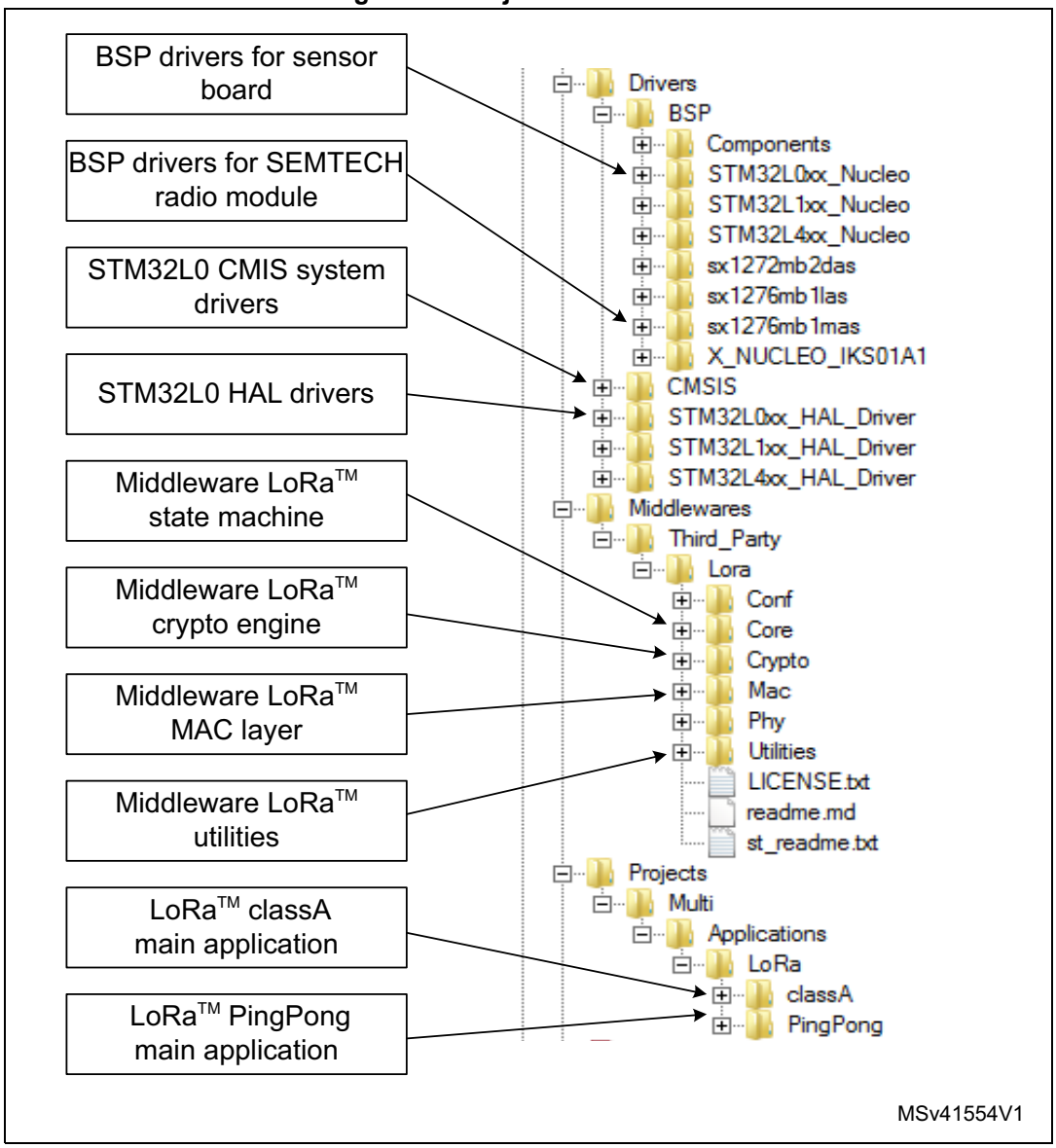
This package offers a LoRa™ stack middleware for STM32 microcontrollers. This middleware is split into several modules:

- LoRaMac layer module
- LoRa™ utilities module
- LoRa™ crypto module
- LoRa™ core module

The LoRa™ core module implements a LoRa™ state machine coming on top of the LoRaMac layer. The LoRa™ stack module interfaces with the BSP SEMTECH radio driver module.

This middleware is provided in source-code format and is compliant which the STM32Cube Hal driver.

Figure 10. Project files structure



The I-CUBE-LRWAN package is composed by:

- LoRa™ stack middleware
  - LoRaWAN™ layer
  - LoRa™ utilities such as timer server, power management and delay management
  - LoRa™ software crypto engine
  - LoRa™ state machine
- Board support package
  - Radio SEMTECH drivers
  - Sensor STM drivers
- STM32L0 HAL drivers
- LoRa™ main application example

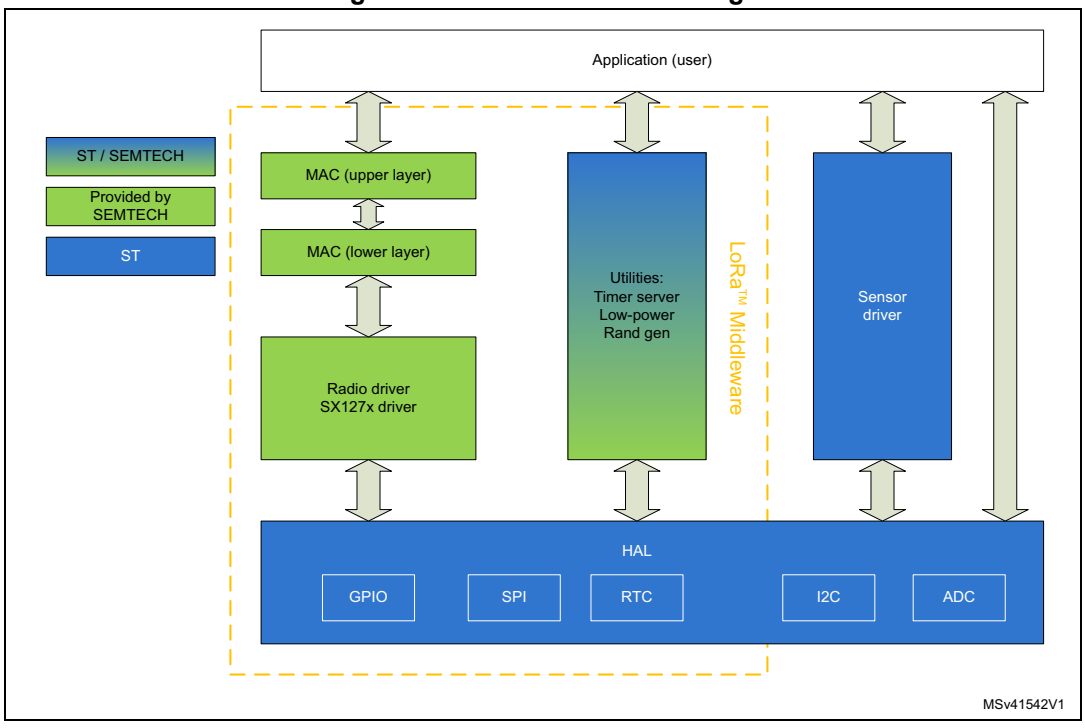
3.2 Features

- Compliant with the specification for the LoRa™ Alliance protocol named LoRaWAN™ (version V1.0 January 2015)
- On-board LoRaWAN™ class A and class C protocol stack
- EU 868MHz ISM band ETSI compliant
- EU 433MHz ISM band ETSI compliant
- US 915MHz ISM band FCC compliant
- End-device activation either through over-the-air activation (OTAA) or through activation-by-personalization (ABP)
- Adaptive data rate support
- LoRaWAN™ test application for certification tests included
- Low-power optimized

3.3 Architecture

The picture below depicts the firmware's main design for the I-CUBE-LRWAN application.

Figure 11. Firmware's main design



The HAL uses cube API to drive the MCU hardware required by the application. Only specific hardware is included in the LoRa™ middleware as it is mandatory to run a LoRa™ application.

The RTC provides a centralized time unit which continues to run even in low-power mode (stop mode). The RTC alarm is used to wake up the system at specific timings managed by the timer server.

The radio driver uses the SPI and the GPIO HW (see *Figure 11*) to control the radio, it also provides a set of API to be used by higher level software. The LoRa™ radio is provided by SEMTECH, though the API have been slightly modified to interface with Cube HAL.

The radio driver is split in two parts:

- The sx1276.c (or sx1272.c) contains all functions which are radio dependent only.
- The sx1276mb1mas.c, sx1276mb1las and sx1272mb2das contain all the radio board dependent functions.

The MAC controls the PHY using 802.15.4 model. It interfaces with the PHY driver and uses the timerServer to add or remove timed tasks, and to take care of the 'Tx time on air'. This action ensures that the duty-cycle limitation mandated by the ETSI is respected. It also carries out AES encryption/decryption algorithm to cypher the MAC header and the payload.

Since the state machine that controls the LoRa™ class A is sensitive, an intermediate level of software has been inserted (lora.c) between the MAC and the application (refer to MAC's "upper layer" on *Figure 11*) to ease the user life. Since the set of API is limited as of now, the user is free to implement the class A state machine at application level.

The application is built around an infinite loop. It manages the low-power, runs the interrupt handlers (alarm or GPIO), and calls the LoRa™ class A if any task has to be done. This application also implements the sensor read access.

3.4 Hardware related components

3.4.1 Radio reset

One GPIO from the MCU is used to reset the radio. It is done once at the initialization of the hardware. Refer to *Table 24: LoRa™ radio expansion boards characteristics on page 30* and to *Section 5.1.4: Hardware mapping on page 31*.

3.4.2 SPI

The sx127x radio registers are accessed through the SPI bus at 1Mbps. Refer to *Table 24: LoRa™ radio expansion boards characteristics on page 30* and to *Section 5.1.4: Hardware mapping on page 31*.

3.4.3 RTC

The RTC calendar is used as a timer engine running in all power modes from the 32kHz external oscillator. By default, it is programed to provide 1024 ticks (sub-seconds) per second. It is programed once at initialization of the hardware when the MCU starts for the first time. Its output is limited to a 32 bits timer, that is, around a 48 day period.

In the case that the user needs to change the tick duration, note that the tick duration should remain below 1ms.

3.4.4 Interrupt lines

Four GPIO lines are dedicated to receive the interrupts from the radio (refer to *Table 24: LoRa™ radio expansion boards characteristics on page 30* and to *Section 5.1.4: Hardware mapping on page 31*). The DIO0 is used to signal that the LoRa™ radio has successfully completed a requested task (TxDone or RxDone). The DIO1 is used to signal that the radio has failed to complete a requested task (RxTimeout).

In FSK mode it is a FIFO-level interrupt signaling that the FIFO-level has reached a predefined threshold and needs to be flushed.

The DIO2 is used in FSK mode. It signals that the radio has successfully detected a preamble and the DIO3 is reserved for future use.

Note: The FSK mode in LoRaWAN™ has the fastest data rate at 50kbps.

4 LRWAN middleware programming guidelines

This section gives a description of the LoRaMac layer APIs. The PHY layer being proprietary (see *Section 2.1: Overview on page 8*), it is out of the scope of this user manual. The PHY has to be viewed as a black box.

4.1 Middleware initialization

The initialization of the LoRaMac layer is done through the "LoraMacinitialization" function. This function does the preamble run time initialization of the LoRaMac layer and initializes the callback primitives of the MCPS and MLME services.

Table 4. Middleware initialization function

Function	Description
LoRaMacStatus_t LoRaMacInitialization (LoRAMacPrimitives_t *primitives, LoRaMacCallback_t *callback);	Do initialization of the LoRaMac layer module (see *Section 4.3: Middleware MAC layer callbacks*)

4.2 Middleware MAC layer functions

The provided APIs follow the definition of “primitive” defined in IEEE802.15.4-2011 (see*Section 1.2: References on page 7*).

The interfacing with the LoRaMac is made through the request-confirm and the indication-response architecture. The application layer can perform a request, which the LoRaMAC layer confirms with a confirm primitive. Conversely, the LoRaMAC layer notifies an application layer with the indication primitive in case of any event.

The application layer may respond to an indication with the response primitive. Therefore all the confirm/indication are implemented using callbacks.

The LoRaMAC layer provides MCPS services, MLME services and MIB services.

4.2.1 MCPS services

In general, the LoRaMAC layer use the MCPS services for data transmissions and data receptions.

Table 5. MCPS services function

Function	Description
LoRaMacStatus_t LoRaMacMcpsRequest (McpsReq_t *mcpsRequest);	Requests to send Tx data

4.2.2 MLME services

The LoRaMAC layer uses the MLME services to manage the LoRaWAN network.

Table 6. MLME services function

Function	Description
LoRaMacStatus_t LoRaMacMlmeRequest (MlmeReq_t *mlmeRequest);	Used to generate a join request or request for a link check

4.2.3 MIB services

The MIB stores important runtime information (such as MIB_NETWORK_JOINED, MIB_NET_ID) and holds the configuration of the LoRaMAC layer (for example the MIB_ADR, MIB_APP_KEY). The following APIs are provided:

Table 7. MIB services functions

Function	Description
LoRaMacStatus_t LoRaMacMibSetRequestConfirm(MibRequestConfirm_t *mibSet);	To set attributes of the LoRaMac layer
LoRaMacStatus_t LoRaMacMibGetRequestConfirm(MibRequestConfirm_t *mibGet);	To get attributes of the LoRaMac layer

4.3 Middleware MAC layer callbacks

Refer to *Section 4.1: Middleware initialization* for the description of the LoRaMac user event functions primitives and the callback functions.

4.3.1 MCPS

In general, the LoRaMAC layer uses the MCPS services for data transmission and data reception.

Table 8. MCPS primitives

Function	Description
void (*MacMcpsConfirm) (McpsConfirm_t *McpsConfirm) ;)	Event function primitive for the called callback to be implemented by the application. Response to a McpsRequest
Void (*MacMcpsIndication) (McpsIndication_t *McpsIndication);	Event function primitive for the called callback to be implemented by the application. Notifies application that a received packet is available

4.3.2 MLME

The LoRaMAC layer uses the MLME services to manage the LoRaWAN network.

Table 9. MLME primitive

Function	Description
void (*MacMlmeConfirm)(MlmeConfirm_t *MlmeConfirm);	Event function primitive so called callback to be implemented by the application

4.3.3 MIB

N/A

4.3.4 Battery level

The LoRaMAC layer needs a battery-level measuring service.

Table 10. Battery level function

Function	Description
uint8_t HW_GetBatteryLevel(void)	Get the measured battery level

4.4 Middleware utilities functions

4.4.1 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 low-power modes.

The timer server provides a reliable clock for the user and the LoRa™ stack. The user can request as many timers as the application requires.

Four APIs are provided:

Table 11. Timer server functions

Function	Description
void TimerInit(TimerEvent_t *obj, void (*callback)(void));	Initialize the timer and associate a callback function when timer elapses
void TimerSetValue(TimerEvent_t *obj, uint32_t value);	Set the timer a timeout value on milliseconds
void TimerStart(TimerEvent_t *obj);	Start the timer
void TimerStop(TimerEvent_t *obj);	Stop the timer

The timer server is located in Middlewares\Third_Party\Lora\Utilities.

4.5 Middleware utilities callbacks

4.5.1 Delay Rx window

Refer to *Section 2.2.2: End-device classes*.

Table 12. Delay Rx functions

Function	Description
void OnRxWindow1TimerEvent(void);	Set the RxDelay1 (ReceiveDelayX - RADIO_WAKEUP_TIME)
void OnRxWindow2TimerEvent(void);	Set the RxDelay2

4.5.2 Delay for MAC layer state checking

Table 13. Delay for MAC layer state checking function

Function	Description
void OnMacStateCheckTimerEvent(void);	Check the state of the MAC every "MAC_STATE_CHECK_TIMEOUT"

4.5.3 Delay for Tx frame transmission

Table 14. Delay for Tx frame transmission function

Function	Description
void OnTxDelayedTimerEvent(void);	Set timer for Tx frame transmission
void OnTxNextPacketTimerEvent(void);	Set timeout for next packet Tx scheduling

4.5.4 Delay for Rx frame

Table 15. delay for Rx frame function

Function	Description
void OnAckTimeoutTimerEvent(void);	Set timeout for received frame acknowledgment

4.6 Middleware low-power functions

The following APIs allows to manage the low-power mode of the core MCU.

Table 16. Middleware low-power functions

Function	Description
void LowPower_Disable(e_LOW_POWER_State_Id_t state);	Disable the low-power mode following the state parameter
void LowPower_Enable(e_LOW_POWER_State_Id_t state);	Enable the low-power mode following the state parameter
uint32_t LowPower_GetState(void);	Get the current low-power state
void LowPower_Handler(void);	Cortex M deep-sleep management

4.7 Middleware class A application function

The interface to the MAC is done through the MAC interface file "LoRaMac.h".

Standard mode

In standard mode, an interface file (see MAC upper layer in *Figure 11 on page 18*) is provided to let the user start without worrying about the LoRa™ state machine. It is located in Middlewares\Third_Party\Lora\Core\lora.c.

It implements an example of finite state machine managing the LoRa™ MAC. It also implements the LoRa™ certification test cases that are not visible to the application layer.

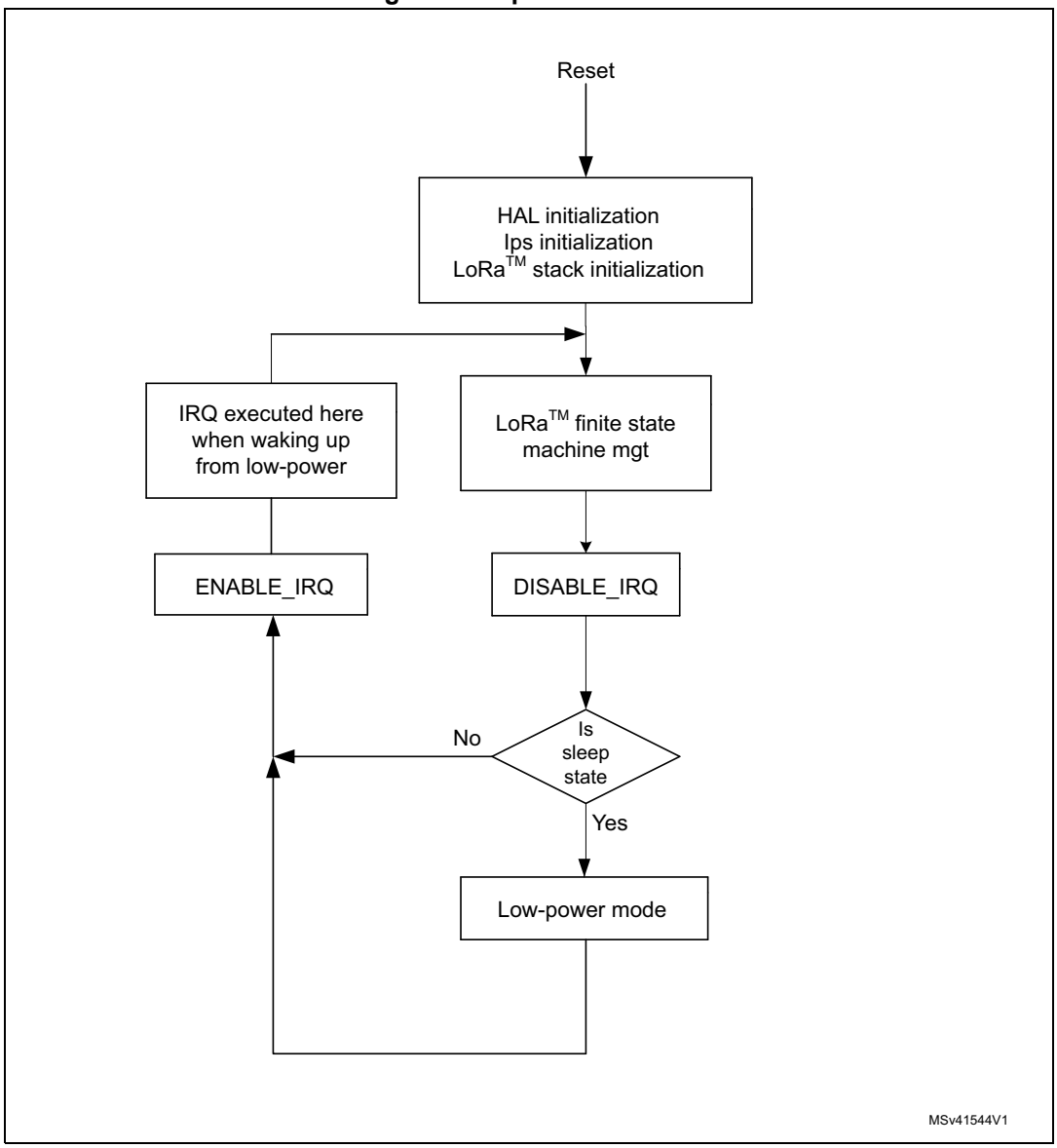
Advanced mode

In this mode, the user can access directly the MAC layer by including the MAC in the user file.

Model of operation

The model of operation proposed for this LoRa™ class A is based on finite state machine (FSM) running in the main thread of the application. LoRa™ FSM transition is triggered either by a timer event or by a radio event.

Figure 12. Operation model



LoRa™ state machine

The following diagram describes the state machine of LoRa™.

On reset after system initialization done, LoRa™ FSM goes into “DEVICE_STATE_INIT”.

It does a join network request when using the “over_the_air_activation” method and goes into “DEVICE_STATE_SLEEP”.

When using the “activation by personalization”, the network is already joined and therefore jumps directly to state send.

From “DEVICE_STATE_SLEEP”, if the end-device has joined the network when a “TimerEvent” occurs, the LoRa™ FSM goes into a temporary DEVICE_STATE_JOINED before going into DEVICE_STATE_SEND.

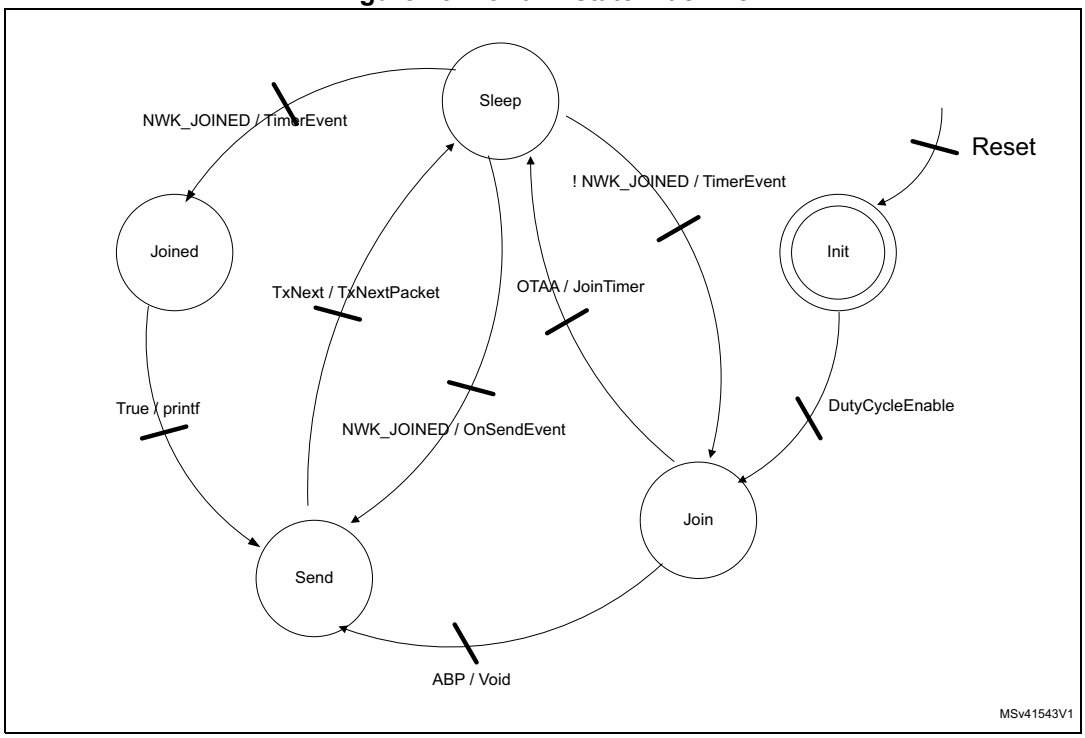
From “DEVICE_STATE_SLEEP”, if the end-device has joined the network when an “OnSendEvent” occurs, the LoRa™ FSM goes into DEVICE_STATE_SEND.

From “DEVICE_STATE_SLEEP”, if the end-device has not joined the network when a “TimerEvent” or “OnSendEvent” occurs, the LoRa™ FSM goes into DEVICE_STATE_JOIN. In this case it will request a new join network request.

From “DEVICE_STATE_SEND”, it prepares and sends Tx packets (if any). Then it goes into “DEVICE_STATE_SLEEP”.

On “OnSendEvent” it goes back to the “DEVICE_STATE_SEND” to send next scheduled packet.

Figure 13. LoRa™ state machine



4.7.1 LoRa™ class A initialization

Table 17. LoRa™ class A initialization function

Function	Description
void lora_Init (LoRaMainCallback_t *callbacks, LoRaParam_t* LoRaParamInit);	Initialization of the LoRa™ class A finite state machine

4.7.2 LoRa™ class A FSM entry point

Table 18. LoRa™ class A FSM entry point function

Function	Description
void lora_fsm(void);	LoRa™ class A FSM entry point

4.8 LIB class A application callbacks

4.8.1 Current battery level

Table 19. Current battery level function

Function	Description
uint8_t HW_GetBatteryLevel (void);	Get the battery level

4.8.2 Board unique ID

Table 20. Board unique ID function

Function	Description
void HW_GetUniqueId (uint8_t *id) ;	Get a unique Identifier

4.8.3 Board random seed

Table 21. Board random seed function

Function	Description
uint32_t HW_GetRandomSeed (void);	Get a random seed value

4.8.4 Make Tx frame

Table 22. make Tx frame function

Function	Description
void LoraTxData(lora_AppData_t *AppData, FunctionalState* IsTxConfirmed);	To prepare and format a CONFIRMED or UNCONFIRMED frame application to be sent. The user is free to implement its own code here

4.8.5 Make Rx frame

Table 23. Make Rx frame

Function	Description
void LoraRxData (lora_AppData_t *AppData) ;	To prepare and format a CONFIRMED or UNCONFIRMED frame application to be sent. The user is free to implement its own code here

5 Getting started

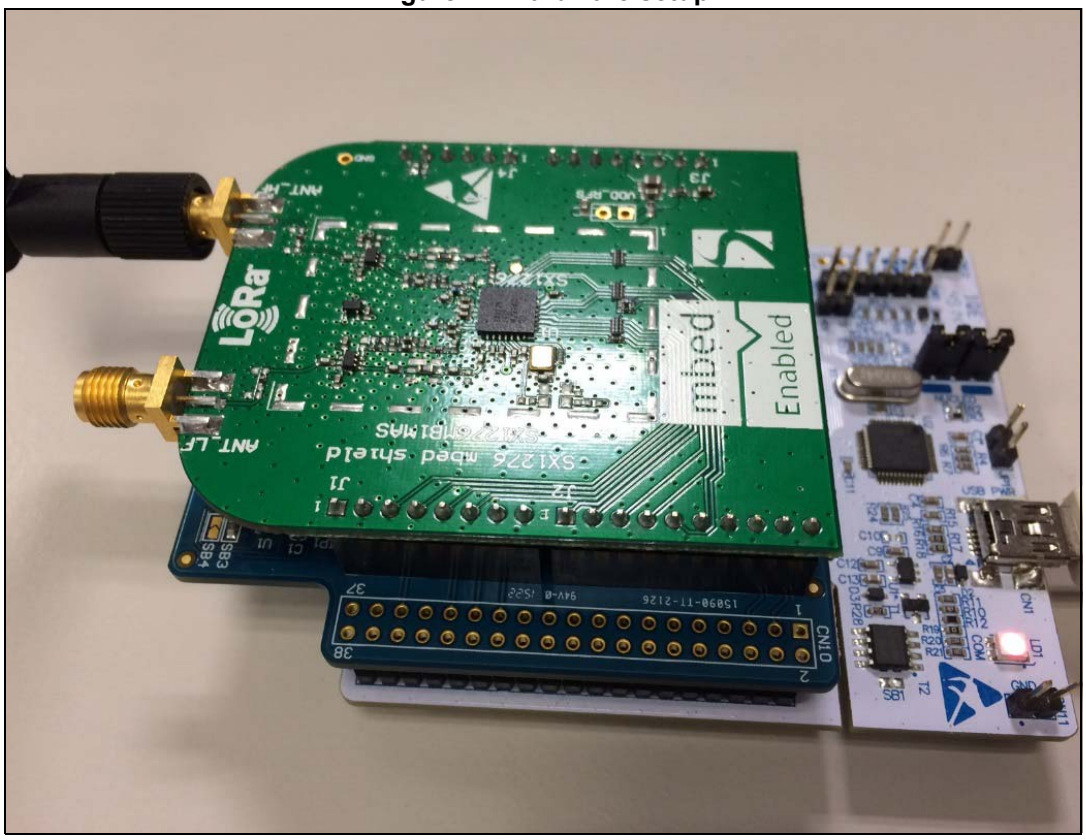
5.1 Hardware description

This application is based on an STM32 NUCLEO platform. It is stacked with an ST X-NUCLEO sensor expansion board and a LoRa™ radio expansion board.

This application reads the temperature, humidity and atmospheric pressure from the sensors through i2c. The MCU measures the supplied voltage to calculate the battery level. These four data are sent periodically to the LoRa™ network using the LoRa™ radio in class A at 868MHz.

The set-up is shown below:

Figure 14. Hardware setup



Only the top left SMA is used to connect the 868MHz antenna.

5.1.1 STM32 Nucleo boards

This package supports STM32L053R8, STM32L152RE and STM32L476RG. The application running on the MCU is Cube compliant. It is described in *Section 4*.

In case of a battery operated device on NUCLEO, the following must be done:

- On JP5, select E5V to connect a ~5 volt supply, though 4.5 V is enough
- Plug the supply on E5V
- For real low-power use case, the LED2 may be disconnected as the current flowing through it is of around 3mA.

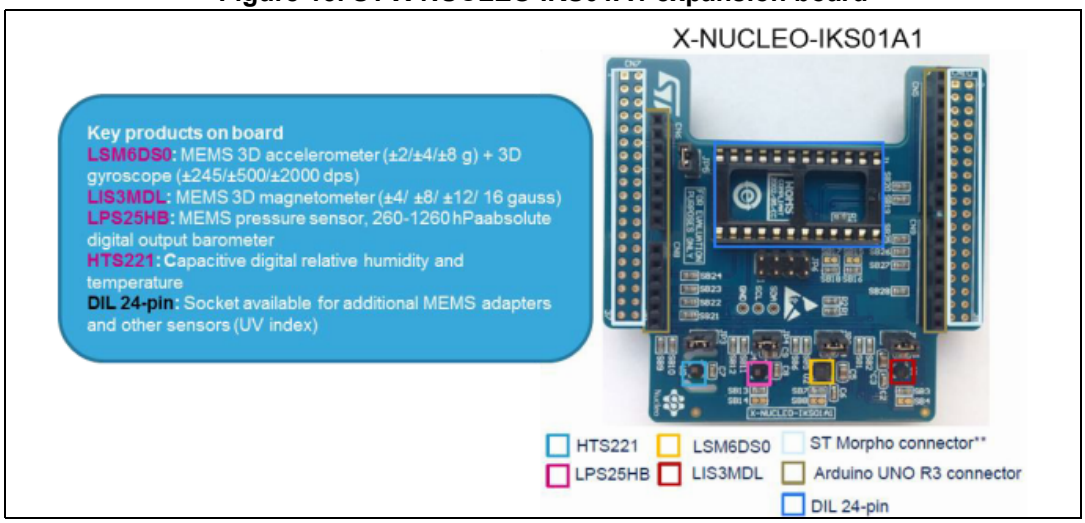
5.1.2 ST X-NUCLEO IKS01A1 expansion board

It hosts four sensors:

- The humidity and temperature sensor (hts221),
- The pressure sensor (lps25H)
- The 3D accelerometer (LSM6DS0)
- A3D magnetometer (LIS3MDL)

The expansion board is depicted below:

Figure 15. ST X-NUCLEO IKS01A1 expansion board



Only hts221 and lps25H are used in this application. The sensors are accessed by I2C interface and the unused interrupt (for now) are routed to the MCU.

5.1.3 LoRa™ radio expansion board

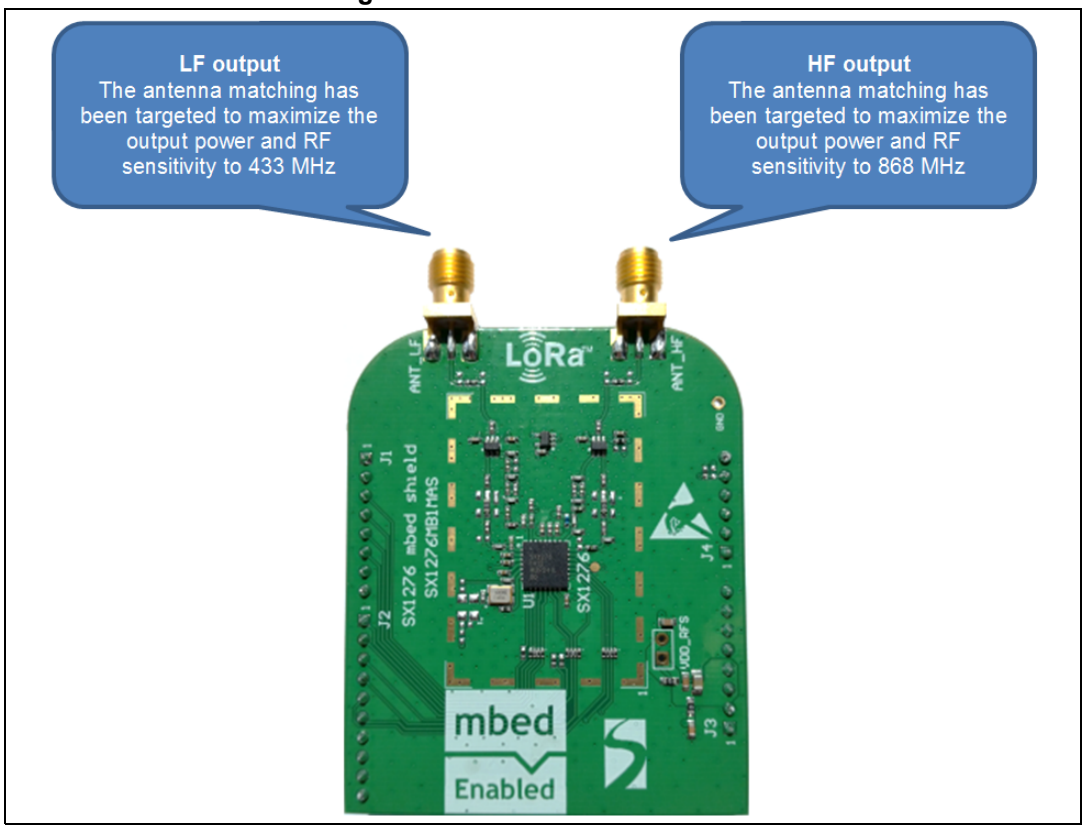
The I-CUBE-LRWAN package supports three LoRa™ radio expansion boards: the sx1276mb1mas, the sx1276mb1las and the sx1272mb2das.

The main characteristic are described in the table below:

Table 24. LoRa™ radio expansion boards characteristics

Board	Characteristics
sx1276mb1mas	868MHz (HF) at 14dBm and 433(LF) at 14dBm
sx1276mb1las	915MHz (HF) at 20dBm and 433(LF) at 14dBm
sx1272mb2das	915MHz and 868MHz at 20dBm

Figure 16. SX1276MB1LAS board



The radio interface is described below:

- Registers are accessed through SPI.
- Three interrupt lines DIO0, 1, 2 are mandatory. DIO3, 4 and 5 are not essential to run a LoRa™ link. Only DIO0, 1, 2 are used in this application.
- There is an optional antenna switch to switch between Rx and Tx mode. Although the sx1276 can control the RF switch directly, the RF switch is controlled by the MCU (soldered this way on the SX1276MB1MAS by default).

5.1.4 Hardware mapping

The hardware mapping is shown below:

Table 25. Hardware mappings: used I/Os

	PA	PB	PC
0	RADIO_RESET	RADIO_DIO_4b	LIS3MDL_DRDY removed
1	-	-	ANT_SWITCH (LIS3MDL_INT1 removed)
2	-		-
3	-	RADIO_DIO_1	-
4	-	LPS25H (*RADIO_DIO_3 removed*)	-

Table 25. Hardware mappings: used I/Os (continued)

	PA	PB	PC
5	RADIO_SCLK	RADIO_DIO_2	-
6	RADIO_MISO	RADIO_NSS	-
7	RADIO_MOSI	-	RADIO_DIO_5
8	-	I2C_SCL	-
9	RADIO_DIO_4a	I2C_SDA	-
10	RADIO_DIO_0	HTS221 INT	-
11	-	-	-
12	-	DBG	-
13	SWDIO	DBG	-
14	SWCLK	DBG	-
15	-	DBG	-

Table 26 shows the interrupt priorities level applicable for the Cortex system processor exception and the STM32L0x LoRa™ application-specific interrupt.

Table 26. STM32L0xx IRQ priorities

	Preempt priority	Sub priority	DIO#0,2,3,4	DIO#1
Systick	0	N/A	-	-
RTC	0	N/A	-	-
EXTI2_3	0	N/A	-	assigned
EXTI4_15	0	N/A	assigned	-

5.1.5 Hardware modifications

In order to fit both expansion boards, some modification are needed on the X-NUCLEO and on the sx1276mb1mas boards:

X-NUCLEO modifications:

- Remove i2c from accel and mag sb1, sb2, sb5, sb6
- Remove interrupt from accel and mag sb21, sb22, sb27
- Remove supply to accel and mag by removing jumper JP1 and JP2

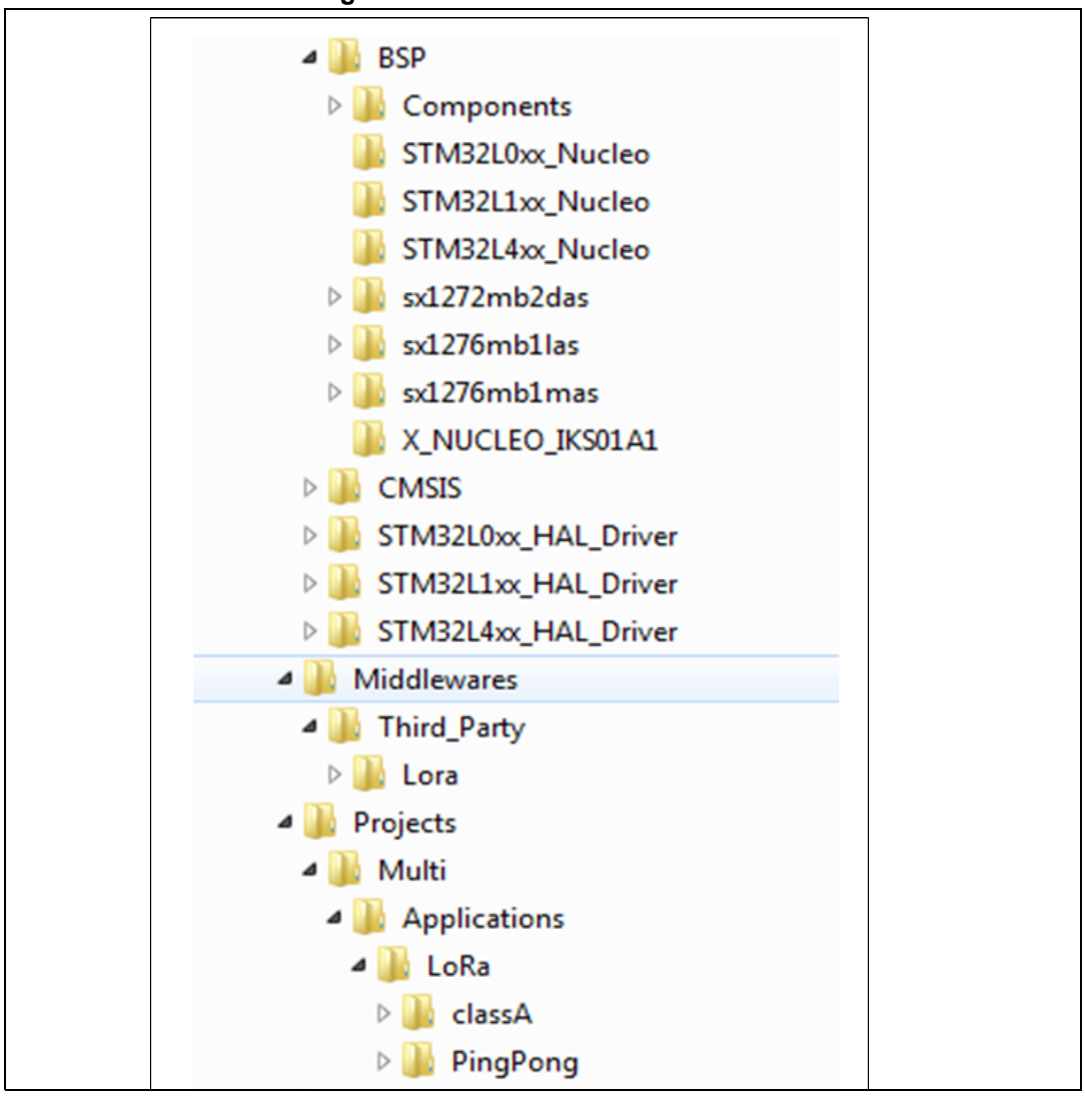
sx1276mb1mas modifications:

- Cut DIO5, DIO4a, DIO4b, DIO3.
- Optionally remove leaky resistor on switch control.

5.2 Software description

When the user unzips the I-CUBE-LRWAN, the package presents the following structure:

Figure 17. I-CUBE-LRWAN structure



The I-CUBE-LRWAN package contains two applications: Class A and PingPong project.

In order to launch the LoRa™ Class A project, the user should go to the dedicated IDE folder (MDK-ARM or IAR IDE environment) and activate either the “Lora.uvprojx” file to start the MDK-ARM IDE debugger or the “Lora.eww” file to start the IAR IDE debugger.

In order to launch the PingPong project, the user must go to the PingPong folder and follow the same procedure as for LoRa™ Class A project to launch the associated debugger. The description of the PingPong application is presented on *Section 5.3*.

5.2.1 Compilation switches

Activation methods and keys

There are two ways to activate a device on the network, either by OTAA or by ABP.

The file projects\Multi\Applications\LoRa\classA\comissioning.h gathers all the data related to the device activation.

The chosen method, along with the commissioning data, are printed on the virtual port running at 921600bauds and is visible on a terminal.

Debug switch

In \Projects\Multi\Applications\LoRa\classA\inc\hw_conf.h, one can enable the debug mode or/and the trace mode by commenting out #define DEBUG/ #define TRACE.

The debug mode enables the DBG_GPIO_SET and DBG_GPIO_RST macros as well as the debugger mode even when the MCU goes in low-power.

The trace mode enables the DBG_PRINTF macro.

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

Sensor switch

In \Projects\Multi\Applications\LoRa\classA\inc\hw_conf.h, when no sensor expansion board is plugged on the set-up, #define SENSOR_ENBALED shall be commented out.

Table 27 provides a summary of the main options for the application's configuration.

Table 27. Switch options for the application's configuration

	Switch option	Definition	Where
LoRa™ stack	OVER_THE_AIR_ACTIVATION	Application uses over-the-air activation procedure	Commissioning
	STATIC_DEVICE_EUI	Static or dynamic end-device identification	Commissioning
	STATIC_DEVICE_ADDRESS	Static or dynamic end-device address	Commissioning
	USE_BAND_868	Enable the EU band selection	Compiler option setting
	USE_BAND_433	Enable the EU band selection	Compiler option setting
	USE_BAND_915	Enable the US band selection	Compiler option setting
Debug	DEBUG	Enable “Led on/off”	hw_conf.h
	TRACE	Enable “printf”.	hw_conf.h
Sensor	SENSOR_ENABLED	Enable the call to the sensor board	hw_conf_.h

5.3 PingPong application description

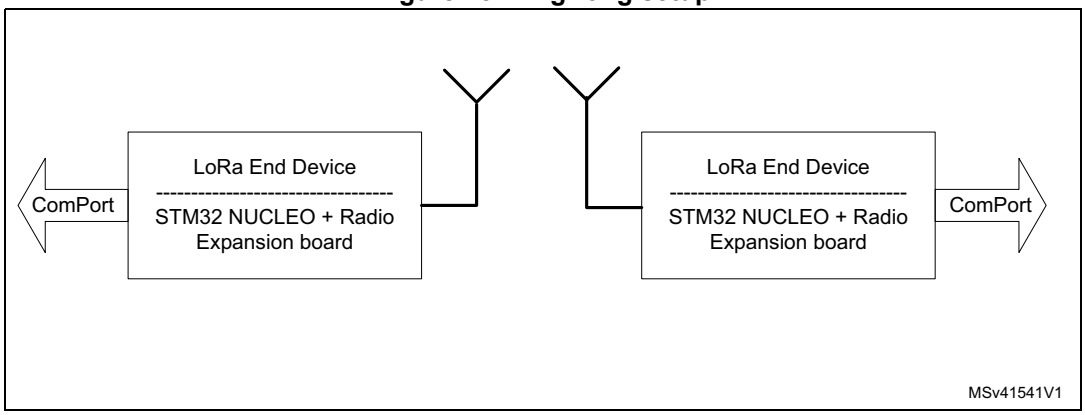
This application is a simple Rx/Tx RF link between two LoRa™ end-devices. By default, each LoRa™ end-device starts as a master and will transmit a "Ping" message, and then wait for an answer. The first LoRa™ end-device receiving a "Ping" message becomes a slave and answers the master with a "Pong" message. The PingPong is then started.

Hardware and Software set-up environment

STM32LXxx-Nucleo Set-up:

- Connect the NUCLEO board to the PC with a USB cable type A to mini-B to ST-LINK connector (CN1). Ensure that the ST-LINK connector CN2 jumpers are fitted.

Figure 18. PingPong setup



The STM32L0-NUCLEO, STM32L1-NUCLEO and STM32L4-NUCLEO boards can be easily tailored to any other supported device and development board.

6 System performances

6.1 Memory footprints

The values in *Table 28* have been measured for the following configuration:

Compiler: Keil (ARM Compiler 5.05)

- Optimization: optimized for size level 3
- Debug option: off
- Trace option: off
- Expansion board: / SX1276MB1MAS

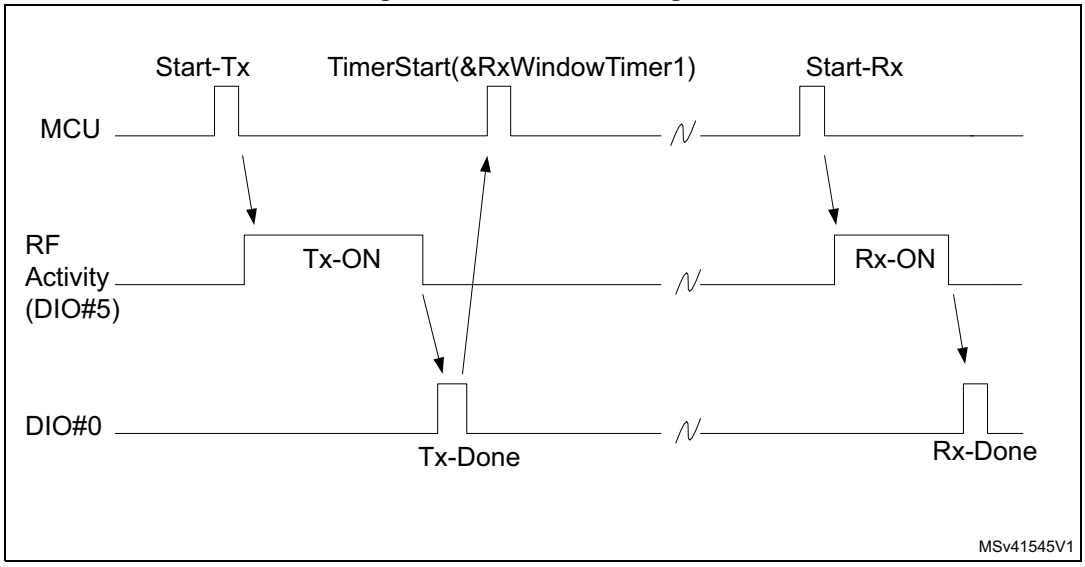
Table 28. Memory footprint measured values

	Flash (bytes)	RAM (bytes)	Description
Application (user) +LoRaClassA +LoRa stack	35596	4256	Memory footprint for the overall Application (App_user + LoRa ClassA + LoRa stack)

6.2 Real-time constraints

LoRa™ RF's asynchronous protocol implies to follow a strict TX/Rx timing recommendation. The SX1276mb1mas expansion board has been optimized for user-transparent low-lock time and fast auto-calibrating operation. The LoRa™ software expansion design integrates the transmitter's startup time and the receiver's startup time constraints.

Figure 19. Rx/Tx time diagram



Rx window channel start

The Rx window opens RECEIVE_DELAY1 (1sec) or JOIN_ACCEPT_DELAY1 (5sec) seconds (+/- 20 microseconds) after the end of the uplink modulation.

The current scheduling interrupt-level priority has to be respected. In other words, all new user interrupts must have interrupt priority > DI0#n interrupt (see *Section Table 26.: STM32L0xx IRQ priorities on page 32*) in order to avoid stalling the received startup time.

6.3 Power consumption

The power consumption measurement has been established for Nucleo board series associated to the SX1276MB1MAS shield.

Measurements setup:

- No DEBUG
- No TRACE
- No SENSOR_ENABLED

Measurements results:

- Typical consumption in stop mode: 4.3 uA
- Typical consumption in run mode: 8.0 mA

Measurements figures:

- Instantaneous consumption over 30 seconds

Figure 20. STM32L0 current consumption against time

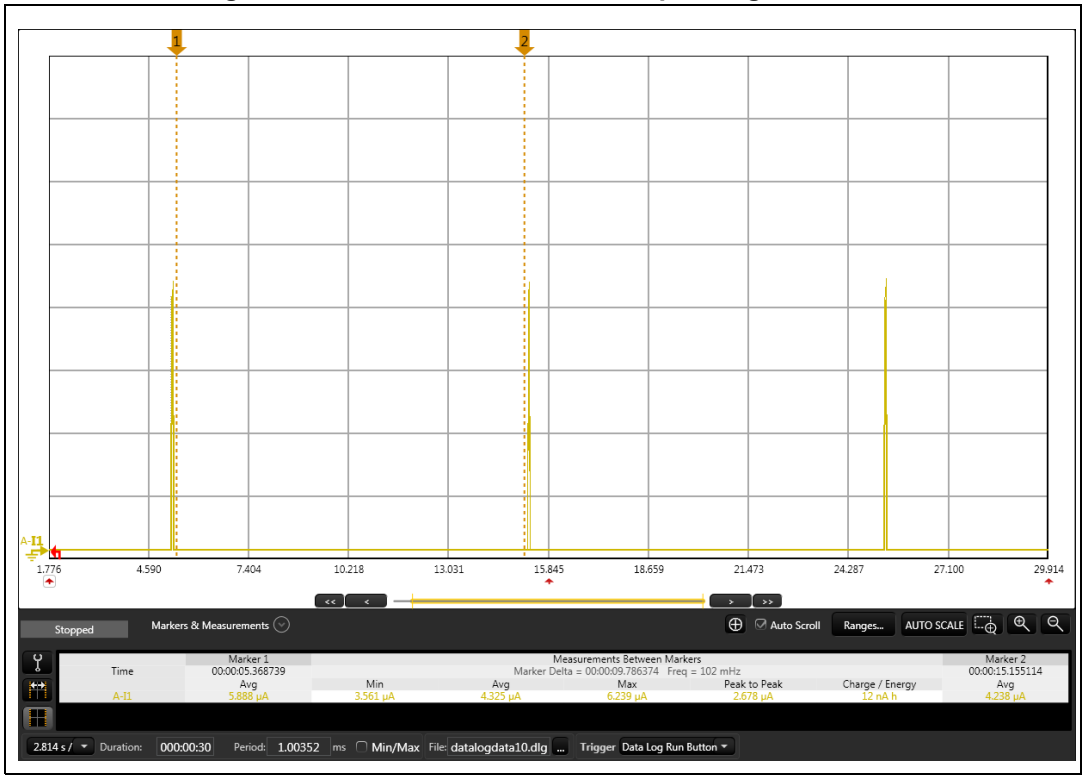
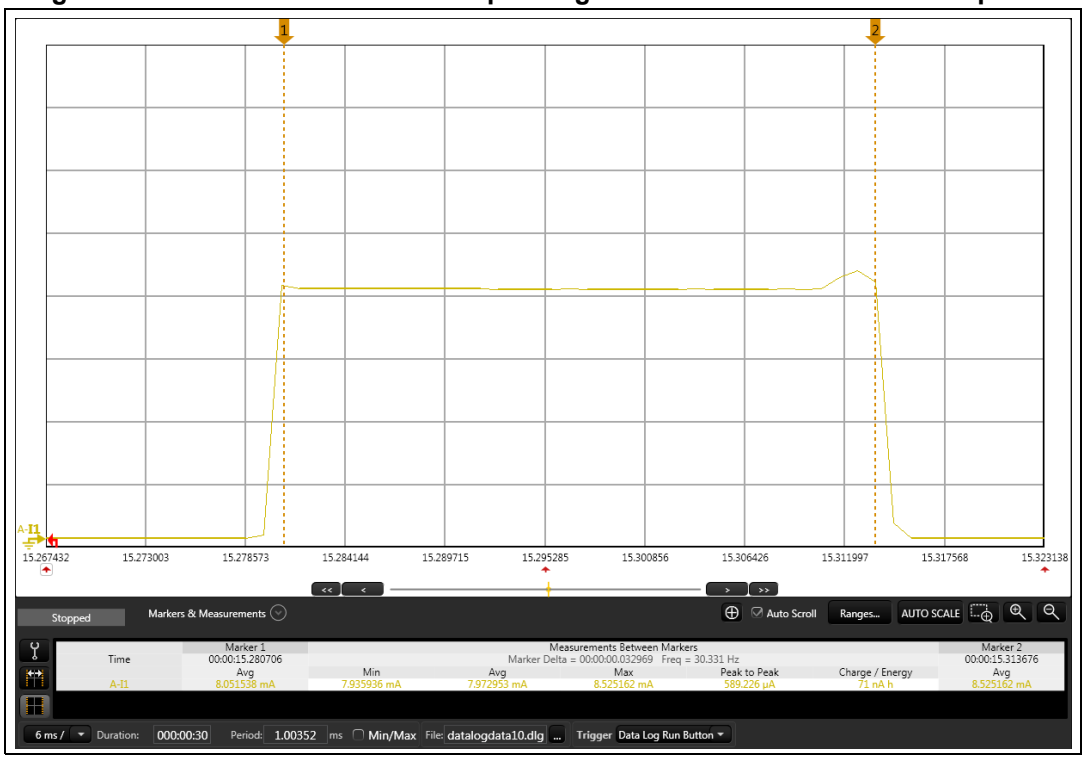


Figure 21. STM32L0 current consumption against time: zoom on run mode period



7 Revision history

Table 29. Document revision history

Date	Revision	Changes
27-Jun-2016	1	Initial release.