UM1904

User manual

Getting started with X-CUBE-SUBG1, Sub-1 GHz RF software expansion for STM32Cube

Introduction

X-CUBE-SUBG1 is an expansion software package for STM32Cube. The software runs on the STM32 and includes drivers that recognize the Sub-1 GHz RF communication for SPIRIT1 SPSGRF modules and S2-LP.

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

The software comes with sample applications of P2P, wM-Bus and 6LoWPAN communication protocols, running on a compatible SPIRIT1 or S2-LP expansion board when connected to a compatible STM32 Nucleo development board.

1 Acronyms and abbreviations

Table 1. List of acronyms

Acronym	Description
AMR	Automatic meter reading
BSP	Board support package
EEPROM	Electrically erasable programmable read-only memory
GHz	Giga Hertz
GUI	Graphical user interface
HAL	Hardware abstraction layer
LED	Light emitting diode
MCU	Microcontroller unit
P2P	Point-to-Point communication
RF	Radio frequency communication
SPI	Serial peripheral interface
USB	Universal serial bus
wM-Bus	Wireless metering bus
WSN	Wireless sensor network

2 What is STM32Cube?

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

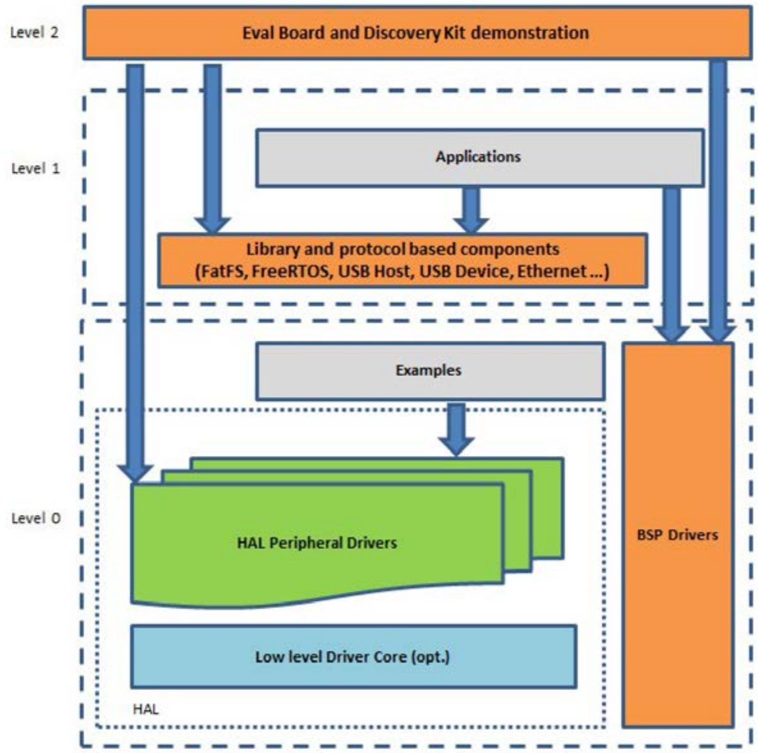
STM32Cube version 1.x includes:

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

2.1 STM32Cube architecture

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

Figure 1. Firmware architecture


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

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

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

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

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

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

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

3 X-CUBE-SUBG1 software expansion for STM32Cube

3.1 Overview

X-CUBE-SUBG1 is a software package that expands the functionality of STM32Cube.

The key features of the package are:

- Firmware package to start developing using SPIRIT1 or S2-LP expansion boards
- Complete middleware to build wireless meter bus (wM-Bus) applications using the wM-Bus library (X-NUCLEO-S2868A1 and X-NUCLEO-IDS01A4 only)
- Middleware library with Contiki OS and Contiki 6LoWPAN protocol stack 3.x (NUCLEO-F401RE and NUCLEO-L152RE only)
- Point-to-point communication sample application for simple buffer transmission and acknowledgement implementation
- Low-power optimizations for the STM32 MCU family
- Easy portability across different MCU families thanks to STM32Cube
- PC-based application (Windows®) for wM-Bus to log meter data
- Free user-friendly license terms
- Sample implementation available on X-NUCLEO-IDS01A4 or X-NUCLEO-IDS01A5 and X-NUCLEO-S2868A1 expansion boards when connected to NUCLEO-F401RE, NUCLEO-L053R8 or NUCLEO-L152RE boards

Startting from this software, t is possible to develop other applications, such as:

- automatic meter reading
- home and building automation
- WSN (wireless sensors network)
- industrial monitoring and control
- wireless fire and security alarm systems

The firmware partitioning among the STM32 microcontroller on the STM32 Nucleo development boards, the SPIRIT1 and the S2-LP is:

- STM32 MCU
  - P2P application implementation
  - low power mode handling
  - interrupt services
- SPIRIT1 role
  - basic/Stack modes
  - header, sync and trailer fields
  - encoding/decoding
  - sync detection
  - TX and RX FIFO
- S2-LP role
  - basic/Stack modes
  - header, sync and trailer fields
  - encoding/decoding
  - sync detection
  - RX and TX 128 bytes FIFO buffers
  - IEEE 802.15.4g hardware packet support with whitening, FEC, CRC and dual sync word detection.

3.2 Architecture

This software is fully compliant with and expands on STM32Cube (see Section 2 What is STM32Cube?) to enable development of applications using X-NUCLEO-IDS01Ax (X- NUCLEO-IDS01A4 or X-NUCLEO-IDS01A5) or X-NUCLEO-S2868A1 boards hosting the SPIRIT1 and S2-LP devices.

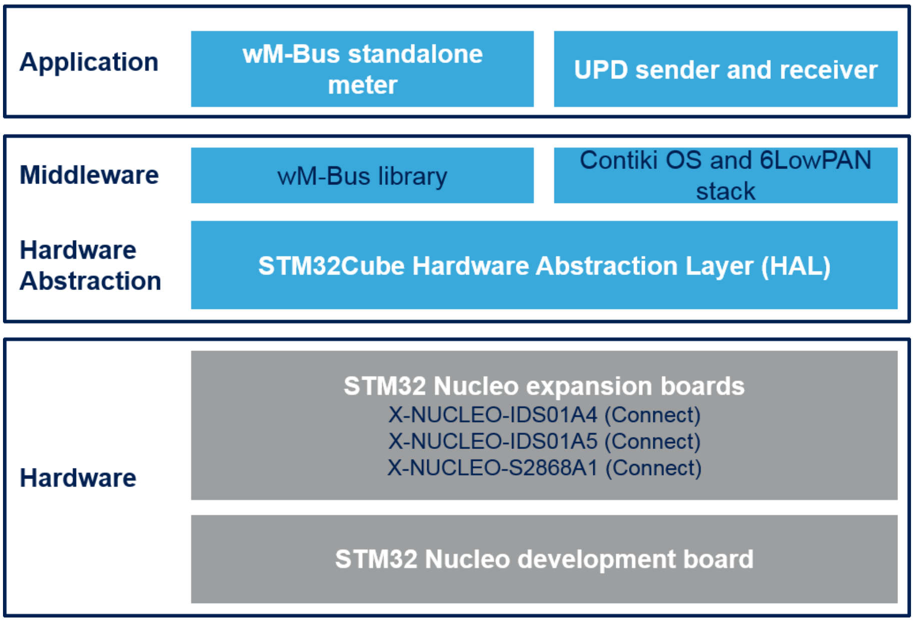
The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a board support package

(BSP) for the SPIRIT1 or S2-LP expansion board and some example firmware for P2P communication.

The software layers used by the application software to access and use the SPIRIT1 or S2-LP expansion board are:

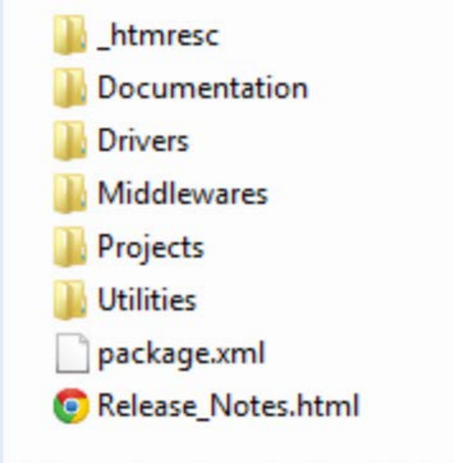
- STM32Cube HAL layer: provides a generic, multi-instance set of APIs to interact with the upper layers (the application, libraries and stacks). It consists of generic and extension APIs based on a common architecture which allows other layers like the middleware layer to function without specific Microcontroller Unit (MCU) hardware configurations. This structure improves library code reusability and guarantees easy device portability.
- Board support package (BSP) Layer: includes the software to support the peripherals on the STM32 Nucleo board (apart from the MCU). It is a set of APIs which provides a programming interface for certain board-specific peripherals (LED, user button etc.). The BSP firmware layer of the X-NUCLEO-IDS01Ax boards contains APIs for the hardware components and consists of two parts:
  - Component: this is the driver related to the external device on the board and not related to the STM32. The SPIRIT1 BSP driver is known as the firmware component. The SPIRIT1 component driver provides specific APIs and can be ported to and used on any board.
  - BSP driver: enables the component driver to be linked to a specific board and provides a set of user-friendly APIs.
- Middleware: includes the wM-Bus, USB, touch sensing etc. libraries. There is no middleware component For Point-to-Point applications as the demo/application layer interacts with the SPIRIT1 link layer directly
- Application layer: provides a Point-to-Point communication example which involves sending a buffer from one node to another and acknowledgments using the features in the SPIRIT1 link layer.

Figure 2. **X-CUBE-SUBG1 software architecture**


Application
wM-Bus standalone meter
UPD sender and receiver
Middleware
wM-Bus library
Contiki OS and 6LowPAN stack
Hardware Abstraction
STM32Cube Hardware Abstraction Layer (HAL)
Hardware
STM32 Nucleo expansion boards
X-NUCLEO-IDS01A4 (Connect)
X-NUCLEO-IDS01A5 (Connect)
X-NUCLEO-S2868A1 (Connect)
STM32 Nucleo development board

3.3 Folder structure

Figure 3. X-CUBE-SUBG1 package folder structure



_htmresc
Documentation
Drivers
Middlewares
Projects
Utilities
package.xml
Release_Notes.html

The following folders are included in the software package:

- ‘Documentation’: contains a compiled HTML file generated from the source code and detailed documentation of the software components and APIs
- ‘Drivers’: contains the HAL drivers and the board-specific drivers for supported board and hardware platforms, including those for the on-board components and the CMSIS vendor-independent hardware abstraction layer for the Cortex-M processor series
- ‘Middlewares’: contains libraries for wM-Bus and 6LoWPAN protocol stack
- ‘Projects’: contains a sample application used for wM-Bus and P2P firmware examples for the NUCLEO-L053R8 and NUCLEO-F401RE or NUCLEO-L152RE platforms with three development environments, IAR Embedded Workbench for ARM (IAR-EWARM), RealView Microcontroller Development Kit (MDK-ARM-STM32), System Workbench for STM32 (SW4STM32)
- ‘Utilities’: this folder contains a ‘PC_software’ subfolder with a Windows PC utility for wM-Bus usage and testing.

3.4 APIs

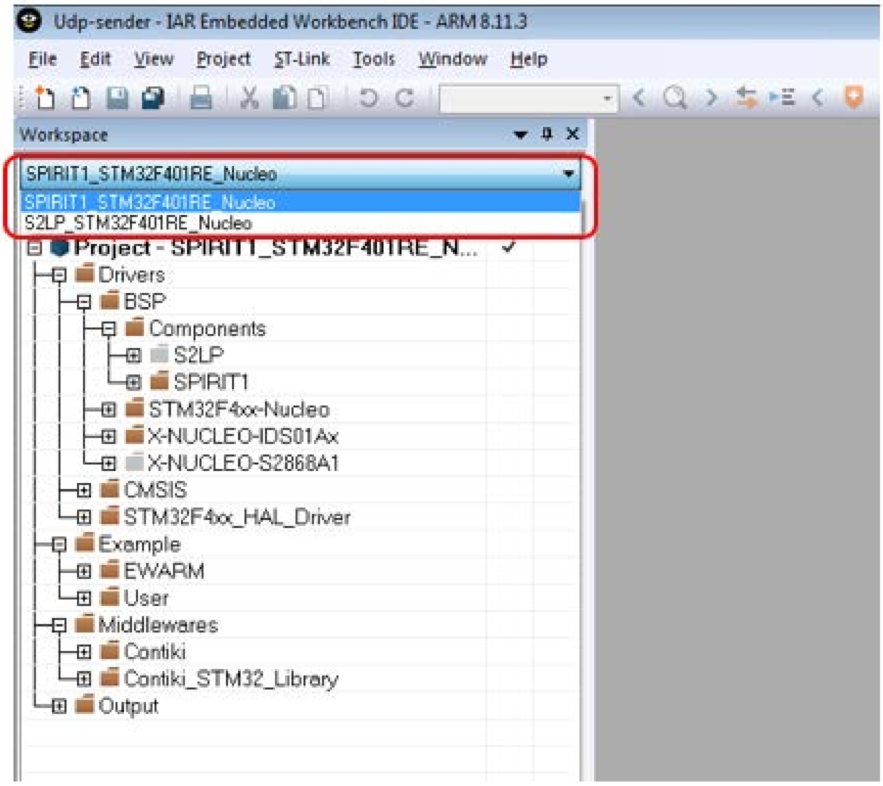
Detailed descriptions of all the functions and parameters of the user APIs user can be found in a compiled HTML file located inside the ‘Documentation’ folder.

3.5 Selecting radio board configuration

X-CUBE-SUBG1 software package supports both SPIRIT1 and S2-LP radio application in a single package.

The figure below shows how to select the firmware configuration in the workspace, according to the radio used (SPIRIT1 or S2-LP).

Figure 4. Selecting the radio board firmware configuration



Udp-sender - IAR Embedded Workbench IDE - ARM 8.11.3

SPIRIT1_STM32F401RE_Nucleo

SPIRIT1_STM32F401RE_Nucleo
S2LP_STM32F401RE_Nucleo

```
Project - SPIRIT1_STM32F401RE_N...  ✓
├ Drivers
│ ├ BSP
│ │ ├ Components
│ │ │ ├ S2LP
│ │ │ └ SPIRIT1
│ │ ├ STM32F4xx-Nucleo
│ │ ├ X-NUCLEO-IDS01Ax
│ │ └ X-NUCLEO-S2868A1
│ ├ CMSIS
│ └ STM32F4xx_HAL_Driver
├ Example
│ ├ EWARM
│ └ User
├ Middlewares
│ ├ Contiki
│ └ Contiki_STM32_Library
└ Output
```

4 Point-to-Point (P2P) demo firmware description

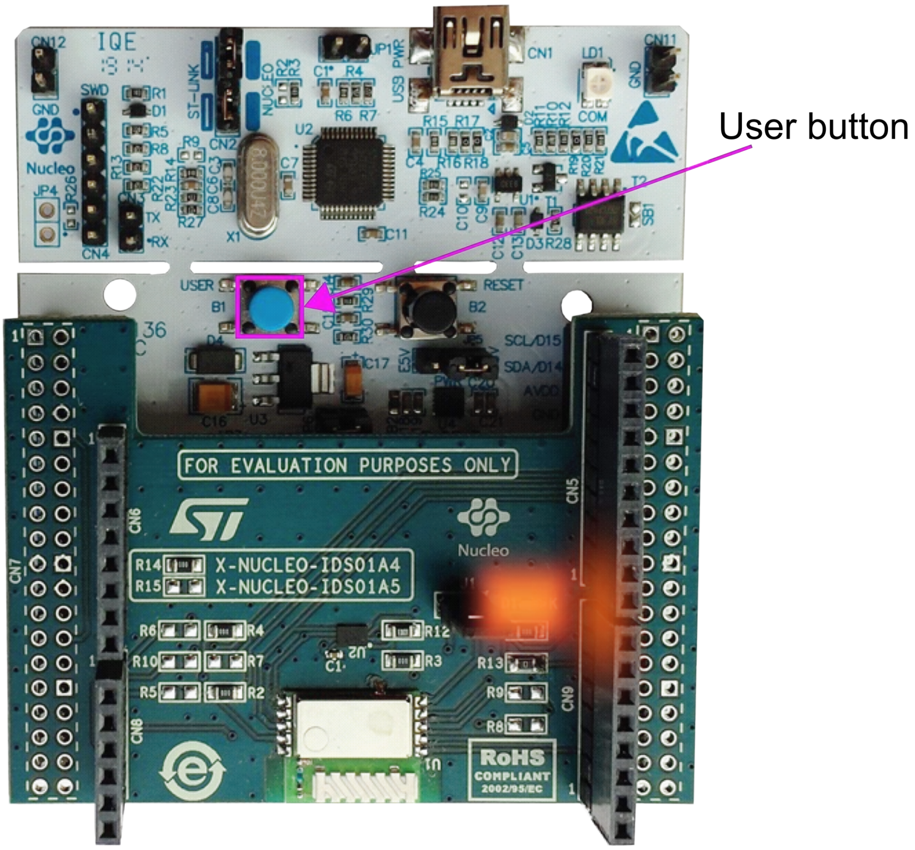
The following section explains how the demo firmware is implemented, the user settings and configurations available and how to modify the firmware for other applications.

4.1 P2P application details

The P2P application operates using two nodes (STM32 Nucleo board plus SPIRIT1 or S2-LP expansion board) as follows:

1. by pressing the Nucleo board user button (shown in the picture below), each node can transmit a buffer to the other node
2. on receiving the signal, the receiver node LED lights up and an acknowledgment (ACK) signal is returned to the transmitter node
3. on reception of the ACK signal, the transmitter node LED flashes four times and switches off after a delay period

Figure 5. X-NUCLEO-IDS01Ax plus STM32 Nucleo used as a node (transmitter/receiver) in P2P communication



4.2 Application state diagram

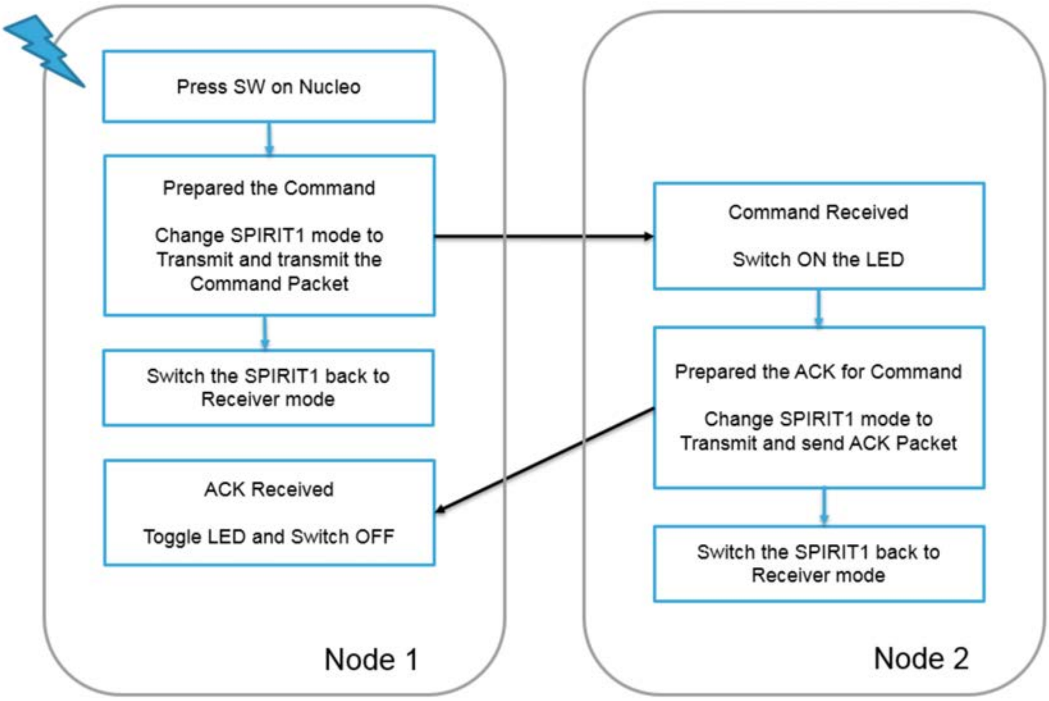
This section explains how to run the demo sample with the STM32 Nucleo boards.

SPIRIT1 or S2-LP remains by default in receive mode but changes to transmit mode when the user button is pressed. Once transmission has terminated, the transceiver returns to its default receive mode. On successful completion of the two-way communication (Command/ Ack), the MCU enters low-power mode.

To limit low-power mode current consumption, the LED is switched off by default.

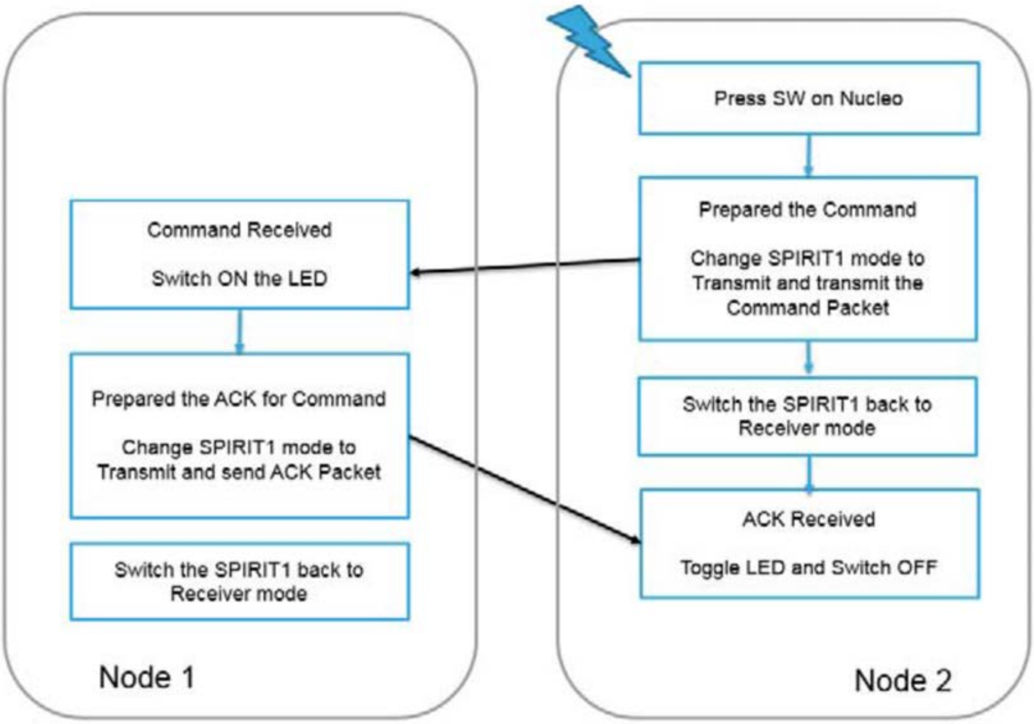
P2P nodes have the same functionality; the address of each node is set in the firmware by the user.

Figure 6. **Application state diagram when Node 1 user button is pressed**


Press SW on Nucleo
Prepared the Command
Change SPIRIT1 mode to Transmit and transmit the Command Packet
Switch the SPIRIT1 back to Receiver mode
ACK Received
Toggle LED and Switch OFF
Node 1
Command Received
Switch ON the LED
Prepared the ACK for Command
Change SPIRIT1 mode to Transmit and send ACK Packet
Switch the SPIRIT1 back to Receiver mode
Node 2

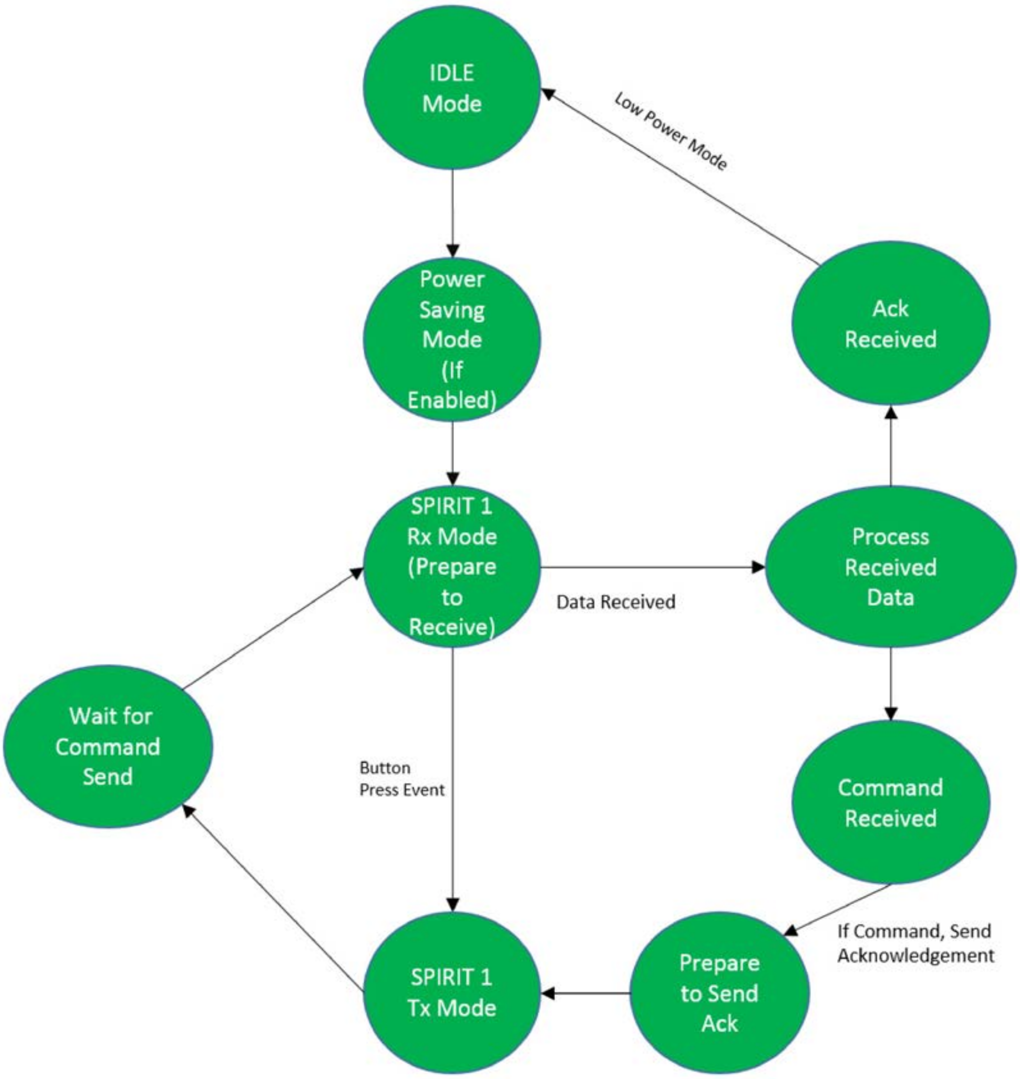
If the user presses the other node user button, the functionality is the same: Node 2 wakes up from low-power mode, prepares the command for transmission, sends the data packet and waits for acknowledgment.

Figure 7. **Application state diagram when Node 2 user button is pressed**


Press SW on Nucleo
Prepared the Command
Change SPIRIT1 mode to Transmit and transmit the Command Packet
Switch the SPIRIT1 back to Receiver mode
ACK Received
Toggle LED and Switch OFF
Command Received
Switch ON the LED
Prepared the ACK for Command
Change SPIRIT1 mode to Transmit and send ACK Packet
Switch the SPIRIT1 back to Receiver mode
Node 1
Node 2

The following diagram shows the transmit and receive states for data communication in the firmware low-power mode.

Figure 8. Application state diagram (low-power mode)



4.3 SPIRIT1 packet handler overview

Before on-air transmission, raw data is arranged in a packet structure. SPIRIT1 offers a highly flexible and fully programmable packet which lets you configure the structure of the packet, the number, the type, and the dimension of the fields inside the packet.

Through a register, the user can choose from one of the formats shown in the tables below.

Table 2. Stack

Preamble	Sync	Length	Destination address	Source address	Control	Seq. no.	No ACK	Payload	CRC

Table 3. wM-Bus

Preamble	Sync	Payload	Postamble

Table 4. **Basic**

Preamble	Sync	Length	Destination address	Control	Payload	CRC

See SPIRIT1 datasheet for further details on the embedded packet handler.

Since P2P communication requires the receiving node destination address, the P2P demo is based on stack and basic packet handlers.

Note: The wM-Bus packet format is not used in this sample demonstration.

Table 5. **Packet handler feature comparison**

Features	Stack	wM-Bus	Basic
Destination address filtering	Yes	No	Yes
Broadcast and multicast addressing	Yes	No	Yes
Source address filtering	Yes	No	No
Custom filtering	Yes	No	Yes
CRC filtering	Yes	No	Yes
LLP: automatic acknowledgment(1)	Yes	No	No
LLP: automatic acknowledgment with piggybacking(1)	Yes	No	No
LLP: automatic retransmission(1)	Yes	No	No

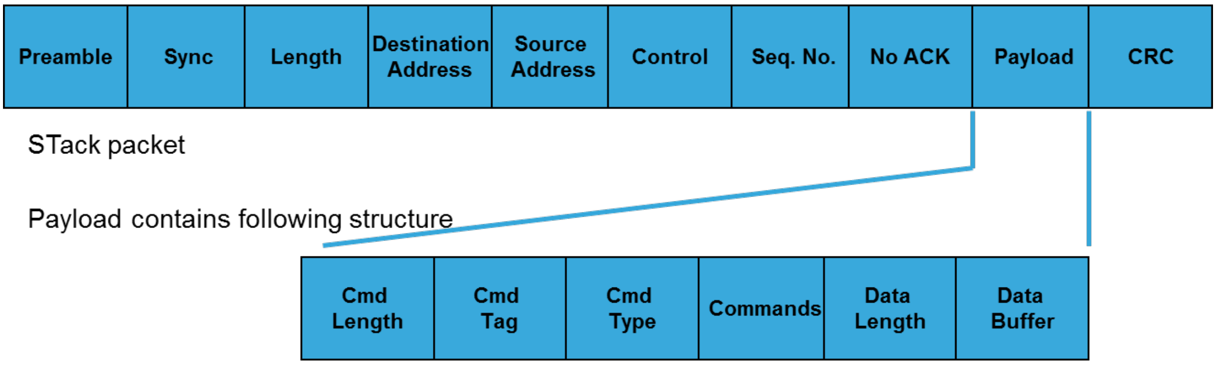
1. Link layer protocol

4.4 Transmit and receive (command and response) packet structure

Command packet features:

- command with data sent at the same time
- SPIRIT1 can handle 65535 bytes of data
- customizable command structure
- customizable data packet maximum size

Figure 9. **Command data packet structure**



Preamble	Sync	Length	Destination Address	Source Address	Control	Seq. No.	No ACK	Payload	CRC

STack packet

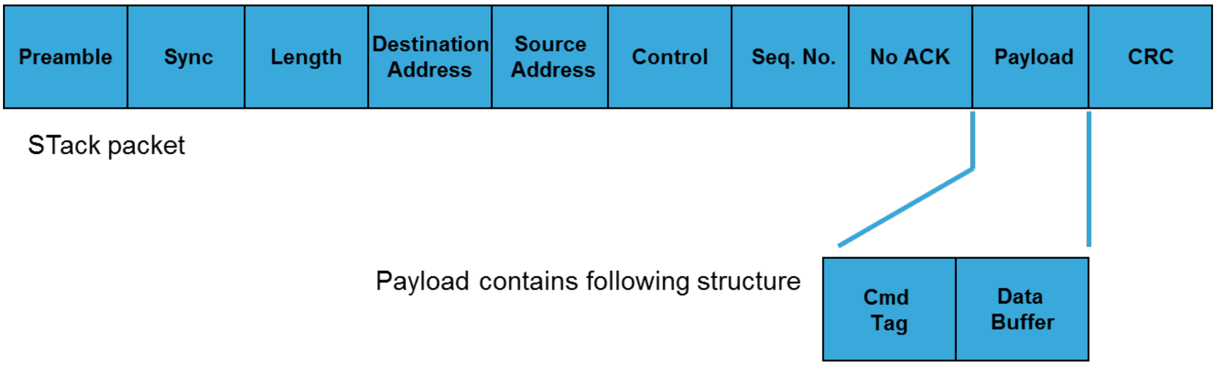
Payload contains following structure

Cmd Length	Cmd Tag	Cmd Type	Commands	Data Length	Data Buffer

Response packet features:

- data buffer is replied from the node
- tag contains the number associated with the command so the receiver can associate the response with the specific command

Figure 10. **Response packet structure**



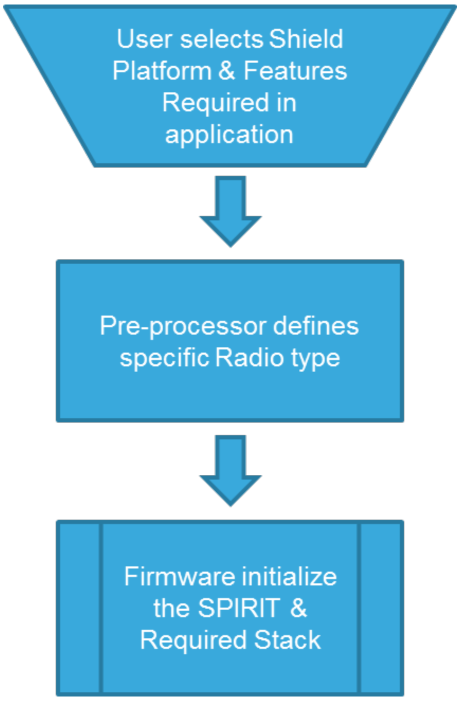
4.4.1 Packet field description

- **Cmd Length**: the basic command is 1 byte long, but you can set multiple command bytes
- **Cmd Tag**: a unique tag number is linked to each command issued from the node and the response must replicate the same number
- **Cmd Type**: flag to identify application level or network command
- **Commands**: the actual command set sent from the source to destination (it may include parameters)
- **Data Length**: the data packet length
- **Data buffer**: the actual data associated with the command

4.5 User configuration

You can modify the configuration file spirit1_appli.h on the basis of the application used.

Figure 11. **User configuration**


User selects Shield Platform & Features Required in application
Pre-processor defines specific Radio type
Firmware initialize the SPIRIT & Required Stack

4.5.1 Selecting the SPIRIT1 expansion board platform

You can select the desired SPIRIT1 expansion board platform by uncommenting the macros:

```
/* Platform definition: Uncomment the used expansion board*/
#define X_NUCLEO_IDS01A4
// #define X_NUCLEO_IDS01A5
#if defined(X_NUCLEO_IDS01A4)
#define USE_SPIRIT1_868MHz
#elif defined(x_NUCLEO_IDS01AS)
#define USE_SPIRIT1_915MHz
#else
#error SPIRIT1 Nucleo expansion board undefined or unsupported
#endif
```

Once the SPIRIT1 expansion board platform is selected, the operating frequency is handled by the firmware itself.

For example, if X-NUCLEO-IDS01A4 is selected as the SPIRIT1 expansion board, the operating frequency automatically selected by the firmware is 868 MHz.

4.5.2 Selecting packet handler

The user can select the desired features by uncommenting the relevant macros:

```
/* Uncomment the Link Layer features to be used */
// #define USE_AUTO_ACK
// #define USE_AUTO_ACK_PIGGYBACKING
// #define USE_AUTO_RETRANSMISSION

#if     defined(USE_AUTO_ACK)&& defined(USE_AUTO_ACK_PIGGYBACKING)&&
defined(USE_AUTO_RETRANSMISSION)
#define USE_STack_PROTOCOL

/* LLP configuration parameters */
#define EN_AUTOACK              S_ENABLE
#define EN_PIGGYBACKING         S_ENABLE
#define MAX_RETRANSMISSIONS          PKT_N_RETX_2
#else
#define USE_BASIC_PROTOCOL
#endif
```

By default, the SPIRIT1 works with the basic packet handler.

SPIRIT1 uses the STack packet handler only if the link layer features (such as auto-ack, piggybacking and auto-retransmission) are defined.

4.5.3 Setting low-power mode

The P2P application supports low-power mode, enabled by default. It allows the MCU to either enter stop or sleep mode.

```
/* Uncomment the system Low Power Operating mode */
#define USE_LOW_POWER_MODE

#if defined (USE_LOW_POWER_MODE)
#define LPM_ENABLE
//#define MCU_STOP _MODE
#define MCU_SLEEP MODE
//#define RF _STANDBY
#endif
```

SPIRIT1 can be set to standby and after to lower-power consumption mode.

4.5.4 Setting radio configuration parameters

You can set the radio parameters in the configuration file, even though it is not recommended to change them.

```
/* Radio configuration parameters */
#define XTAL_OFFSET_PPM                        0
#define INFINITE_TIMEOUT                             0.0

#ifdef USE_RADIO_433MHz
#define BASE_FREOUENCY                         433.0e6
#endif
#ifdef USE_RADIO_868MHz
#define BASE_FREQUENCY                         868.0e6
#endif
#ifdef USE_RADIO_915MHz
#define BASE_FREOUENCY                         915.0e6
#endif

#define CHANNEL_SPACE                          20e3
#sefine CHANNEL_NUMBER                         0
#define MODULATION_SELECT                      FSK
#define DATARATE                                       38400
#define FREQ_DEVIATION                                 20e3
#define BANDWIDTH                                      100E3
#define POWER_DBM                                      11.6
#define POWER_INDEX                                    7
#define RECEIVE_TIMEOUT                        2000.0/*change the value for requir
ed timeout period*/


/* Radio configuration parameters */
```

4.5.5 Setting packet configuration parameters

You can set the packet configuration, even though it is not recommended to change default settings.

```
*/ Packet configuration parameters */
#define PREAMBLE_LENGTH                           PKT_PREAMBLE_LENGTH_04BYTES
#define SYNC_LENGTH                               PKT_SYNC_LENGTH_4BYTES
#define SYNC_WORD                                 0x1A2635A8
#define LENGTH_TYPE                               PKT_LENGTH_VAR
#define LENGTH_WIDTH                              7
#define CRC_MODE                                  PKT_CRC_MODE_8BITS
#define CONTROL_LENGTH                            PKT_ CONTROL_LENGTH_0BYTES
#define EN_ADDRESS                                S_DISABLE
#define EN_FEC                                    S_DISABLE
```

4.5.6 Setting address of the nodes

Node ddresses can be set in following section of the system setup guide.

```
*/ Addresses configuration parameters */
#define EN_FILT_MY_ADDRESS                             S_DISABLE
#define MY_ADDRESS                                     0x34
#define EN_FILT_MULTICAST_ADDRESS                      S_DISABLE
#define MULTICAST_ADDRESS                              0xEE
#define EN_FILT_BROADCAST_ADDRESS                      S_DISABLE
#define BROADCAST_ADDRESS                              0xFF

#define DESTINATION_ADDRESS                            0x44
#define EN_FIL T _SOURCE_ADDRESS                       S_DISABLE
#define SOURCE ADDR MASK                               0xf0
```

4.5.7 User defined commands and macros

```
/* User Command */
#define APPLI_CMD                           0x11
#define NWK_CMD                             0x22
#define LED_TOGGLE                          0xff
#define ACK_OK                              0x01
#define MAX_BUFFER_LEN                      96
#define TIME_TO_EXIT_RX                     3000
#define DELAY_RX_LED_TOGGLE                 200
#define DELAY_TX_LED_GLOW                   1000
#define LPM WAKEUP TIME                     100
```

4.6 Hardware configuration

4.6.1 STM32 Nucleo platform

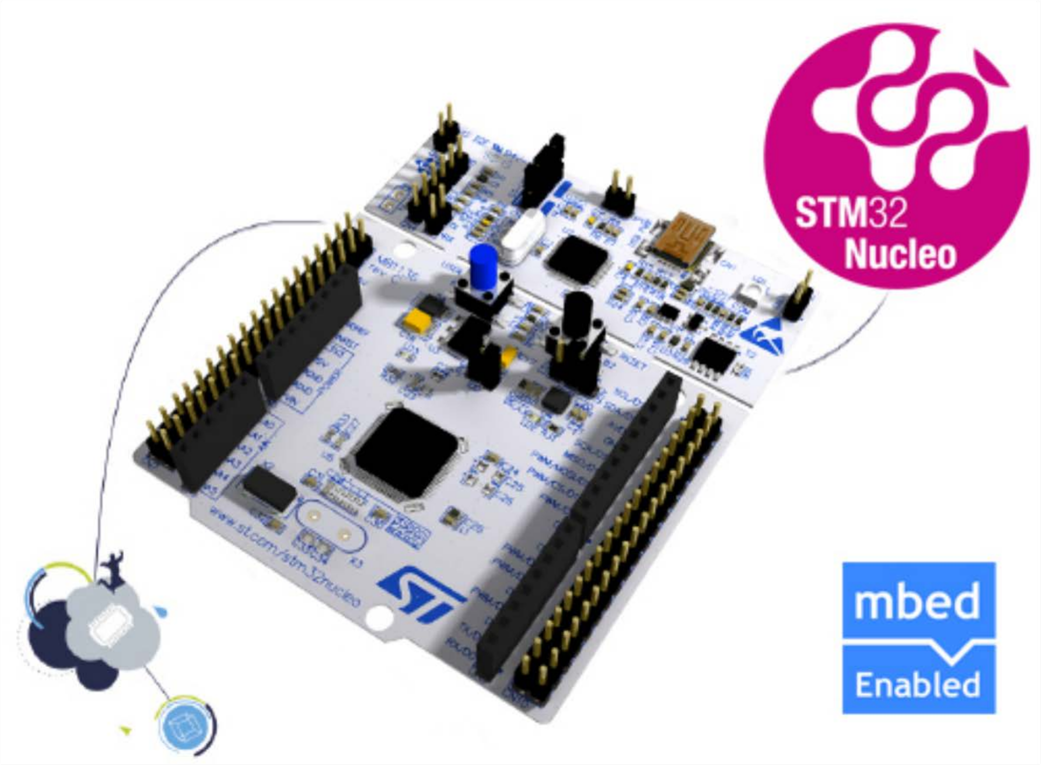
STM32 Nucleo development boards provide an affordable and flexible way for users to test solutions and build prototypes with any STM32 microcontroller line.

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

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

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

Figure 12. STM32 Nucleo board



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

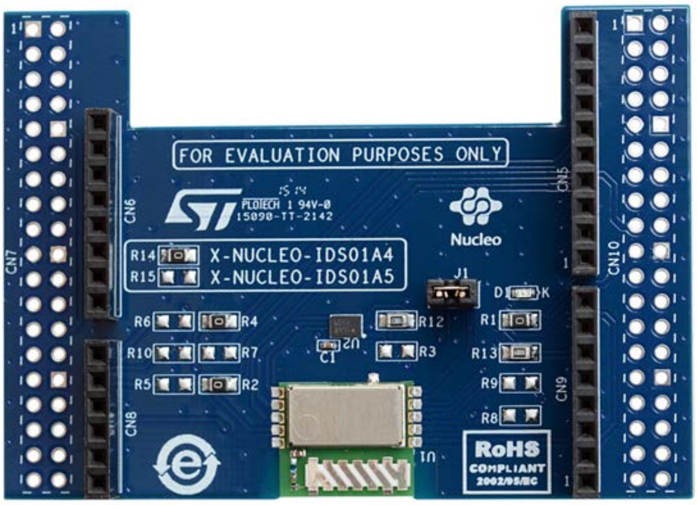
4.6.2 X-NUCLEO-IDS01Ax expansion board

The X-NUCLEO-IDS01A4 and X-NUCLEO-IDS01A5 evaluation boards allow you to evaluate the features and capabilities of the SPIRIT1 low data rate, low power sub-1 GHz transceiver device.

These expansion boards include on-board SPI EEPROM to store parameters and user interface signal LED.

The X-NUCLEO-IDS01A4 board operates the SPIRIT1 transceiver at 868MHz, while the X-NUCLEO-IDS01A5 board operates the SPIRIT1 transceiver at 915MHz.

Figure 13. X-NUCLEO-IDS01Ax expansion board



Information regarding the X-NUCLEO-IDS01A4 and X-NUCLEO-IDS01A5 expansion boards is available on www.st.com at www.st.com/x-nucleo.

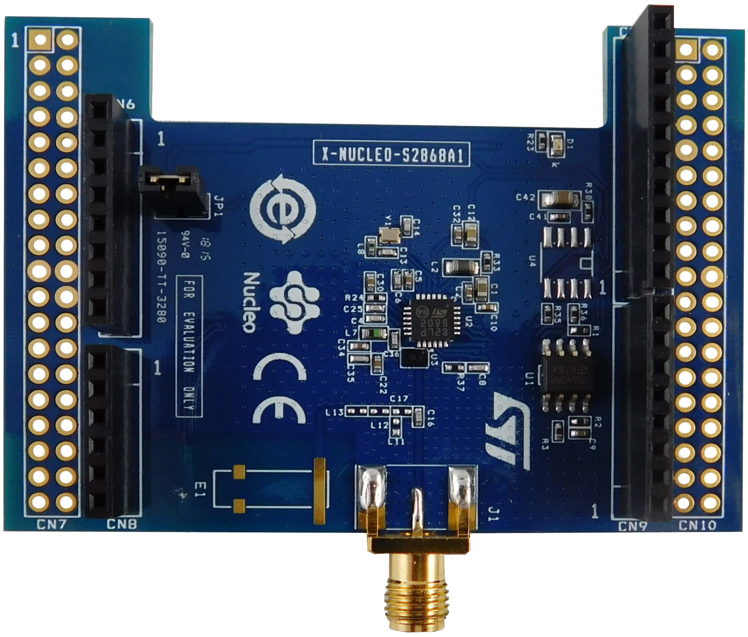
4.6.3 X-NUCLEO-S2868A1 expansion board

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

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

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

Figure 14. **X-NUCLEO-S2868A1 expansion board**



4.7 Software description

To use STM32 Nucleo development boards with the X-NUCLEO-IDS01Ax or X-NUCLEO-S2868A1 expansion boards, the following software specification are required:

- **X-CUBE-SUBG1** expansion for STM32Cube. The X-CUBE-SUBG1 firmware and related documentation is available on st.com.
- **Development tool-chain and Compiler** supported by the STM32Cube expansion software:
  - IAR Embedded Workbench for ARM® (IAR-EWARM) toolchain + ST-LINK
  - RealView Microcontroller Development Kit (MDK-ARM-STM32) toolchain + ST-LINK
  - System Workbench for STM32 (SW4STM32) + ST-LINK

4.8 Hardware setup

The following hardware components are required:

- an STM32 Nucleo development board (order code: NUCLEO- F401RE or NUCLEO-L053R8)
- a SPIRIT1 or an S2-LP expansion board (order code: X-NUCLEO-IDS01A4, X- NUCLEO-IDS01A5 or X-NUCLEO-S2868A1)
- a USB type A to Mini-B USB cable to connect the STM32 Nucleo to the PC

4.9 Board setup

Step 1. Check that the jumper on J1 connector is connected to provide the required voltage to the board devices.

Step 2. Connect the X-NUCLEO-IDS01Ax or X-NUCLEO-S2868A1 to the STM32 Nucleo board

Figure 15. X-NUCLEO-IDS1Ax expansion board plugged to STM32 Nucleo board

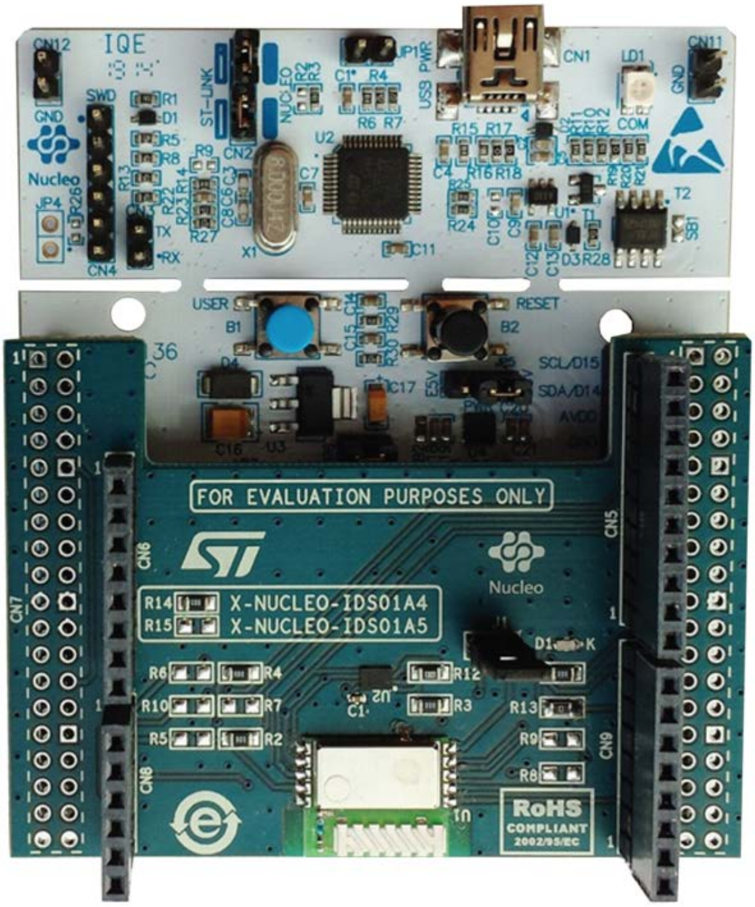
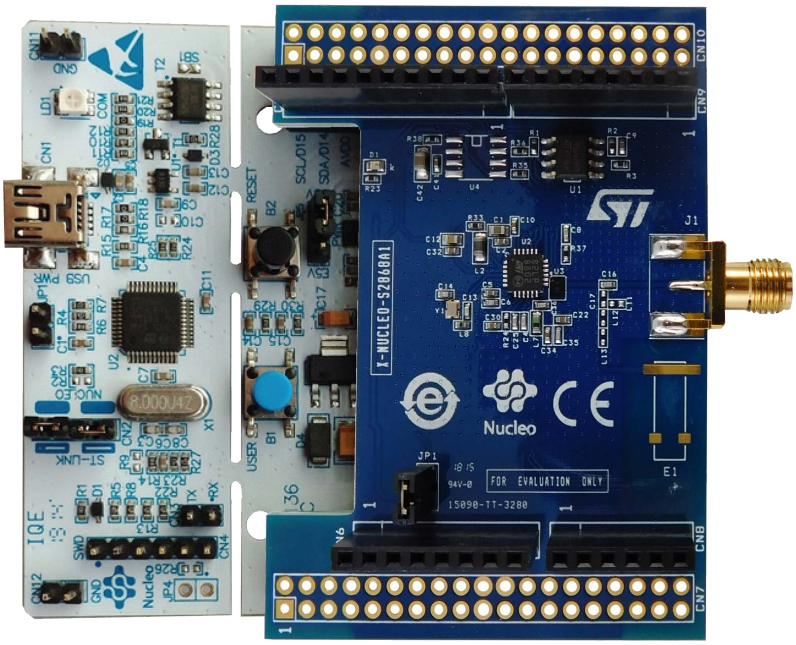


Figure 16. X-NUCLEO-S2868A1 expansion board plugged to STM32 Nucleo board



Step 3. Power the Nucleo board using the Mini-B USB cable

Step 4. Program the firmware in the STM32 on the Nucleo development board using the firmware sample provided

Step 5. Press the reset button on the Nucleo board

The demonstration kit is ready-to-use

4.10 6LoWPAN application

4.10.1 Contiki6LP software description

Contiki6LP is a middleware library ready to be integrated in projects based on STM32Cube and X-CUBE-SUBG1 expansion software.

The software includes samples for sending messages via UDP over 6LoWPAN, using the SPIRIT1 or S2-LP sub-1GHz radio transceiver.

The key features are:

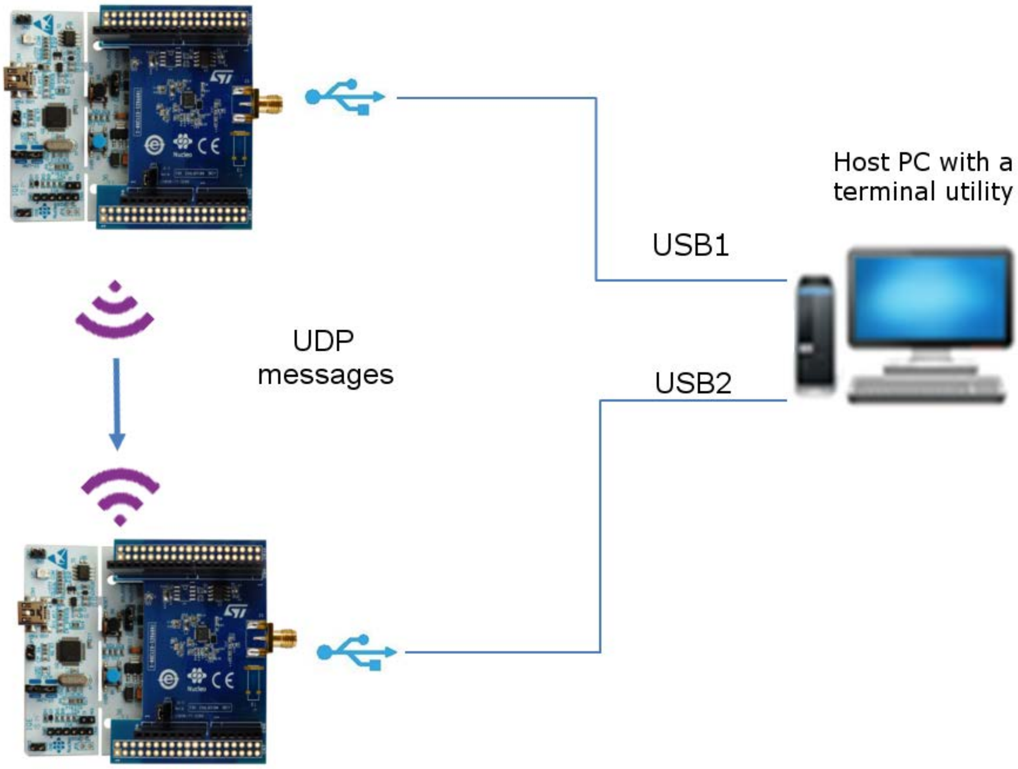
- Middleware library with Contiki OS and Contiki 6LoWPAN protocol stack 3.x
- Support for mesh networking technology via the standard RPL protocol
- Built-in support for STM32 L1 and F4 platforms
- Sample applications (such as UPD sender and receiver, serial siffer and border router)
- Samples available for NUCLEO-F401RE and NUCLEO-L152RE
- Easy portability across different MCU families, thanks to STM32Cube
- Free and user-friendly license terms

4.10.2 UDP sender and receiver sample application overview

This sample application works as follows:

1. the UDP sender node transmits the packets continuously over the air and wait for any receiver node to receive the data packets
2. the receiver node is indefinitely listening for UDP packets, until it receives the data packets from the sender node
3. the receiver node sends acknowledgement and outputs the message packet received in the terminal window
4. the sender node prints the data successfully sent to the receiver node address

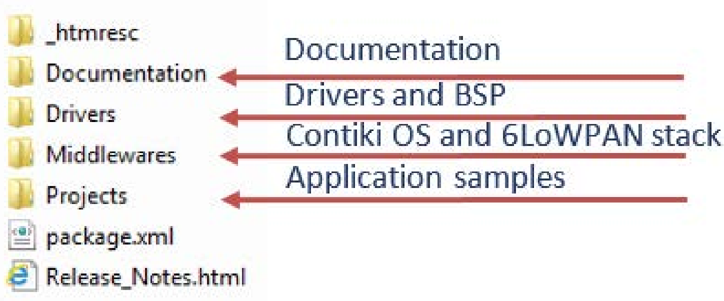
Figure 17. **6LoWPAN UDP sender and receiver node communication with a PC**



4.10.3 Run the application firmware

Step 1. Download and unpack X-CUBE-SUBG1 package.

Figure 18. **X-CUBE-SUBG1 package folders**

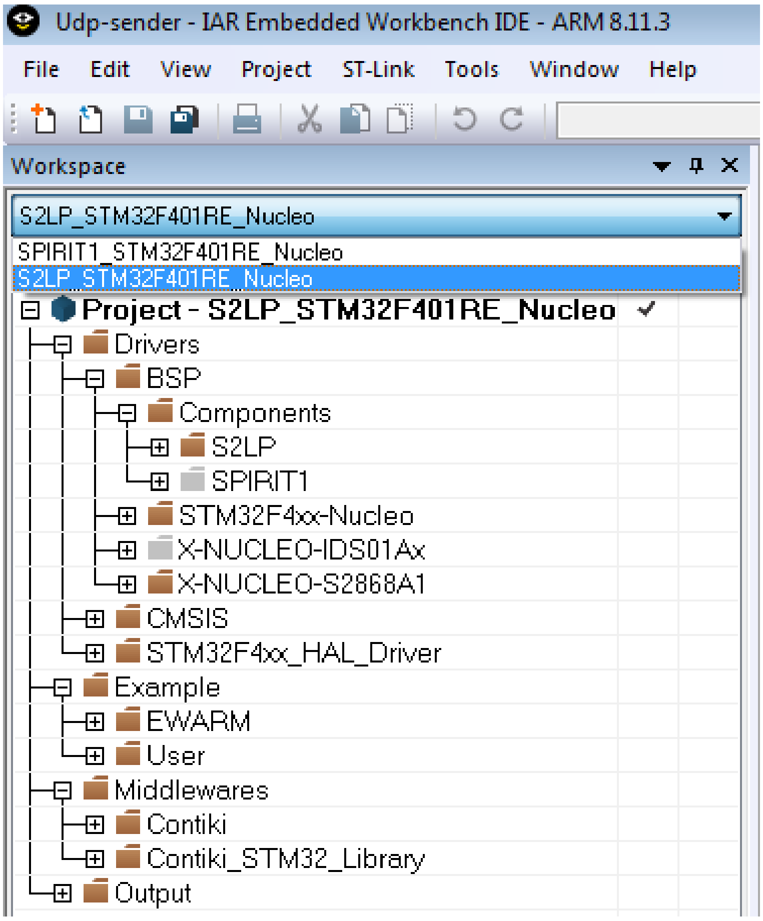


Step 2. Select the UDP receiver application and build the project using a supported IDE.

Alternatively you can use a pre-built binary provided to run this application with the selected STM32 Nucleo board.

Step 3. Select the radio configuration to be used.

Figure 19. **Radio configuration selection**



S2LP_STM32F401RE_Nucleo

SPIRIT1_STM32F401RE_Nucleo
S2LP_STM32F401RE_Nucleo

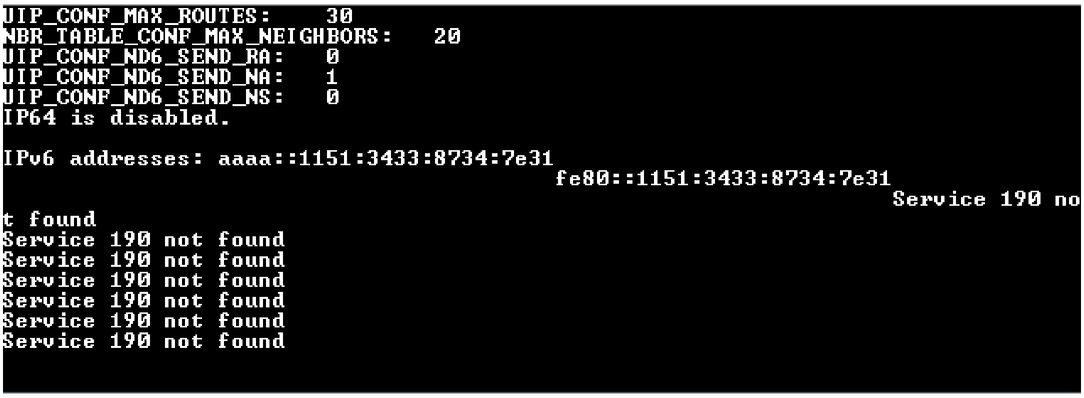
```
⊟ Project - S2LP_STM32F401RE_Nucleo ✓
├⊟ Drivers
│ ├⊟ BSP
│ │ ├⊟ Components
│ │ │ ├⊞ S2LP
│ │ │ └⊞ SPIRIT1
│ │ ├⊞ STM32F4xx-Nucleo
│ │ ├⊞ X-NUCLEO-IDS01Ax
│ │ └⊞ X-NUCLEO-S2868A1
│ ├⊞ CMSIS
│ └⊞ STM32F4xx_HAL_Driver
├⊟ Example
│ ├⊞ EWARM
│ └⊞ User
├⊟ Middlewares
│ ├⊞ Contiki
│ └⊞ Contiki_STM32_Library
└⊞ Output
```

Step 4. Compile the firmware for UDP receiver node.

Step 5. Repeat the same steps for the UDP sender application node.

Step 6. Launch the terminal application and set the UART port to 115200 bps, 8 bit, No Parity, 1 stop bit.
The terminal shows the window below

Figure 20. **UDP sender window**

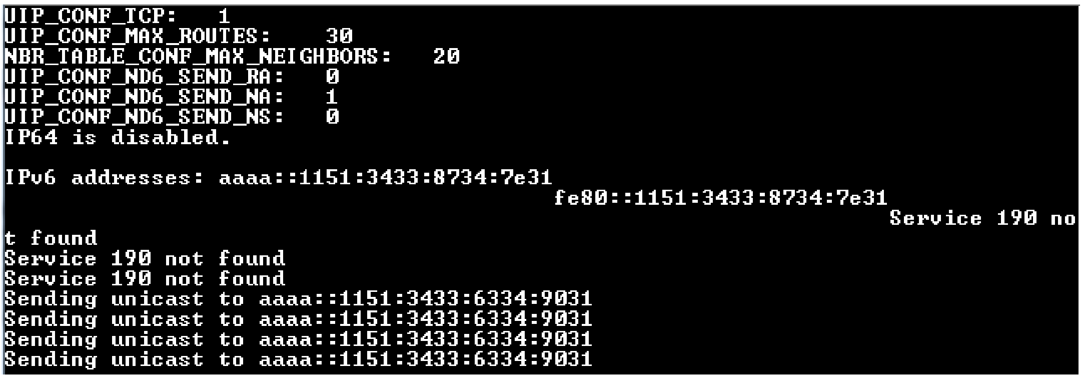


```
UIP_CONF_MAX_ROUTES:    30
NBR_TABLE_CONF_MAX_NEIGHBORS:   20
UIP_CONF_ND6_SEND_RA:   0
UIP_CONF_ND6_SEND_NA:   1
UIP_CONF_ND6_SEND_NS:   0
IP64 is disabled.

IPv6 addresses: aaaa::1151:3433:8734:7e31
                                         fe80::1151:3433:8734:7e31
                                                                  Service 190 no
t found
Service 190 not found
Service 190 not found
Service 190 not found
Service 190 not found
Service 190 not found
Service 190 not found
```

After setting the right parameters, the terminal output becomes

Figure 21. UDP sender terminal output

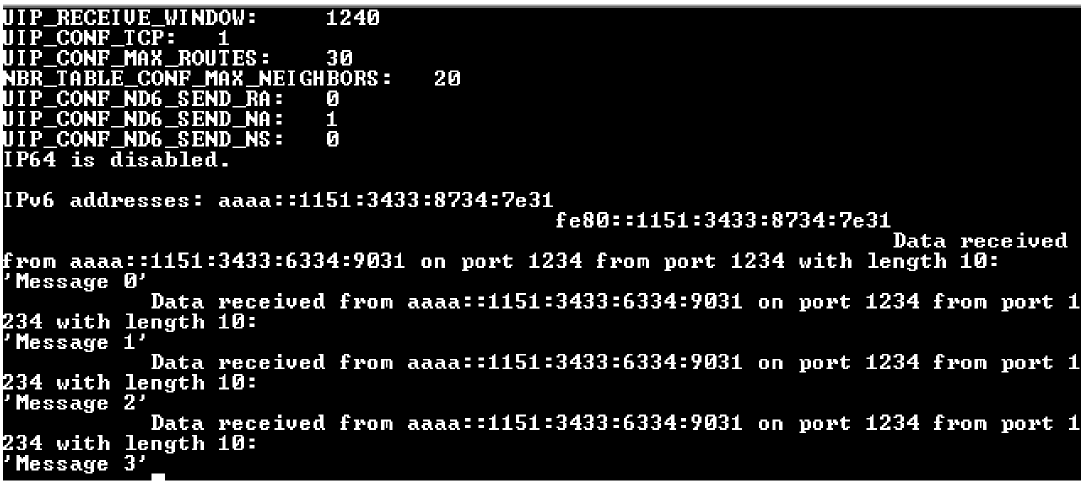


```
UIP_CONF_TCP:   1
UIP_CONF_MAX_ROUTES:    30
NBR_TABLE_CONF_MAX_NEIGHBORS:   20
UIP_CONF_ND6_SEND_RA:   0
UIP_CONF_ND6_SEND_NA:   1
UIP_CONF_ND6_SEND_NS:   0
IP64 is disabled.

IPv6 addresses: aaaa::1151:3433:8734:7e31
                                         fe80::1151:3433:8734:7e31
                                                                  Service 190 no
t found
Service 190 not found
Service 190 not found
Sending unicast to aaaa::1151:3433:6334:9031
Sending unicast to aaaa::1151:3433:6334:9031
Sending unicast to aaaa::1151:3433:6334:9031
Sending unicast to aaaa::1151:3433:6334:9031
```

The received UDP messages are shown as

Figure 22. UDP receiver window



```
UIP_RECEIVE_WINDOW:     1240
UIP_CONF_TCP:   1
UIP_CONF_MAX_ROUTES:    30
NBR_TABLE_CONF_MAX_NEIGHBORS:   20
UIP_CONF_ND6_SEND_RA:   0
UIP_CONF_ND6_SEND_NA:   1
UIP_CONF_ND6_SEND_NS:   0
IP64 is disabled.

IPv6 addresses: aaaa::1151:3433:8734:7e31
                                         fe80::1151:3433:8734:7e31
                                                                  Data received
from aaaa::1151:3433:6334:9031 on port 1234 from port 1234 with length 10:
'Message 0'
           Data received from aaaa::1151:3433:6334:9031 on port 1234 from port 1
234 with length 10:
'Message 1'
           Data received from aaaa::1151:3433:6334:9031 on port 1234 from port 1
234 with length 10:
'Message 2'
           Data received from aaaa::1151:3433:6334:9031 on port 1234 from port 1
234 with length 10:
'Message 3'
```

5 Reference

Freely available on www.st.com:

1. SPIRIT1 device datasheet
2. SPSGRF module datasheet
3. STM32 Nucleo board datasheet
4. UM1872: Getting started with the Sub-1 GHz expansion board based on the SPSGRF- 868 and SPSGRF-915 modules for STM32 Nucleo
5. S2-LP datasheet

Revision history

Table 6. **Document revision history**

Date	Revision	Changes
10-Jun-2015	1	Initial release.
09-Jun-2017	2	Updated text in Introduction, Section 2.1: Overview, and Section 2.3: Folders structure. Replaced Architecture and Application state diagram. Minor text updates throughout the document.
14-May-2018	3	Text and formatting changes throughout document. Added references to X-NUCLEO-S2868A1 and S2-LP. Added Section 3.5 Selecting radio board configuration and Section 4.10 6LoWPAN application.

Contents

List of figures

List of tables

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.