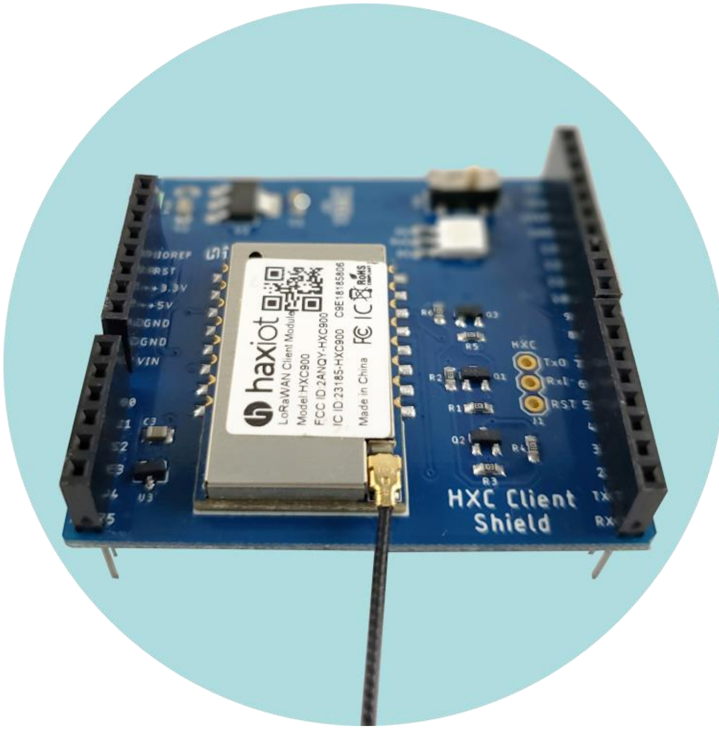
HXC Client Expansion Board User Guide



Version 1.02

Disclaimer and Copyright Notice

LoRa is a registered trademark of Semtech Corporation
LoRaWAN is a registered trademark of the Lora Alliance
Haxiot is a registered trading name of Iotek Systems, LLC

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. IOTEK SYSTEMS MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Iotek Systems disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Iotek Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Iotek Systems intellectual property rights unless otherwise stated.

Table of Contents

List of Figures

List of Tables

1 Introduction

The HXC Client Shield provides an affordable and flexible way for users to try out LoRaWAN and build prototypes around it. The Arduino™ Uno V3 and the STM32 Nucleo headers allow users to easily expand the functionality of the prototype. A comprehensive software library is available for both STM32 and Arduino, together with various packaged software examples, as well as direct access to the Arm® Mbed™ online resources.

2 Features

The HXC Client Shield board offers the following features:

- Built-in Analog Temperature sensor (MCP9700).
- Slide switch for digital input.
- 1 RGB LED.
- Access to Nucleo user and reset push-buttons.
- On-board 3.3V regulator for HXC Client.
- On-board logic converter to support both 5V and 3.3V I/O.
- Jumpers are available to use any MCU pins to connect to HXC Client.
- Flexible power-supply options.
- Comprehensive free HXC Client libraries and examples.
- Support of Arduino, Atollic TrueSTUDIO, and Arm® Mbed™ IDE.

3 Conventions

Table 1 provides the conventions used for the jumper/solder bridge ON and OFF settings in the present document.

Table 3-1: ON/OFF Conventions

Convention	Definition
Solder Bridge ON	Solder bridge connection is closed by solder
Solder Bridge OFF	Solder bridge connection is left open

4 Quick Start

This section will cover how to get started with an HXC Client Shield. The following instructions are for Nucleo-L053R8 development board using Mbed compiler.

4.1 System Requirements

- Windows® OS (7, 8 and 10), Linux® 64-bit or macOS®
- USB Type-A to Mini-B cable

4.2 Development toolchains

- Arm® Mbed™ online (http://mbed.com)

4.3 Getting started

Follow the sequence below to configure the Nucleo board and HXC Client Shield.

1. Nucleo board provides three different USB interfaces: Virtual COM port, Mass storage, and Debug port. To utilize Virtual COM port and Debug port, the user will have to install Nucleo driver. Our example utilizes Mass storage of the Nucleo board, hence doesn't require any driver.

2. Connect HXC Client Shield on top of the Nucleo board, using the Arduino connector.

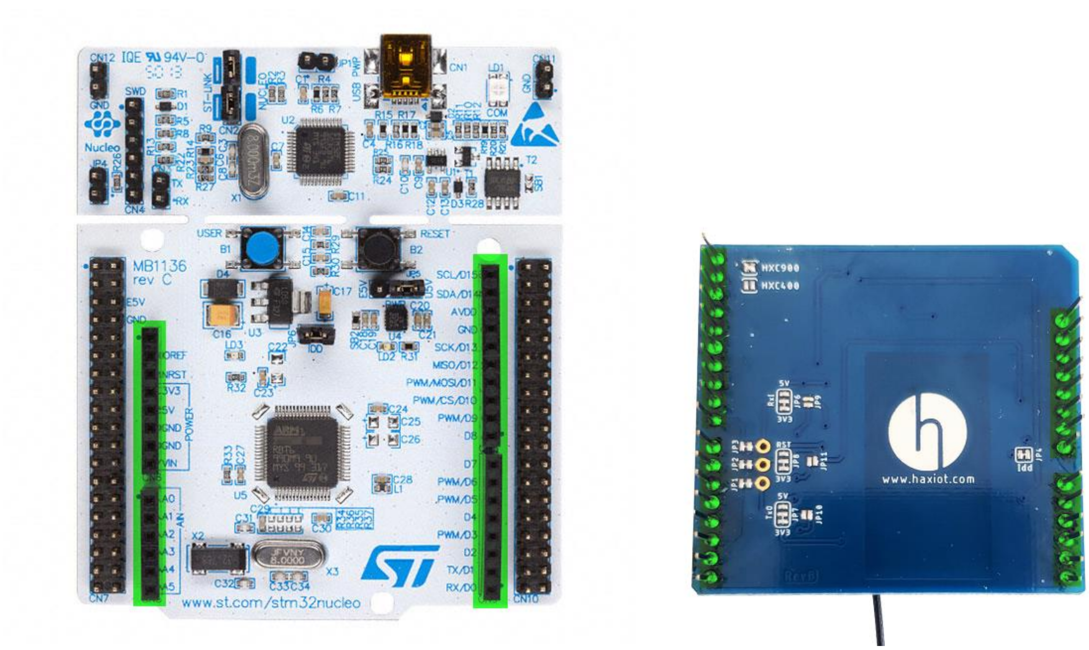


Figure 4-1: Nucleo-L053R8 and HXC Client Expansion Board

Figure 4-2: HXC Shield on top of Nucleo

3. Connect the Nucleo board to a PC with a USB cable ‘Type-A to Mini-B’ through USB connector CN1 to power the board. The LD1 LED (COM) on Nucleo board and D2 LED (PWR) on Shield board should light up.

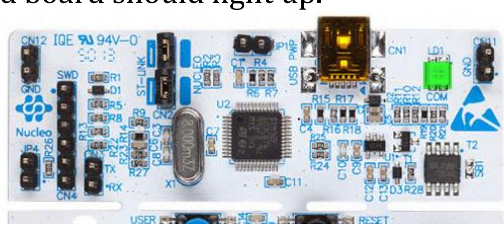


Figure 4-3: LD1 LED (COM)

4. The Nucleo board will also show up as Mass Storage drive. In our example, it showed up as Drive E (NODE_L053R8)

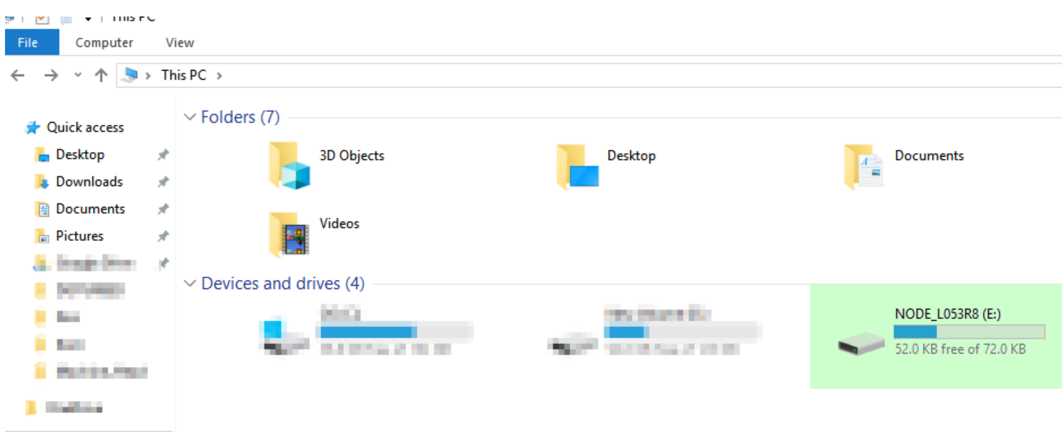


Figure 4-4: Nucleo as Mass Storage Drive

5. Go to "`https://os.mbed.com/compiler`" and open an account. Now login.

6. Depending on where on the website you logged in from, you might end up at the compiler page or at your profile page. If it is the profile page, click on '*Compiler*' to go to the compiler page.

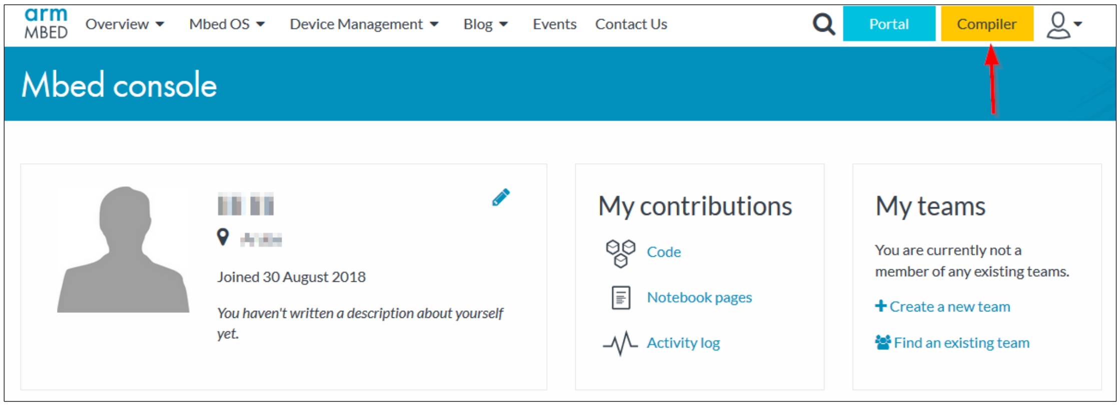


Figure 4-5: Mbed user profile page

7. On the compiler page, click on '`No device selected`' to choose our development platform.

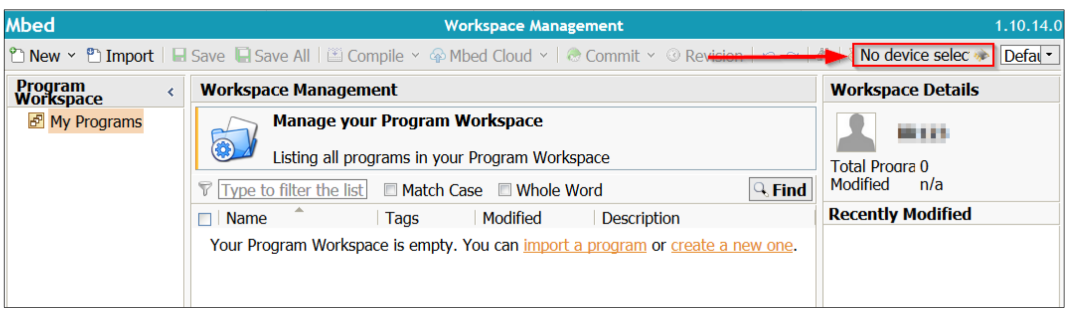


Figure 4-6: Mbed compiler window

8. Click on '`Add Board`'. This will open a new window with a list of Mbed supported development platform.

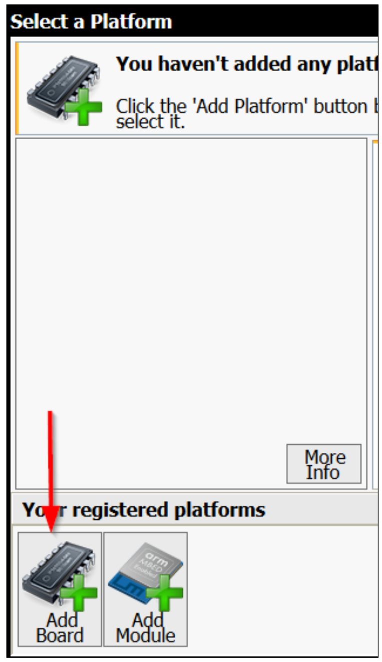


Figure 4-7: Select a platform window

9. Select 'STMicroelectronics' from the 'Target vendor'. This will narrow down the platform list. Now click on 'NUCLEO-L053R8'. That will take you to the platform's dedicated page.

Figure 4-8: List of Mbed supported STM32 boards

10. Click on 'Add to your Mbed Compiler'. A confirmation message will pop-up if it is successfully added to your compiler. Now if you go back to your compiler, it will show what hardware platform you are using now.

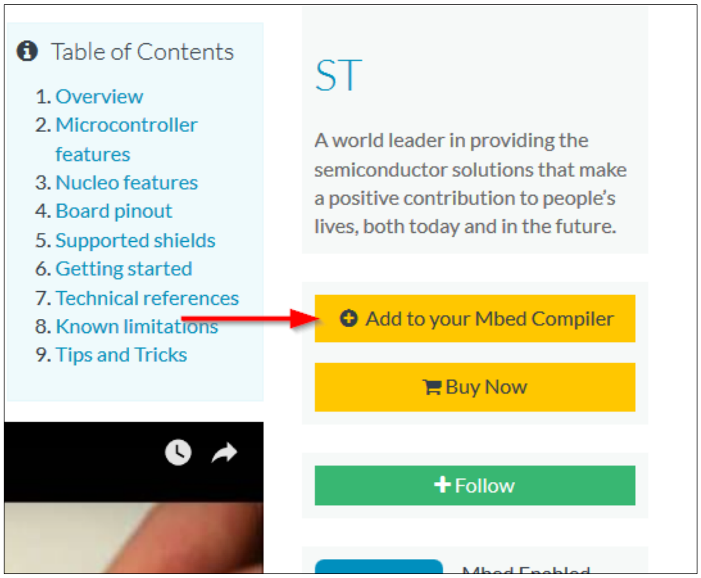


Figure 4-9: Add hardware platform to your compiler

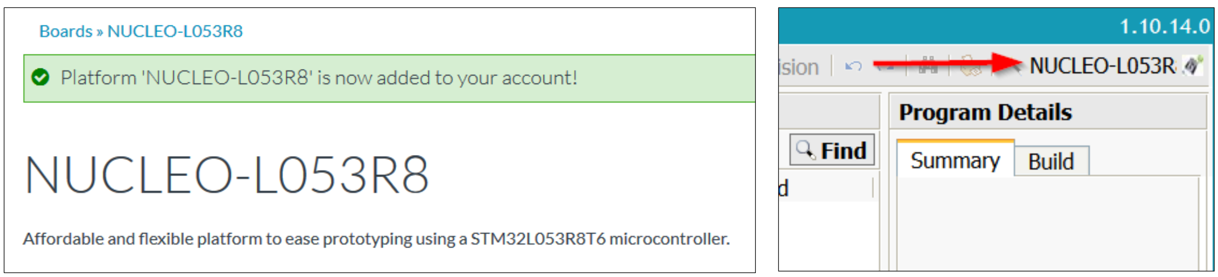


Figure 4-10: Nucleo board was successfully added

11. Now we will import the example code. Go to [this repository](https://os.mbed.com/users/fahadmirza/code/Nucleo_HXC900/) (https://os.mbed.com/users/fahadmirza/code/Nucleo_HXC900/) and click '`Import into Compiler`'.

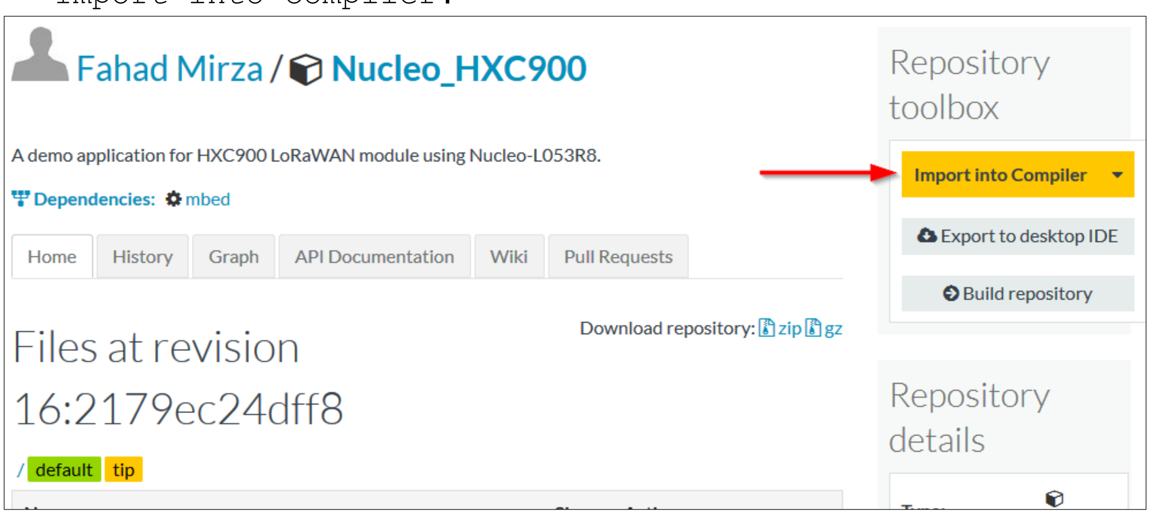


Figure 4-11: HXC Client example repository

12. You can rename your project. Tick the update box and then click '`Import`'.

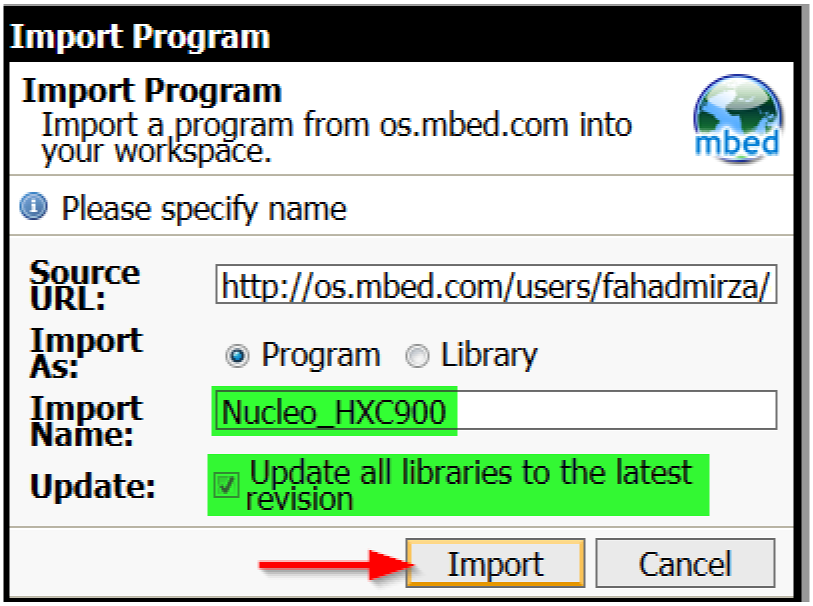


Figure 4-12: Import program window

13. Let's change the LoRaWAN configurations. Open 'lora_conf.h' and scroll down until you reach 'LoraConfigParam'. Choose a JoinMode (OTAA/ABP), enable/disable ADR and choose a Class (A/C).

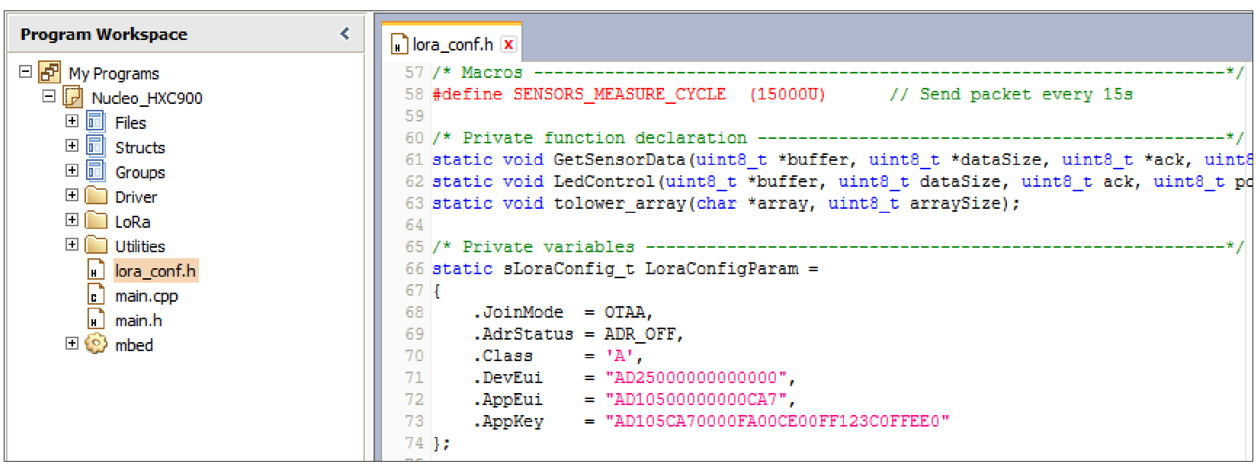


Figure 4-13: main.cpp window

14. Make sure you already have the keys (AppEUI, DevEUI, and AppKey) from our X-ON server (us1.haxiot.com). Replace these default keys with the one you have from X-ON. For this example, we will be using these default keys which are already provisioned on X-ON.

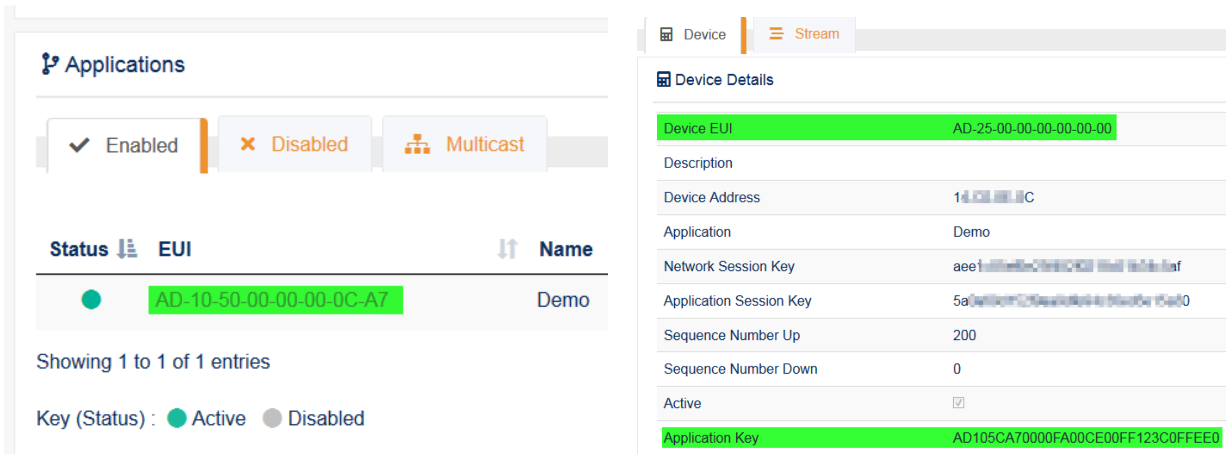


Figure 4-14: Keys on X-ON server

15. Now ‘`Compile`’.

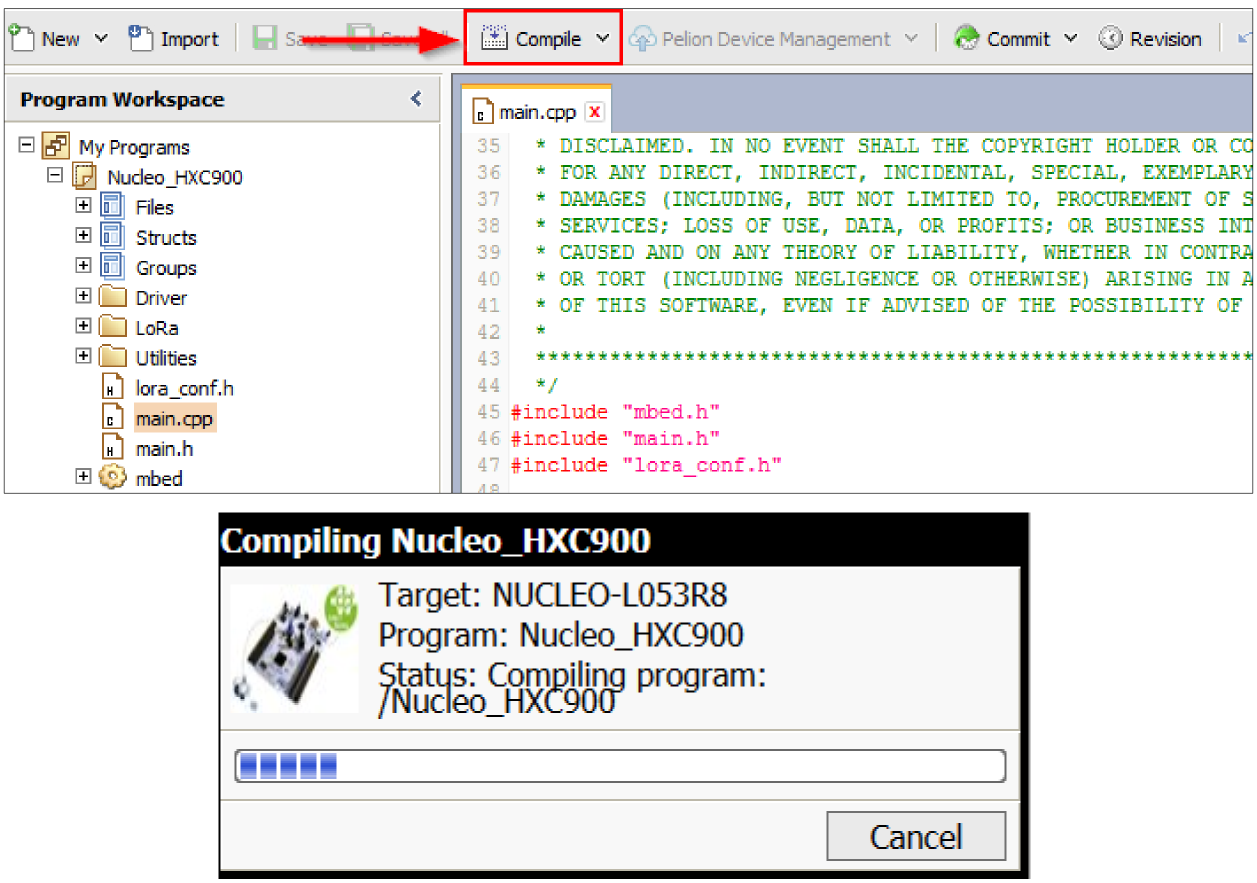


Figure 4-15: Code compilation

16. Save the generated bin file directly into the Mass storage drive (Drive E: in our case).

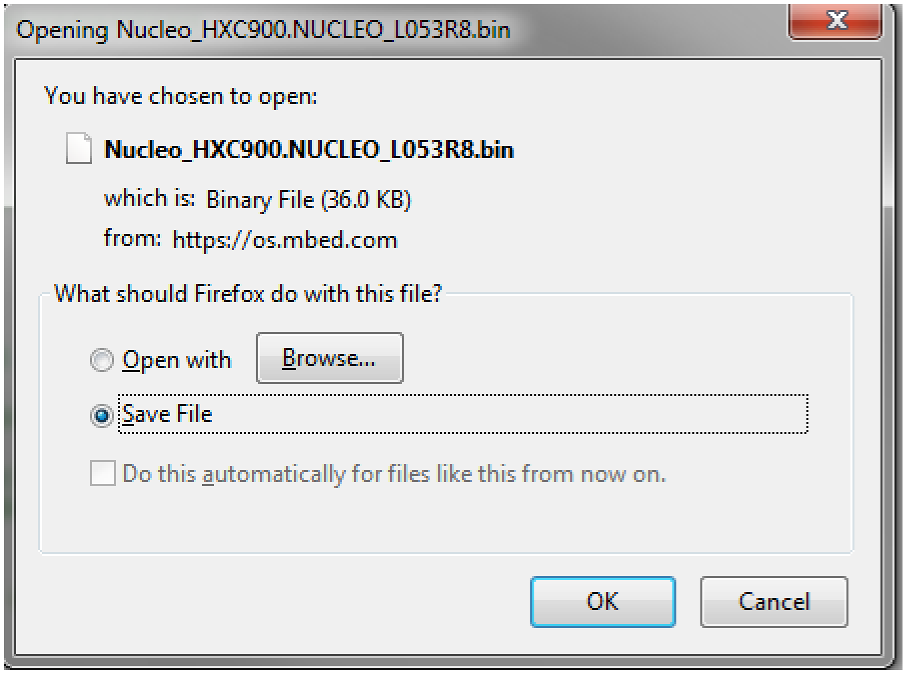


Figure 4-16: Saving the bin file

17. The device will restart itself once the flashing is done. Make sure one of our gateways is running.

18. If the device successfully joins the network server, the D2 LED (RGB) will light up as pink. The device will start sending uplinks with sensor data every 15 seconds (default).

19. Go to X-ON. On your device page, click on 'Stream'. Now you will be able to see data coming from the device.

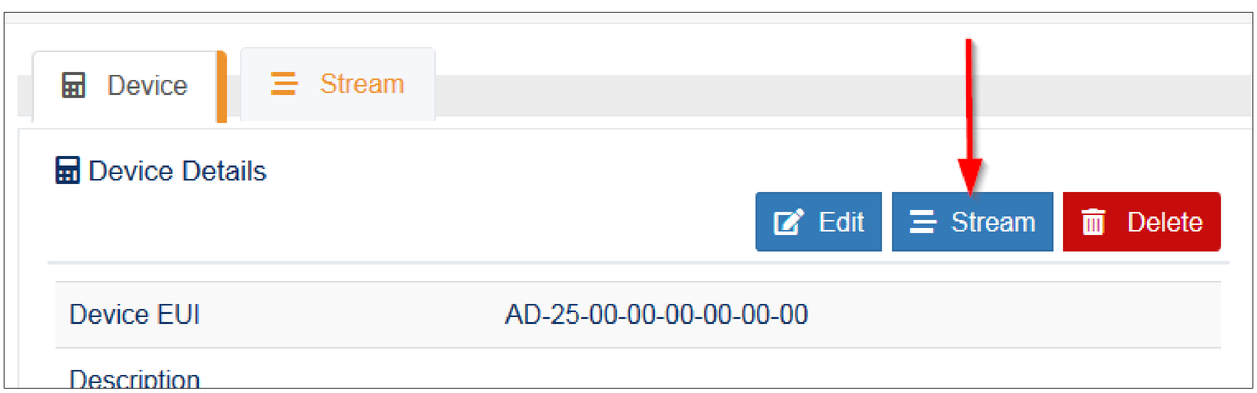


Figure 4-17: Stream tab on X-ON

5 Hardware Layout and Configurations

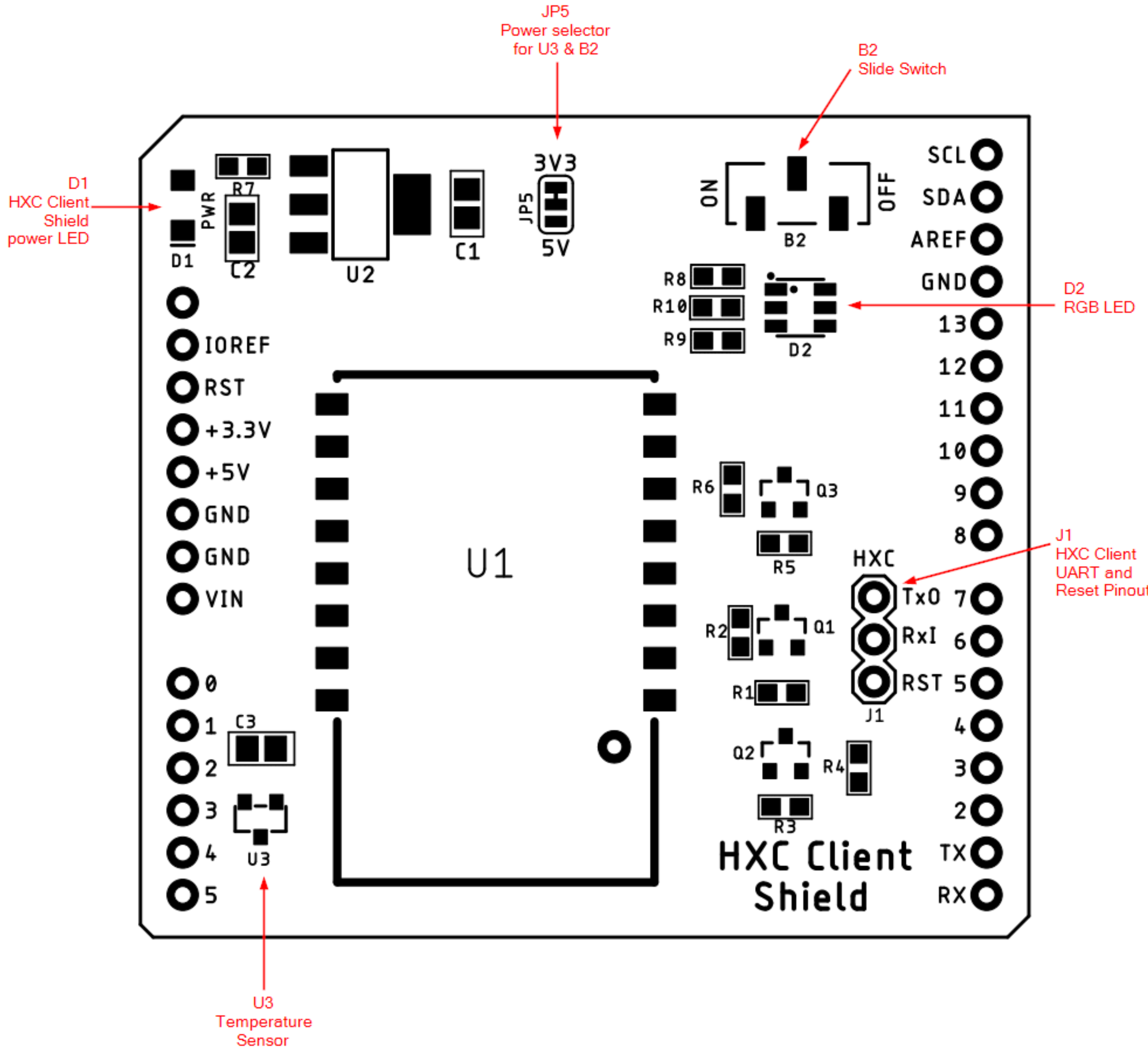


Figure 5-1: HXC Client Expansion Board Top Layout

Table 5-1: Mechanical Dimension

Dimension	Min (mm)	Max(mm)
Height	20	21
Width (Fig1 - W)	49.5	50
Length (Fig1 - L)	53.5	54

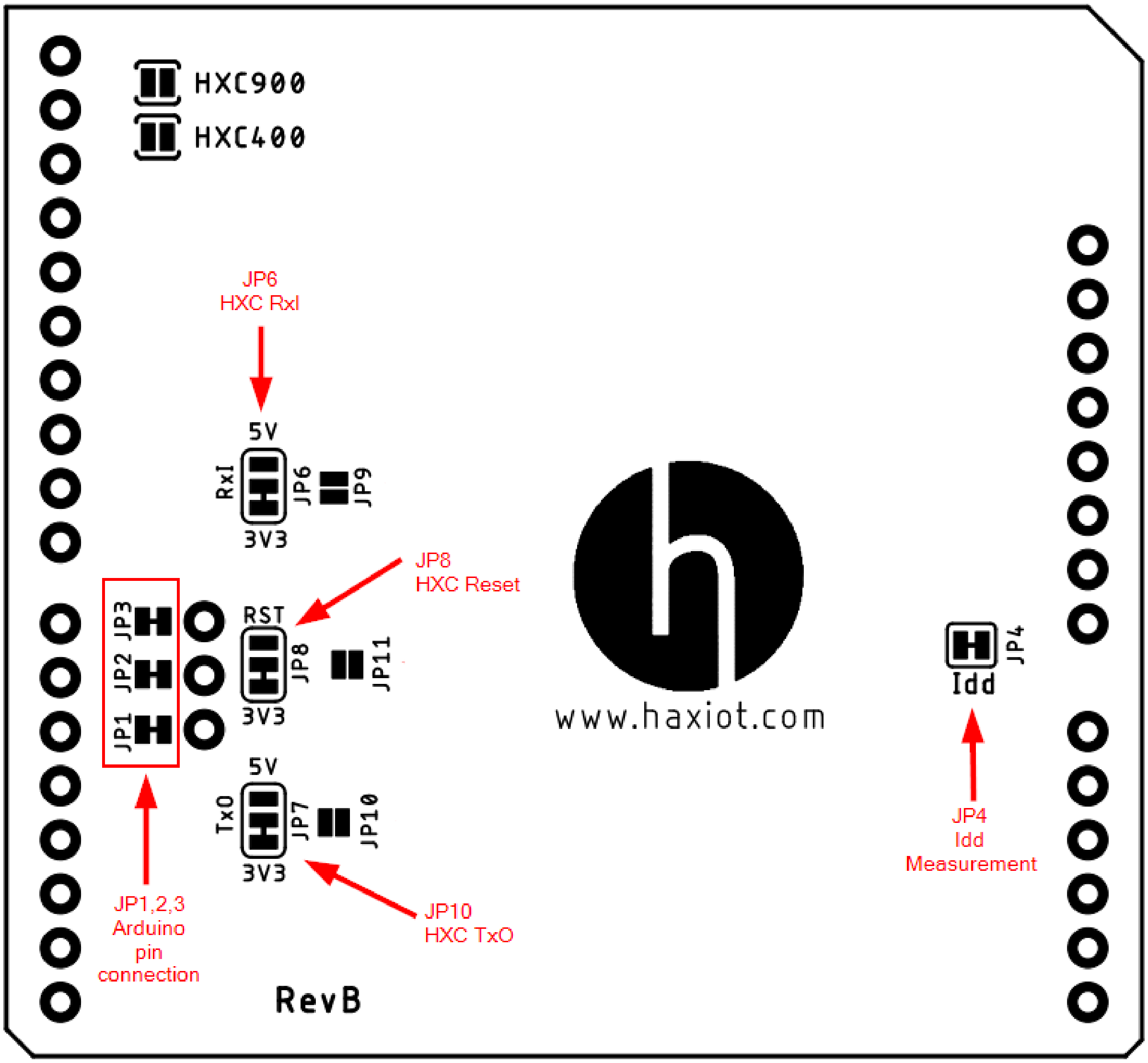


Figure 5-2: HXC Client Expansion Board Bottom Layout

5.1 Nucleo Board (or 3.3V I/O) Connection

Nucleo boards use 3.3V I/O. The default solder bridges (as shown in Figure 5-1 and Figure 5-2) are for Nucleo boards or any other Arduino form factor board (e.g. Metro board from Adafruit) that uses 3.3V I/O.

5.2 Arduino Uno Board (or 5.5V I/O) Connection

Arduino Uno uses 5V I/O. But the HXC Client module uses 3.3V. The shield has three logic converter circuits to accommodate TxO, RxI and Reset pin. Cut the traces of JP6, JP7, and JP8 from the 3.3V pad and solder them with the 5V pad. Also solder JP9, JP10, and JP11.

Table 5-2: Solder bridge connection for 5V I/O

Jumper	Solder Bridge
JP6	Middle-Top --> SB on Middle-Bottom --> SB off
JP7	Middle-Top --> SB on Middle-Bottom --> SB off
JP8	Middle-Top --> SB on Middle-Bottom --> SB off
JP9	SB on
JP10	SB on
JP11	SB on

5.3 Idd measurement

Jumper JP4 is used to measure the HXC Client consumption by cutting the trace and connecting an ammeter.

SB ON: HXC Client is powered (default).
SB OFF: an ammeter must be connected to measure the HXC Client current. If there is no ammeter, the client is not powered.

5.4 Different I/O for HXC Client UART

By default, HXC Client's TxO is connected to D2, RxI is connected to D8 and Reset is connected to D6. If users want to use different pins, cut the trace of JP1, JP2, and JP3. Use J1 to connect HXC Client to any other pins using wires.

6 Extension Connectors

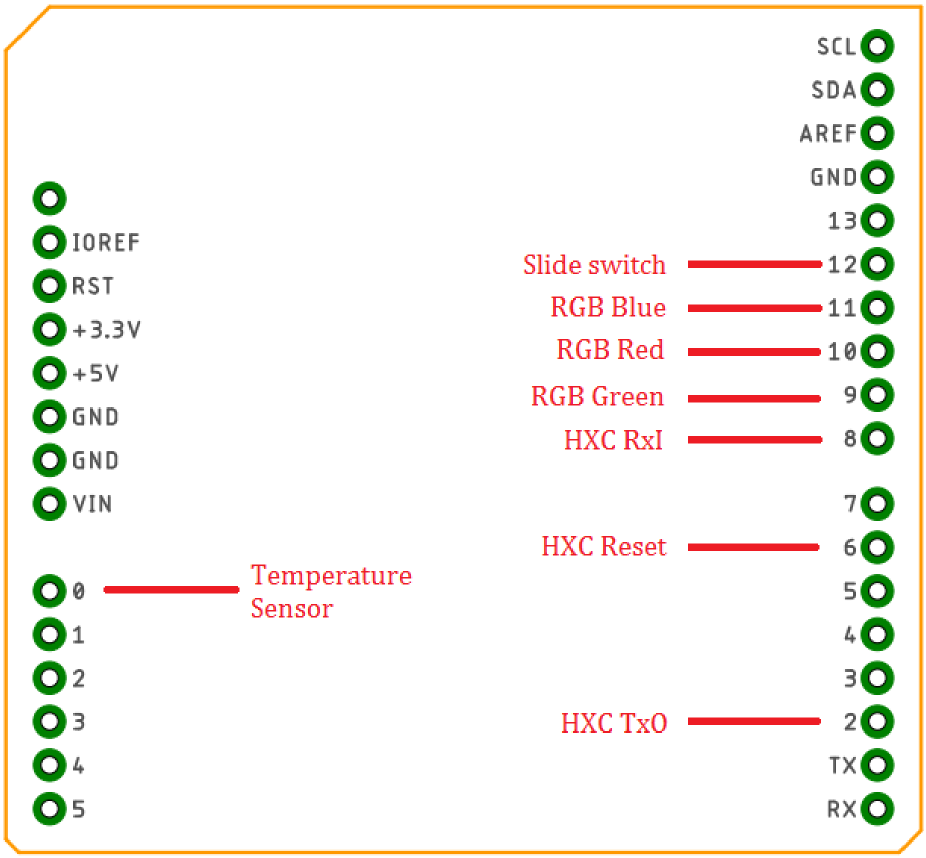


Figure 6-1: HXC Client Expansion Board Pin Layout

7 LoRa Standard Overview

7.1 Overview

This section provides a general overview of the LoRa and LoRaWAN recommendations, focusing on the LoRa end-device which is the core subject of this user manual. LoRa is a type of wireless telecommunications 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. Table 7-1 shows the LoRa classes usage definition. Refer to *Section 7.2.2* for further details on these classes.

Table 7-1 LoRa Classes

Class Name	Intended Usage
A – All	▪ Battery powered sensors or actuators with no latency constraint. ▪ Most energy efficient communication class. ▪ Supported by all devices.
C – Continuous	▪ Main powered actuators. ▪ Devices which can afford to listen continuously. ▪ No latency for downlink communication

7.2 Network Architecture

LoRaWAN network is structured in a star of starts topology, where the end-devices are connected via a single LoRa link to one gateway as shown in Figure 7-1.

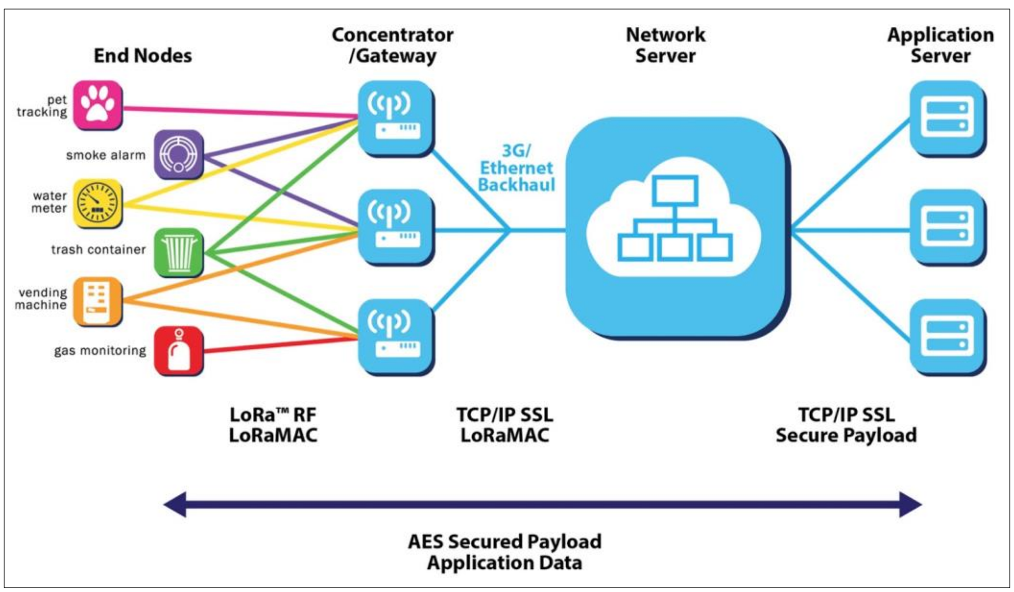


Figure 7-1 LoRaWAN Network Architecture (source: Semtech)

7.2.1 End Device Architecture

The end-device should contain an HXC Client module, which will take care of all the LoRaWAN RF communication, and an MCU to communicate with HXC and optionally any sensor drivers if there are any.

7.2.2 End Device Classes

The HXC Client Module supports two classes of end-point devices, addressing the different needs reflected in the wide range of applications.

7.2.2.1 Class A: Bi-directional end devices

- Class A operation is the lowest-power end-device system.
- Each end-device uplink transmission is followed by two downlink receive windows.
- Downlink communication from the server goes down shortly after the end-device has sent an uplink transmission.

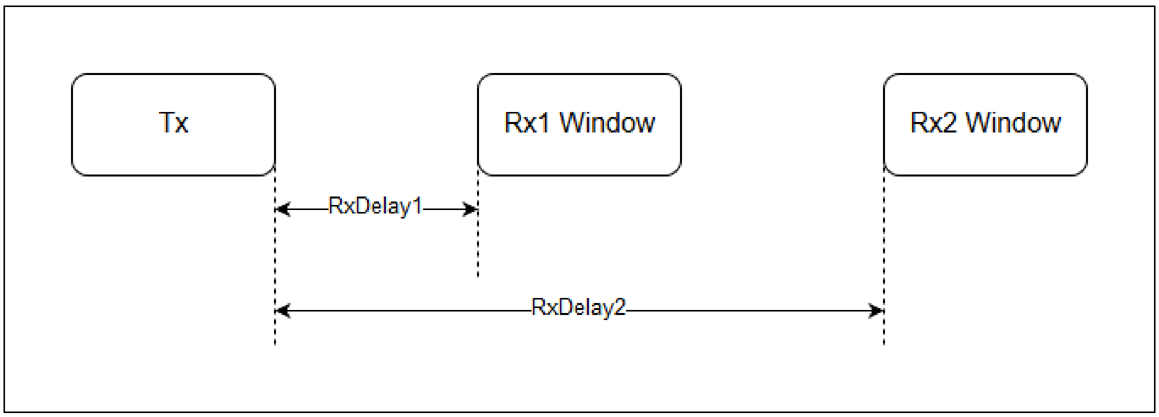


Figure 7-2 Class A Tx/Rx diagram

7.2.2.2 Class C: Bi-directional end-devices with continuous receive slots

- Class C devices are always listening hence they have large power consumption.
- Not suitable for a battery-powered device.
- The receive window is always open except during transmission.

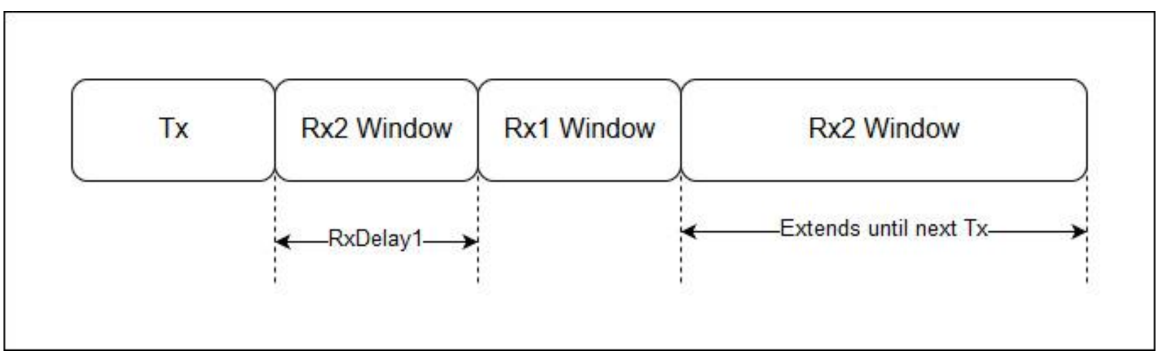


Figure 7-3 Class C Tx/Rx diagram

7.2.3 End Device Activation / Joining a Network

7.2.3.1 Over-the-air Activation (OTAA)

The OTAA 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 *AppKey*. During a joining procedure, the LoRa end-device and the application server exchange inputs to generate two session keys:

- a network session key (*NwkSKey*) for MAC commands encryption.
- an application session key (*AppSKey*) for application data encryption.

7.2.3.2 Activation by personalization (ABP)

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

7.2.4 Regional Support

There are two different models of HXC Client Module, HXC900 and HXC400, which supports 900MHz and 400MHz bands respectively. Table 7-2 shows the LoRaWAN regions supported by the HXC Client Module.

Table 7-2 HXC Client Module supported regions

HXC Client Module	LoRaWAN Region
HXC900	US915
HXC400	CN470

7.3 Message Flow

This section describes how the information flow between an end user and a network server.

7.3.1 End device activation/joining

Before communicating on the LoRaWAN network, the end-device must be associated or activated following one of the two activation methods described in section 7.2.3. Figure 7-4 shows the OTAA activation.

7.3.2 End device data communication

The end-device transmits data by one of the following methods: through a confirmed-data message method (Figure 7-5) or through an unconfirmed-data message (Figure 7-6). 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 confirmed-data, the end device should wait at least Rx1Delay to receive the acknowledgment. If the acknowledgment frame is received, then the transmission is successful, else the transmission failed. HXC Client takes eight attempts to get "ACK" from a network server.

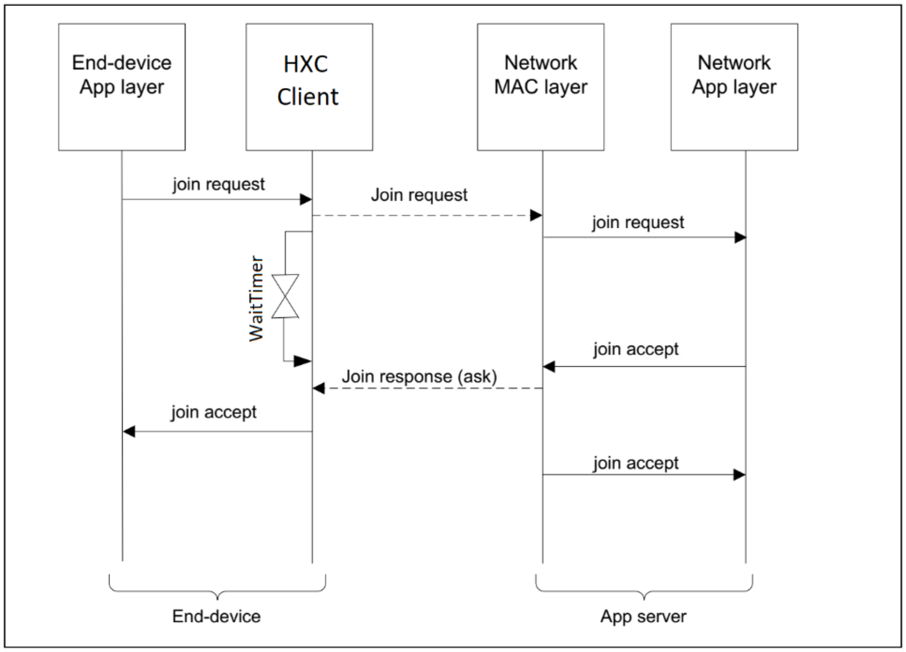


Figure 7-4 Message sequence chart for joining

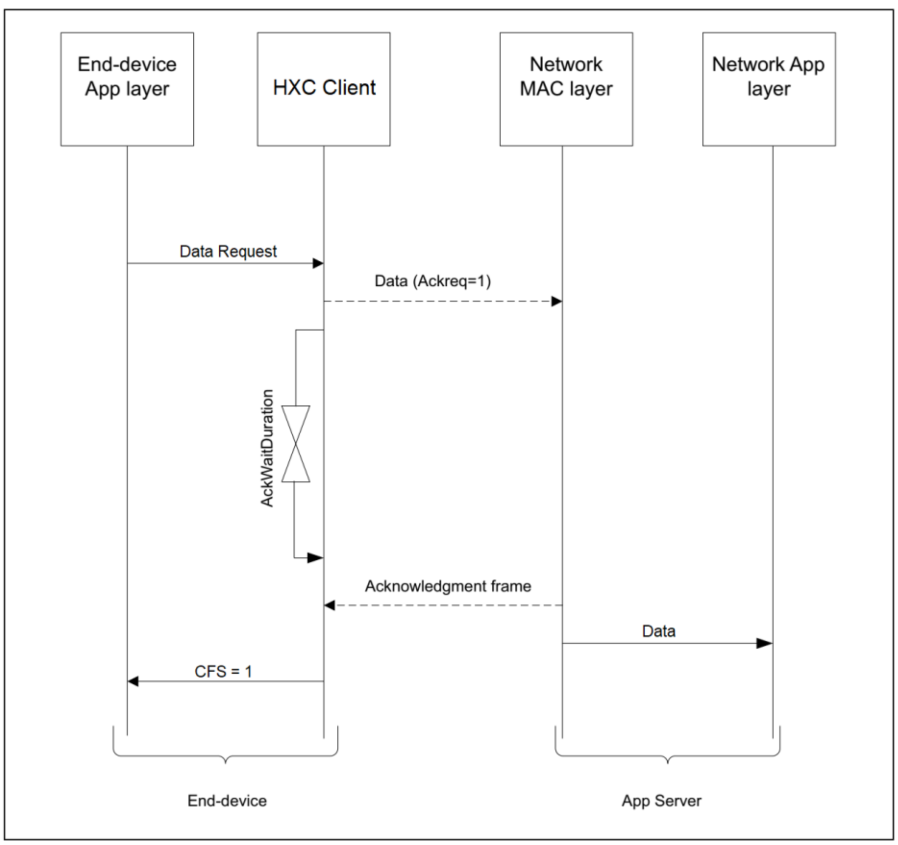


Figure 7-5 Message sequence chart for confirmed-data

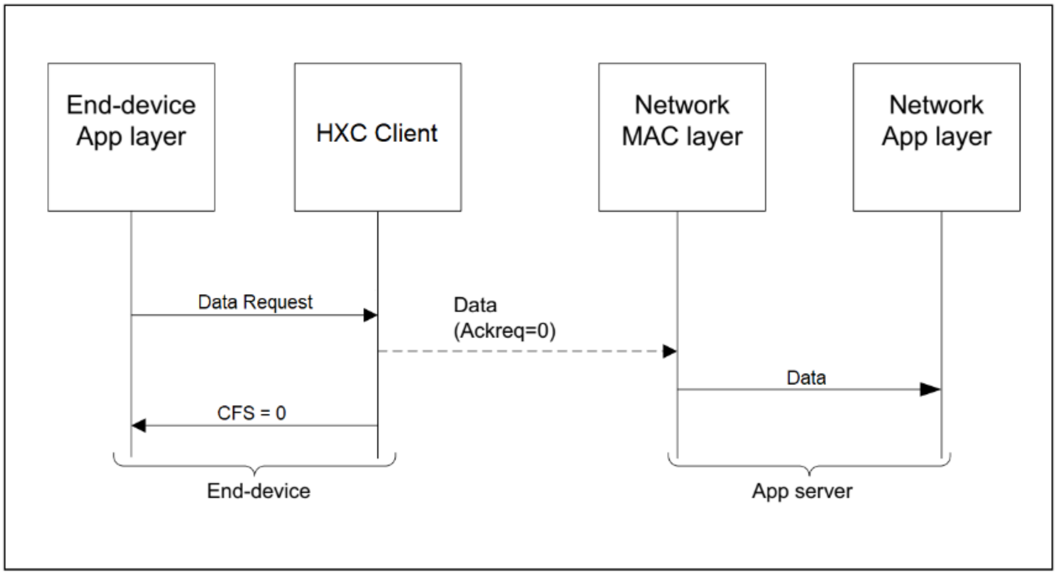


Figure 7-6 Message sequence chart for unconfirmed-data

8 Software

A comprehensive Embedded API and a demo application are available at our Mbed Repository. Chapter 4 explains how you can import the repository and flash our dev kit. The detailed explanation of our Embedded API can be found at our support site.

8.1 API Layer

There are three layers in the API:

- Application Layer: Handles the end-device application (e.g. reading sensors).
- Lora Driver Layer: Executes LoRa state machine and handles uplink and downlink communication between the application layer and PHY layer.
- PHY Layer: Handles the physical communication with the HXC Client Module.

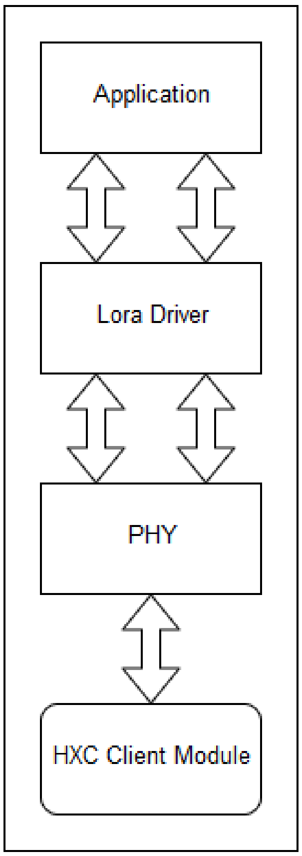


Figure 8-1 HXC Client API Layers

8.2 Project Structure

The LoRa driver module (lora_driver) implements a LoRa state machine. The HXC Client PHY module (hxc_client) communicates directly with HXC Client modem. PHY module uses STM32Cube HAL libraries. Refer to Figure 8-2 for the structure of the project files.

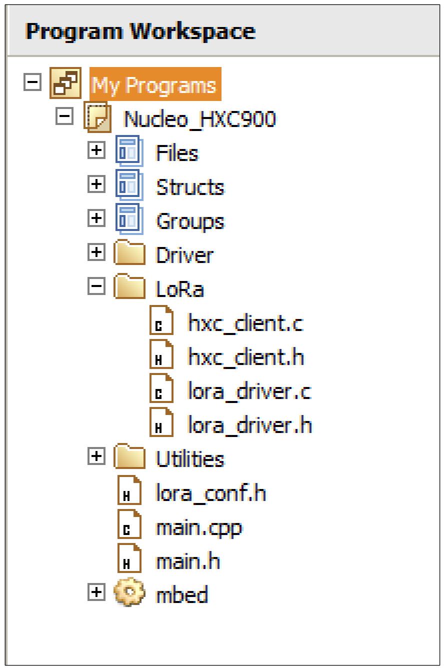


Figure 8-2 Project structure

8.3 LoRa State Machine

The state machine is implemented following best practices to develop a LoRaWAN client application node. Users can use this state machine as it is or tailor it for their own application. The best practices are listed below.

8.3.1 Best Practices to Develop LoRaWAN Client Application

- Keep the payload as small as possible and use the lowest datarate for uplinks. This will ensure low on-air time, will preserve power and will be within duty cycle restriction (if there is any). Table 8-1 and 8-2 shows the relation between payload size and datarate for HXC900 and HXC400, respectively.

Table 8-1: HXC400 Data Rate Table for Uplink

Datarate	Spreading Factor	Payload Size (Bytes)
DR0	SF12	51
DR1	SF11	51
DR2	SF10	51
DR3	SF9	115
DR4	SF8	222
DR5	SF7	222

Table 8-2: HXC900 Data Rate Table for Uplink

Datarate	Spreading Factor	Payload Size (Bytes)
DR0	SF10	11
DR1	SF9	53
DR2	SF8	125
DR3	SF7	242
DR4	SF8	242

- Use binary/hex instead of ASCII to prepare the payload. This will ensure smaller payload size. For example, one byte can represent any value between 0 to 255, whereas, an ASCII '255' will take three bytes.
- For OTAA authentication scheme, use a random delay to initiate the Join request to avoid synchronization between devices. To know more click here.
- Every client node should Re-Join occasionally, to update the security keys. The deciding factor can be a time or a certain number of uplinks. In our API, client node initiate Join request every 7-days.
- Any sorts of trigger value should be updatable thru downlink. For example, the uplink polling period, Re-Join interval etc.

8.3.2 Embedded API

Our API is implemented following all the best practices. Figure 8-3 shows the LoRa state machine execution flow.

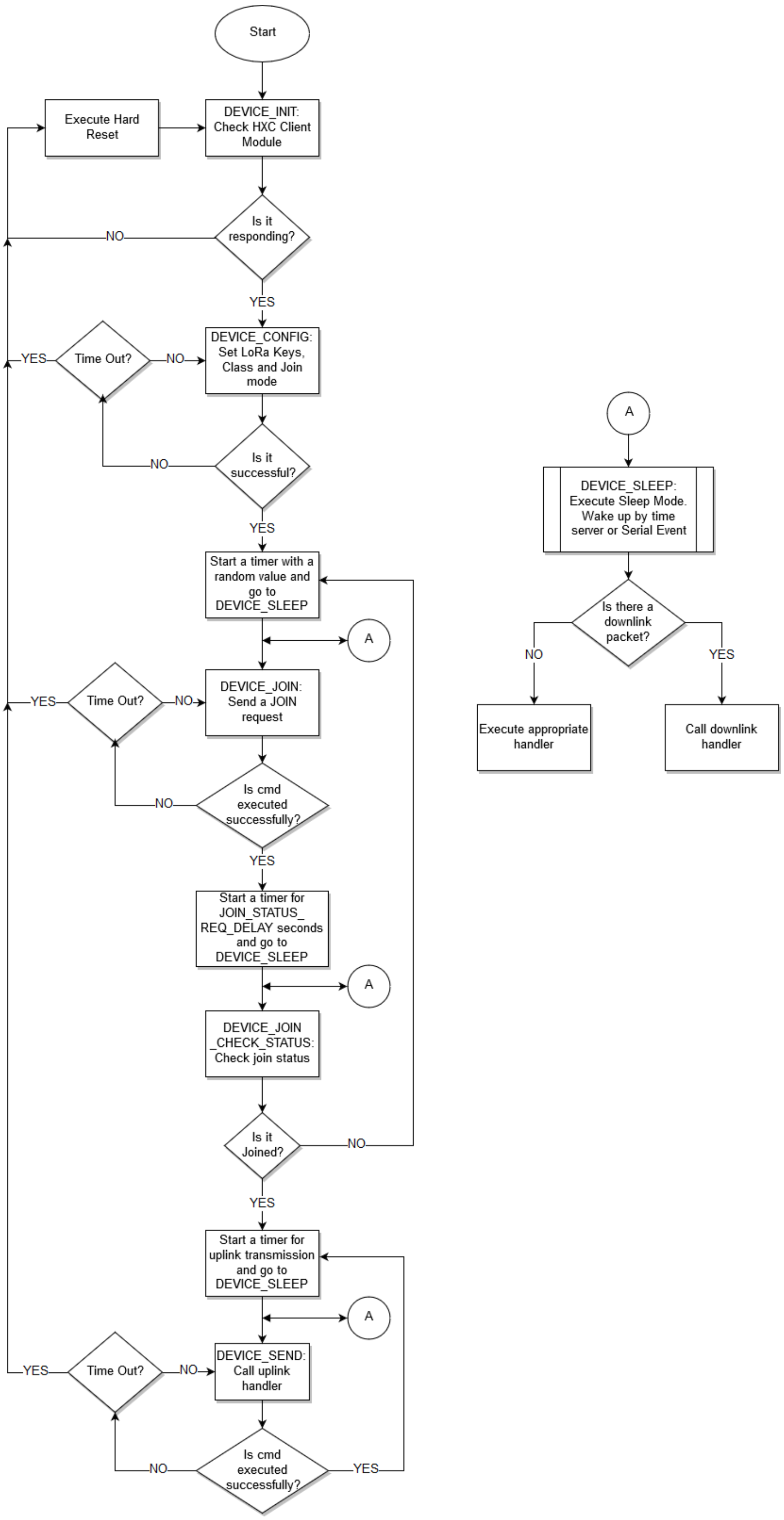


Figure 8-3 LoRa state machine flowchart

8.4 Demo application payload format

The uplink payload consists of a temperature sensor output (MCP9700) and a slide switch status. The payload consists of 3-bytes. 2-bytes for the temperature sensor and 1-byte for slide switch status.

Table 8-3: Solder bridge connection for 5V I/O

Byte #	Mnemonic	Description	Units
1	Temp_MSB	Temperature MSB	mV
2	Temp_LSB	Temperature LSB	
3	Switch_Status	Slide Switch Status	0 or 1

To convert temperature sensor output to an actual temperature user will need to use a formula on their application interface.

$$Vout = ((Temp\_MSB << 8) + Temp\_LSB) / 1000$$

$$Ta = (Vout - 0.5)/ 0.01$$

Where, Ta = Ambient Temperature in degree Celsius

8.5 How to update LoRaWAN configuration

Our Embedded API abstracted all the complexities from the users and made our client module integration a seamless experience. It cannot get easier than this.

It is very easy to change end device keys, Class, ADR capability and Join Mode. Just go to 'lora_conf.h'. Scroll down until you find `LoraConfigParam`. Switch between `OTAA` or `ABP` Join Mode, or Class A/C by updating `Class` etc.

You can also change the uplink interval by updating SENSORS_MEASURE_CYCLE.

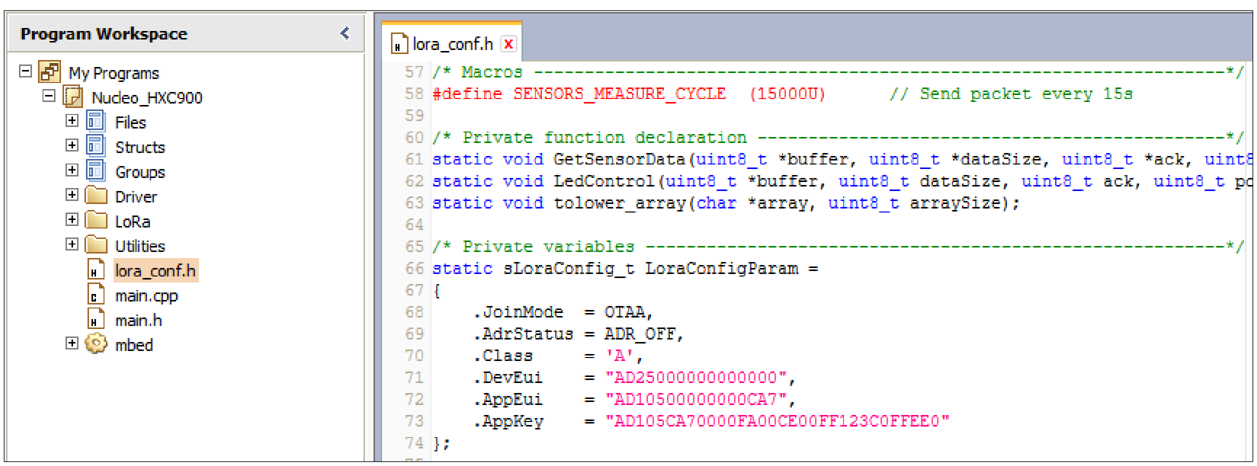


Figure 8-4: LoRaWAN Configuration

8.6 Setting up uplink and downlink

The demo application uses a state machine to schedule uplink, parse downlink and sleep in-between. Uplink and downlinks are handled by callback functions. Users can easily change these without fiddling the state machine.

```
/* Private variables ---------------------------------------
static sLoraConfig_t LoraConfigParam =
{
    .JoinMode  = OTAA,
    .AdrStatus = ADR_OFF,
    .Class     = 'A',
    .DevEui    = "AD25000000000000",
    .AppEui    = "AD10500000000CA7",
    .AppKey    = "AD105CA70000FA00CE00FF123C0FFEE0"
};

static sLoraDriverParam_t LoraDriverParam =
{
    .SensorMeasureTime  = SENSORS_MEASURE_CYCLE,
    .SendDataHandler    = GetSensorData,
    .ReceiveDataHandler = LedControl
};
```

Figure 8-5: Uplink and downlink callback functions

`SendDataHandler` and `ReceiveDataHandler` callback pointers handle the uplink and downlink, respectively. To modify `GetSensorData` and `LEDControl` go to '`lora_conf.h`' and scroll until you reach `GetSensorData` function.

```
/****************************************************************************
 * @Brief : Uplink packet handler for lora_driver
 * @Param : Pointer for payload buffer, data size, ack configuration and port
 * @Return: None
****************************************************************************/
static void GetSensorData(uint8_t *buffer, uint8_t *dataSize, uint8_t *ack, uint8_t *port)
{
    /* Prepare an unconfirmed uplink packet for port 2 */
    uint8_t size = 0;

    // Converts and read the analog input value (value from 0.0 to 1.0)
    float temperatureValue = temperatureSensor.read();
    // Change the value to be in the 0 to 3300 range
    uint16_t temperatureValueInt = temperatureValue * 3300;
    // Checkout our user manual to convert temperatureValueInt into degree Celcius

    buffer[size++] = (temperatureValueInt >> 8) & 0xFF;
    buffer[size++] = temperatureValueInt & 0xFF;
    buffer[size++] = slideSwitch;

    *dataSize = size;
    *ack = (uint8_t)UNCONFIRMED;
    *port = 2;
}
```

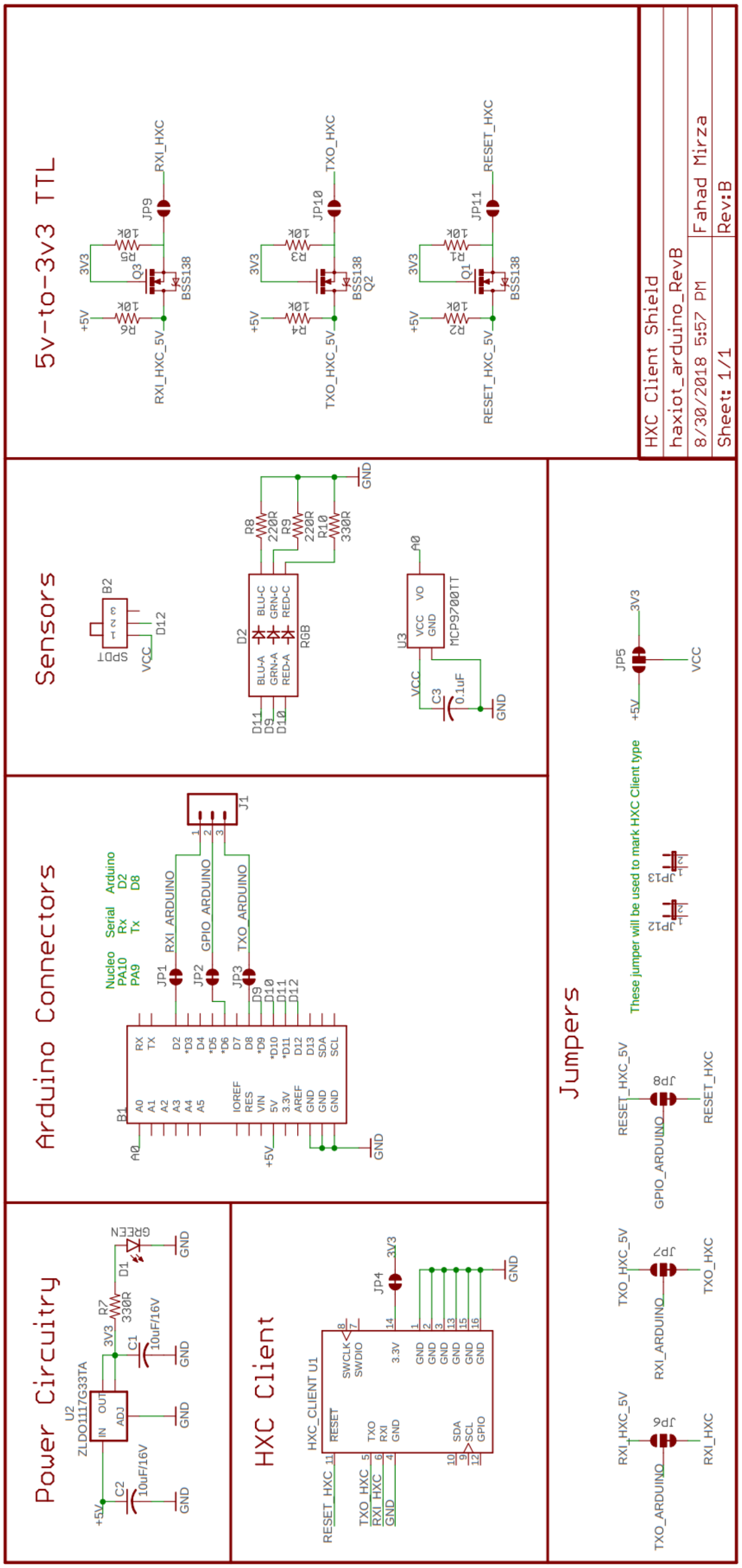
Figure 8-6: Uplink callback function

```
/******************************************************************************
 * @Brief : Downlink packet handler for lora_driver
 *          Valid downlink messages are: 'red', 'green', 'blue' and 'off'
 * @Param : Payload buffer, data size, ack configuration and port
 * @Return: None
******************************************************************************/
static void LedControl(uint8_t *buffer, uint8_t dataSize, uint8_t ack, uint8_t port)
{
    tolower_array((char *)buffer, dataSize);

    if(strncmp("red", (const char *)buffer, 3) == 0)
    {
        redLED   = 1; greenLED = 0; blueLED  = 0;
    }
    else if(strncmp("green", (const char *)buffer, 5) == 0)
    {
        redLED   = 0; greenLED = 1; blueLED  = 0;
    }
    else if(strncmp("blue", (const char *)buffer, 4) == 0)
    {
        redLED   = 0; greenLED = 0; blueLED  = 1;
    }
    else if(strncmp("off", (const char *)buffer, 4) == 0)
    {
        redLED   = 0; greenLED = 0; blueLED  = 0;
    }
}
```

Figure 8-7: Downlink callback function

9 Appendix A: Electrical schematics



10 Appendix B: Document Information

10.1 Version History

Version	Date	Author	Description
V0.01	08/25/18	Fahad Mirza	• Initial Release
V1.00	08/31/18	Fahad Mirza	• Added more images
V1.01	09/03/18	Fahad Mirza	• Added a Software Chapter
V1.02	10/19/18	Fahad Mirza	• Added Chapter 7 LoRaWAN Overview • Added explanation about the API layer • Added best practices to develop LoRaWAN client application • Updated images according to updated API

10.2 List of Abbreviations

Acronym	Definition
LoRa®	Long range radio modulation scheme
LoRaWAN™	LoRa® wide-area network protocol
RF	Radio frequency
OTAA	Over-the-air Activation
ABP	Activation by personalization
NS	Network Server