Open Innovation Platform for IoT-Big data in Sub-Sahara Africa

Grant Agreement Nº 687607

Agriculture MVP

Low-Cost Weather station with Lora

Responsible Editor:	Unparallel Innovation, Lda
Contributors:	
Document Reference:	Agriculture MVP: Low-Cost Weather station with Lora
Distribution:	Confidential/Public/Restricted
Version:	1.2
Date:	January, 17th , 2017

Table of Contents

LIST OF FIGURES

LIST OF TABLES

1. RELEASE NOTES & KNOW ISSUES

1.1. What's New

- Added this chapter 1 with the release notes and also the know-issues detected that will resolved in the near future.
- Resizing and adjusting in some figures in order to make this document better structured.
- Added some missing references of the figures in the text.
- Added new chapter Adafruit RTC5 that describes the Adafruit DS3231 RTC that needed to be added to the implementation. This chapter contains information and related links about the DS3231 RTC.
- Change in the default conversion values for rainfall and wind speed. Specifically change from inches to millimetres (for rain) and miles per hour to kilometres per hour (for wind speed). These changes were also added to the firmware's functions that returns wind speed, gust and rainfall.
- Change the length of time that the system falls asleep. From 4 minutes to 10 minutes by default.
- Added the necessary information to table 5 in order to allow to connecting the DS3231 RTC to the implementation made until this moment.
- Change to new figures that already have the DS3231 RTC in the implementation.
- Added the new "RTCLib" library in to the firmware to allow the access and communication with the DS3231 RTC.
- System running update with the new specifications.
- Information's update about the previous tests.
- Added new tests, with their respective conclusions, using the RTC implementation.

1.2. Know Issues

- So far, the system is not yet autonomous. As such it is intended to use a solar panel.
- The system is dependent on the amount of rain and wind. With larger amounts of wind and rain the system will consume more energy and will have less durability.
- Some sensors are attached to the Weather Shield's PCB. It Can be a problem because the acquired measurements may not be as accurate as possible. In a future version it is intended to resort to the implementation of external sensors.
- So far, the entire software solution occupies all the memory that allows efficient system operation (72% of full memory load). It is intended to resort in the next weather station version other board with more memory capacity.

2. INTRODUCTION

This document aims to describe and detail the development of a low cost weather station, based on the Adafruit Feather board. This board communicates with any LoRa intermediate gateway node. It describes the creation of a vigilant weather supervision, composed by multi-tech sensors as part of the IoT sensor node with LoRa network integration. This work was done in the context of WAZIUP Research Project, which has received funding from the European Union's H2020 Programme for research, technological development and demonstration under grant agreement No 687607.

With expert level architecture, the challenge here is to be able to gather data from the surrounding area and then make it available to the WAZIUP cloud. The transmission is carried out by the LoRa bi-directional communication, representing a low cost technologic long range communication. Also challenging is to provide a final sensor node as a light and scalable solution with low energy dependencies.

Nowadays there are many advanced and well-integrated weather stations. However, they were discarded for use in WAZIUP due to the need of having an IoT solution that could be qualified as "low-cost" and with built-in LoRa radio. So, and in order to decrease costs, we decided to use several low-cost components and integrate them to provide a WAZIUP Weather Station node. This node, was developed using the requirements received from the Project Use cases.

This document provides a step-by-step tutorial, a simple user guide, on how built the WAZIUP Weather Station). The code used for this weather station is open source and hosted in the GitHub repository [1].

[1] https://github.com/bmpalmeida/UI_Waziup_Weather_Station/

3. MAIN ARCHITECTURE

The objective is to develop a low-cost and sustainable solution capable of reading real-time data typical of a weather station, using different sensors, and capable of communicating via LoRa. After a review of all known hardware available on the market, all the components strictly necessary to the solution were defined, which in turn fulfilled the requirements of the above: The hardware chosen was:

- Adafruit Feather 32u4 RFM95 LoRa Radio with female pin headers[2]
- Adafruit RTC DS3231[3]
- Sparkfun Weather Shield with RJ11 female connectors[4]
- Wind and Rain sensors kit[5]
- Antenna 868 Mhz and SMA cable

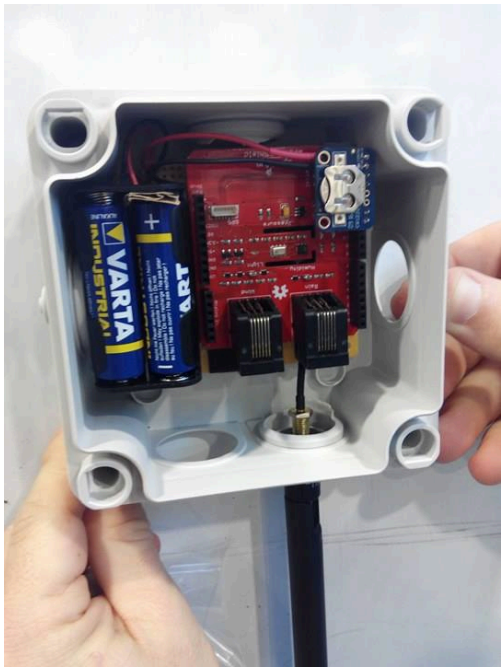

Figure 1 – Weather Station v1.0 – IoT solution for the Agriculture MVP.

In Figure 2, it is depicted the working model of the data acquisition and supervision system, constituted by the previously mentioned hardware, together with software capable of monitoring and supervising the sensory variables and the devices.

[2] https://www.adafruit.com/product/3078
[3] https://www.adafruit.com/product/3013
[4] https://www.sparkfun.com/products/12081
[5] https://www.sparkfun.com/products/8942

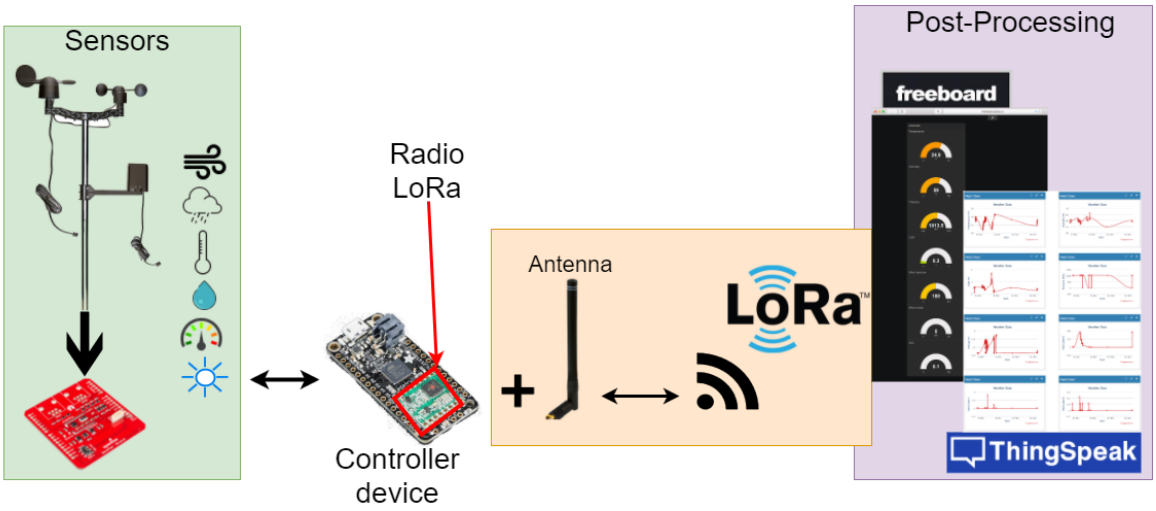

Figure 2 – Overview of the overall solution for the Agriculture MVP

3.1. Core System controller

In terms of the core system within the Weather Station solution, it is composed by the Feather32u4, which that takes a specialized role in the system where it performs control functions through software, with processing power, enabling the sensory devices to gather data from the environment, using specific libraries. This system also has built-in communication capabilities.

3.2. Data Acquisition

The weather shield is an integrated module with several built-in sensors capable of collecting data, such as temperature, humidity, luminosity, barometric pressure and altitude. Along these sensors, the weather shield also enables the integration of three more different sensors to collect data regarding wind direction, wind speed and amount of rain.

Based on the proposed model, it becomes clear the connection between the controller and the Weather Shield. This connection is established with the I2C protocol that allow this digital integrated circuit to communicate with one or more masters. It is used this type of protocol because it's only intended short distance communications within a single device and only requires two signals to exchange information.

The software controller uses the library "Wire.h" that is dedicated to the I2C logic protocol. The embedded software requires the “WeatherConfig.h” library in order to call all the functions responsible for activating and reading the sensors signals coupled to the weather shield. This library also contains some logical functions dedicated to executing the desired specifications.

Until now, the communication system build between Adafruit Feather (ATmega 32u4) and all the sensors who take measurements (include WeatherStation library).

3.3. Data Communication

Like as expected in the proposed system model, the controller will send data to the outstation, based on the information collected from the weather shield module. For this it makes use of the "SPI.h" library to run the communication with the radio module RFM9x LoRa 868/915. The LoRa radio must communicate with the LoRa gateway, specified by the system, and for that will interact with the "featherLora.h" library. This library was created because of the Adafruit Feather's specifications and was originally developed by Congduc.

The data will be collected according to the time windows described, already considered in the project. At the end of each time window will be sent the package with the message containing the information collected.

In data sharing with outstation it was established to send an acknowledge information packet like a result of the incoming data from the different sensors. The typical message to be sent from one gateway to another is based on the type message as described in the following example:

Example of the message send in the package:

\\!TC/18.6/HU/85/LU/56/DO/7.7/AZO/87/WD/90/WC/5.55/RC/0.55

The following table outlines the type and content of the information sent in each package:

Table 1 - Communication package.

Communication package	
TC/18.6/	Temperature
HU/85/	Humidity
LU/56/	Luminosity
DO/7.7/	Pressor
AZO/87/	Supply
WD/90/	Wind direction
WC/5.55/	Wind Speed
RC/0.55/	Amount of rain

In that way, the LoRa gateway receivers this message by post-processing python scripts to be able to correctly and sequentially inspect all the information, in order to post in interface

platform the values to be appreciated. Off course some parts need your own customization for cloud access.

The following example shows the result, for a short period of time, from weather station in the Cloud platform:

Figure 3 – Represented the ThingSpeak Cloud platform.[6]

[6] https://thingspeak.com/channels/184796

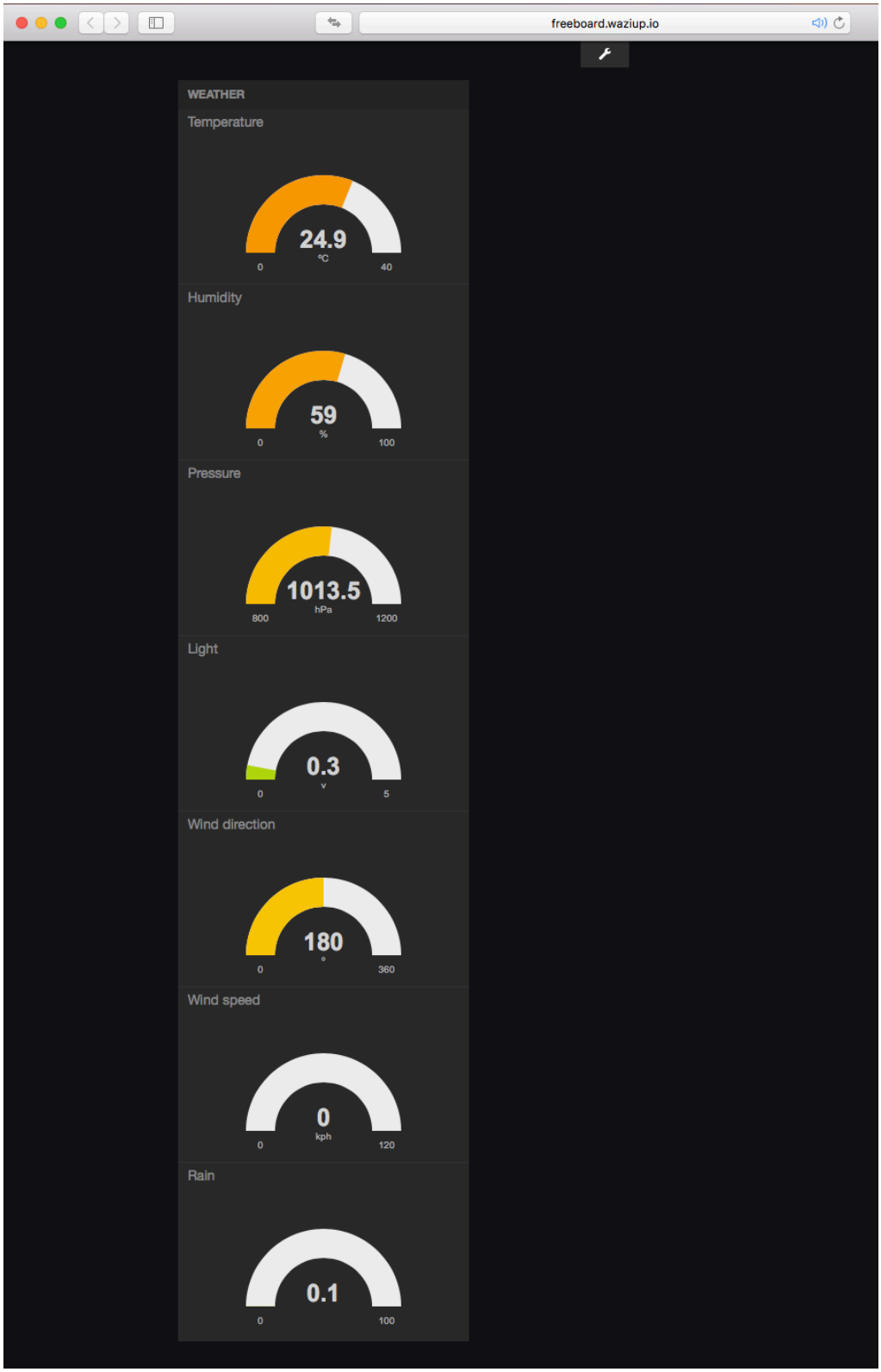

Figure 4 – Represented the Freeboard Cloud platform.[7]

7

http://freeboard.waziup.io/index.html#source=http://thingproxy.freeboard.io/fetch/https://www.dropbox.com/s/4wrwte3qqaevmxb/UI_WEATHER.json?dl=1

So far, the LoRa communication between Feather (with RF95 integrated) and Raspberry PI (integrated with Hope RFM9X LoRa Radio) with the "FeatherLib" library works with a success of 100%. To develop and building LoRa end-devices you can follow Congduc's tutorials.[8] [9]

The figure shows the architecture of proposed model and it's operating flow with the set specifications:

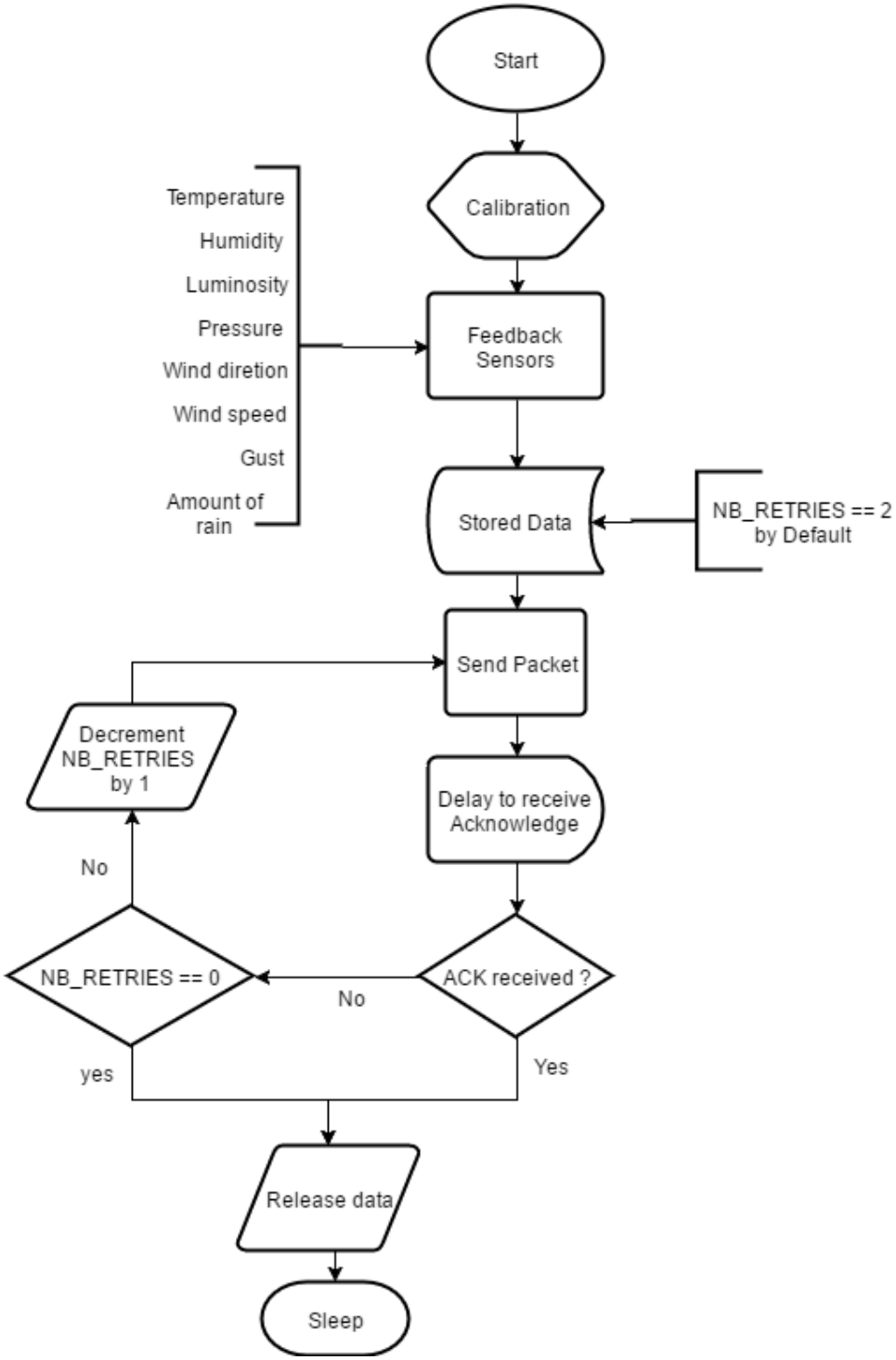

Figure 5 – Architecture model with the set specifications.

[8] http://cpham.perso.univ-pau.fr/LORA/LoRaDevices.html
[9] https://github.com/CongducPham/tutorials

It is important to understand how the LoRa communication protocol is constituted. It starts by sending a transmission with the master. Next, an acknowledge is expected. We can choose how many times the Feather tries to receive the ACK by the gateway (NB_RETRIES=2 by default). Only then the Slave fall asleep.

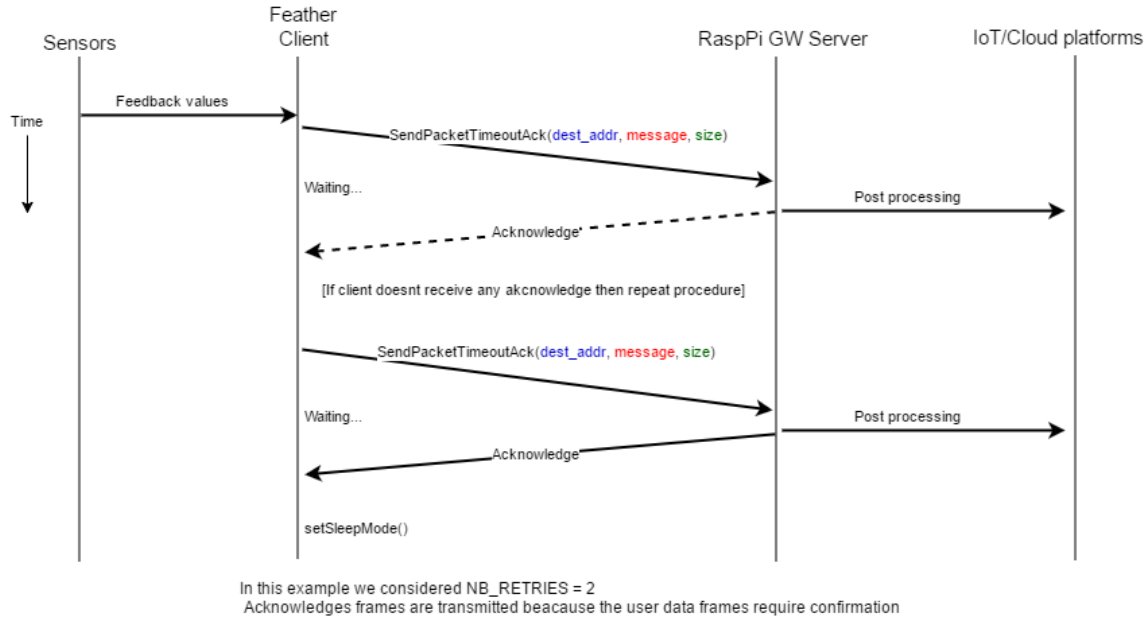

Figure 6 – Operation flow in LoRa communication protocol.

3.4. Particular Specifications

After some analysis about the requirements and implications to be had in this system, some reserved variables containing useful parameters for the developed model were verified.

- We can choose how long the feather sleep between two communications (four minutes by default).
- We can choose to show in console the information about what is happening on communication in real time (warning: increase high load memory).
- When somehow the wires in the circuit were built wrong it returns to warning with a message "I2C communication to sensors is not working. Check solder connections "and red led from Feather device start blink.

4. ADAFRUIT FEATHER

The following Figure 7 shows the Adafruit Feather 32u4 LoRa Radio (RFM9x) that is a development board from Adafruit. It's a portable microcontroller with new standards, great for communications in networks without high power requirements of WI-FI, it has a "Long Range" (LoRa) packet radio transceiver built in USB and battery charging.

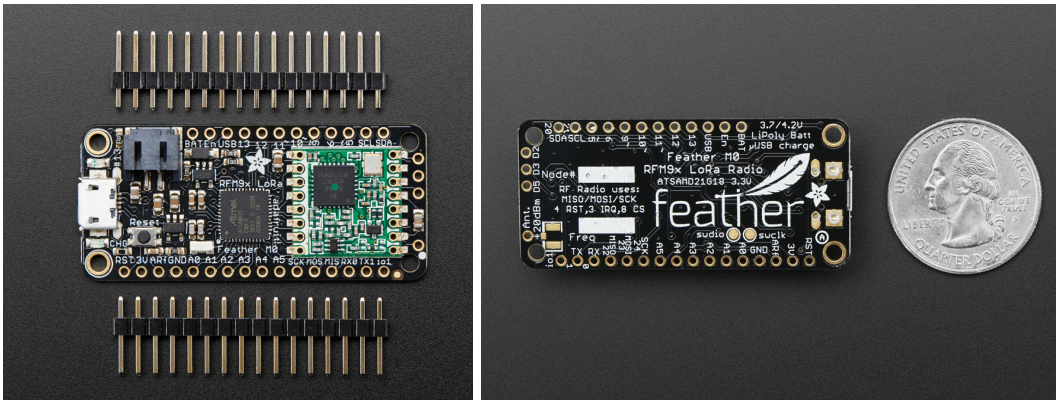

Figure 7 – Adafruit Feather 32u4.[10]

The 900 MHz radio version can be used for either 868MHz or 915MHz transmission/reception. The exact Radio Frequency is determined when you load the software. The Feather 32u4 have a ATmega32u4 processor clocked at 8 MHz with a 3.3V logic, 32K of flash and 2K of RAM, with built in USB. It have a USB-to-Serial program & debug capability built in with no need for an FTDI-like chip.

To make it easy to use for portable projects was added a connector for any of 3.7V Lithium polymer batteries and built in battery charging. Don't need a battery because it will run just fine straight from the micro USB connector. With a battery we can take it on the go, then plug in the USB to recharge. The Feather will automatically switch over to USB power when its available. We also tied the battery thru a divider to an Analog Pin, so we can measure and monitor the battery voltage to detect when you need a recharge. In the following table you can find the Feather 32u4's specifications:

Table 2 - Feather 32u4's Specifications.

Feather 32u4's Specifications	
Measures	51mm x 23mm x 8mm
Weight	5.5 grams

[10] https://www.adafruit.com/product/3078

Microcontrollers	ATmega32u4 @ 8MHz with 3.3V logic/power
Supply	3.3V regulator with 500mA peak current output
USB support	Comes with USB bootloader and serial port debugging
GPIO pins	x20
Hardware	Serial support, I2C support, SPI support
PWM pins	x80
Analog inputs	x10
Reset button	Yes

This Feather uses a RFM9x LoRa 868/915 MHz radio module. These radios are not good for transmitting audio or video, but they do work quite well for small data packet transmission when you need more range than 2.4 GHz. That provides ultra-long range spread spectrum communication and high interference immunity whilst minimising current consumption.

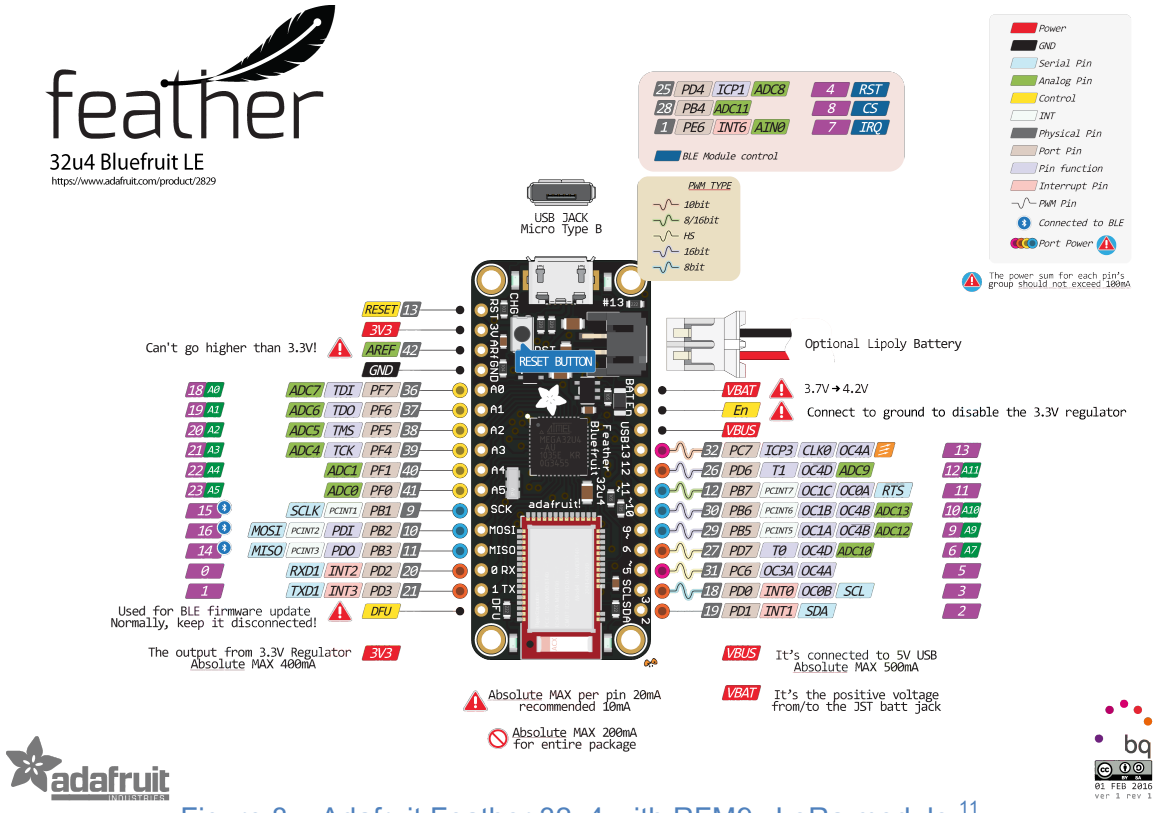

Figure 8 – Adafruit Feather 32u4 with RFM9x LoRa module.[11]

[11] https://www.adafruit.com/product/3078

5. ADAFRUIT RTC

This is a great battery-backed Real Time Clock that helps in microcontroller projects to keep track of time even if it is reprogrammed, or if the power is lost. A real time clock is basically just like a watch: it runs on a battery and keeps time for you even when there is a power outage.

Using an RTC, you can keep track of long timelines, even if you reprogram your microcontroller or disconnect it from USB or a power plug. Most microcontrollers, including the Arduino, have a built-in timekeeper called millis() and there are also timers built into the chip that can keep track of longer time periods like minutes or days. The biggest reason for using a RTC is because millis() only keeps track of time since the Arduino was last powered.

That means that when the power is turned on, the millisecond timer is set back to 0. The Arduino doesn't know that it's 'Tuesday' or 'March 8th', all it can tell is 'It's been 14,000 milliseconds since I was last turned on'.

So what if you wanted to set the time on the Arduino? You'd have to program in the date and time and you could have it count from that point on. But if it lost power, you'd have to reset the time. Much like very cheap alarm clocks: every time they lose power they blink 12:00.

While this sort of basic timekeeping is OK for some projects, some projects such as data-loggers, clocks, etc will need to have consistent timekeeping that doesn't reset when the Arduino battery dies or is reprogrammed. Thus, we include a separate RTC! The RTC chip is a specialized chip that just keeps track of time. It can count leap-years and knows how many days are in a month.

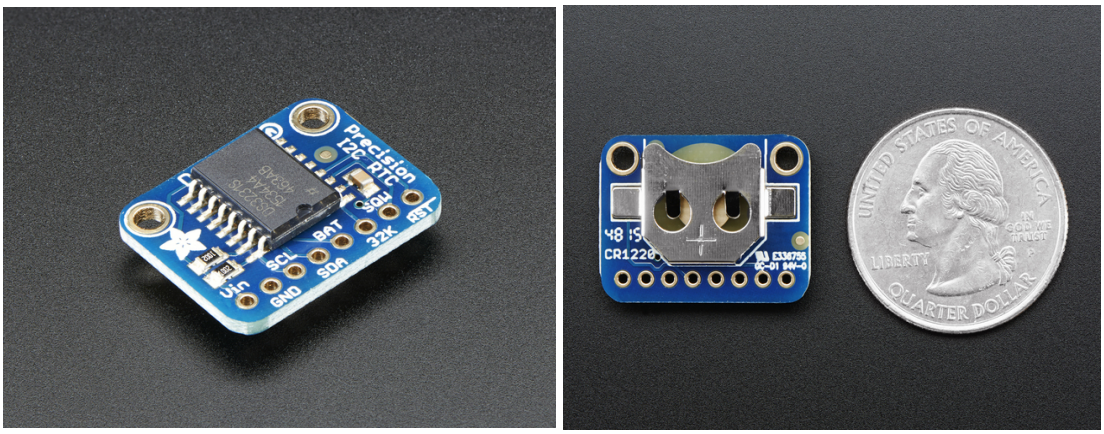

Figure 9 – Adafruit DS3231 RTC.[12]

[12] https://www.adafruit.com/product/3013

Most RTC's use an external 32kHz timing crystal that is used to keep time with low current draw. And that's all well and good, but those crystals have slight drift, particularly when the temperature.

This DS3231 Adafruit RTC is in a beefy package because the crystal is inside the chip! And right next to the integrated crystal is a temperature sensor. That sensor compensates for the frequency changes by adding or removing clock ticks so that the timekeeping stays on schedule.

This is the finest RTC you can get, and become with a coin cell plugged into the back so we can get years of precision timekeeping, even when main power is lost. Great for datalogging and clocks, or anything where you need to really know the time. Comes as a fully assembled and tested breakout plus a small piece of header. In the following table you can find the Feather 32u4's specifications:

Table 3 - Adafruit DS3231 RTC Specifications.

DS3231 RTC Specifications	
Measures	23mm x 17.6mm x 7.2mm
Weight	2.1 grams
Supply	From 2.3V to 5.5V. Do not need regulator. Use the same power as the logic level of microcontroller.

6. WEATHER SHIELD

The Weather Shield was designed by the SparkFun. Is an easy to use Arduino shield that grants you access to temperature, luminosity, barometric pressure, altitude and humidity. There are also connections on this shield to sensors as wind speed, direction, rain gauge and GPS for location and super accurate timing.

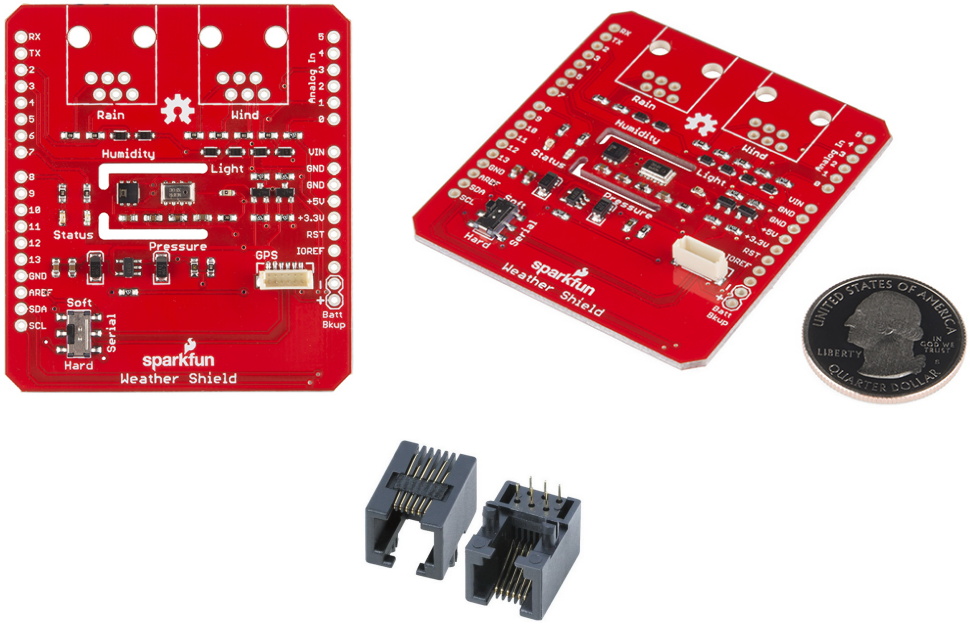

Figure 10 – Sparkfun Weather Shield.[13]

These Weather Shields utilize the “HTU21D Humidity”, “MPL3115A2 Barometric Pressure”, and “ALS-PT19” light sensors and relies on the “HTU21D” and “MPL3115A2” Arduino libraries. Each shield comes with two unpopulated RJ11 connector spaces (for hook up of rain and wind sensors). Finally, it can operate from 3.3V up to 16V and has built in voltage regulators and signal translators.

The following table shows the measurements specifications:

Table 4 - Weather shield measurements specifications.

Measurements Specifications	
Temperature	Return a float containing the temperature in degrees [-40ºC; +125ºC] and maximum error of +/-0.4ºC.
Humidity	Return a float containing the humidity as a percentage with maximum

[13] https://www.sparkfun.com/products/12081

	error of +/-3%.
Pressure	Return a float with barometric pressure in hPa (1 hPa = 100 Pa) with maximum error of +/-0,5 hPa.
Altitude	Not used in this project.
Light	Return a float with range between 0 and 3.

In the next picture you can consult the Sparkfun Weather Shield's schematic:

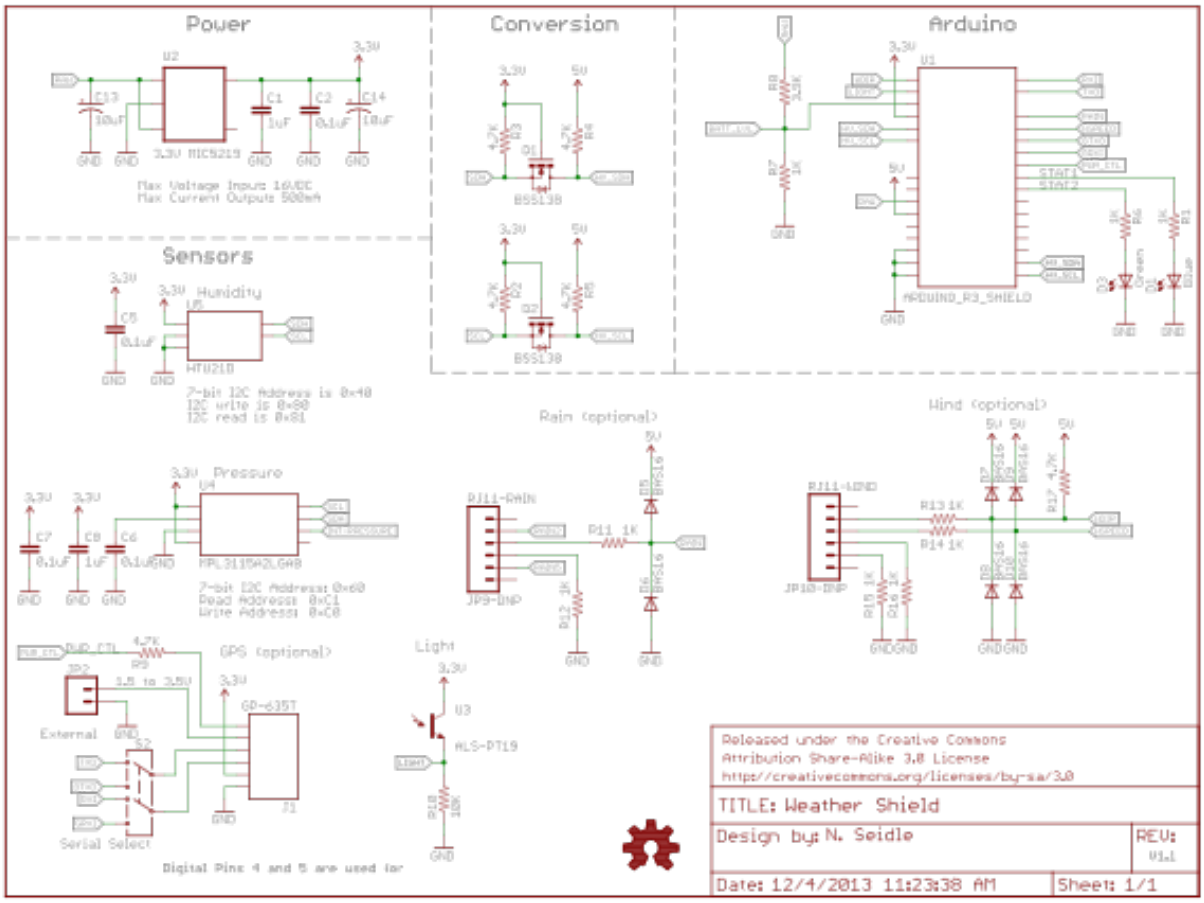

Figure 11 – Sparkfun Weather Shield's schematic.

6.1. Wind and Rain Sensors Kit

This kit from Argent Data System represents the three core components of weather measurement: wind speed, wind direction and rainfall. None of the sensors in this kit contain active electronics, instead they use sealed magnetic reed switches and magnets so we'll need to source a voltage to take any measurements.

The positive side of this is that the sensors are easy to interpret. The rain gauge is a self-emptying bucket-type rain gauge which activates a momentary button closure for each 0.2794mm (0.011") of rain that are collected. The anemometer (wind speed meter) encodes

the wind speed by simply closing a switch which each rotation. A wind speed of 2.4 m/h (1.492 MPH) produces a switch closure once per second.

Finally, the wind vane reports wind direction as a voltage which is produced by the combination of resistors inside the sensor. The vane's magnet may close two switches at once, allowing up to 16 different positions to be indicated. For more specification you can read the paper available at the link in the notes.[14]

It's necessary to pay attention to the measurements specifications of wind, gust wind and rain shown below:

- **Wind**: The sensor wind works with interrupts and return values of direction and speed in metres per hour every ten minutes (by default). The cup-type anemometer measures wind speed.
  The wind vane has eight switches, each connected to a different resistor to measures the direction.
  The ADC arrangement is shown in the diagram bellow. Cause the circuit was built for 5V supply we needed to rewrite arrangement values.

Table 5 - ADC arrangement for 5V and 3.3V.

	Angle (º)	ADC with 5Volts (Old Version)	ADC with 3.3Volts (Our version)
North	0	913	747
	22.5	680	483
	45	746	522
	67.5	393	252
West	90	414	258
	112.5	380	239
	135	508	325
	157.5	456	283
South	180	615	399
	202.5	551	368
	225	833	638
	247.5	801	617

[14] http://cpham.perso.univ-pau.fr/LORA/resources/RPIgateway.pdf

East	270	990	873
	292.5	940	776
	315	967	821
	337.5	878	689

- **Gust**: This parameter starts from sensor wind who return maximum values of wind in metres per hour every ten minutes.

- **Rain**: This sensor rain works with interrupts and return values in inches every ten minutes. The rain gauge is a self-emptying tipping bucket type. Each 0.2794 mm (0.011”) of rain causes one momentary contact closure that can be recorded with a digital counter or microcontroller interrupt input. Cause the circuit was built for 5V supply we needed to force the Pull Up.

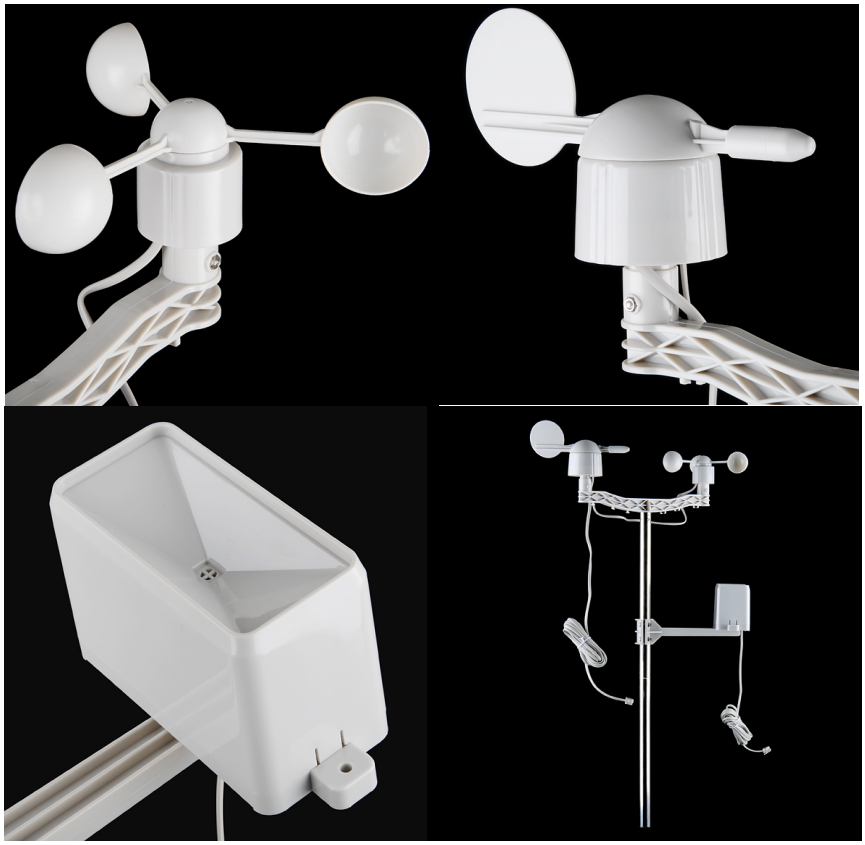

Figure 12 – Wind and Rain Sensors Kit.

7. LORA COMMUNICATION

LoRa RF it's a radio modulation format that gives significantly longer range than straight FSK modulation, a technique that compete with others technologies, specific intended for wireless battery operated Things.

Lora significantly improves the receiver and as with other spread-spectrum modulation techniques uses the entire channel bandwidth to broadcast the signal, making it robust to channel noise and insensitive to frequency offsets caused from the use of low cost crystals. The selection of the data rate is a trade-off between communication range and message duration.

The LoRa gateways are designed to use in long range start network architectures and are utilized in a LoRaWAN system. The gateways use different RF components than the end-point to enable high capacity and serve as a transparent bridge relaying messages between end-devices and a central network server in the backend.

The Gateways are connected to the network server via standard IP connections while the end-devices use single-hop wireless communication to one or many gateways.

The LoRa modulation is defined by the following basic parameters:

- The Bandwidth (BW), meaning the difference in minimum and maximum frequency;
- The Spreading Factor (SF), is a measure for the number of bits encoded per symbol;
- The Coding Rate (CR), is a measure for the amount of forward error correction;
- Preamble Length and SyncWord value;
- Whether an explicit header is sent with the message, this header contains information about the parameter of the rest of the message (payload length, the CR parameter, CRC presence);
- Presence of a 16-bit CRC and;
- The maximum packet length is 256 bytes in LoRa mode.

Instead of using a Received Signal Strength Indicator (RSSI) method to identify if a signal is present the Channel Activity Detector (CAD) is used to detect the presence of a LoRa signal.

The CAD detection has the advantage of being much faster than a typical RSSI detection and the capability to differentiate between noise and a desired LoRa signal. The CAD process requires two symbols, and if the CAD is detected, the "CAD_Detected" interrupt will be asserted and the device will stay in RX mode to receive the data payload.

Regarding software, we use "FeatherLib" library that is the base of the Congduc library to support this kind of communication. This library has some specific features to the hardware used in this case, and uses a modified and improved version of the "SX1272" library originally developed by Libelium for its "SX1272" based LoRa module.

8. BUILD & IMPLEMENTATION

Here we will show you step by step how to build a Low Cost Weather Station with LoRa communication to collect data and send them to gateway. First of all, you need to assembling the hardware, and then programming the software and tested it.

1) **Connect Feather 32u4 with DS3231 RTC and Weather Shield**:

   Try using different colours when connecting the circuits. You can use the following colours, like the next table. You have to go through some delicate but simple soldering task to attach female header pins to both devices. It's not difficult, but you have to train a little before.

Table 6 - Pin's Connection.

Pin's Connection		
Feather 32u4	**DS3231 RTC**	**Weather Shield**
3V	3V	5V
GND	GND	GND
SCL (3)	SCL	SCL
SDA (2)	SDA	SDA
12	-	7
13	-	8
Int2 (0) in Vin with a 10KΩ resistor	-	-
Int2 (0)	-	2
Int3 (1)	-	3
A0	-	A0
A1	-	A1
A2	-	A2
A3	-	A3
A4	-	Vin

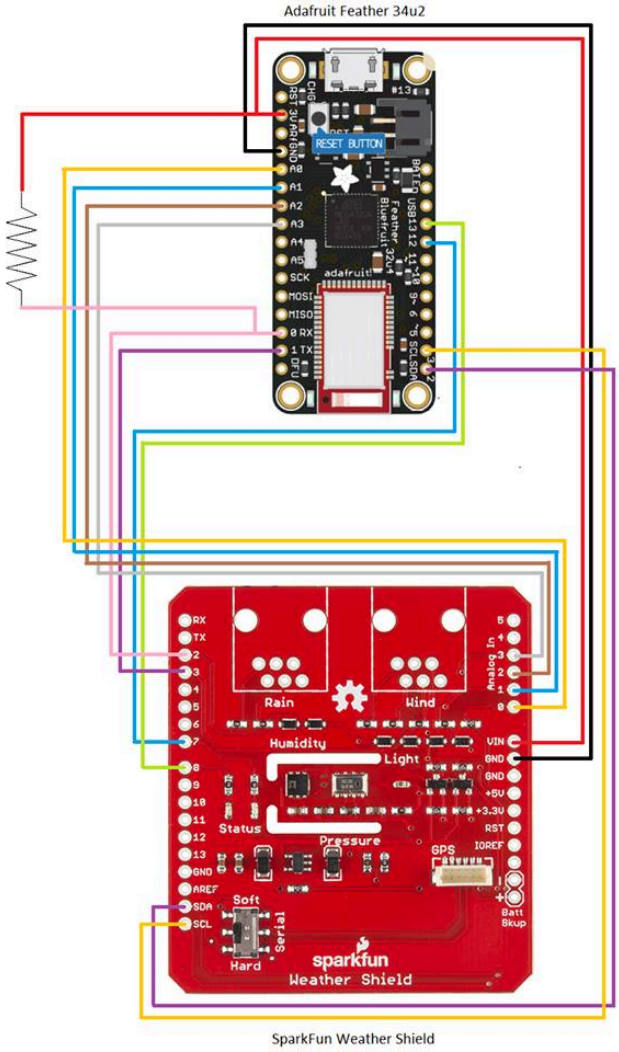

Figure 13 – Schematic with the Pin's connection between Feather and Weather Shield.

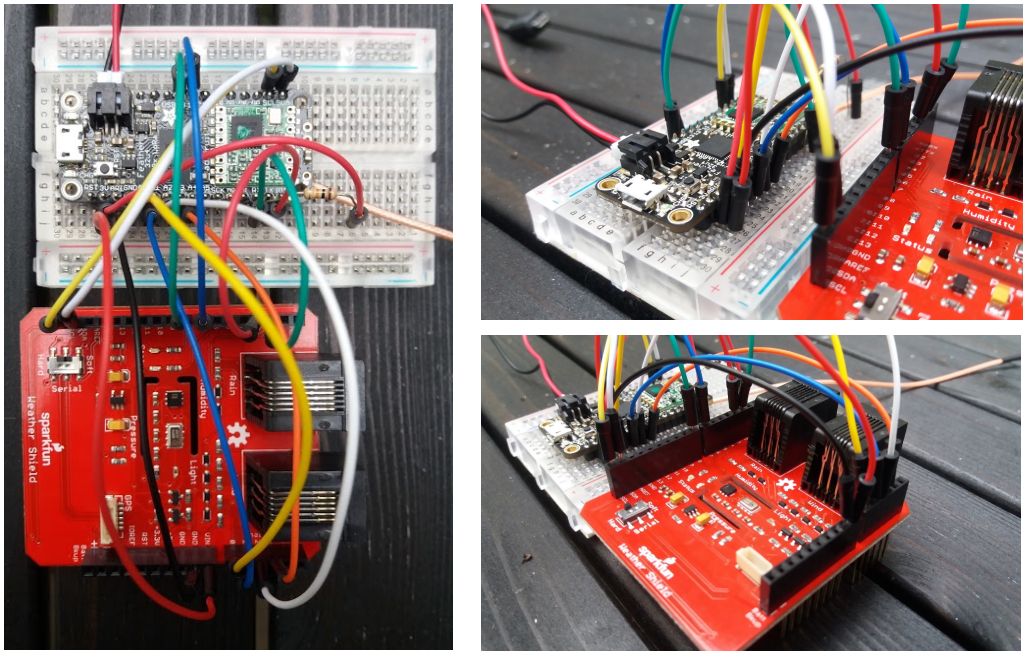

Figure 14 – Some connection's pictures in testing phase.

2) **Connect the other sensors**:

Figure 15 – Wind and Rain Sensors.

3) **Connect the Antenna 868 MHz in Feather**:

The Feather 32u4 comes fully assembled radio and tested. You will need to cut and solder the SMA plug cable in order to create your antenna.

Figure 16 – 868 MHz Antenna.

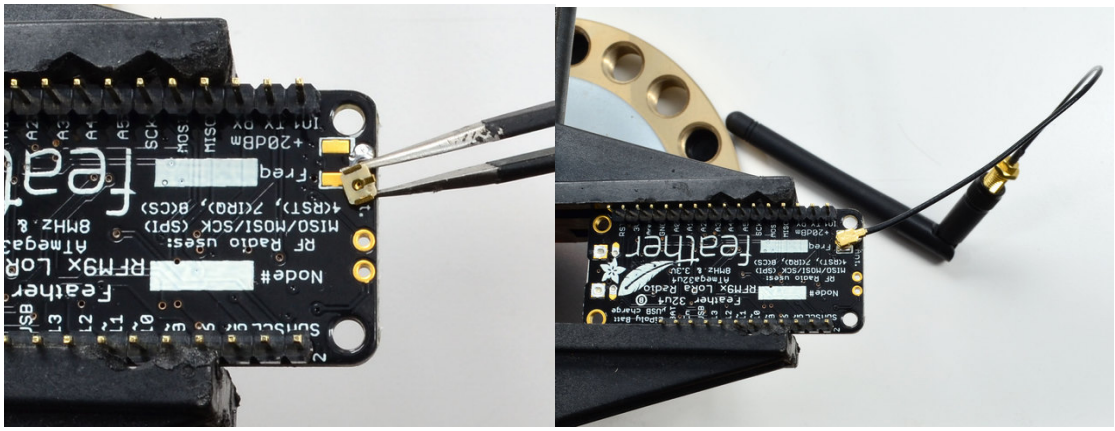

Figure 17 – Connect the 868 MHz Antenna to Feather.

4) **Weather Station built with off-the-shelves components, so far**:

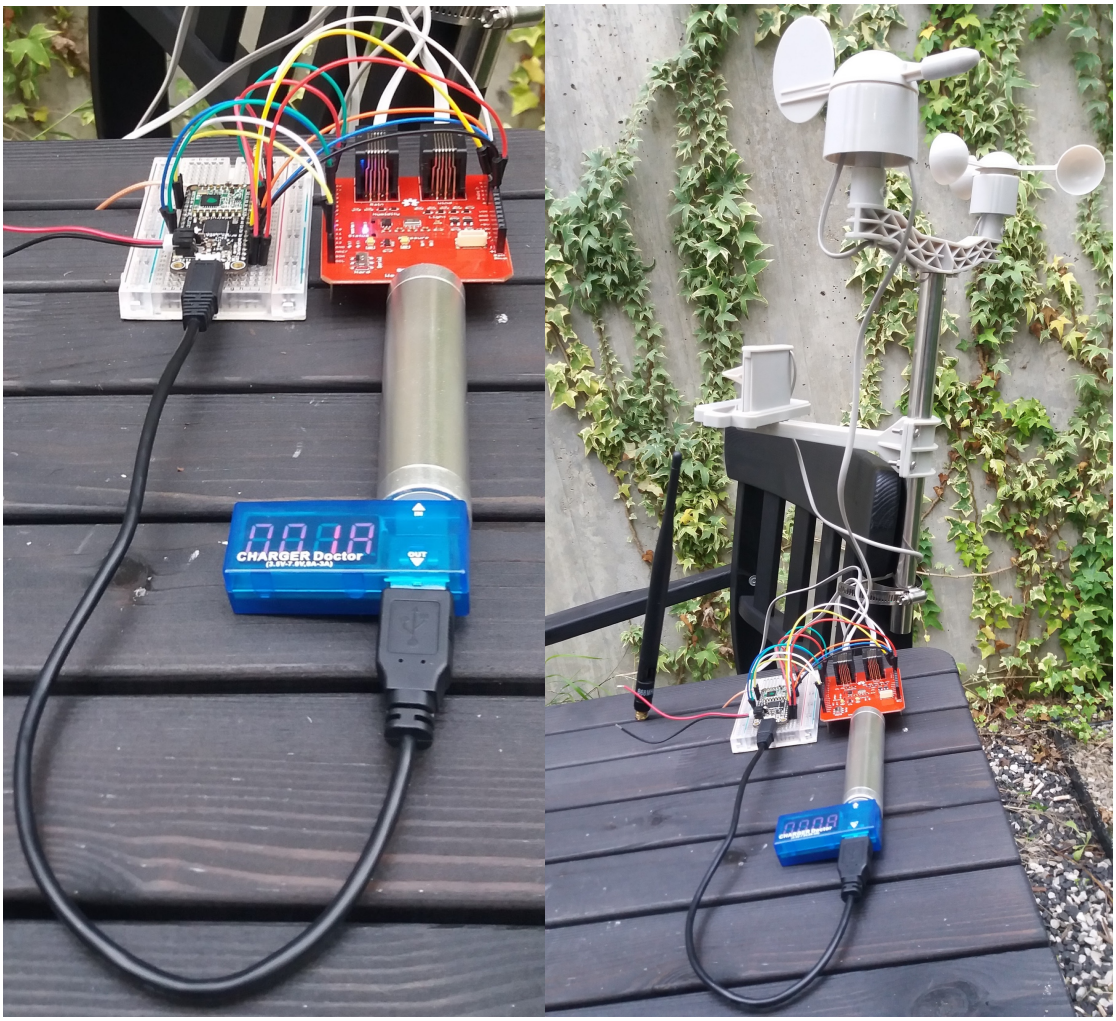

Figure 18 – Implemented Weather Station in testing phase.

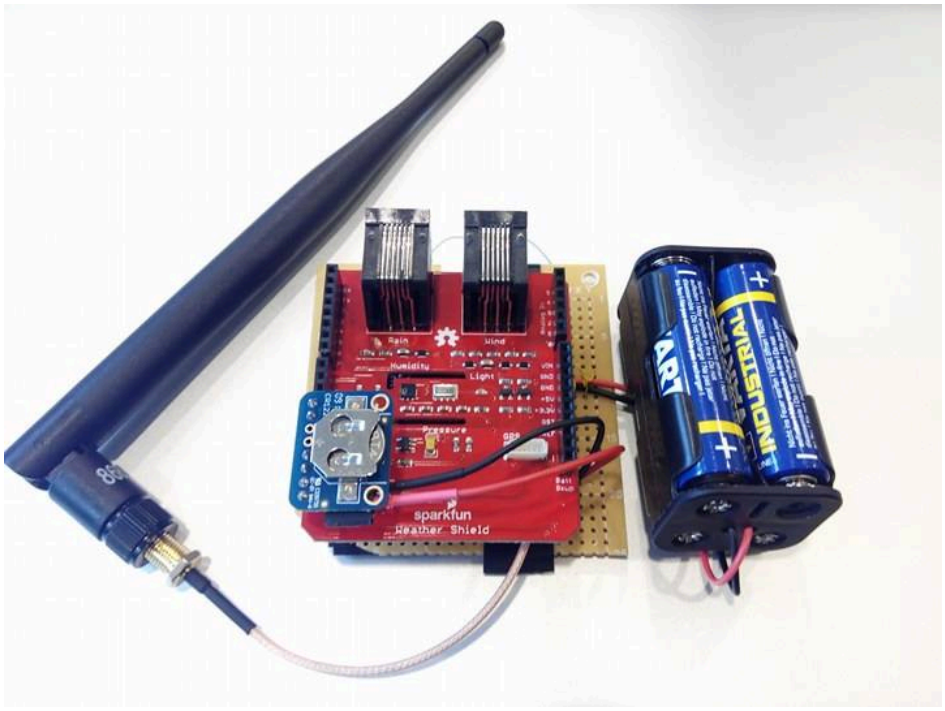

Figure 19 – Implemented Weather Station in final prototype.

5) **Download the IDE and specific software**:

First, you will need to download the Arduino IDE 1.6.6 or later. Using the Library Manager from Arduino IDE to include 'SPI.h' and 'Wire.h' for protocol communication. Found in the "Sketch" menu under 'Include Library', 'Manager Libraries...'.

When you open the 'Include Library' or even 'Library Manager' you will find a large list of Libraries ready for one-click install call. Click in that button, and the library should install automatically. When installation finishes, close the Library Manager.

Next, you'll need to get "FeatherLib", "RTCLib" and "WeatherStation" libraries from our Github.[15]
The "FeatherLib" library includes LoRa Communication and was built because of its specification; The "RTCLib" allow the interaction with the DS3231 RTC and; The "WeatherConfig" library includes all necessary sensor control functions. For these libraries you will need to install by 'add .ZIP Library ...

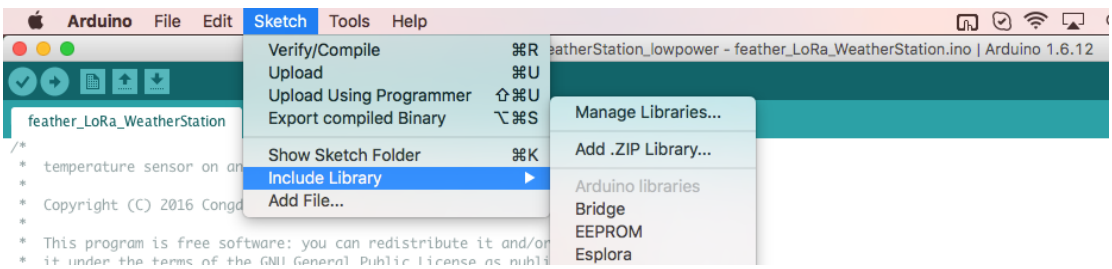

Figure 20 – Step to include the Libraries.

[15] https://github.com/unparallel-innovation/UI_Waziup_Weather_Station

6) **Compiling the software**:

Write a script that include both of these libraries and “SPI” and “I2C” libraries. In Github you will find an example of this project.

Copy the “Feather_Lora_WeatherStation_LowPower_RTC.ino” file into your “sketch” folder. This archive contains the latest sketch (firmware) that the USB Feather board uses to sample the sensors and output data with Lora communication.

You can use this sketch as is, modify it to suit your own purposes, or write your own sketches. Connect the USB end to your computer and the USB port should be detected in the Arduino IDE. For that follow the same steps of the next picture:

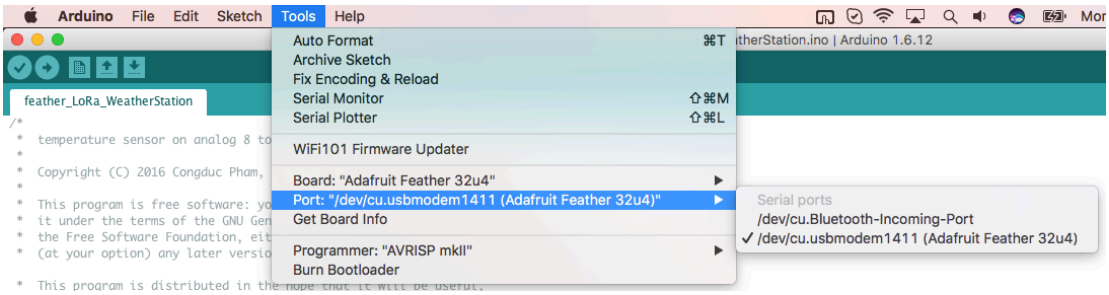

Figure 21 – Steps to include the Libraries.

Then, click on the «verify» button. After it says “Done compiling” Select the serial port for your device. It may have another name than what is shown in the example. Then click on the «upload» button.

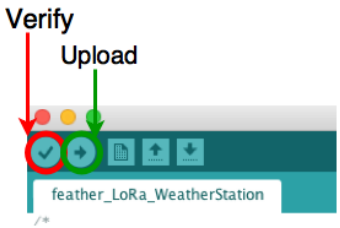

Figure 22 – Steps to Verify and Upload the software.

Then, click on the «verify» button. After it says “Done compiling” Select the serial port for your device. It may have another name than what is shown in the example. Then click on the «upload» button.

7) **Serial Monitor**:

You can see the sensors output and LoRa communication state if you use serial communication. Use the Arduino IDE «serial monitor» to get such output, just to verify that the sensor is running fine. Be sure to use 38400 bauds.

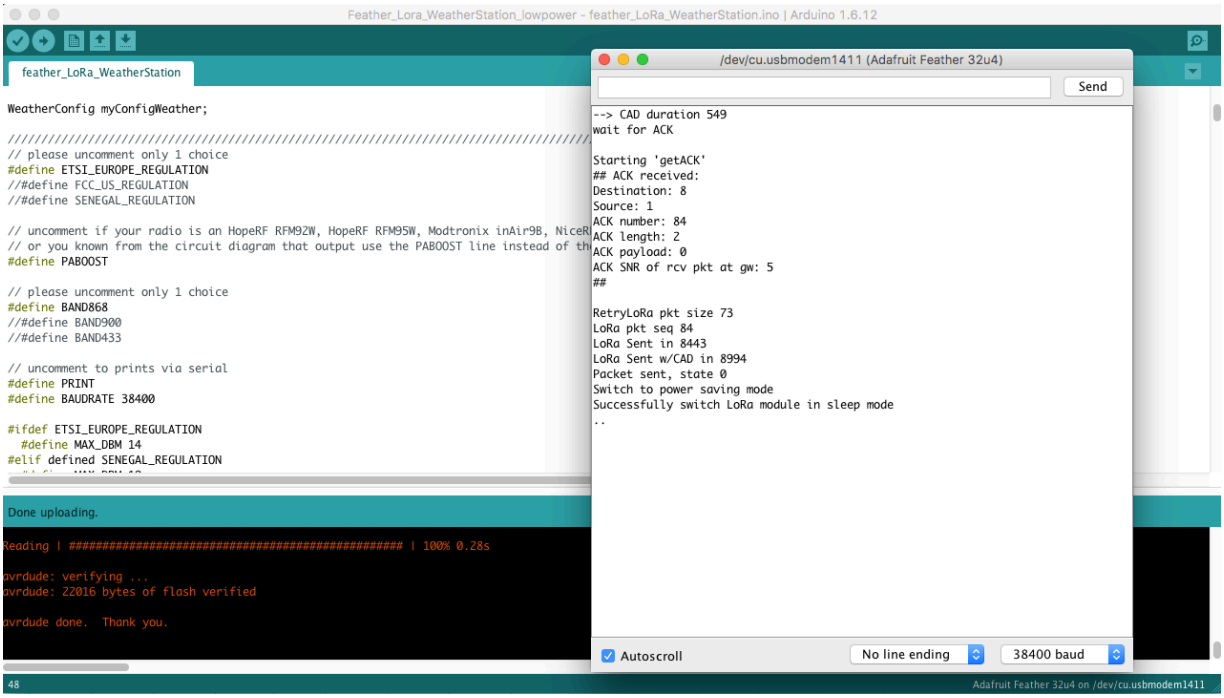

Figure 23 – Serial Monitor results.

Default Configuration:

The following code it's a default configuration. Other types of default configuration, like I/O pins or other defines, steels like the way it is.

```
// please uncomment only 1 choice
#define ETSI_EUROPE_REGULATION
//#define FCC_US_REGULATION
//#define SENEGAL_REGULATION

// uncomment if your radio is an HopeRF RFM92W, HopeRF RFM95W, Modtronix inAir9B, NiceRF1276
// or you know from the circuit diagram that output use the PABOOST line instead of the RFO line
#define PABOOST

// please uncomment only 1 choice
#define BAND868
//#define BAND900
//#define BAND433

// uncomment to prints via serial
//#define PRINT
#define BAUDRATE 38400

// CHANGE HERE THE LORA MODE, NODE ADDRESS
#define LORAMODE  1
#define DEFAULT_DEST_ADDR 1
#define node_addr 8

// CHANGE HERE THE TIME IN MINUTES BETWEEN 2 READING & TRANSMISSION
unsigned int idlePeriodInMin = 4;


#ifdef WITH_APPKEY

// CHANGE HERE THE APPKEY, BUT IF GW
CHECKS FOR APPKEY, MUST BE

 // IN THE APPKEY LIST MAINTAINED BY GW.

uint8_t my_appKey[4]={5, 6, 7, 8};


#ifdef WITH_ACK

#define NB_RETRIES 2
```

9. TESTING & IMPLEMENTATION RESULTS

9.1. System Running

1) Power-on the Feather. The script starts automatically when feather is powered on;

2) At the start and during its automatic sensors calibration process the green led appears on (take about 5 seconds).

3) Wakes-up every 15 min, take all the measurements and send to Gateway.

4) At the moment of reading values by the sensors and send a message to LoRa gateway the blue led turns on and then:

   I- If feather does not receive acknowledge then the blue led turns off and Feather sleeps.

   II- If feather receive acknowledge from master gateway then the blue led switch to green for 2 seconds, and feathers start sleeping.

5) When Feather sleeps all led's are off.

6) Feather wake up after 15 minute (by default), take the measurements and send back to master gateway until sleeping again.

9.2. Test Without LowPower & Without RTC

This test was performed without putting the Feather to sleep and without resorting to the RTC implementation. Were obtained the following data and conclusions:

- Sketch "Feather_Lora_WeatherStation.ino";
- 70% of full memory load;
- With radio LoRa Sleep mode policy;
- Without feather low-power policy;
- Wake-up's 4 minutes at a time;
- +14dbm of power;
- 10mA with radio in sleep mode;
- 115mA when is active and sending;
- 20mA when waiting for ACK;

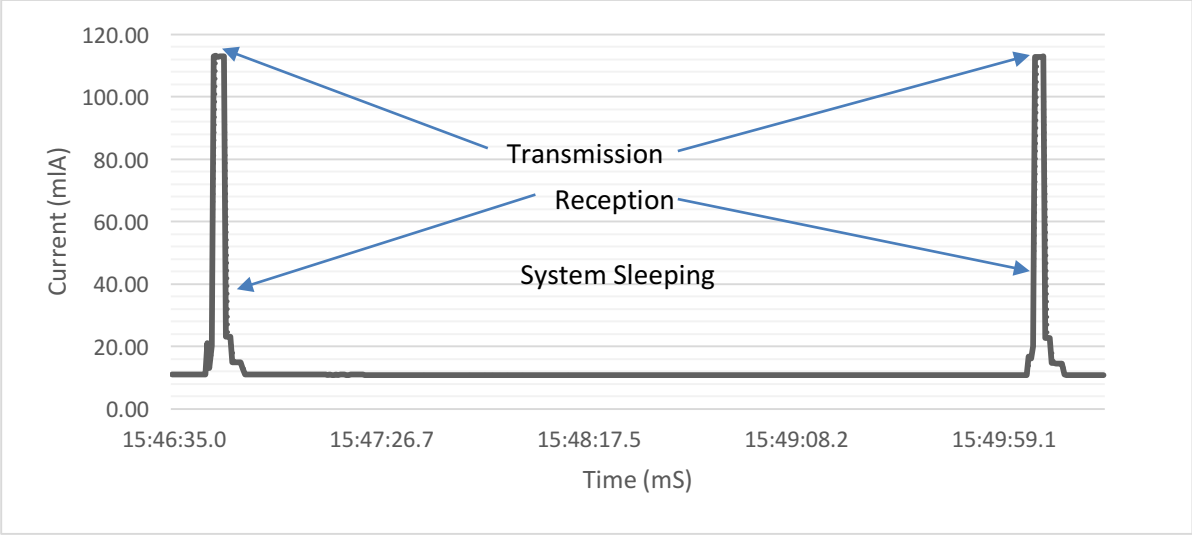

Figure 24 – Weather Station test without LowPower & without RTC during 4 minutes.

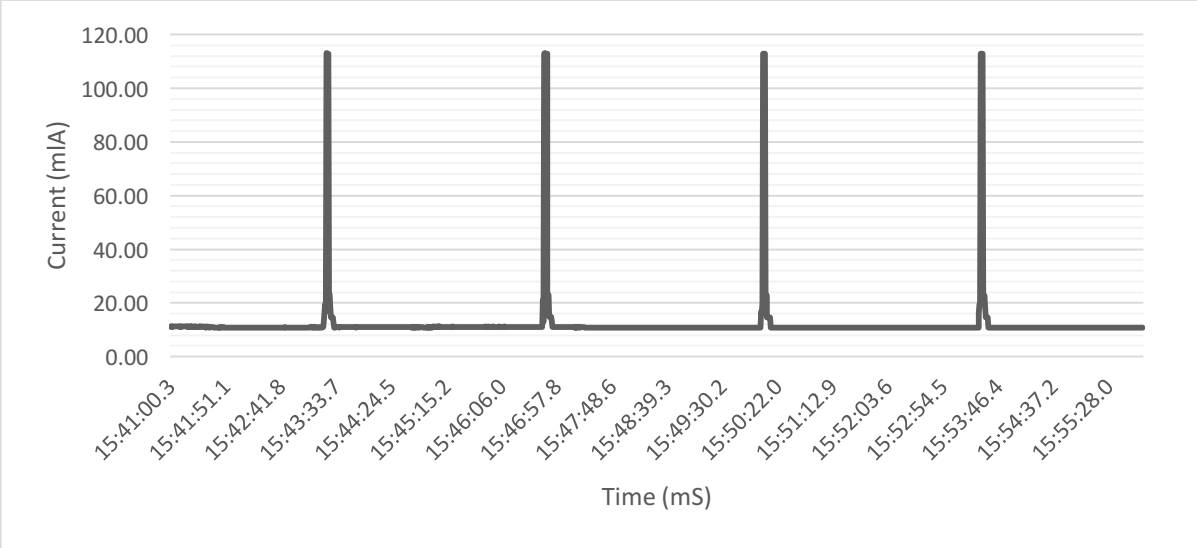

Figure 25 – Weather Station test without LowPower & without RTC during 15 minutes.

In this case we didn't realize any advanced power saving mechanisms such as putting the micro-controller in deep sleep mode or lower frequency, or performing ADC reduction, just powering off the radio module using your sleep mode.

It's expected that the baseline consumption can be further decreased with more advanced power management policy.

9.3. Test With LowPower & Without RTC

The following test was performed by putting the Feather to sleep mode and without resorting to the RTC implementation. It was possible to obtained to some data and conclusions:

- Sketch “Feather_Lora_WeatherStation_LowPower.ino”;
- 70% of full memory load;
- With radio LoRa Sleep mode policy;
- Wake-up’s 4 minutes at a time;
- +14dbm of power;
- 7.60mA with radio in sleep mode and feather too;
- 120mA when is active and sending;
- 24mA when waiting for ACK;
- 9.15mA average.

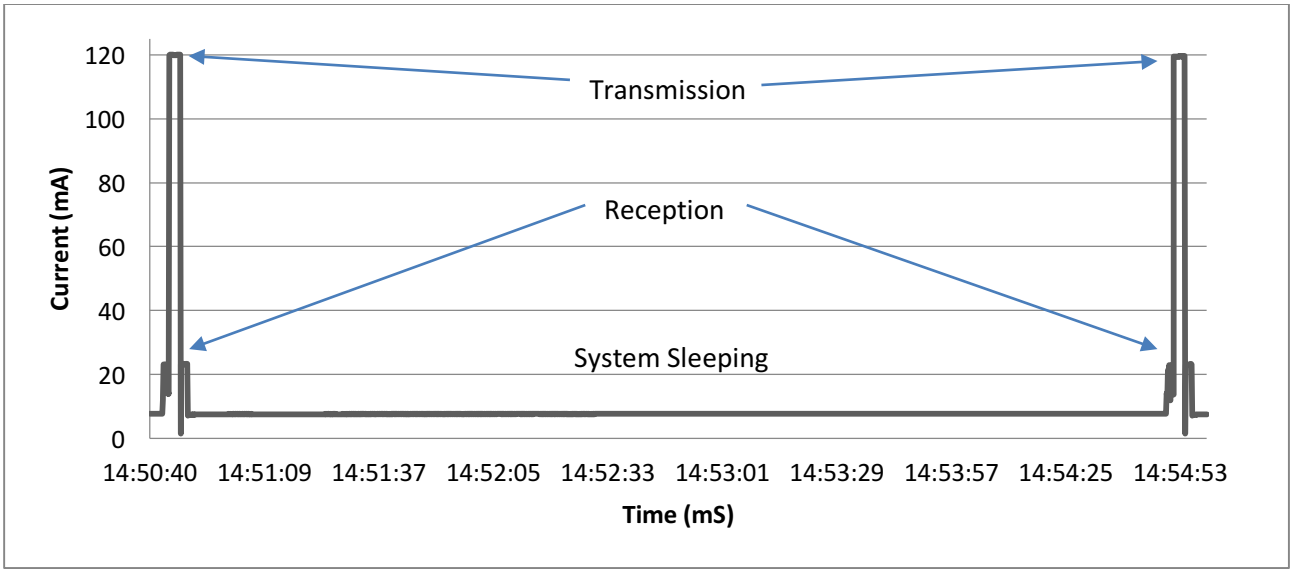

Figure 26 – Weather Station test with LowPower & without RTC during 4 minutes.

After removing the energy consumed by the feather, we found that an entire cycle for data acquisition, encoding and transmission consumes about 120mA.

The largest consumed energy part on the system comes from weather shield (about 80% of then). The receiving communication actually consumes less than half that amount of energy.

- Sketch “Feather_Lora_WeatherStation_LowPower.ino”;
- 70% of full memory load;
- With radio LoRa Sleep mode policy;
- Wake-up’s 15 minutes at a time;
- +14dbm of power;
- 7.60mA with radio in sleep mode and feather too;
- 118mA when is active and sending;
- 24mA when waiting for ACK;
- 8.23mA average.

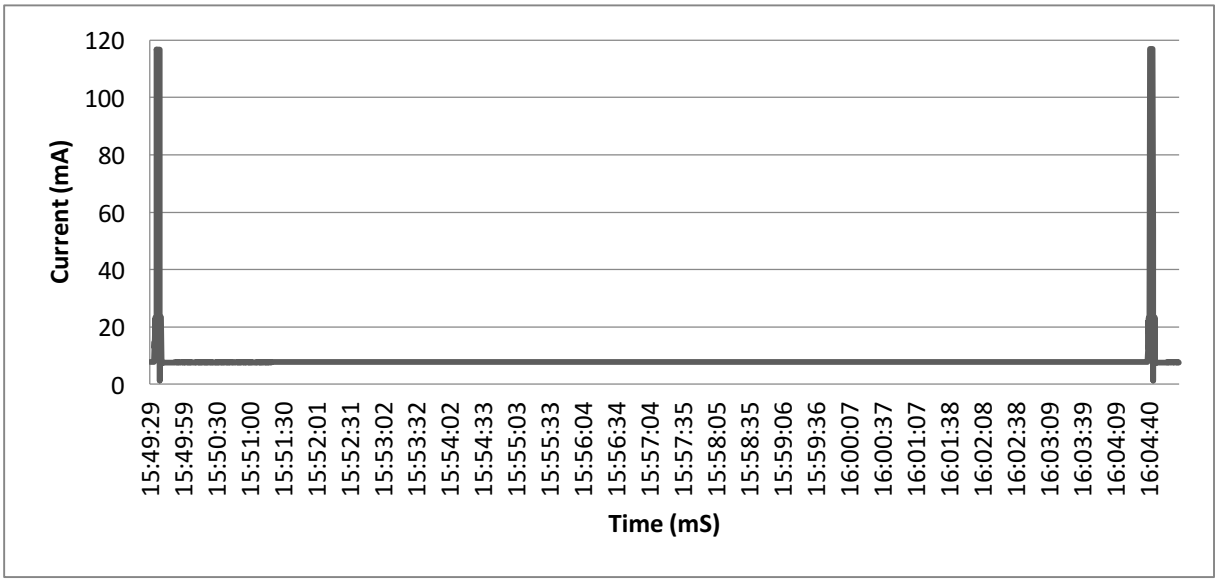

Figure 27 – Weather Station test with LowPower & without RTC during 15 minutes.

In this case, you can see that the sleeping intervals aren't being respected. This is because the relationship between the sensor interrupts and LowPower hasn't yet been resolved, and the whole system wakes up at the moment an interrupt occurs.

9.4. Test With LowPower & With RTC

After the implementation of the RTC it was possible to acquire the following data and also some conclusions:

- Sketch "Feather_Lora_WeatherStation_LowPower_RTC.ino";
- 72% of full memory load;
- With radio LoRa Sleep mode policy;
- Wake-up's 4 minutes at a time;
- +14dbm of power;
- 0.25mA with radio in sleep mode and feather too;
- 119mA when is active and sending;
- 23mA when waiting for ACK;
- 1.98mA average.

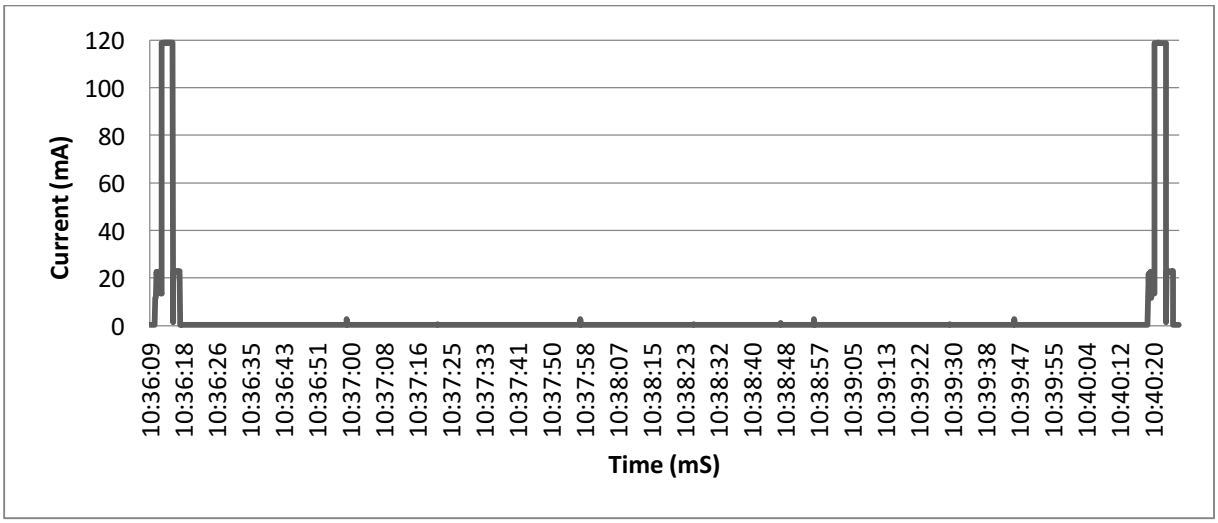

Figure 28 – Weather Station test with LowPower & with RTC during 4 minutes.

- Sketch “Feather_Lora_WeatherStation_LowPower_RTC.ino”;
- 72% of full memory load;
- With radio LoRa Sleep mode policy;
- Wake-up’s 15 minutes at a time;
- +14dbm of power;
- 0,25mA with radio in sleep mode and feather too;
- 120mA when is active and sending;
- 23mA when waiting for ACK;
- 0,80mA average.

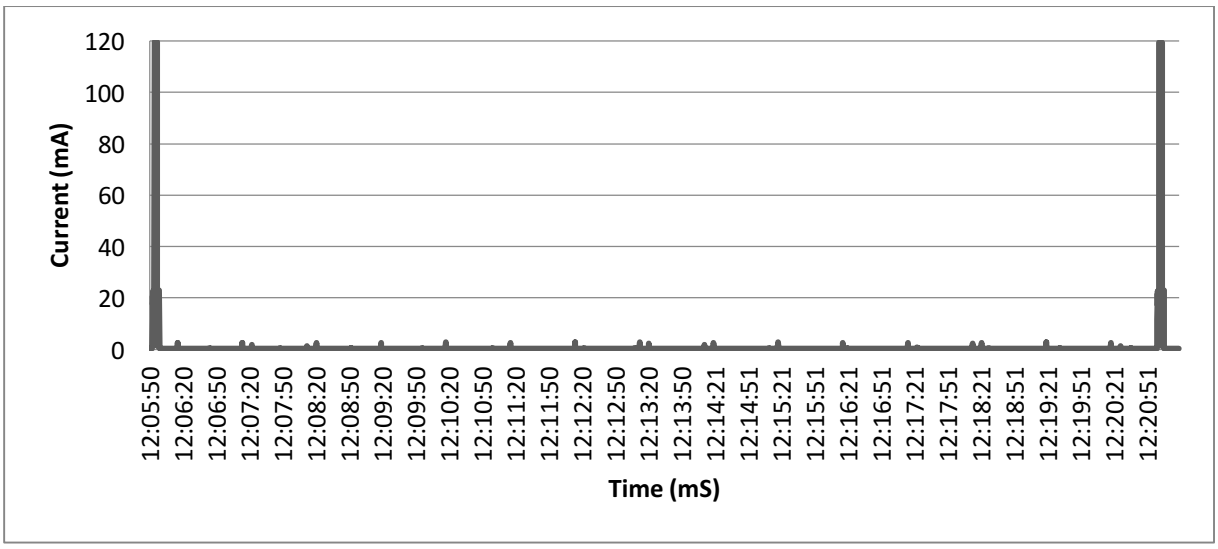

Figure 29 – Weather Station test with LowPower & without RTC during 15 minutes.

By putting the micro-controller in deep sleep, using the LowPower mode, it was possible to reduce the energy consumed to 0,25mA. It was also possible to decrease the energy consumed in an entire cycle for data acquisition, encoding and transmission in about 120mA.

Based on the RTC implementation it was possible to respect the sleep intervals. At this moment when the sensor interrupts it's active, the feather wakes up, records the change of state that this interrupt affects and if hasn't passed the sleep time forces the microcontroller to sleep again.

Using four batteries AA 2600mAh and this specification:

- with 4 minutes sleeping time can run approximately 40 days;
- with 15 minutes sleeping time can run approximately 115 days.

ACKNOWLEDGEMENT

This document has been produced in the context of the H2020 WAZIUP project. The WAZIUP project consortium would like to acknowledge that the research leading to these results has received funding from the European Union's H2020 Research and Innovation Programme under the Grant Agreement H2020-ICT-687670.