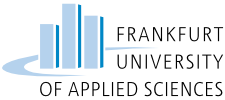
http://www.daadvn.net/images/upload/Logos/VGU_logo_01.gif ****

Vietnamese − German University

Electrical Engineering and Information Technology Study Program

Frankfurt University of Applied Science

Faculty of Electrical Engineering

**DESIGN, IMPLEMENTATION, AND CHARACTERISATION  
OF AN AUTONOMOUS WIRELESS AGROMETEOROLOGY STATION**

by

DO NGUYEN HOANG

Matriculation number: 1235052

First supervisor: Dr. Udo Klein

Second supervisor: M.Sc. Tran Quang Nhu

BACHELOR THESIS

Submitted in partial fulfillment of the requirements

for the degree of Bachelor Engineering in study program

Electrical Engineering and Information Technology,

Vietnamese − German University, 2023

Binh Duong, Vietnam, September 2023

*(This page is intentionally left blank)*

# DISCLAIMER

I declare that this thesis is the product of my work, unless otherwise referenced. All opinions, result, conclusions, and recommendations are my own to the best of my knowledge and may not represent the policies or opinions of the Vietnamese-German University.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Do Nguyen Hoang

# ACKNOWLEDGEMENTS

khgkfbgksdgkadgklbaskgkjlahrgklj

# ABSTRACT

By applying technological advances to support, and even replace traditional methods of human observations and experience, precision agriculture could be realised for better productivity. Therefore, this project aims at the construction of a sensory unit that can help monitor the meteorology of a cultivation area with high consistency while keeping error to the lowest.

The design includes a set of sensors to sample targeted environmental parameters and allows the data to be accessed online. The finding of this thesis is that the design for a such goal is viable, but not without some limitations.

# TABLE OF CONTENTS

[DISCLAIMER i](#_Toc145663581)

[ACKNOWLEDGEMENTS ii](#_Toc145663582)

[ABSTRACT iii](#_Toc145663583)

[TABLE OF CONTENTS iv](#_Toc145663584)

[LIST OF FIGURES vi](#_Toc145663585)

[LIST OF TABLES viii](#_Toc145663586)

[ABBREVIATIONS ix](#_Toc145663587)

[1. Introduction 1](#_Toc145663588)

[1.1. Background 1](#_Toc145663589)

[1.2. Objectives 1](#_Toc145663590)

[1.3. Thesis structure 1](#_Toc145663591)

[2. Module overview 2](#_Toc145663592)

[2.1. STM32F103C8T6 Microcontroller 2](#_Toc145663593)

[2.2. Sensor Units 3](#_Toc145663594)

[2.2.1. Anemometer 3](#_Toc145663595)

[2.2.2. Wind Vane 4](#_Toc145663596)

[2.2.3. Rain Gauge 6](#_Toc145663597)

[2.2.4. BME280 8](#_Toc145663598)

[2.2.5. DS18B20 9](#_Toc145663599)

[2.3. SX1278 for Wireless Communication 11](#_Toc145663600)

[2.4. Other Modules 12](#_Toc145663601)

[2.4.1. DS3231 12](#_Toc145663602)

[2.4.2. microSD Card 13](#_Toc145663603)

[2.4.3. Analogue Low-pass Filter (LPF) 13](#_Toc145663604)

[2.5. System Powering 16](#_Toc145663605)

[2.6. Server 16](#_Toc145663606)

[3. Implementation 18](#_Toc145663607)

[3.1. STM32F103C8T6 Microcontroller 18](#_Toc145663608)

[3.2. Sensor units 21](#_Toc145663609)

[3.2.1. Anemometer 21](#_Toc145663610)

[3.2.2. Wind Vane 25](#_Toc145663611)

[3.2.3. Rain Gauge 35](#_Toc145663612)

[3.2.4. BME280 36](#_Toc145663613)

[3.2.5. DS18B20 38](#_Toc145663614)

[3.3. SX1278 42](#_Toc145663615)

[3.4. Other Modules 42](#_Toc145663616)

[3.4.1. DS3231SN 42](#_Toc145663617)

[3.4.2. microSD Card 43](#_Toc145663618)

[3.5. System Powering 43](#_Toc145663619)

[3.6. Gateway and Server 45](#_Toc145663620)

[4. Experimental Characterisation 53](#_Toc145663621)

[4.1. Anemometer 53](#_Toc145663622)

[4.2. Wind Vane 60](#_Toc145663623)

[4.3. Rain Gauge 66](#_Toc145663624)

[4.4. Temperature Sensors – Hygrometer – Barometer 67](#_Toc145663625)

[4.5. LoRa Connectivity – ThingSpeak 71](#_Toc145663626)

[4.6. LoRa Range Test 73](#_Toc145663627)

[5. Conclusion and Future Work 78](#_Toc145663628)

[APPENDIX A. HAL and DMA Configuration Functions for ADC x](#_Toc145663629)

[APPENDIX B. CA3140EZ Voltage Buffer Test Result xiv](#_Toc145663630)

[APPENDIX C. Op-Amp Pick-up Test for Voltage Buffer at 3.3-V Power Supply xvii](#_Toc145663631)

[APPENDIX D. Wind Vane Test Result for Circuitry of Figure 3‑14 xx](#_Toc145663632)

[APPENDIX E. Python Program on Raspberry Pi 4B xxiv](#_Toc145663633)

[APPENDIX F. LoRa Range Test – Node’s Log xxvii](#_Toc145663634)

[APPENDIX G. LoRa Range Test – Gateway’s Log xxxi](#_Toc145663635)

[REFERENCES xxxvi](#_Toc145663636)

# LIST OF FIGURES

[Figure 2‑1. The STM32F103C8T6-based Blue Pill board in the project 2](#_Toc145675737)

[Figure 2‑2. The counterfeit read as an STM32F103 Medium Density device with a 128-KB Flash memory 3](#_Toc145675738)

[Figure 2‑3. The three-cup anemometer [1] 4](#_Toc145675739)

[Figure 2‑4. “The internal reed switch of the anemometer” [1] 4](#_Toc145675740)

[Figure 2‑5. The wind vane [1] 5](#_Toc145675741)

[Figure 2‑6. Electrical circuit of the wind vane [1] 5](#_Toc145675742)

[Figure 2‑7. Examples of wind vane-reading circuits 6](#_Toc145675743)

[Figure 2‑8. The structure of the rain gauge [1] 7](#_Toc145675744)

[Figure 2‑9. Rain gauge’s reed switch on a PCB upon removal of base chassis cover cap [1] 7](#_Toc145675745)

[Figure 2‑10. Orifice dimensions (in mm) of the funnel [1] 7](#_Toc145675746)

[Figure 2‑11. Dimensions of a BME280 sensor [18] 8](#_Toc145675747)

[Figure 2‑12. BME280 module with header strip [20] 8](#_Toc145675748)

[Figure 2‑13. DS18B20 hardware 10](#_Toc145675749)

[Figure 2‑14. Circuitry examples for communication with DS18B20 [1], [22] 11](#_Toc145675750)

[Figure 2‑15. SX1278 LoRa module in test 12](#_Toc145675751)

[Figure 2‑16. DS3231SN pinout module 13](#_Toc145675752)

[Figure 2‑17. microSD card module from Thegioiic [46] 14](#_Toc145675753)

[Figure 2‑18. Passive first-order RC LPF circuit 14](#_Toc145675754)

[Figure 2‑19. Frequency response of an example RC LPF with and under simulation [1] 16](#_Toc145675755)

[Figure 2‑20. 18-V 5-W solar panel from Thegioiic [50] 17](#_Toc145675756)

[Figure 2‑21. 18650 batteries and charger board included in the design 17](#_Toc145675757)

[Figure 2‑22. The nodeMCU ESP8266 used at the server side of the project [1] 18](#_Toc145675758)

[Figure 3‑1. The schematic of the STM32F103C8T6-housed Blue Pill board [7] 20](#_Toc145675759)

[Figure 3‑2. Simplified Blue Pill schematic with removed unused parts 21](#_Toc145675760)

[Figure 3‑3. Bus connections with the Blue Pill board 22](#_Toc145675761)

[Figure 3‑4. Circuitry for testing the anemometer in ideal conditions 23](#_Toc145675762)

[Figure 3‑5. “An instance of noise spikes captured in an oscilloscope” [1] 23](#_Toc145675763)

[Figure 3‑6. Bypass capacitor test circuitry using the reed switch from the anemometer 24](#_Toc145675764)

[Figure 3‑7. Reed switch debounced by the 68-nF bypass capacitor 25](#_Toc145675765)

[Figure 3‑8. Circuitry for interfacing with the anemometer 25](#_Toc145675766)

[Figure 3‑9. Anemometer setup routine and interrupt service routine upon input event 26](#_Toc145675767)

[Figure 3‑10. Voltage divider circuitry with ADC module 27](#_Toc145675768)

[Figure 3‑11. Derivation of the Norton equivalent of the voltage divider 28](#_Toc145675769)

[Figure 3‑12. An instance of Rexternal = 3.0kΩ input to the voltage step-calculating sheet 29](#_Toc145675770)

[Figure 3‑13. “A noise instance captured during a test on the wind vane” [1] 30](#_Toc145675771)

[Figure 3‑14. Circuitry for reading the wind vane 31](#_Toc145675772)

[Figure 3‑15. Frequency response of the active 11.587-kHz RC LPF 32](#_Toc145675773)

[Figure 3‑16. Step response of the active 11.587-kHz RC LPF 33](#_Toc145675774)

[Figure 3‑17. Circuitry for interfacing with the rain gauge 36](#_Toc145675775)

[Figure 3‑18. Anemometer setup routine and interrupt service routine upon input event 37](#_Toc145675776)

[Figure 3‑19. BME280 setup for the I2C protocol 38](#_Toc145675777)

[Figure 3‑20. The concept of a DS18B20 powered via an external power supply VP 39](#_Toc145675778)

[Figure 3‑21. Final circuitry for interfacing the DS18B20 with the microcontroller 40](#_Toc145675779)

[Figure 3‑22. Setup and reading routines for DS18B20 42](#_Toc145675780)

[Figure 3‑23. Hardware setup for SX1278 LoRa module 43](#_Toc145675781)

[Figure 3‑24. SX1278 software setup and message-handling routines 46](#_Toc145675782)

[Figure 3‑25. Hardware setup with the DS3231SN RTC module 47](#_Toc145675783)

[Figure 3‑26. Physical pin connections of the microSD card module with the microcontroller 48](#_Toc145675784)

[Figure 3‑27. Block diagram of the system powering circuitry 49](#_Toc145675785)

[Figure 3‑28. Hardware setup for the gateway 50](#_Toc145675786)

[Figure 3‑29. ThingSpeak channel’s “API Keys” tab 52](#_Toc145675787)

[Figure 3‑30. Gateway setup and loop routines 54](#_Toc145675788)

[Figure 4‑1. Axial Fan Module MFP107 by TecQuipment [76] 57](#_Toc145675789)

[Figure 4‑2. Near-linear association between the anemometer’s and the Axial Fan Module’s wind speeds 59](#_Toc145675790)

[Figure 4‑3. Wind speed factors by the read wind speed from the anemometer 59](#_Toc145675791)

[Figure 4‑4. Curve Fitting result for N = 1 61](#_Toc145675792)

[Figure 4‑5. Curve Fitting result for N = 2 62](#_Toc145675793)

[Figure 4‑6. Curve Fitting result for N = 3 63](#_Toc145675794)

[Figure 4‑7. Number of correct and bad readings when sampling from the voltage divider 64](#_Toc145675795)

[Figure 4‑8. Number of correct and bad readings when sampling from the passive 11.587-kHz LPF 66](#_Toc145675796)

[Figure 4‑9. Op-Amp setup for the voltage buffer test 67](#_Toc145675797)

[Figure 4‑10. Number of correct and bad readings when sampling from an active 11.587-kHz LPF using a CA3140EZ Op-Amp 67](#_Toc145675798)

[Figure 4‑11. Number of correct and bad readings when sampling from an active 11.587-kHz LPF using a MCP6002-I/P Op-Amp 68](#_Toc145675799)

[Figure 4‑12. Final wind vane test result from the exact circuitry as Figure 3‑14 69](#_Toc145675800)

[Figure 4‑13. The mini digital scale used in rain gauge test 70](#_Toc145675801)

[Figure 4‑14. Extech 445815 Humidity Alert II Hygro-Thermometer unit used as reference 72](#_Toc145675802)

[Figure 4‑15. Temperature readings from the compound test 73](#_Toc145675803)

[Figure 4‑16. Humidity readings from the compound test 73](#_Toc145675804)

[Figure 4‑17. Barometric pressure readings from the compound test 74](#_Toc145675805)

[Figure 4‑18. Channel settings on ThingSpeak for LoRa connectivity and server test 75](#_Toc145675806)

[Figure 4‑19. Private view of the server test channel on ThingSpeak 76](#_Toc145675807)

[Figure 4‑20. Example of Chart Option window on ThingSpeak 77](#_Toc145675808)

[Figure 4‑21. LoRa Range Test – logged locations of LoRa transmissions by the node 79](#_Toc145675809)

[Figure 4‑22. RSSI and SNR of the gateway displayed on ThingSpeak 81](#_Toc145675810)

# LIST OF TABLES

[Table 2‑1. Wind directions by resistor values [15] 6](#_Toc145632656)

[Table 2‑2. BME280 setup parameters [19] 9](#_Toc145632657)

[Table 2‑3. Comparison of different wireless communication technologies for IoT in Vietnam [31]–[35] 11](#_Toc145632658)

[Table 3‑1. Choosing bypass capacitor value based on available resistors from 30kΩ to 50kΩ 23](#_Toc145632659)

[Table 3‑2. Voltage steps of the voltage divider by Rexternal 29](#_Toc145632660)

[Table 3‑3. A part of the component pick-up table for the passive LPF 30](#_Toc145632661)

[Table 3‑4. ADC functions for wind vane reading software 34](#_Toc145632662)

[Table 3‑5. Methods of the OneWire class used in the software for DS18B20 40](#_Toc145632663)

[Table 3‑6. ROM and function commands in use for the DS18B20 40](#_Toc145632664)

[Table 4‑1. Anemometer test data with the Axial Fan Module MFP107 54](#_Toc145632665)

[Table 4‑2. Output by the microcontroller for the 270˚ direction in the test with the passive LPF 61](#_Toc145632666)

[Table 4‑3. LoRa range test result 68](#_Toc145632667)

# ABBREVIATIONS

|  |  |  |
| --- | --- | --- |
| GDP | Gross Domestic Product | |
| IoT | Internet of Things |  |
| ADC | Analogue-to-Digital Converter | |
| I2C | Inter-Integrated Circuit | |
| U(S)ART | Universal (Synchronous) Asynchronous Receiver/Transmitter | |
| SPI | Serial Peripheral Interface | |
| PCB | Printed Circuit Board | |
| RH | Relative Humidity |  |
| IIR | Infinite Impulse Response | |
| 1-Wire | One-Wire |  |
| MOSFET | Metal-Oxide Semiconductor Field-Effect Transistor | |
| LoRa | Long-Range |  |
| Wi-Fi | Wireless Fidelity |  |
| NFC | Near Field Connection | |
| RFID | Radio-frequency Identification | |
| RTC | Real-time Clock |  |
| TCXO | Temperature-compensated Crystal Oscillator | |
| LPF | Low-pass Filter |  |
| MS | Most Significant |  |
| LS | Least Significant |  |
| LoRaWAN | Long Range Wide Area Network | |
| ISR | Interrupt Service Routine | |
| CSV | Comma-separated Values | |
| ROM | Read-only Memory |  |

# Introduction

## Background

## Objectives

The main objective of this project is to develop a working design for an Autonomous Wireless Agrometeorology Station. Therefore, the viability of each included module needs evaluations.

In general, the Autonomous Wireless Agrometeorology Station should satisfy the following requirements:

* The station is automatic and operates off-grid.
* The station is able to monitor the following parameters and send the readings wirelessly to a server once every 5 minutes.
  + Temperature at ground level and 1 metre above the ground within the range of -50 ℃ to 70 ℃ at a resolution of 0.04 ℃ with an accuracy of at least 0.5 ℃.
  + Barometric pressure within the range of 900 hPa to 1100 hPa at a resolution of 0.5 hPa with an accuracy of at least 4 hPa.
  + Humidity within the range of 0 %RH to 100 %RH at a resolution of 0.03 %RH with an accuracy of at least 3.5 %RH.
  + Wind direction by at least 8 principle directions, including the 4 basic cardinal directions of North, East, South, West, and the 4 ordinal directions Northeast, Southeast, Southwest, Northwest.
  + Wind speed with a maximum of 25 m/s and a relative accuracy of 1%
  + Precipitation data at the site of installation.
* The station is able in the Southeast Asia region.
* The station is small and simple enough to be deployed by 1 person with little to no experience.

Furthermore, since this project provides an opportunity, a side objective is set to be the evalution of the newly developed STM32duino firmware by STMicroelectronics in a real application.

## Thesis structure

# Module overview

## STM32F103C8T6 Microcontroller

There are numerous factors when it comes to choosing a microcontroller for a project, including the project requirements, physical and electrical characteristics of a microcontroller (size, architecture, built-in features, upgradability, etc.), development platforms, and supply chain factors (cost and availability) [1]–[5]. Based on the objectives of this thesis (Section 1.2) and the characteristics of other modules explored later in this chapter, it is determined that the microcontroller of choice must have at least 19 port pins that have the capability for handling GPIO, external interrupts, ADC, SPI, and I2C. Furthermore, since this thesis has a side objective of testing the STM32duino firmware, the options are limited to the STM32 family by STMicroelectronics.

All the devices belonging to the STM32 family run on a 2.0-3.6V power supply, which is relatively more power-efficient than higher-voltage microcontrollers (i.e., the 5-V AVR family) [6]. For testing and prototyping, one of the popular STM32 options in Vietnam that satisfies the port pin quantity requirement is the STM32F103C8T6 available on a Blue Pill board. However, upon purchased, the units appear to contain a counterfeit (Figure 2‑1) which bears much similarities with an authentic STM32F103C8T6 apart from its doubled Flash memory size of 128 KB (Figure 2‑2). For the sake of simplicity, this thesis shall still call the counterfeit an STM32F103C8T6 microcontroller.

|  |  |
| --- | --- |
|  |  |
| (a) The Blue Pill board in testing and prototyping | (b) The counterfeit with the markings   like an authentic STM32F103C8T6 |

Figure 2‑1. The STM32F103C8T6-based Blue Pill board in the project

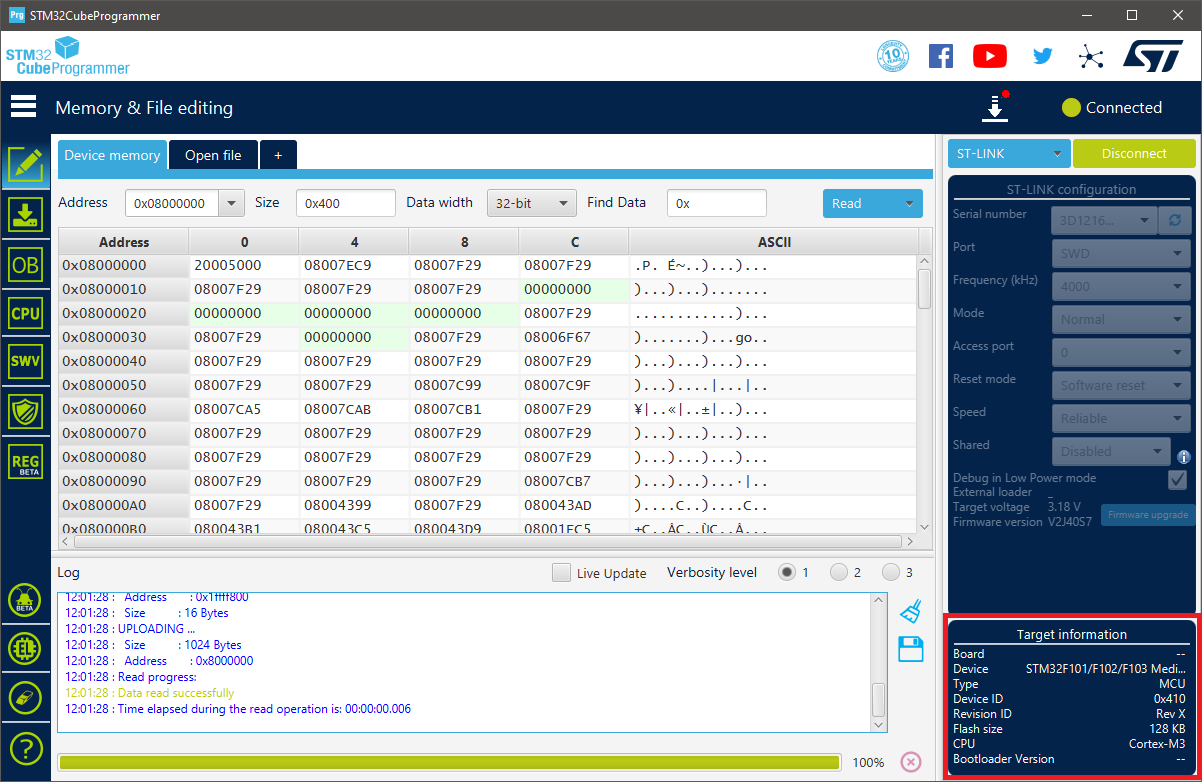


Figure 2‑2. The counterfeit read as an STM32F103 Medium Density device with a 128-KB Flash memory

Each Blue Pill board offers 32 input/output port pins available for use with interrupt capability, 18 of which are 5-V tolerant [7]–[9]. The STM32F103C8T6 microcontroller is built with 2 12-bit Analogue-to-Digital Converter (ADC) modules sharing 10 physical input pins (from PA0 to PA7, and PB0, PB1), 7 timers whose resolutions range from 16-bit to 24-bit, and remappable communication interfaces including I2C, U(S)ART, and SPI. As a result, the STM32F103C8T6 is determined to be able to handle communications with other devices used in this design.

## Sensor Units

In agriculture, the weather factor plays a key role in the growth and development of plants, since it affects both the environment (e.g., soil, fungi, pests) and the plants themselves (e.g., the integrity of branches and leaves) [10]–[12]. By monitoring the corresponding meteorological parameters, agronomy helps to predict weather events and to plan suitable adjustments to an area for the maximised agric productivity. Some such qualities includes wind data, rainfall, temperature, humidity, and atmospheric pressure, which are to be monitored by the Autonomous Wireless Agrometeorology Station via a set of sensors which are looked into in this section.

### Anemometer

The Autonomous Wireless Agrometeorology Station includes a sensor kit SP-WS02 for the Wireless Weather Station WH2081 by Misol Electronics. The kit consists of a thermo-hygrometer, an anemometer, a wind vane, and a rain gauge [13]. However, the temperature – humidity sensor is not used since it is of unknown type and encased along with the processing unit of Misol’s Weather Station. Section 2.2 therefore explores the other 3 devices, first of which is the anemometer for reading wind speed.

The anemometer follows the standard design for 3-cup anemometers, which was built to replace the traditional 4-cup design due to its superior aerodynamic performance [14], with an addition of a magnet and a reed switch. The leads of the reed switch is further extended by an RJ11 cable to allow reading of the switch electrically with a suitable circuitry.



Figure 2‑3. The three-cup anemometer [1]

|  |  |
| --- | --- |
|  |  |
| 1. The internal reed switch shown upon   the removal of bottom cap | 1. The connection between the reed switch and the RJ11 connector [15] |

Figure 2‑4. “The internal reed switch of the anemometer” [1]

The reed switch is observed to be normally open upon removal of the PCB (Figure 2‑4a) from the sensor body. When the magnet fixed under the rotor passes, the reed switch produces a click, based on which it is found to close twice per revolution of a cup.

### Wind Vane

The second device from the sensor kit SP-WS02 is a wind vane for reading wind directions. Underneath the rotor of the wind vane is there a magnet which closes up to 2 out of 8 normally-open reed switches on the PCB inside the body of the sensor (Figure 2‑6). Each reed switch is paired with a unique resistor whose value is read to determine the position of the rudder blade, thus the wind direction.



Figure 2‑5. The wind vane [1]

|  |  |
| --- | --- |
|  |  |
| 1. Reed switches inside the wind vane | 1. Schematic of the wind vane’s circuit from [15] |

Figure 2‑6. Electrical circuit of the wind vane [1]

The wind vane has an RJ11 port for users to connect the anemometer and combine the 2 sensors into 1 output via the wind vane’s RJ11 cable as illustrated in Figure 2‑6b. When set up accordingly to the 4 directions marked on the sensor’s body, the aforementioned resistors give the direction as shown in Table 2‑1. It has been pointed out that the magnet could close 2 reed switches simultaneously, thus parallel resistors formed and 8 additional wind directions to be read.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Direction (single resistor) | | | R | Direction (parallel resistors) | | | R |
| Cardinal point | | Azimuth degrees | Cardinal point | | Azimuth degrees |
| North | N | 0˚ | 33 KΩ | North-Northeast | NNE | 22.5˚ | 6.57 KΩ |
| Northeast | NE | 45˚ | 8.2 KΩ | East-Northeast | ENE | 67.5˚ | 891 Ω |
| East | E | 90˚ | 1 KΩ | East-Southeast | ESE | 112.5˚ | 688 Ω |
| Southeast | SE | 135˚ | 2.2 KΩ | South-Southeast | SSE | 157.5˚ | 1.41 KΩ |
| South | S | 180˚ | 3.9 KΩ | South-Southwest | SSW | 202.5˚ | 3.14 KΩ |
| Southwest | SW | 225˚ | 16 KΩ | West-Southwest | WSW | 247.5˚ | 14.12 KΩ |
| West | W | 270˚ | 120 KΩ | West-Northwest | WNW | 292.5˚ | 42.12 KΩ |
| Northwest | NW | 315˚ | 64.9 KΩ | North-Northwest | NNW | 337.5˚ | 21.88 KΩ |

Table 2‑1. Wind directions by resistor values [15]

Table 2‑1 data is given by Argent Data Systems and verified by the use of a multimeter. In a microcontroller-based system, reading resistor value, thus the wind direction, from the wind vane could be done by mimicking the behaviour of a multimeter in resistance measurement mode: injecting a constant current to get a voltage across the internal resistors [16]. Another approach is to use an external resistor to form a voltage divider [15]. Either way, the resistive value accoss the RJ11 inductors of the wind vane must be determined by an ADC module.

|  |  |
| --- | --- |
|  |  |
| 1. Current mirror circuit as  a constant current source | 1. Voltage divider with an external resistor |

Figure 2‑7. Examples of wind vane-reading circuits

### Rain Gauge

The third device from the sensor kit SP-WS02 used in this project is a rain gauge as a self-emptying tipping bucket. It is comprised of 2 small buckets mounted on a fulcrum like a seesaw underneath a funnel (Figure 2‑8). The design allows the buckets to tip over a side when one is full to be emptied through the opennings on the base of the rain gauge, while the other in turn collects rain water. The magnet fixed to the bucket (Figure 2‑8b), on the other hand, also moves pass the reed switch hidden inside the case on the rain gauge base and momentarily closes it. Since the reed switch leads are extended via an RJ11 cable, with a suitable circuitry, electrical pulses could be generated and read by the microcontroller.

|  |  |
| --- | --- |
|  |  |
| 1. Top view: the orifice of the funnel | 1. Disassembly: the buckets on a pivot |

Figure 2‑8. The structure of the rain gauge [1]

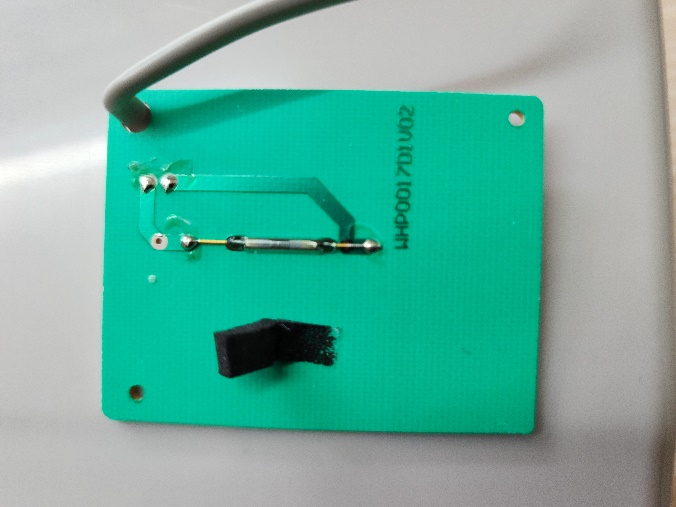


Figure 2‑9. Rain gauge’s reed switch on a PCB upon removal of base chassis cover cap [1]

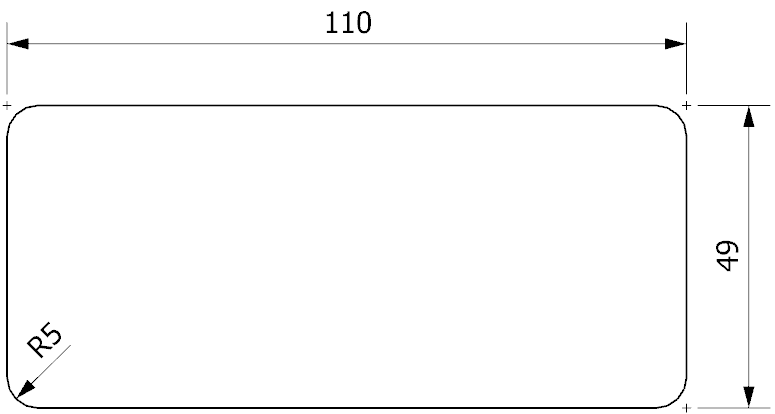


Figure 2‑10. Orifice dimensions (in mm) of the funnel [1]

Misol Electronics states that the rain gauge produces 1 pulse for every 0.3 mm of rain [17], while Argent Data System provides a value of 0.011”, or 0.2794 mm [15]. However, since the resolution provided by the latter is converted from inches, the difference may be neglected, and the value of 0.3 mm rainfall per tip could later be used to construct the software for the Autonomous Wireless Agrometeorology Station.

### BME280

BME280 is a self-calibrating environmental sensor by Bosch Sensortec GmbH. It is a combined digital sensor for barometric pressure, relative humidity, and ambient temperature, developed for multiple targets including mobile devices and wearables, thus the advantages of small size and low power consumption [18], [19]. The device in this design, arrives on a pinout board with a 2.54-mm single row header strip (Figure 2‑12) for convenience in prototyping and testing.

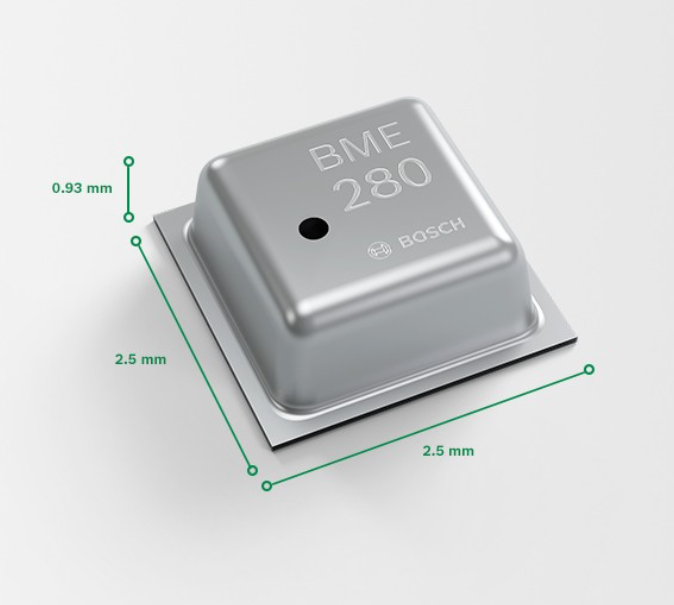


Figure 2‑11. Dimensions of a BME280 sensor [18]

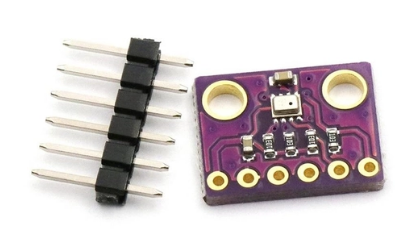


Figure 2‑12. BME280 module with header strip [20]

According to [19], the product specifications of the BME280 are as follows:

* Operational voltage: 1.71 V to 3.6 V
* Communicational voltage: 1.2 V to 3.6 V
* Typical current consumption: 0.1 µA (sleep mode) to 714 µA (pressure measurement)
* Communication interface: SPI and I2C
* Full-range output:
  + Temperature: -40 ℃ to +85 ℃
  + Humidity: 0 %RH to 100 %RH
  + Pressure: 300 hPa to 1100 hPa
* Data resolution:
  + Temperature: 0.01 ℃
  + Humidity: 0.008 %RH
  + Pressure: 0.18 hPa (minimum)
* Data accuracy:
  + Temperature: ±0.5 ℃ (0 ℃ to 65 ℃) up to ±1.5 ℃ (-40 ℃ to -20 ℃)
  + Humidity: ±3 %RH
  + Pressure: ±1 hPa

As an IC sensor, the BME280 is built with not only the sensing units but also mode control, data compensation, and noise reduction by oversampling and a digital filter. During software setup, the BME280 could be configured with the parameters shown in Table 2‑2.

|  |  |
| --- | --- |
| Parameter | Valid values |
| Sensor mode | Sleep mode, Normal mode, Forced mode |
| Pressure sampling rate | None (sampling is turned off), x1, x2, x4, x8, x16 |
| Temperature sampling rate |
| Humidity sampling rate |
| IIR filter coefficient | Off (IIR filter is turned off), x2, x4, x8, x16 |
| Standby time for normal mode (in milliseconds) | 0.5, 10, 20, 62.5, 125, 250, 500, 1000 |

Table 2‑2. BME280 setup parameters [19]

### DS18B20

DS18B20 is a family of digital temperature sensors by Maxim Integrated that follows the design of DS1820, a predecessor introduced by Dallas Semiconductor in 1998 [21]. Although the DS18B20 sensors are offered in only 3 types of packages by the manufacturer [22], a waterproof version with a long cable built from the 3-pin TO-92 configuration DS18B20 is available and has had its application explored off a PCB in different environments (e.g., liquids) [21], [23]–[25]. Because of this version, the DS18B20 is chosen to monitor the ambient temperature at ground level for this project. Although there are other choices of sensors with long cables, they either need heavily calibrating (e.g., Resistance Temperature Detector PT100 [26]), or have higher price and unnecessary functions (e.g., the AM2315 temperature and humidity sensor priced at 830,000 VND compared to the 71,000-VND DS18B20 temperature sensor with 1-m long cable [27], [28]).

|  |  |
| --- | --- |
|  |  |
| 1. TO-92 configuration | 1. Waterproof version with 1-m long cable |

Figure 2‑13. DS18B20 hardware

According to [2], the specifications of the sensor are as follows:

* Operating voltage: 3.0 V to 5.5 V
* Active current: typically, 1 mA (VDD = 5 V)
* Standby current: typically, 750 nA
* Sink current: maximum, 4.0 mA
* Full-range temperature output: -55 ℃ to +125 ℃
* Data resolution: programmable; 0.5 ℃, 0.25 ℃, 0.125 ℃, and 0.0625 ℃
* Minimum thermometer error: ±0.5 ℃ (from -10 ℃ to +85 ℃)

Communication with a DS18B20 is done asynchronously via the bi-directional One-Wire (1-Wire) protocol, which works under the principle of timing, digital input/output, and interchanging control of a single wire among masters (microcontrollers) and slaves (1-Wire devices). Although the standard communication speed is announced to be a maximum of 16.3 kbps, a high-speed mode “overdrive” of up to 90 kbps is available [29], [30]. The 1-Wire protocol allows a DS18B20 to operate not only on an external power supply but also in a parasite mode as depicted in Figure 2‑14 for lower power consumption [22]. However, it is found that the circuitry in Figure 2‑14b is only applicable if the microcontroller turns on the MOSFET before releasing the 1-Wire bus for the sensor when required. Alternatively, the MOSFET could be replaced by software (Figure 2‑14c) when the microcontroller outputs “HIGH” on its digital pin of the 1-Wire bus as soon as releasing control.

|  |  |
| --- | --- |
|  | |
| (a) DS18B20 on an external power supply | |
|  |  |
| (b) DS18B20 in parasite mode with a MOSFET | (c) DS18B20 in parasite mode  (software-controlled) |

Figure 2‑14. Circuitry examples for communication with DS18B20 [1], [22]

## SX1278 for Wireless Communication

In the design of an Autonomous Wireless Agrometeorology Station, the wireless communication between the station and a server is done via LoRa. The choice for LoRa is done from the comparison below.

|  |  |  |  |
| --- | --- | --- | --- |
| Technology | Highest data rate (uplink) | Maximum coverage | Power consumption |
| Wireless Fidelity (Wi-Fi) | 93.38 Mbps | 90 m | Medium to high |
| Bluetooth | 1 – 3 Mbps | 100 m | Low |
| Cellular network | 18.27 Mbps | Several kilometres | Medium to high |
| LoRa | 37.5 kbps | Several kilometres | Low |
| Zigbee | 250 kbps | 50 m | Low |
| Near Field Communication (NFC) | 424 kbps | 10 cm | Low |
| Radio-frequency Identification (RFID) | 424 kbps | 10 cm | Low |

Table 2‑3. Comparison of different wireless communication technologies for IoT in Vietnam [31]–[35]

It could be observed that Wi-Fi provides the highest communication speed, while data-exchange is slowest via LoRa. The data collected and transmitted by the system are just sensor readings, so the data rate does not need to be high [36].

The next criterium to be taken into account is the range over which communications are performed. LoRa and Zigbee could be seen to allow data exchange range. Despite the maximum range of 90 m, Wi-Fi signals could typically reach a maximum of only 45 m [37], which should not be a problem for an agricultural application. The common Bluetooth modules HC-05 and HC-06 provide a connection up to 100 m [38], so Bluetooth remains an option. Only NFC and RFID technologies are invalid since they are applicable for only short-range communications.

In terms of power consumption, Wi-Fi and cellular network are not ideal for a self-contained off-grid design. Bluetooth, in spite of the low power consumption, is said to be more “power hungry” compared to LoRa [39]. That leaves LoRa and Zigbee the remaining choices for wireless communications.

For this project, LoRa and Zigbee are lastly compared over product prices. A major supplier in Ho Chi Minh City, Vietnam chosen for referencing is Thegioiic. The LoRa modules SX1278 are sold at around 100,000 VND, while prices for the Zigbee modules go from 183,000 VND [40], [41]. As a result, LoRa is the wireless communication technology chosen for this design, and 2 LoRa modules SX1278 are picked for the station and the server side. Each module is a PCB module with an 8-pin 2.54-mm dual row header strip for convenience in prototyping and testing.

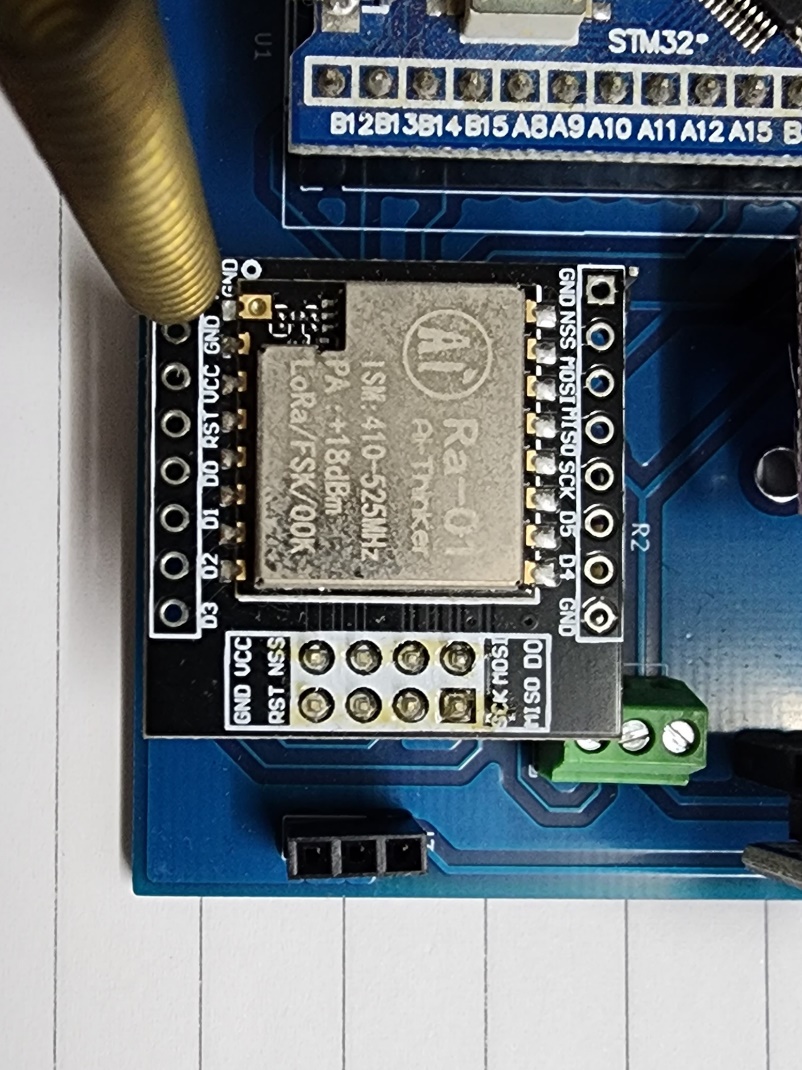


Figure 2‑15. SX1278 LoRa module in test

The product specifications for LoRa mode from [42] are as follows:

* Operating voltage: 1.8 V to 3.7 V
* Current consumption: 0.2 µA (sleep mode), up to 12 mA in receive mode or 120 mA during data transmissions
* Communication protocol: SPI
* Spreading factors: 6 to 12
* Cyclic coding rate: 4/5 to 4/8
* Signal bandwidth (in kHz): 7.8, 10.4, 15.6, 20.8, 31.2, 41.7, 62.5, 125, 250, and 500
* Package preamble length (in symbols): 6+4 to 65535+4
* User-defined package synch word: 8-bit synch word

## Other Modules

### DS3231

Time keeping is an extended feature for the Autonomous Wireless Agrometeorology Station. Although the STM32F103C8T6 microcontroller already has a built-in real-time clock (RTC), an external module is explored for higher precision of time data.

The DS3231SN is an RTC from the DS3231 product lines by Maxim Integrated, clocked at 32,768-Hz by an internal crystal. Another member from the DS3231 product lines is the DS3231M. A major difference between DS3231SN and DS3231M is that the former is built with a temperature-compensated crystal oscillator (TCXO) for the improved quality of time keeping [43], which could be checked via its 32K pin. As a result, the accuracy is proven to be much higher than that of the DS3231M [44]. For testing purposes, a DS3231SN pinout module (Figure 2‑16) is in use.

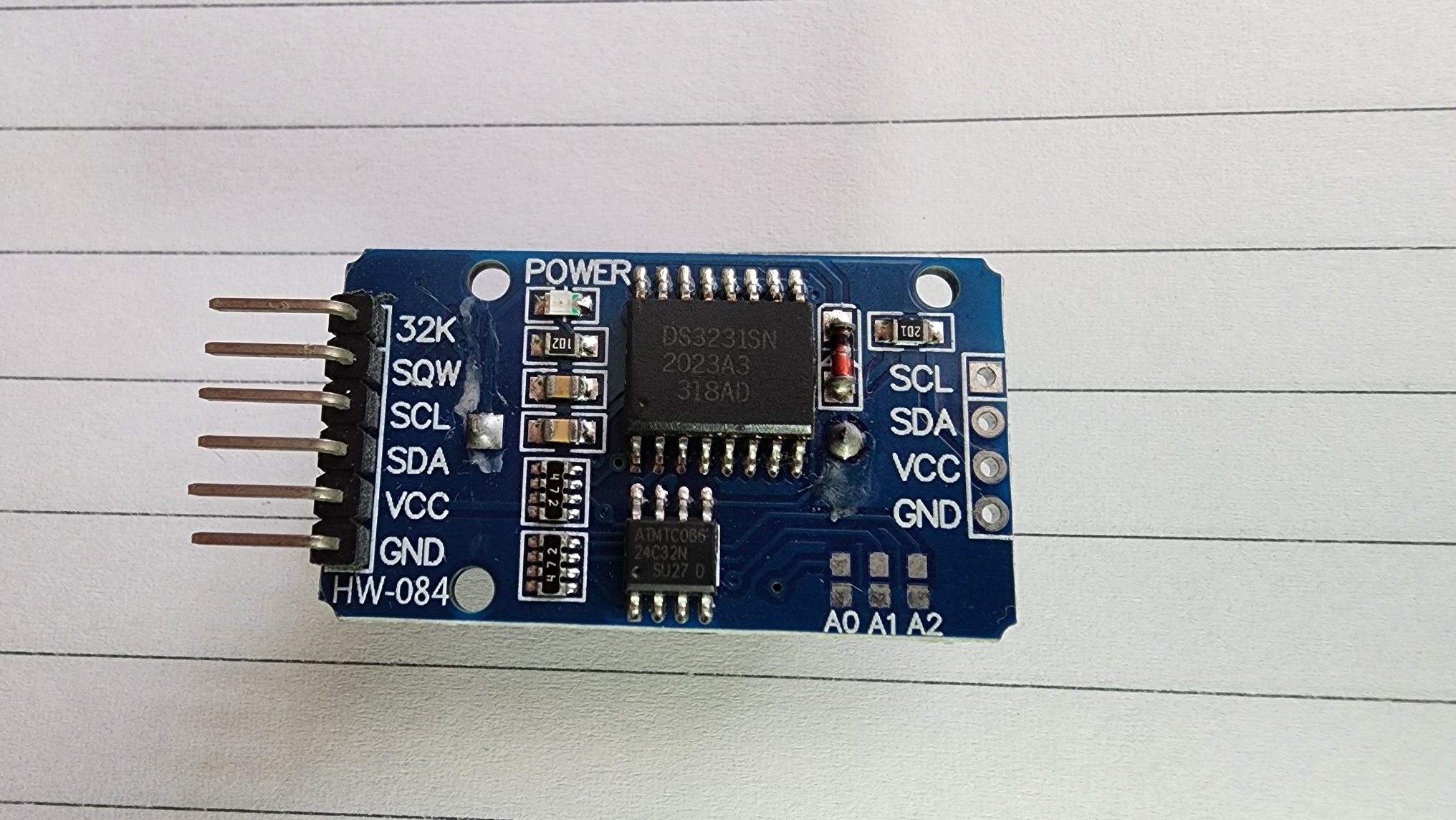


Figure 2‑16. DS3231SN pinout module

The key specifications of the DS3231SN from [45] are as follows:

* Operating voltage: 2.3 V to 5.5 V
* Battery voltage: 2.3 V to 5.5 V
* Maximum active current: 200 µA at 3.63 V
* Maximum standby current: 110 µA at 3.63 V
* Typical standby current from battery: 0.84 µA
* Effective time keeping range: from 1970 to 2100
* Compensated base clock output on 32kHz pin
* Programmable square wave output or alarm on pin
* I2C communication interface at up to 400 kHz

### microSD Card

In order to avoid the loss of data in case of dropped uplink connection from the Autonomous Wireless Agrometeorology Station to its designated server, a microSD card is included in its design. The microSD card module for this project is obtained from Thegioiic.com to be added as a module for easy prototyping and redesigning should anything goes wrong.

A microSD could be accessed via either the Secure Digital Input Output (SDIO) interface or SPI. Since the STM32F103C8T6 in this project is not designed with the SDIO interface, the microSD card module provides an SPI solution.

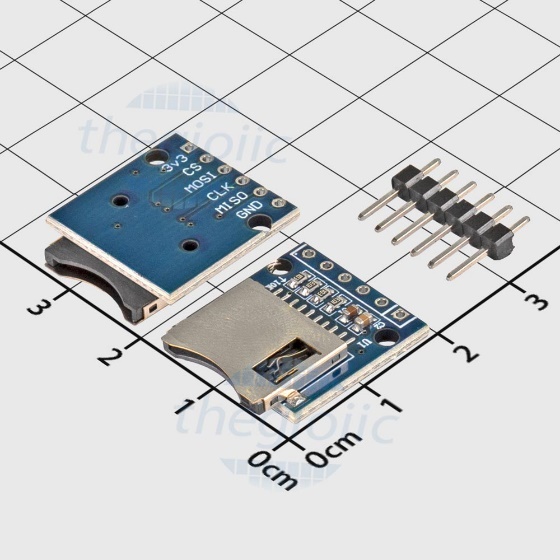


Figure 2‑17. microSD card module from Thegioiic [46]

### Analogue Low-pass Filter (LPF)

Any real-life systems are always faced with the noise problem which roots from various sources like external interference or the components of the system itself [47], [48]. Producing clean electrical signals to use in an application although is not possible, reducing the effects of noise to reach a such goal remains practical. Since noise are usually unwanted high frequency signals injected into the desired ones, the method of using analogue filters could be applied. The simpliest form of LPFs against noise is a passive first-order RC LPF as demonstrated in Figure 2‑17.

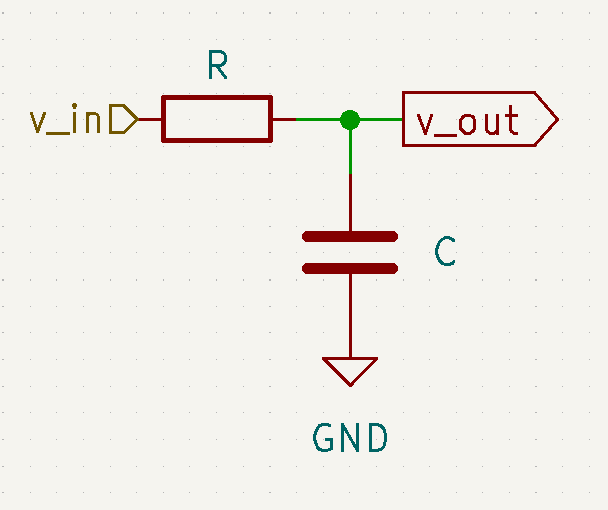


Figure 2‑18. Passive first-order RC LPF circuit

The circuit above acts under the principle of a voltage divider:

, where and is the frequency of the input signal .

The cut-off frequency of a LPF is defined to be the frequency at which the voltage gain on the logarithmic scale of the output is -3 dB [49], or . As the result, the cut-off frequency of a passive RC LPF is . For example, a LPF with and has the cut-off frequency .

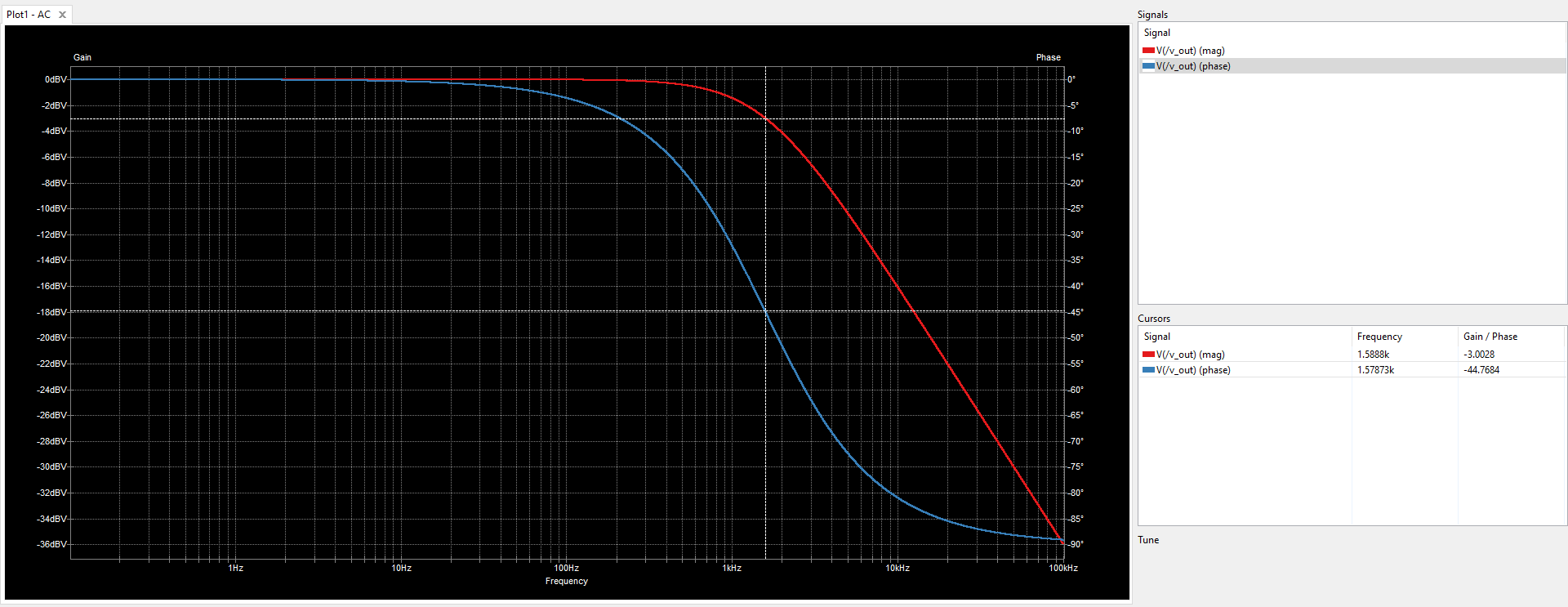


Figure 2‑19. Frequency response of an example RC LPF with and under simulation [1]

## System Powering

One of the requirements for the Autonomous Wireless Agrometeorology Station is that it is able to operate off-grid. In order to do that, the system needs another power source. The option explored in this project is solar energy, which is harvested by a 18-V 5-W solar panel obtained from Thegioiic:

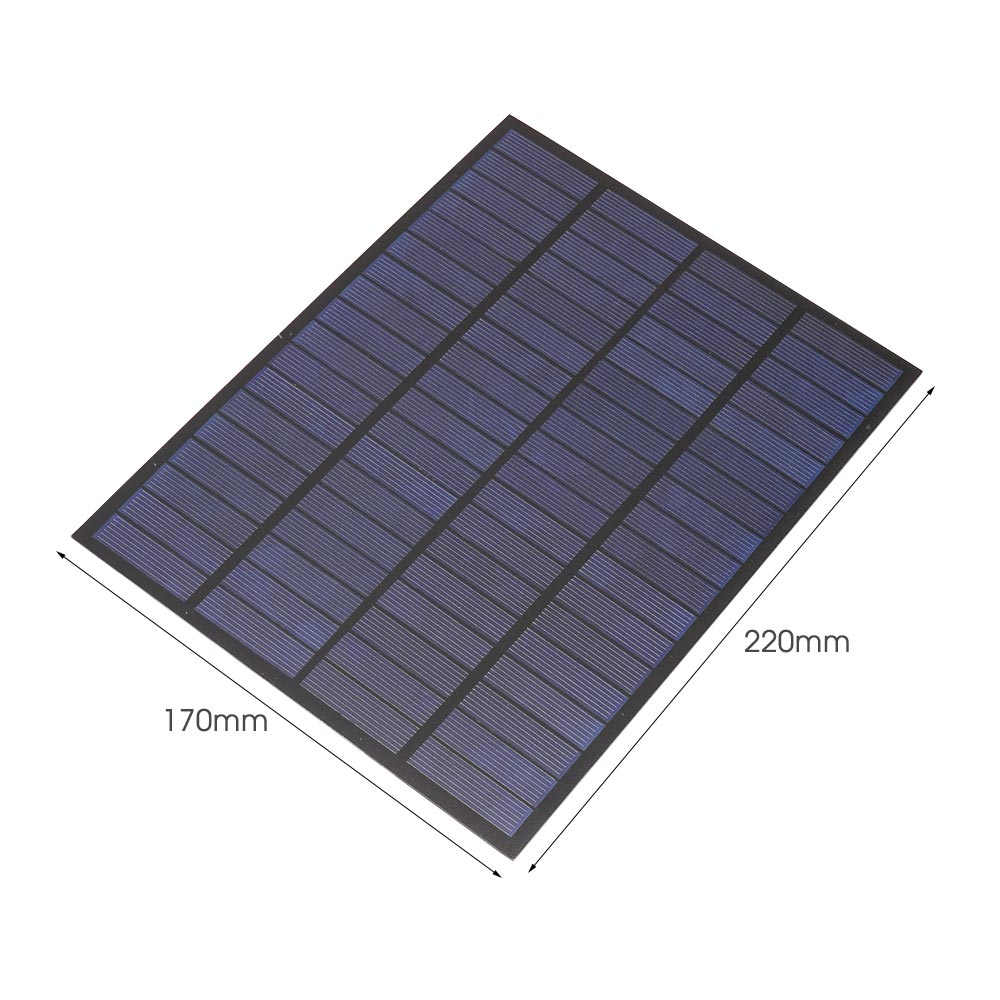


Figure 2‑20. 18-V 5-W solar panel from Thegioiic [50]

Since a solar battery system always contains a power storage unit, this project includes 2 BRC 18650 4800mAh 3.7V Li-ion batteries and an HX-2S-D20 charger board.

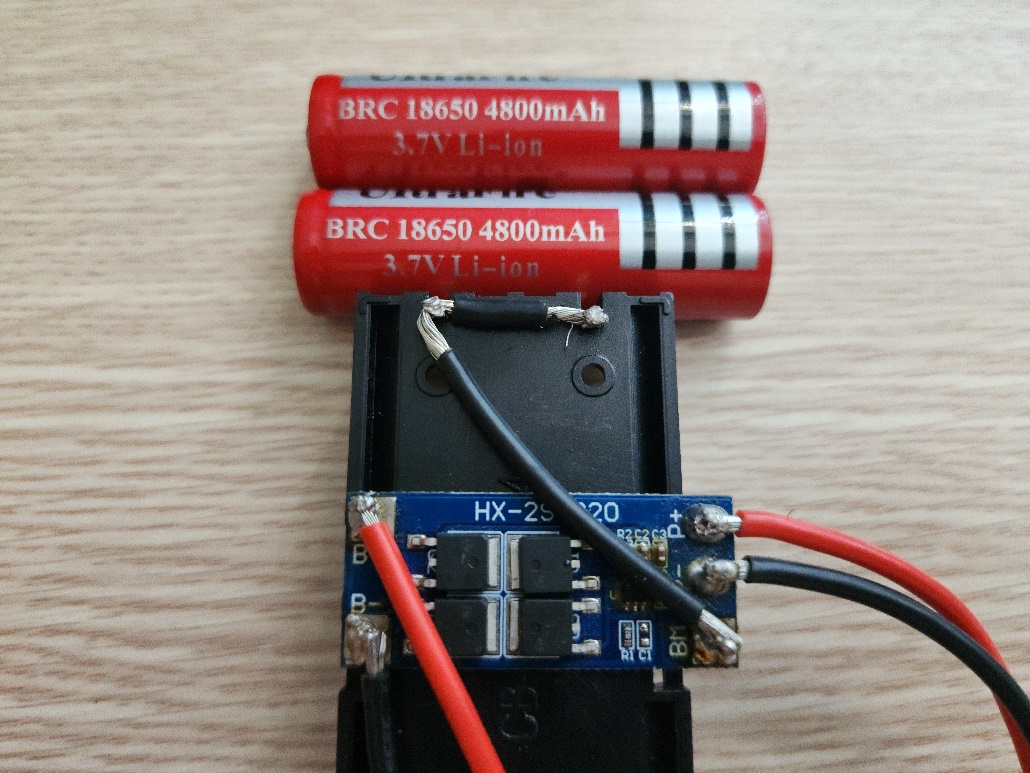


Figure 2‑21. 18650 batteries and charger board included in the design

According to the retailer, the solar panel has the absolute ratings of 18V and 5W output, which translates to a maximum current output of . Like a standard 18650 Li-ion battery, the units in this project have a voltage range from 2.5V to 4.2V, which gives 5.0V to 8.4V when combined in series for the HX-02-D20 charger.

According to [51], the HX-02-D20 is a charger board for 2 18650 Li-ion batteries, taking 8.4V – 9V input to charge the batteries. The board has protective mechanisms against short-circuits, over- and under-voltage. Therefore, it appears to be a good option to be included in the design of the Autonomous Wireless Agrometeorology Station.

## Server

The server side for the Autonomous Wireless Agrometeorology Station consists of:

* A gateway that receives the sensor data wirelessly from the station via LoRa communication,
* And an online database which the gateway forwards the sensor data to and makes the data accessible for users.

The main idea is to involve another microcontroller to use with an SX1278 LoRa module as the gateway. The microcontroller should be able then to connect to the Internet to upload the received data to the online database of choice.

For the sake of simplicity, a nodeMCU ESP8266 is utilised because of its Wi-Fi connection capability and its availability, as well as the Arduino framework for software development. The online database of choice is ThingSpeak by The MathWorks Incorporation, on top of services like Microsoft Azure, Google Firebase, and InfluxDB.

Initially, InfluxDB was in use as the online database for the Autonomous Wireless Agrometeorology Station and proved to be intuitive [1]. However, it is discovered that this service only retains uploaded data for 30 days for free accounts, which makes it less ideal for a project of a bachelor thesis. Microsoft Azure and Google Firebase, on the other hand, offer decently free services, but the connection proccesses are quite complicated, thus ruled out. ThingSpeak is the remaining option whose service is free of charge while letting users accessing and handling data without a time limit. The only known downside of ThingSpeak is that each free account is only allowed 4 data channels, which is not an issue in testing and prototyping the Autonomous Wireless Agrometeorology Station.

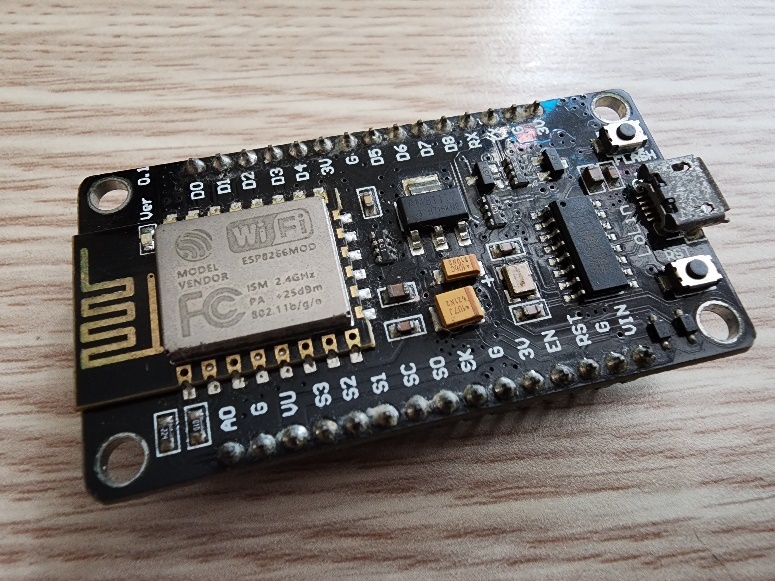


Figure 2‑22. The nodeMCU ESP8266 used at the server side of the project [1]

# Implementation

## STM32F103C8T6 Microcontroller

As mentioned in section 2.1, the STM32F103C8T6 microcontroller in test is shipped on a Blue Pill board (Figure 2‑1) with all the necessary components for its peripherals. However, the implementation in this thesis does not require some included blocks such as the RTC or the voltage regulator, thus some unused hardware pins and/or components. As a result, some parts are removed from the Blue Pill board for testing purpose; and later on, the board is redesigned following Figure 3‑2. The board is then implemented with all the connections shown in Figure 3‑3.

The software allowing the STM32F103C8T6 microcontroller to control all the connected devices is explored in Chapter 3 and constructed upon the STM32duino firmware for the Arduino framework. However, the STM32 core is not a built-in of the Arduino platform, thus board definitions required. By appending the following to the Additional Boards Manager URLs of the Arduino IDE, the STM32duino firmware could be installed via Boards Manager and used for programming in the later sections [52].

|  |  |
| --- | --- |
| https://github.com/stm32duino/BoardManagerFiles/raw/main/package\_stmicroelectronics\_index.json | La tesis de Z3r0 |

Since the microcontroller in use is an STM32F103C8T6 with a Flash memory size of 128 KB (Section 2.1), the Arduino IDE is set up as:

* Board: “Generic STM32F1 series”
* Board part number: “BluePill F103CB (or C8 with 128k)”
* U(S)ART support: “Enabled (generic ‘Serial’)”
* USB support (if available): “None”
* USB speed (if available): “Low/Full Speed”
* Optimize: “Smallest (-Os default)”
* Debug symbols and core logs: “None”
* C Runtime Library: “Newlib Nano + Float Printf”
* Upload method: “STM32CubeProgrammer (SWD)”

The software is built with the intension of debugging via UART, thus the U(S)ART support option being “Enabled (generic ‘Serial’) to make use of the default Serial object of the Arduino framework. Furthermore, the “Newlib Nano + Float Printf” option is chosen for C Runtime Library to make use of the printf(..) method, which is ported from the C language to the C++ language of the Arduino framework exclusively for the STM32duino firmware.

Finally, the software for the Autonomous Wireless Agrometeorology Station makes use of the official STM32duino firmware by STMicroelectronics, so at the time of this thesis, the only known method of uploading to the STM32F103C8T6 microcontroller is via the Serial Wire Debug (SWD) interface with an ST-Link in-circuit debugger/programmer, and the driver for a such hardware is provided by the STM32CubeProgrammer application. Therefore, the Upload method option is chosen to be “STM32CubeProgrammer (SWD)”. For the rest of the tool settings on the Arduino IDE, the default options are left untouched.

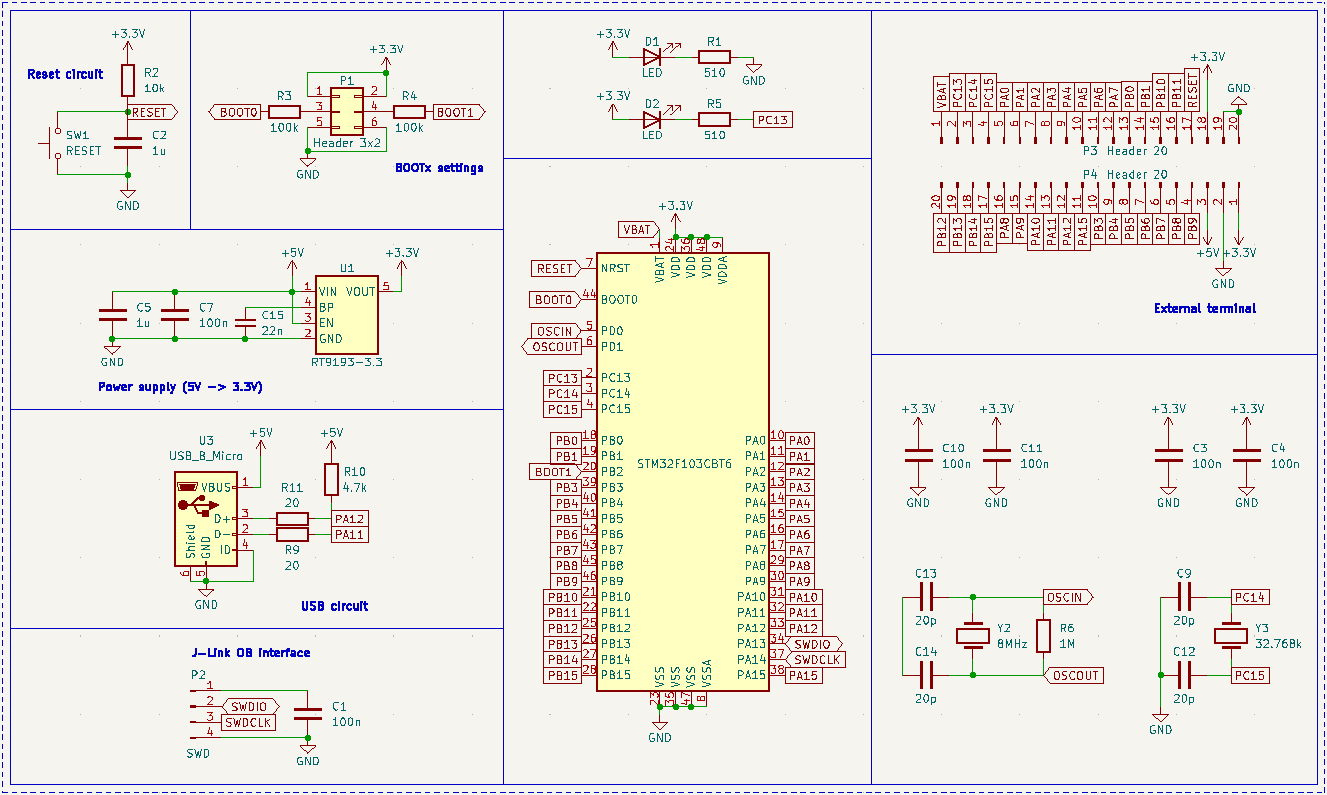


Figure 3‑1. The schematic of the STM32F103C8T6-housed Blue Pill board [7]

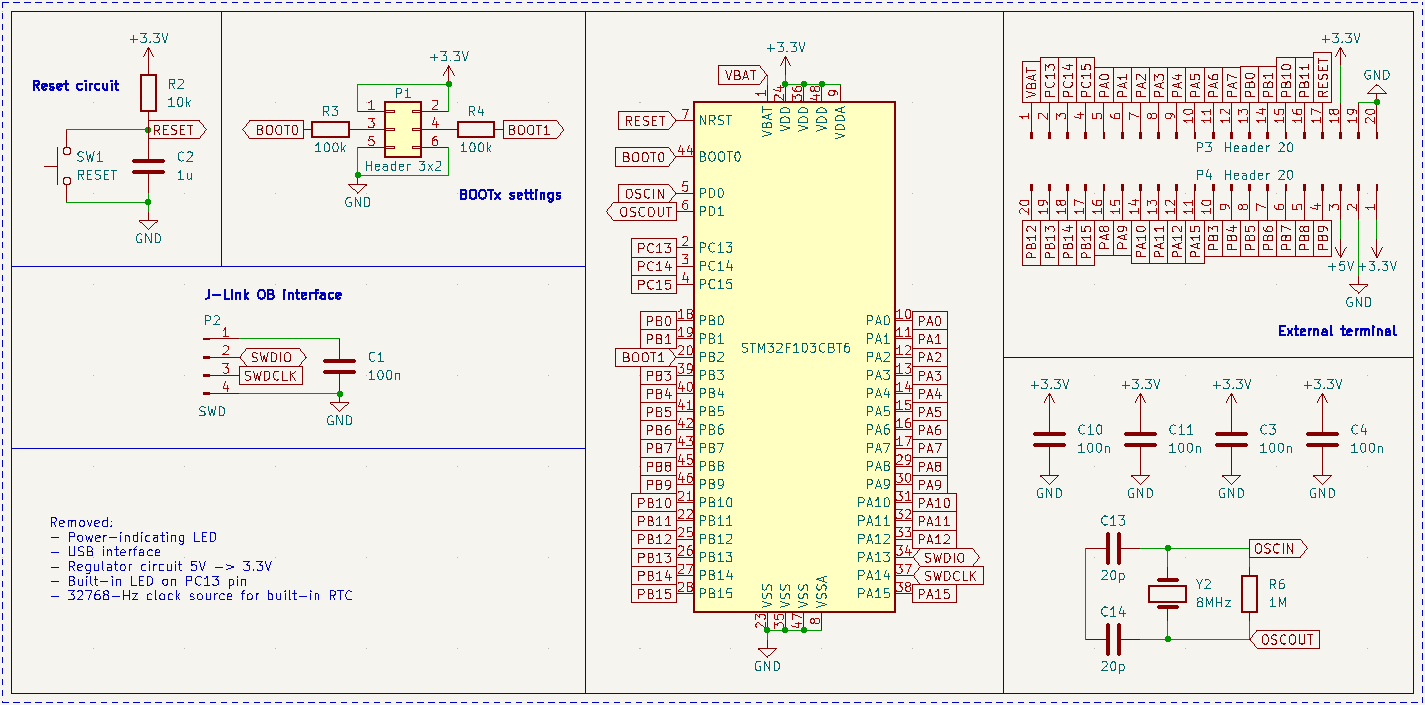


Figure 3‑2. Simplified Blue Pill schematic with removed unused parts

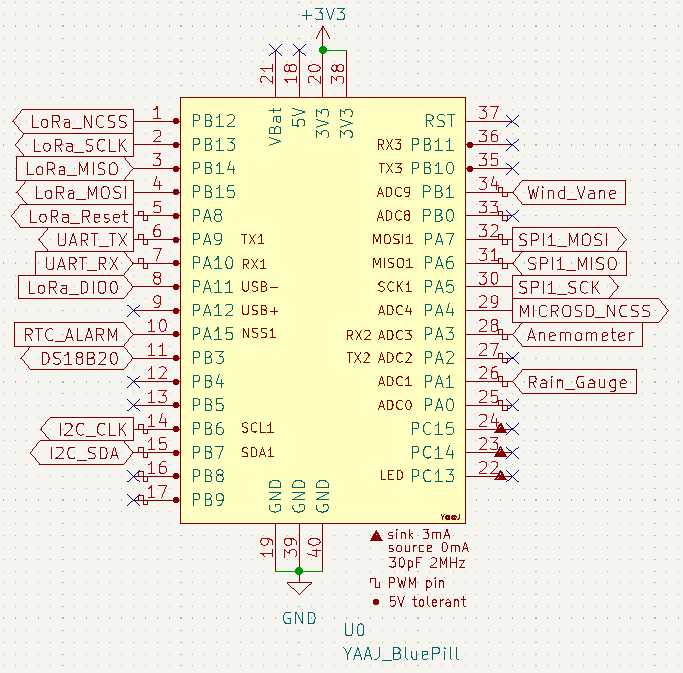


Figure 3‑3. Bus connections with the Blue Pill board

## Sensor units

### Anemometer

#### Hardware dependencies

##### Pull-up resistor

As mentioned in Section 2.2.1, the anemometer could be read through the inner 2 conductors of its RJ11 cable. If it is connected to the wind vane as shown in section 2.2.2, interfacing through the inner 2 conductors of wind vane’s RJ11 cable is done instead. It is known that the electronic part of the anemometer is a mechanical device called the reed switch. As a standalone device, a reed switch does not create any electrical signals to be read, thus the use of a setup as simple as illustrated in Figure 3‑4.

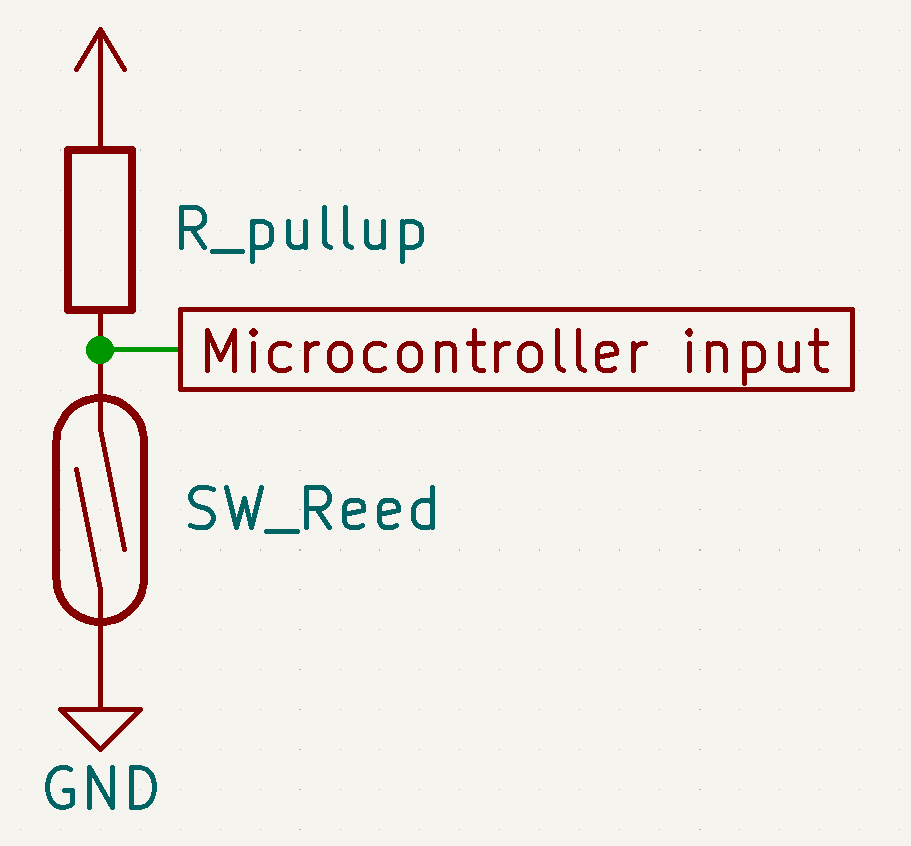


Figure 3‑4. Circuitry for testing the anemometer in ideal conditions

The circuitry in Figure 3‑4 generates active-LOW outputs by the use of the pull-up resistor, which means the microcontroller normally reads logic 1 and detects a logic 0 when the switch closes. Active-HIGH outputs could be achieved by switching the positions of the reed switch (SW\_Reed) and the resistor (R\_pullup); however, since there are potentially faults due to interference while the microcontroller detects logic 1 in that setup, active-HIGH signals are undesirable in this project.

The pull-up resistor value could be chosen for either strong or weak pull-up purpose. A resistor with a high value creates a weak pull-up which results in lower power consumption, and vice versa [53]. If the input impedance on the microcontroller pin is unknown, it is safe to choose a strong pull-up resistor (i.e. 4.7kΩ). However, [9] specifies that a resistor from 30kΩ to 50kΩ could be used to achieve a weak pull-up for an STM32F103CBT6 microcontroller, so the value of 33kΩ is chosen in this project. As a result, the design would be more suitable as a battery-powered application.

##### Bypass capacitor

In [1], [54], it is investigated that as a mechanical switch, a reed switch could produce noise spikes which yield wrong edge detections by a microcontroller, thus wrong data. For example, Figure 3‑5 illustrates a noise instance which could make a microcontroller read 3 falling edges instead of an expected 1 edge within a window of 98.98µs.

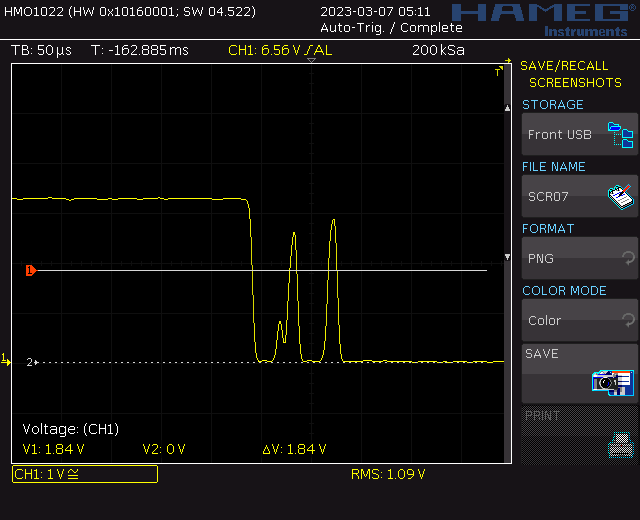
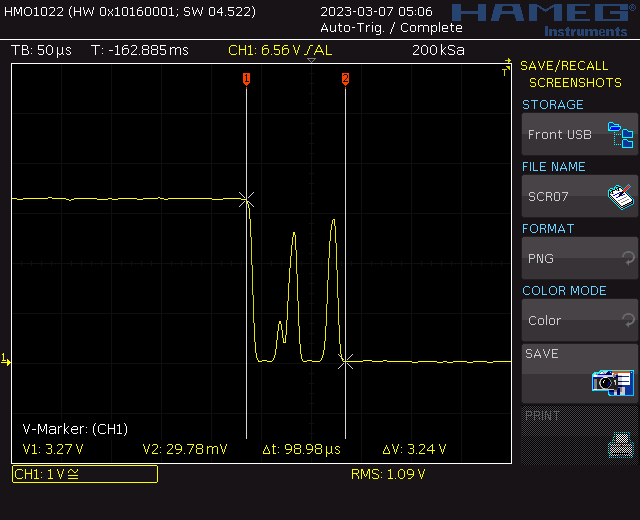


Figure 3‑5. “An instance of noise spikes captured in an oscilloscope” [1]

The solution to a such problem is the implementation of either a LPF to filter out the noise [54] or just a bypass capacitor to debounce the switch. Due to lower edge rise and fall time, and less components involved, the latter is a more preferable method [1]. The choice for a such capacitor is to satisfy that the time constant of the pull-up resistor – bypass capacitor pair is about half of the debouncing time [55].

|  |  |
| --- | --- |
| where | (in seconds) is the time constant of the resistor – capacitor pair, |
| (in Ohms) is the value of the pull-up resistor of the switch circuit, |
| (in Farads) is the value of the decoupling capacitor. |

It is studied that reed switches have bounces up to 4.5ms. Moreover, the pull-up resistor for interfacing the anemometer with the microcontroller is chosen to be 33kΩ. Therefore, the value of the bypass capacitor is . Since a such value does not exist in real life, the closest value of 68nF is chosen for the bypass capacitor instead.

Furthermore, the fact that the pull-up resistor is determined to be 33kΩ relies on the calculation of the bypass capacitor in this part. The bypass capacitor value is derived from a list of manufactored resistor values, which should be available for both testing and prototyping when this thesis is conducted, and is picked from the closest real numbers for a capacitor. Table 3‑1 shows that the resistor – capacitor pair of 33kΩ – 68nF fits the theoratical calculation the most.

|  |  |  |
| --- | --- | --- |
| Real resistor value | Calculated capacitor value | Closest real capacitor value |
| 30kΩ | 75.00nF | 82nF |
| 33kΩ | 68.18nF | 68nF |
| 36kΩ | 62.50nF | 68nF |
| 39kΩ | 57.69nF | 56nF |
| 47kΩ | 47.87nF | 47nF |

Table 3‑1. Choosing bypass capacitor value based on available resistors from 30kΩ to 50kΩ

In order to determine the practicability of the pull-up resistor and the bypass capacitor, a circuit illustrated in Figure 3‑6 is put in use. The reed switch stays attached inside the anemometer and is read from the corresponding pins of the RJ11 connector. The smoothed-out transitions shown in Figure 3‑7 are consistent with the findings in [1] and indicate that the setup is applicable in a real design.

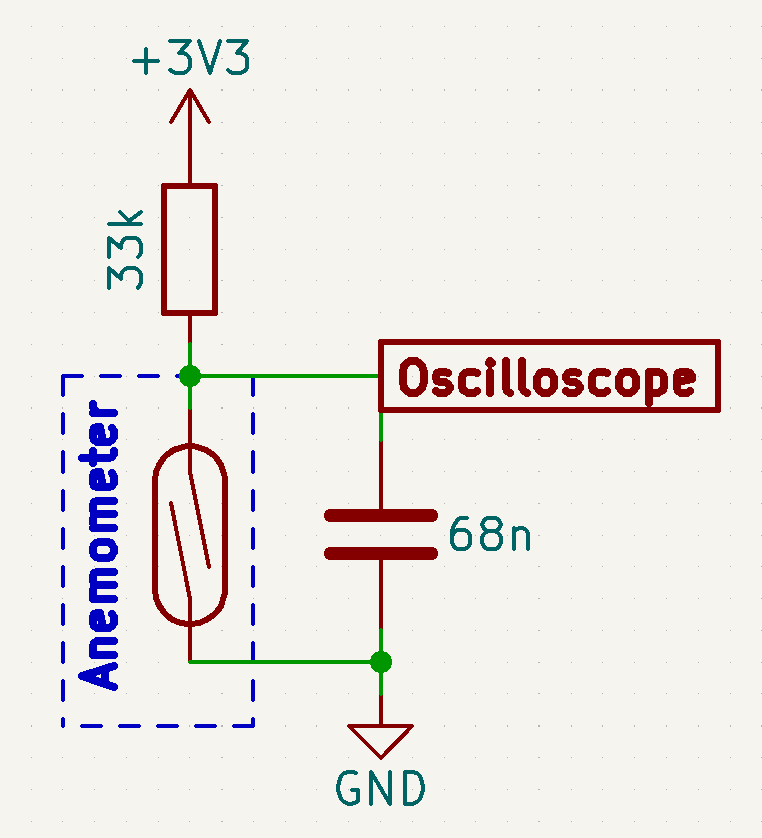


Figure 3‑6. Bypass capacitor test circuitry using the reed switch from the anemometer

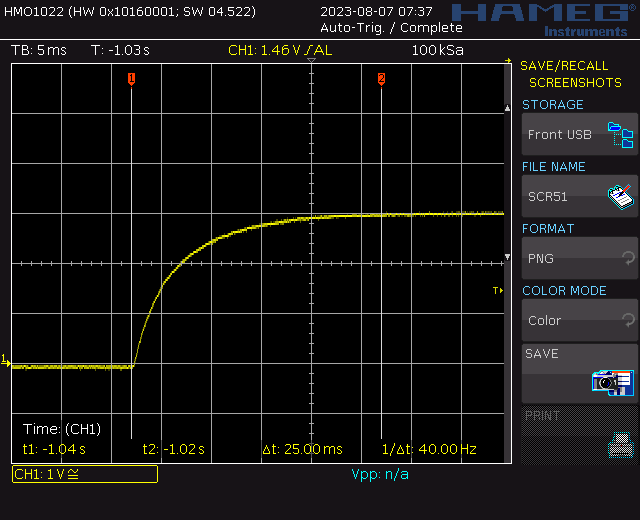
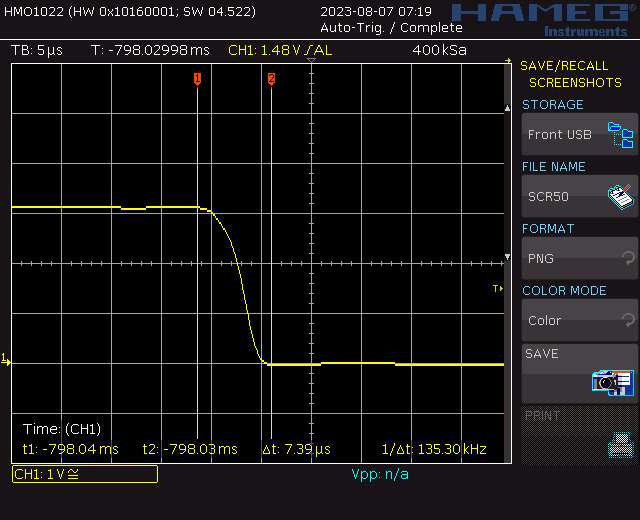


Figure 3‑7. Reed switch debounced by the 68-nF bypass capacitor

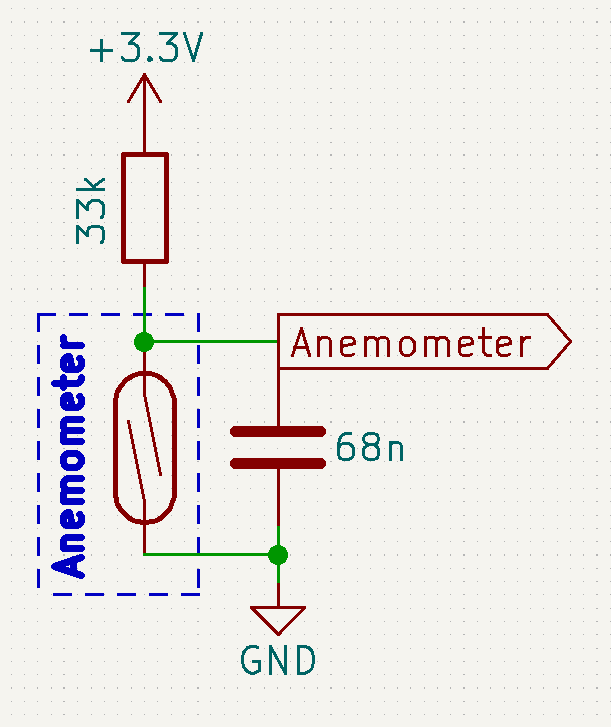


Figure 3‑8. Circuitry for interfacing with the anemometer

#### Software design

[1] determines that the average wind speed monitored via an anemometer is , where is the number of pulses over a period detected from the anemometer by the microcontroller, and is “the radius from the pivot to the edge of a cup” [1], [56]. However, a real anemometer always contain a parameter called the anemometer factor as demostrated by [14]. However, since the determination of the anemometer factor requires a deep study on the aerodynamic characteristics of the cup anemometer itself, the software design shall leave the anemometer factor *K* = 1 and test for the behaviours of the anemometer in Section 4.1.

Since monitoring wind speed is simply counting pulses from the anemometer, the software shall be interrupt-based to avoid blockage of other tasks while keeping the count constantly updated. All the necessary software parts for the anemometer are put into the one-time setup routine on microcontroller’s power-up.

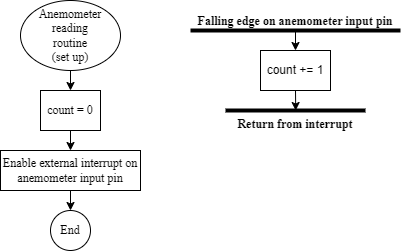


Figure 3‑9. Anemometer setup routine and interrupt service routine upon input event

On the Arduino platform, enabling an external interrupt on the anemometer input pin is done via the built-in attachInterrupt(..) function. After the anemometer input pin is declared as Anemometer\_InputPin, and the count variable is initialised with the value 0, since the hardware designs the anemometer input as active-LOW, the external interrupt is set to detect falling edges by

// Attach an interrupt on anemometer input pin for falling edge detection

  pinMode(Anemometer\_InputPin, INPUT);

attachInterrupt(digitalPinToInterrupt(Anemometer\_InputPin),

anemometerInput\_Detected,

FALLING);

Afterwards, whenever the anemometer rotates and produces a pulse, the interrupt service routine, namely anemometerInput\_Detected(), is automatically called to increment the count variable. On the global scale, the software keeps track of the time and requests all sensor data, including the anemometer’s, once every 5 minutes. As soon as this global event happens, the value of the count variable is stored to a temporary variable, namely count\_temp, and count is reset to 0; count\_temp is used to converting the pulse count to wind speed to respond to the global data request, while count goes on a new counting cycle. By simplifying to is , the latest wind speed in metres per second is returned by a single line

return (count\_temp \* PI \* 0.092 / 300);

where PI is a built-in macro for π on the Arduino platform, 0.092 is the radius in metres of the anemometer (Figure 2‑3b), and 300 is the 5-minute window in seconds. The global sensor data request occurs every 5 minutes because this is the chosen sampling duration for the anemometer to minimise the effects of wind gusts [57].

### Wind Vane

#### Hardware dependencies

##### Voltage divider

In section 2.2.2, it is illustrated that there are 2 circuitries that could be used for the wind vane: a current mirror and a voltage divider. For simplicity, the latter is utilised in this project.

In order to form a voltage divider, an external resistor as shown in Figure 2‑7b is required. The value of a such component in turn needs to satisfy that:

* Current output at the voltage divider is enough for the ADC module to charge its internal capacitor CADC for each conversion.
* Voltage step of the voltage divider values generated by all the wind vane positions is large enough for the microcontroller to distinguish.

Figure 3‑10 illustrates the voltage divider circuit when integrated with the microcontroller’s ADC module. It is assumed that at the beginning of each conversion cycle, the internal capacitor CADC has been fully discharged and the switch (“SW”) is then closed.

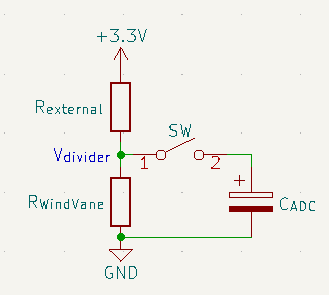


Figure 3‑10. Voltage divider circuitry with ADC module

By applying the source transformation technique, the voltage divider could be changed into a first order RC charging circuit as shown in Figure 3‑11. The Norton equivalent circuit is produced by deriving a Thévenin equivalent circuit of the +3.3V voltage source in series with the resistor *Rexternal*, which yields:

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

The resulted circuitry is then further simplified by combining the parallel resistors into a single value:

At this point, the step response of an RC circuit as given by [58] could be applied:

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

or, based on previously made assumption, and by substituting (3) into (2):

where is the time constant for the RC circuit. It is worth noticing that the voltage up to which the capacitor CADC is charged is:

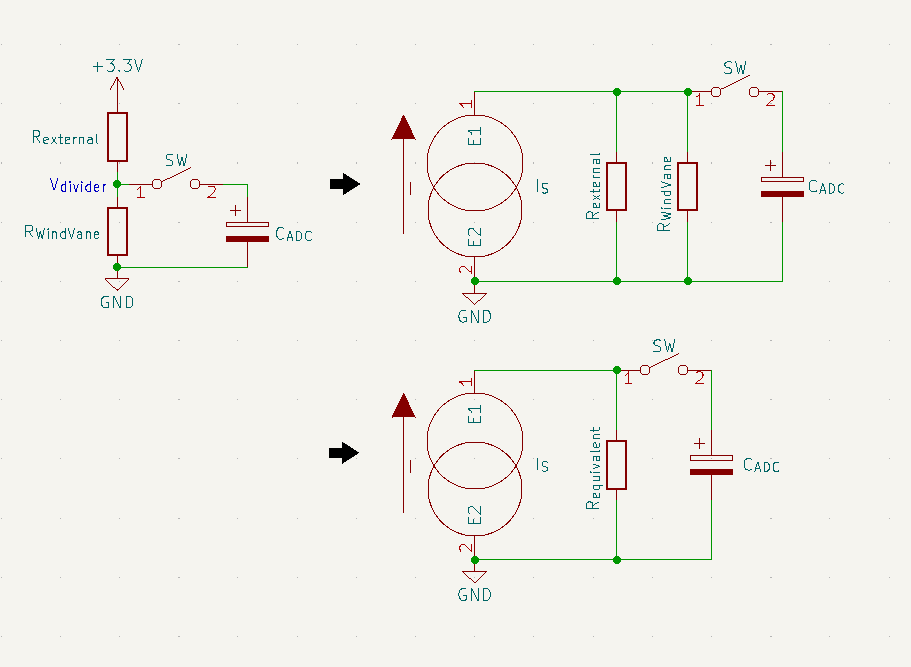


Figure 3‑11. Derivation of the Norton equivalent of the voltage divider

According to the capacitor charging voltage curve shown in [59], CADC would reach its steady state within 4*T* after the switch is closed, and become fully charged at t = 5*T*. It is desirable that the ADC internal capacitor CADC is fully charged during the sampling window of the ADC module. Since the lowest sample rate of the STM32F103C8T6 microcontroller is 1.5 ADC clock cycles [8], there holds a condition for *Rexternal*:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| => |  |  |
| => |  |  |

This project uses an STM32F103CBT6 micrcontroller on the official Arduino core by STMicroelectronics, so the ADC clock frequency could be derived to be 12MHz from the core project on Github [52]. Furthermore, [9] specifies that the internal sample and hold capacitor of the ADC module is guaranteed to be 8pF by design. By substituing the wind vane internal resistance values from Table 2‑1, the condition of is obtained. Afterward, all the manufactored resistor values which meet that condition are put into an Excel sheet to calculate the corresponding voltage step of the voltage divider with a fixed value of +3.3V power supply.

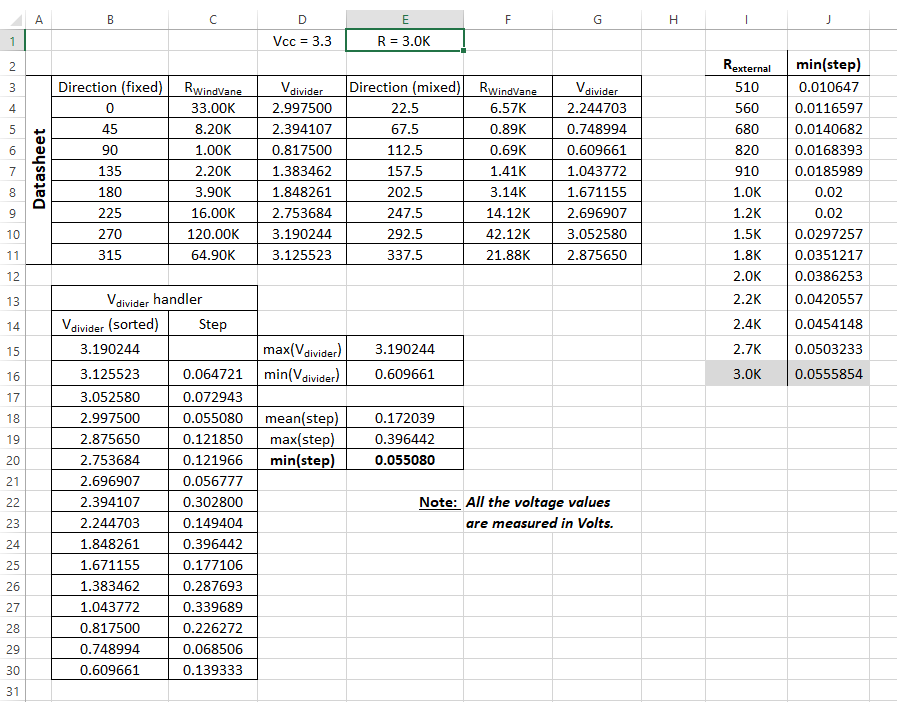


Figure 3‑12. An instance of Rexternal = 3.0kΩ input to the voltage step-calculating sheet

Table 3‑2 records all the minimum voltage steps of the voltage divider for all the resistor values under 3.208kΩ. Since the STM32F103C8T6 microcontroller has 12-bit ADC modules, in theory, it could detect analogue input changes as low as . Therefore, all voltage steps in Table 3‑2 is detectable. However, in practice, there are a number of factors like ADC offset errors, input noise, imprecise analogue reference, etc. that could cause unpredictable behaviours and/or unusable data if an ADC module were applied in ideal conditions. As a result, the largest voltage step by design for an ADC module to detect is always desired, which is 0.0555854V for Rexternal = 3.0kΩ.

|  |  |
| --- | --- |
| **Rexternal (in Ohms)** | **Voltage step (in volts)** |
| 510 | 0.010647 |
| 560 | 0.0116597 |
| 680 | 0.0140682 |
| 820 | 0.0168393 |
| 910 | 0.0185989 |
| 1.0K | 0.02 |
| 1.2K | 0.02 |
| 1.5K | 0.0297257 |
| 1.8K | 0.0351217 |
| 2.0K | 0.0386253 |
| 2.2K | 0.0420557 |
| 2.4K | 0.0454148 |
| 2.7K | 0.0503233 |
| 3.0K | 0.0555854 |

Table 3‑2. Voltage steps of the voltage divider by Rexternal

##### Low-pass filter

It has been mentioned that ADC readings could be different from expected due to input noise, which is inevitable. However, the effects of those unwanted fluctuations could first be reduced by the use of an analogue filter. Since noise is usually high frequency signals, the use of an analogue LPF is desired.

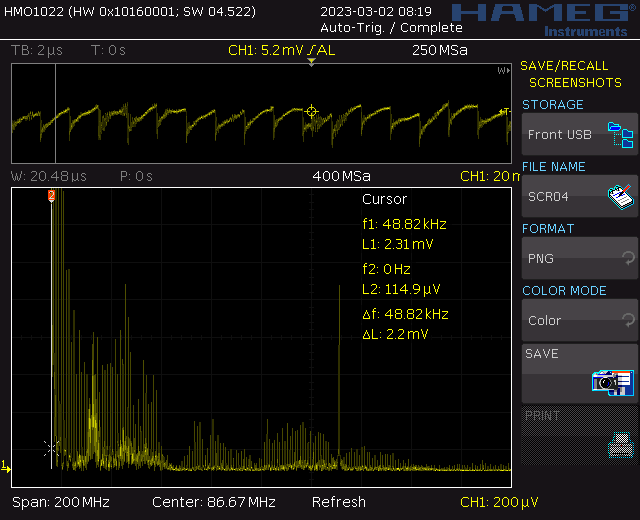


Figure 3‑13. “A noise instance captured during a test on the wind vane” [1]

Designing a LPF firstly requires the cut-off frequency. In order to achieve the best possible ADC accuracy, it is predetermined that the software for reading the wind vane shall utilise the ADC module at the highest sampling time of 239.5 ADC clock cycles [8]. The reviewed STM32duino firmware for the STM32F103C8T6 microcontroller, combined with the reference manual [8] and the STM32CubeMX, gives the ADC clock of 12 MHz and the sampling frequency for the 239.5-cycle sampling time of [52]. Furthermore, [60] states that the sampling frequency should range from 2 to 5 times the frequency limit of the input signal, which results in the cut-off frequency by the LPF to be between 9.524 kHz and 23.810 kHz. The such frequency band is then narrowed down to 9.524 kHz – 12.62 kHz since the latter was observed to be the lowest noise signal when the test for Figure 3‑13 was conducted. A table, a part of which is Table 3‑3, is then constructed with the known manufactured resistor and capacitor values; each LPF option contains only 1 resistor Rfilter, while there could be up to 4 capacitors in parallel for higher capacitance and lower equivalent series resistance (ESR). The final cut-off frequency is chosen based on 2 criteria:

* The theoratical cut-off frequency must be between 9.524 kHz and 12.62 kHz.
* The resistor value must be low to avoid unwanted signal attenuation caused by insufficient input current.

As a result, the LPF option with f-3dB = 11.587 kHz is chosen to be included in the hardware design for the wind vane.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rfilter** | **C1** | **C2** | **C3** | **C4** | **f-3dB** |
| 680 Ω | 10 nF | 4.7 nF | 1 nF |  | 14.908 kHz |
| 10 nF | 3.3 nF | 1 nF |  | 16.367 kHz |
| 10 nF | 4.7 nF | 3.3 nF | 1 nF | 12.318 kHz |
| 10 nF | 4.7 nF | 3.3 nF | 2.2 nF | 11.587 kHz |
| 820 Ω | 10 nF | 4.7 nF | 1 nF |  | 12.363 kHz |
| 10 nF | 4.7 nF | 2.2 nF |  | 11.485 kHz |
| 10 nF | 4.7 nF | 3.3 nF |  | 10.783 kHz |
| 1 kΩ | 10 nF | 4.7 nF | 1 nF |  | 10.137 kHz |
| 10 nF | 3.3 nF | 1 nF |  | 10.130 kHz |
| 10 nF | 2.2 nF | 1 nF |  | 12.057 kHz |

Table 3‑3. A part of the component pick-up table for the passive LPF

Although the contruction of the LPF takes into account the attenuation caused by insufficient current at the ADC input side, the hardware design for the wind vane takes another precaution to avoid any further issues by adding a voltage buffer to make the LPF an active one. Finally, a pull-down resistor is put in parallel with the ADC input of the microcontroller to avoid floating output of the active LPF, completing the hardware design for the wind vane as shown in Figure 3‑14. The choices of the MCP6002-I/P as the operational amplifier (Op-Amp) and the 3.3-kΩ pull-down resistor Rpulldown are based upon the experimental characterisation of the set of options explored in section 4.2.

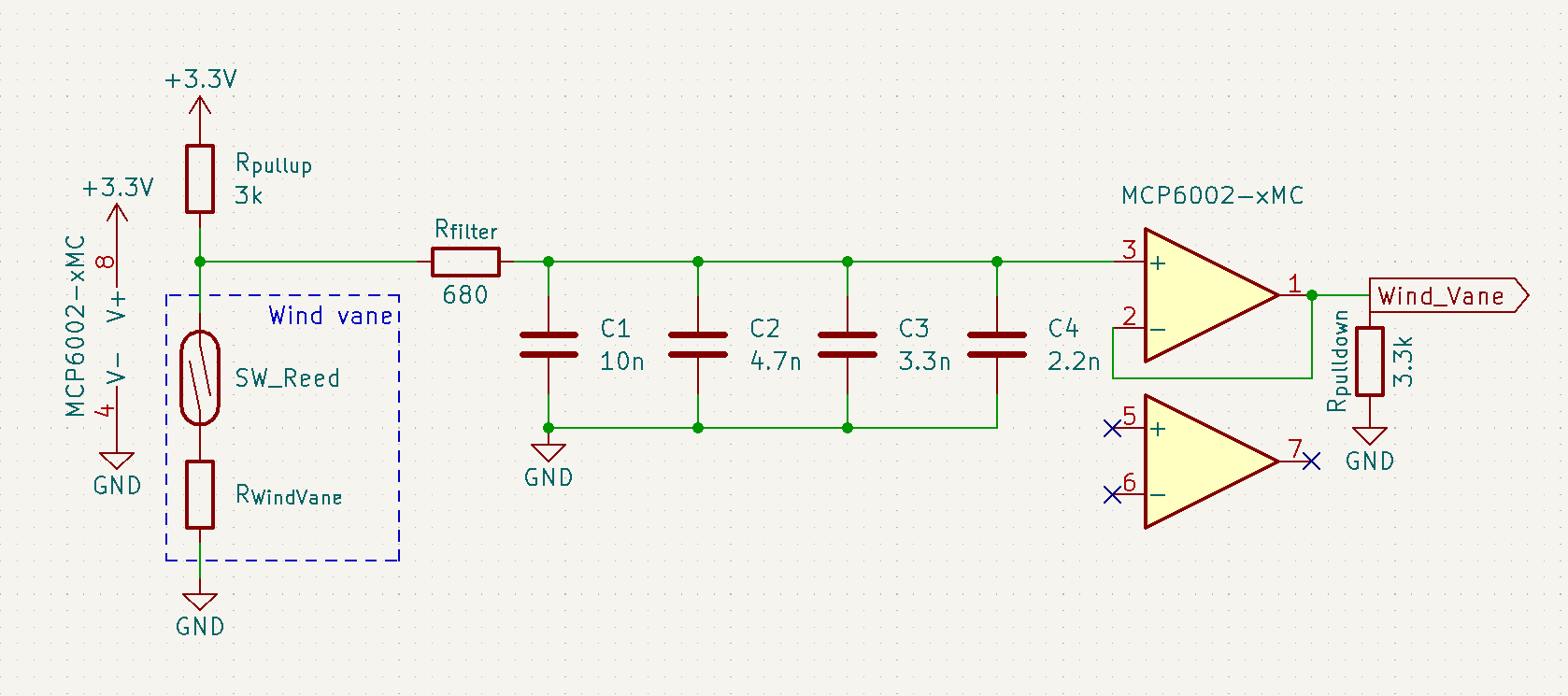


Figure 3‑14. Circuitry for reading the wind vane

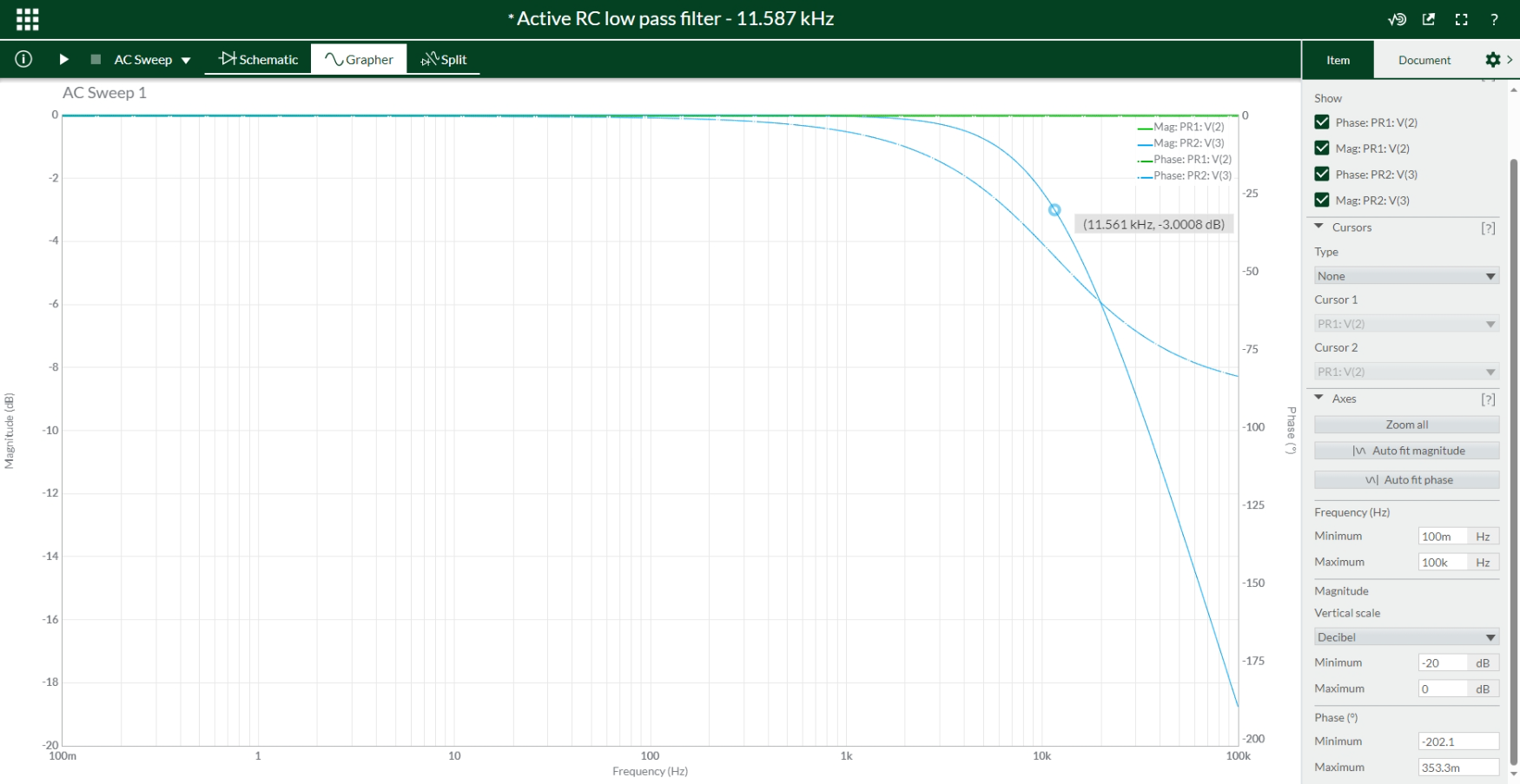


Figure 3‑15. Frequency response of the active 11.587-kHz RC LPF

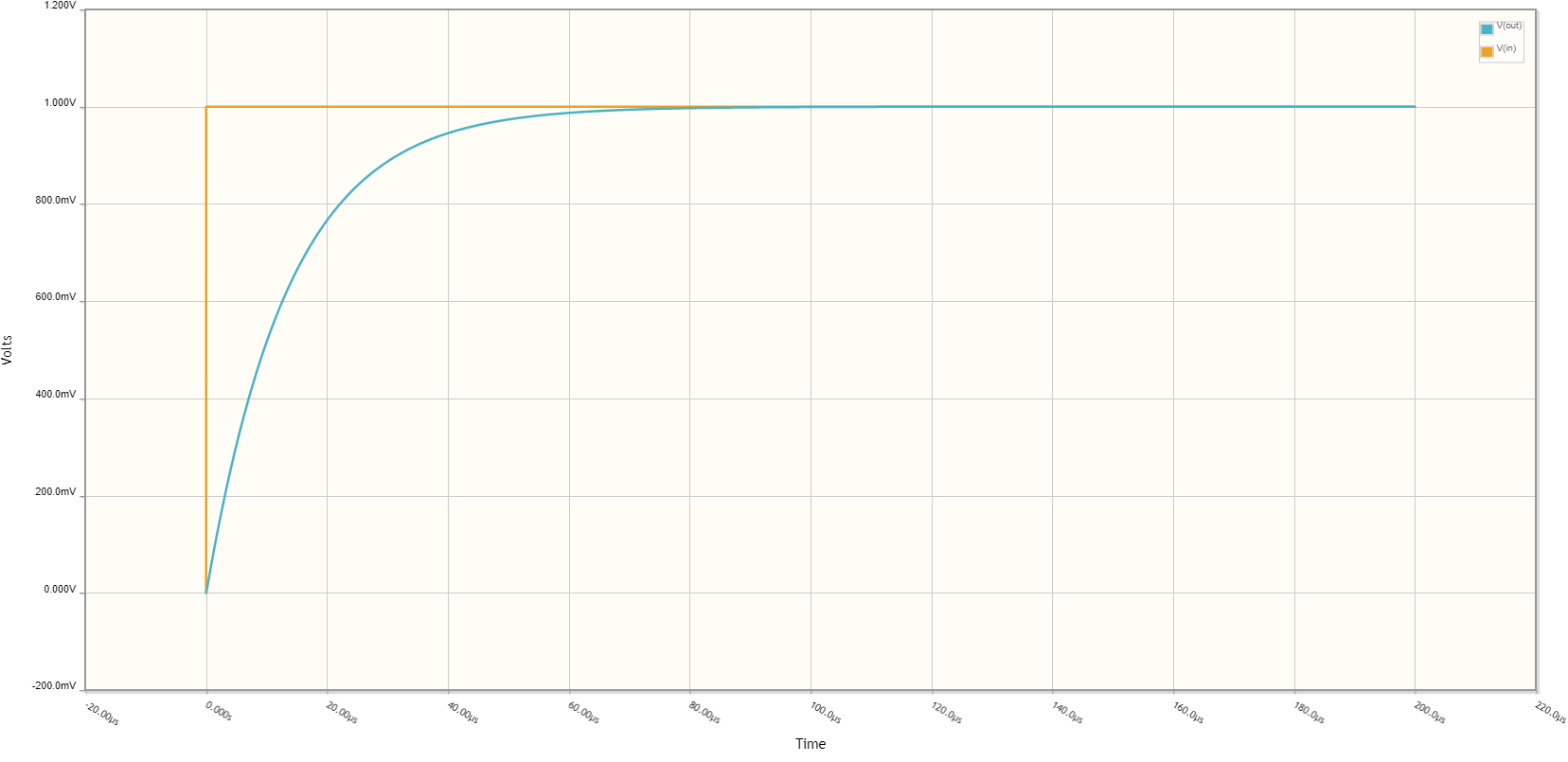


Figure 3‑16. Step response of the active 11.587-kHz RC LPF

#### Software design

Reading the wind direction from the wind vane mostly involves determining the resistive value returned by the sensor, which is done directly via the built-in ADC functionality of the STM32F103C8T6 microcontroller.

Since the wind vane is hooked on the same +3.3V power supply as the microcontroller (Figure 3‑3 and Figure 3‑14), and the STM32F103C8T6 microcontroller uses its power supply as analogue reference [8], at the beginning of each sampling cycle, the apparent value of the +3.3V is checked by the ADC internal reference channel and stored in a temporary variable *Vcc* to overcome the issue of any instabilities on the power supply. Afterwards, the microcontroller continuously samples 21 ADC values from the input by the wind vane, then discards the first entry and takes the mean value of the remaining 20 data points. Since the signal from the analogue ciruitry of the wind vane is essentially a step response, the raw ADC values are expected to remain stable after the settling time, thus the discarded first value.

By applying the formula for the voltage divider, the momentary internal resistor of the wind vane is estimated

where

Finally, the value of is compared with the given values in Table 2‑1. The corresponding Azimuth degree of the given resistor value which is closest to is determined to be the wind direction.

One major part of the software design for the wind vane is the use of the Hardware Abstract Layer (HAL) for ADC and the Direct Memory Access (DMA) controller instead of the simple built-in *readAnalog()* function of the Arduino framework. The DMA controller allows ADC sampling in non-blocking mode, which means the microcontroller could perform other tasks while the sampling operation takes place, and writes the conversion results directly to a pre-defined buffer. The respective functions for HAL and DMA configurations as shown in Appendix A are generated in C language by the STM32CubeMX software and modified to be used in the native C++ language of the Arduino platform. Furthermore, in order for the HAL module to be initialised for ADC during compilation, there must exist a header file named ***hal\_conf\_extra.h*** whose content is a single definition line:

#define HAL\_ADC\_MODULE\_ONLY

The header file ***hal\_conf\_extra.h*** is then included in the main Arduino source file (namely ***Main\_Code.ino*** in this project), following the official instructions from STMicroelectronics [52].

On each wind vane reading cycle, the necessary ADC functions by the STM32duino firmware are shown in Table 3‑4. Upon powered, the microcontroller needs to call the functions in the order of 0-3-1 to initialise the DMA controller and perform a calibration of the ADC module as described in [8]. On each sampling cycle, the function order 2-4-6-7-6-7 is called to read the analogue reference *Vcc*, then 3-4-6-7 for a dummy reading (discarded value). Finally, the main conversion routine takes place by the function order 3-5.

|  |  |  |  |
| --- | --- | --- | --- |
| **Funtion number** | **Function** | **Return type** | **Purpose** |
| 0 | MX\_DMA\_Init(); | None | Initialising the DMA controller for HAL ADC module |
| 1 | HAL\_ADCEx\_Calibration\_Start(&hadc1); | None | Perform a calibration of the ADC module |
| 2 | MX\_ADC1\_Init(INTERNAL\_REFERENCE\_VOLTAGE); | None | Switching ADC to internal channel |
| 3 | MX\_ADC1\_Init(EXTERNAL\_INPUT\_SIGNAL); | None | Switching ADC to external channel |
| 4 | HAL\_ADC\_Start(&hadc1); | None | Initiating a single ADC sampling in blocking mode (used for internal channel when reading analogue reference voltage, and external channel when sampling the then-discarded value) |
| 5 | HAL\_ADC\_Start\_DMA(&hadc1,  (uint32\_t\*)storage,  (uint32\_t)storage\_size); | None | Initiating ADC sampling with the DMA controller. The number of ADC samples (20, in this design) is declared by the storage\_size input parameter. |
| 6 | HAL\_ADC\_PollForConversion(&hadc1, 10); | None | Halting the system for 10 ms to wait for the conversion called by HAL\_ADC\_Start(&hadc1); to finish. The delay time could be changed accordingly to the sampling time. |
| 7 | HAL\_ADC\_GetValue(&hadc1); | uint32\_t | Reading the raw ADC result in blocking mode. This function is called after  HAL\_ADC\_PollForConversion(..); |

Table 3‑4. ADC functions for wind vane reading software

### Rain Gauge

#### Hardware design

Since the electrical parts of the rain gauge and the anemometer are essentially the same, the exact circuitry for the anemometer could be used for the rain gauge.

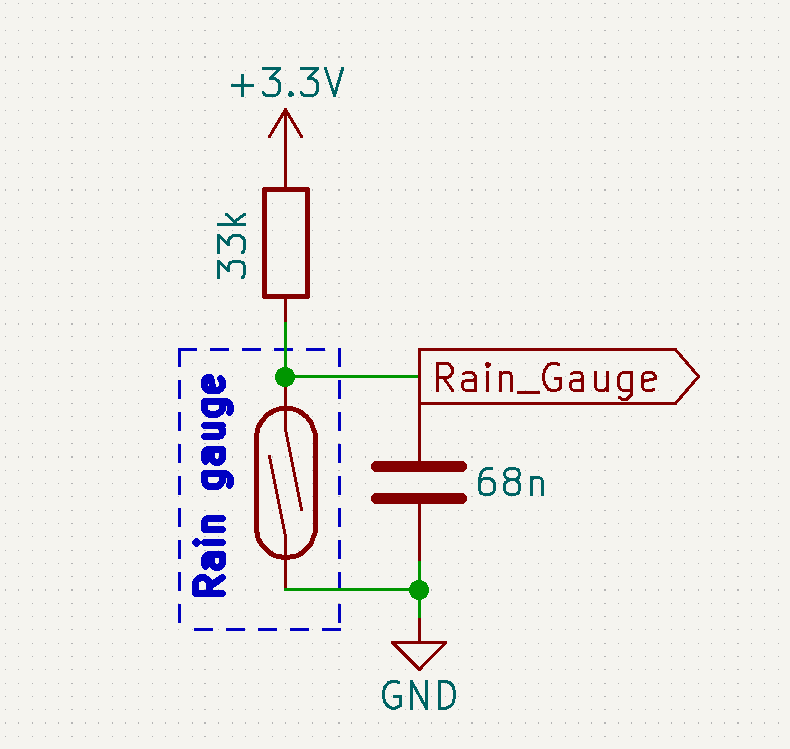


Figure 3‑17. Circuitry for interfacing with the rain gauge

#### Software design

In terms of software, it could be considered that the rain gauge is the anemometer on a bigger scale, since the principle of monitoring rainfall via the tipping bucket is also to count the number of pulses produced over a fixed period of time, only that this period is longer. The typical duration is typically 24, 48, or 72 hours, depending on the site where the observation is done. In this project, the rainfall data is updated daily at 9:00 in the morning.

The software for the rain gauge is also interrupt-based. The setup routine for the rain gauge is called once on microcontroller’s power-up.

// Attach an interrupt on rain gauge input pin for falling edge detection

  pinMode(RainGauge\_InputPin, INPUT);

attachInterrupt(digitalPinToInterrupt(RainGauge\_InputPin),

raingaugeInput\_Detected,

FALLING);

The interrupt service routine for the rain gauge input, namely raingaugeInput\_Detected(), increments the local variable count on a falling edge event on the input pin RainGauge\_InputPin. When the rainfall data is requested, the value of the count variable is stored to a temporary holder count\_temp, then reset to 0. Since the buckets tip once per 0.3 mm of rainfall [17], the rainfall data is returned by

return (0.3 \* count\_temp);

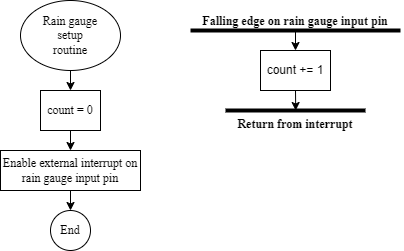


Figure 3‑18. Anemometer setup routine and interrupt service routine upon input event

Unlike other sensors, the software part of the rain gauge does not respond to the 5-minute global sensor data request. Instead, the microcontroller requests for its data specifically as the time keeper announces a daily alarm at 9:00 in the morning to the microcontroller, which is later explored in Section 3.4.1.

### BME280

#### Hardware design

The BME280 is designed with 2 communication interfaces, SPI and I2C, configurable via the shared 4 pins, CSB, SDI, SCK, and SDO [19]. Although the STM32F103C8T6 microcontroller has 2 SPIs, both are reserved for other modules. Moreover, while the number of slaves on each SPI bus is limited by only the number of GPIO pins as SS pins, having multiple devices on the same bus introduces higher current consumption, thus lower communication effectiveness, particularly in low-power applications. The increased power consumption issue by more slaves also exists for I2C buses, but it would be much lower since this interface addresses the slaves by software [61]. As a result, the I2C interface is utilised for BME280 in this thesis.

According to [19], the BME280 is put into I2C mode by keeping the CSB pin “HIGH” at VDDIO; while the SDO pin is pulled either “HIGH” or “LOW” by direct wirings to VDDIO or GND respectively to set the I2C address to 0x77 or 0x76. Since the Arduino library for BME280 chooses 0x76 as the default I2C address for the sensor [62], this thesis designs the hardware as such.

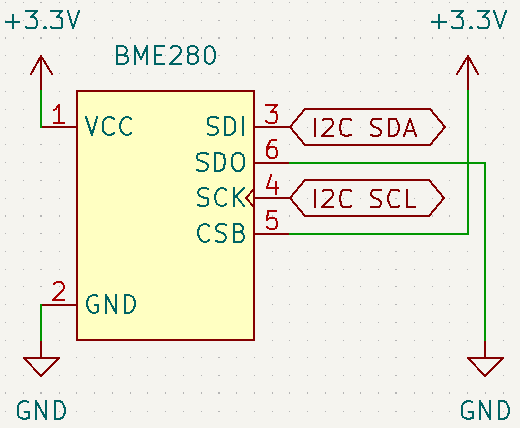


Figure 3‑19. BME280 setup for the I2C protocol

#### Software design

On the Arduino platform, the software handling all the communications with the BME280 sensor could be built upon the “Adafruit BME280 Library”, written by Limor Fried [62] and available both on Github and via the Library Manager of the Arduino IDE.

Before the initialisation of the BME280 sensor, an instance of the class Adafruit\_BME280 must be created. Since the microcontroller communicates with the sensor via I2C, no input parameter is passed in order to call the default constructor of the class:

Adafruit\_BME280 bme280;    // Create an instance to use libraries for BME280

Then, the initialisation of the sensor is done with the begin() method, followed by setSampling(). Since the Autonomous Wireless Agrometeorology Station is to be put outdoor, the suggested settings for weather monitoring by [19] is applied.

// Initialise the BME280 on the I2C bus with the default address

  bme280.begin();

  // Set up the BME280 for weather monitoring

  bme280.setSampling(

    Adafruit\_BME280::MODE\_FORCED,     // Forced mode

    Adafruit\_BME280::SAMPLING\_X1,     // Temperature sampling rate x1

    Adafruit\_BME280::SAMPLING\_X1,     // Pressure sampling rate x1

    Adafruit\_BME280::SAMPLING\_X1,     // Humidity sampling rate x1

    Adafruit\_BME280::FILTER\_OFF       // IIR filter is turned off

  );

Following the anemometer, the BME280 is expected to return the latest readings when there is a global request every 5 minutes. This duration is of no problem since [19] specifies that the weather monitoring mode has the data rate of 1/60 Hz, which means the minimum duration between 2 data sampling by the BME280 is 1 minute. When a request is issued, the microcontroller forces the BME280 sensor to sample the environment parameters, and pulls back the results instantly as soon as the sensor responds with the availability of the readings.

// Issue a data conversion to sensor in forced mode

    bme280\_instance->takeForcedMeasurement();

    // Fetch the latest sensor readings

    temperature = bme280.readTemperature();         // Read temperature in ℃

    pressure    = bme280.readPressure() / 100.0F;   // Read pressure data in hPa

    humidity    = bme280.readHumidity();            // Read humidity data in %RH

### DS18B20

#### Hardware design

In Section 2.2.5, it is shown that a DS18B20 could operate in 2 modes: parasite and local power modes. Due to potential issues caused by software timing and/or electrical delays [1], in this thesis, the implemented DS18B20 temperature sensor shall be powered with an external supply.

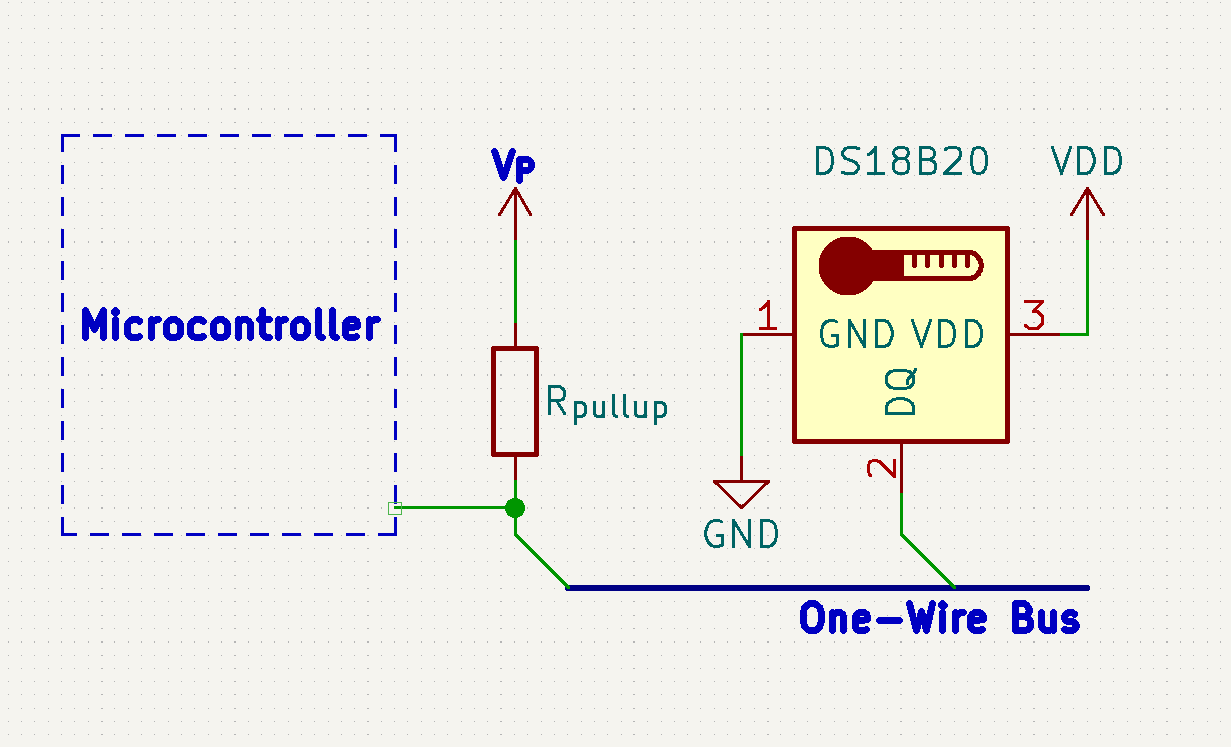


Figure 3‑20. The concept of a DS18B20 powered via an external power supply VP

Given that the DS18B20 communicates via 1-wire protocol, [63] provides the formula to calculate the available current on the bus as

thus the value for the pull-up resistor for the 1-wire bus

Since the DS18B20 is externally powered, the active current ID is provided directly in the Vdd pin, which leaves the “extra power demand” dependent on the input current of the data pin DQ [63]. Maxim Integrated specifies the DQ input current IDQ to be typically 5µA, and the minimum pullup supply voltage to be 3V [22]. The power supply for the DS18B20 is also the 3.3-V supply for the station, which yields the pull-up resistor to be

It is worth noticing that the given DQ input current is the necessary amount for data exchange to be successful, so it could go higher than 5µA but not lower. As a result, the calculated 60-kΩ pullup resistor is the maximum value to achieve 1-wire communication between the microcontroller and the DS18B20 temperature sensor. Since there shall be no other 1-wire devices included in this design, and the active current ID does not contributes in local power mode, the suggested value of 4.7 kΩ for the pullup resistor could be used freely, thus implemented in the design for the Autonomous Wireless Meteorology Station.

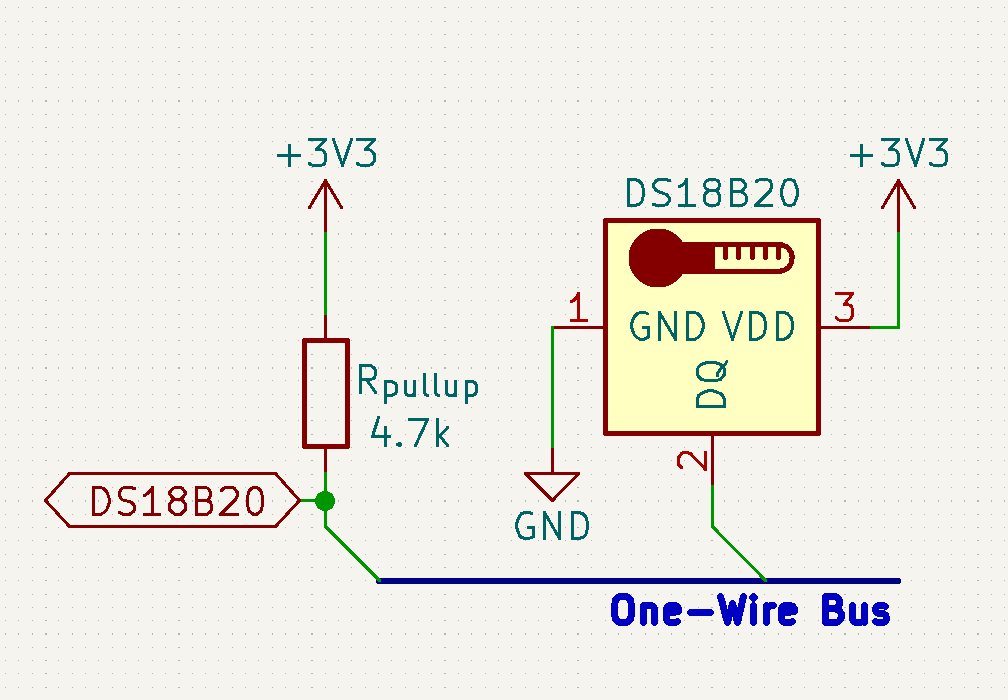


Figure 3‑21. Final circuitry for interfacing the DS18B20 with the microcontroller

#### Software design

There exists multiple Arduino libraries for the DS18B20 sensor, yet this design only uses the “OneWire” library by Paul Stoffregen to handle the 1-wire bus in general. Although it is relatively simple to construct from scratch the firmware for the 1-wire protocol, which is not a built-in of the Arduino platform, a library helps avoid excessive coding.

Like the BME280, the DS18B20 only needs to read the temperature and return its latest data when there is a global request. The software piece contains 2 hard-coded operation procedures as depicted in Figure 3‑22. The setup routine is called once on power-up, while the reading routine is in response to the global sensor data requests. Both routines start with a declaration of a OneWire instance, which is done with a pre-defined GPIO pin DS18B20\_InputPin as the physical pin of the 1-wire bus.

// Declare a OneWire instance on the bus DS18B20\_InputPin

OneWire ds(DS18B20\_InputPin);

Since one of the requirements for the Autonomous Wireless Agrometeorology Station is the temperature at ground level with a resolution of 0.04 ℃, the only thermometer setting that could gives the closest values is the 12-bit resolution for 0.0625 ℃ [22]. Therefore, in the setup routine, there is a byte of 0x7F sent to the DS18B20 by the microcontroller.

In the reading routine, there is no software delay since the microcontroller actively reads 1 bit in a while (..) loop from the sensor after issuing the “Convert T” command. The microcontroller reads 2 bytes from the DS18B20 device to a 2-element byte array data[2]. This is because, out of the 9 bytes of the scratchpad, only the first 2 bytes contains the raw temperature data, all 16 bits of which are used to calculate the temperature in degree Celsius because of the 12-bit thermometer resolution [22]. The temperature is then obtained and returned as response to the global request by

// Obtain the 16-bit raw temperature data

  int16\_t raw = (data[1] << 8) + data[0];

  // Convert from 2's complement to floating-point value

  float temperature = (float)raw / 16.0;

  // Return the temperature as response to global request

  return temperature;

Throughout the procedures of setup and reading routines, the following methods of the OneWire class are used:

|  |  |  |
| --- | --- | --- |
| **Method** | **Example of use** | **Function** |
| .reset() | byte present = ds.reset(); | Performs a reset on the 1-wire bus and returns the presence indicator of 1-wire device(s) on the bus. |
| .write(command) | ds.write(0x44); | Writes a command to the 1-wire device. |
| .read() | byte data = ds.read(); | Reads a single byte from a 1-wire device. |

Table 3‑5. Methods of the OneWire class used in the software for DS18B20

Since the .write(..) method requires a suitable command as the input parameter, Table 3‑6 dives into the ROM and function commands necessary for the software part for the DS18B20 sensor. Although [22] gives a total of 11 commands, this project utilises only 4.

|  |  |  |
| --- | --- | --- |
| **Command** | **Value** | **Description** |
| Skip ROM | 0xCC | Addressing all 1-wire slaves on the bus. Since there exists only 1 device, the DS18B20, “Skip ROM” command makes it faster for data exchange. |
| Read scratchpad | 0xBE | Requesting the content of the sensor’s scratchpad. The data is then transferred from byte 0 in the LSB order every time the master calls a read method. Since the scratchpad contains 9 bytes, the master could only call the read method up to 9 times. |
| Write scratchpad | 0x4E | Overwriting the existing temperature alarm thresholds and thermometer resolution configurations of the DS18B20. Every time the “Write scratchpad” command is issued, the master must write exactly 3 corresponding bytes; otherwise corruption of data may occur. |
| Convert T | 0x44 | Initiating a temperature conversion by the DS18B20 sensor. |

Table 3‑6. ROM and function commands in use for the DS18B20

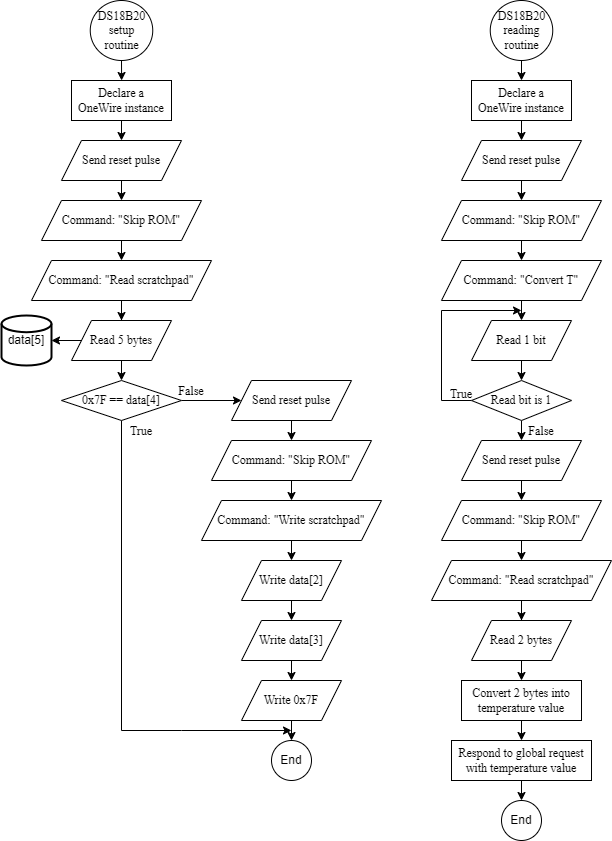


Figure 3‑22. Setup and reading routines for DS18B20

## SX1278

##### Hardware design

Since this design makes use of SX1278 pinout module, further circuitries are not necessary to establish hardware connections. In the design of the Autonomous Wireless Agrometeorology Station, all the communication and control pins of the LoRa module could be connected directly to the corresponding pins of the STM32F103C8T6 microcontroller because of the similar 3.3-V logic level. If the microcontroller and the LoRa module share a voltage regulator as power source, there may be occasions when the latter puts stress on the regulator by drawing too much current during a transmission, resulting in unexpected behaviours like sudden voltage drops and undesired resets, or even damages to other components and devices. In order to prevent such issues, a separate voltage regulator is setup for the SX1278 LoRa module.

Apart from the SPI connections, the SX1278 LoRa module requires 2 more pins from the microcontroller for interrupts and resets. The interrupt pin is controlled by the module to inform the microcontroller of incoming data communication, while the LoRa\_Reset signal line is required and controlled by software.

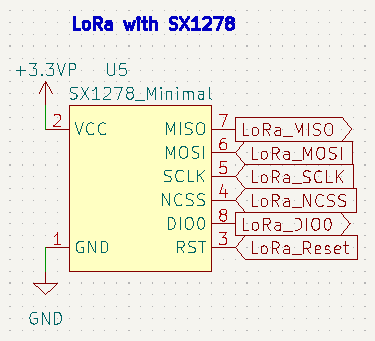


Figure 3‑23. Hardware setup for SX1278 LoRa module

##### Software design

LoRa and LoRaWAN are 2 separate concepts. LoRa refers to the technology of wireless modulation, thus the physical layer of the LoRa Network [64]. LoRaWAN, on the other hand, is a set of protocols developed on top of the LoRa physical layer, also known as the Media Access Control (MAC) layer [64]–[66]. An SX1278 module allows a system to join a LoRa network as a node for transmission of data; however, data exchange could only be done between a node and a gateway as illustrated in [34], [66] because of the star topology by LoRaWAN [64], [67].

In this design, 2 SX1278 modules are supposed to establish communications between the sensor station and a server. Without a LoRa gateway, a such task is unachievable. However, a workaround could be done by mimicking some behaviours of a LoRa gateway at the server side by software.

The software is built on the Arduino library “arduino-LoRa” by Sandeep Mistry. At the initialisation step, the LoRa frequency band on which communication is established must be chosen. Not all the frequency bands of LoRa are available to use due to the Regional Parameters of LoRaWAN [68]. According to [69], [70], the frequency plan in Vietnam is AS923-925 (“AS2”), which ranges from 923.2 MHz to 924.8 MHz [71] and is to be avoided since permission to use is not available. [66] states that the 433 MHz frequency band could be used without a license in Vietnam, so it is put in software setup at both the server side and the sensor station.

Upon choosing the LoRa frequency, the LoRa module is ready to be used in the software design. After the LoRa library is included, the setup parameters for the SX1278 LoRa module is defined. The spreading factor, signal bandwidth, and coding rate are chosen to be 12, 500 kHz, and 4/8 respectively for the maximum data transfer rate. The sync word is picked randomly to be 0x92.

#include <SPI.h>    // Built-in SPI library for LoRa

#include <LoRa.h>   // LoRa library handling SX1278

// LoRa setup parameters

const long frequency = 433E6;   // LoRa frequency

int spreadingFactor = 12;       // LoRa spreading factor

long signalBandwidth = 500E3;   // LoRa signal bandwidth

int codingRate4 = 8;            // Denominator for LoRa coding rate

int syncWord = 0x92;            // Sync word for LoRa communication

Since the LoRa module introduces some more physical connections with the microcontroller, those pins must also be defined to use used in software.

// LoRa module hardware dependencies

const int csPin = PB12;    // SPI NCSS for LoRa

const int resetPin = PA8;  // LoRa reset

const int irqPin = PA11;   // Interrupt by LoRa

It could be seen from the hardware design with the SX1278 LoRa module that the SPI2 port on the STM32F103C8T6 microcontroller is utilised for LoRa. However, the SPI2 is not defined by the STM32duino firmware [52], so it must be user-defined. Firstly, the physical pins of the SPI2 port are to be declared.

uint32\_t SPI2\_MOSI\_Pin = PB15;  // SPI2 MOSI pin

uint32\_t SPI2\_MISO\_Pin = PB14;  // SPI2 MISO pin

uint32\_t SPI2\_SCLK\_Pin = PB13;  // SPI2 SCLK pin

The LoRa library initiates a LoRaClass instance by design, so the LoRa name of the instance could be used directly. After the declaration of parameters and functional pins comes the software setup routine as shown in Figure 3‑24. Before the initialisation of the LoRa module, the LoRa object must know which SPI port it is going to use and what the physical pins between the LoRa module and the microcontroller is, thus the assignments:

// Create an SPI object on the SPI2 pins

  SPIClass\* NewSPI = new SPIClass(SPI2\_MOSI\_Pin, SPI2\_MISO\_Pin, SPI2\_SCLK\_Pin);

  // Assign the newly created SPI object to LoRa object

  LoRa.setSPI(\*NewSPI);

  // Assign LoRa pins

  LoRa.setPins(csPin, resetPin, irqPin);

after which comes the first attempt to initialise the physical LoRa module, where LoRa\_InitStatus is a pre-defined variable of int type storing the initialisation result of the LoRa itself.

LoRa\_InitStatus = LoRa.begin(frequency);

From this point, the LoRa\_InitStatus variable is used with an if clause to guard the software, in case the initialisation fails. For example, within the “Initialise LoRa module at 433MHz” step, at the stage of pushing LoRa setup parameters to the LoRa module, the following operations are done with LoRa\_InitStatus guard:

if (LoRa\_InitStatus) {

    LoRa.setSpreadingFactor(spreadingFactor);

    LoRa.setSignalBandwidth(signalBandwidth);

    LoRa.setCodingRate4(codingRate4);

    LoRa.setSyncWord(syncWord);

  } else {

    // Make another initialisation attempt

    LoRa\_InitStatus = LoRa.begin(frequency);

  }

Then, excluding the guard, the LoRa driver requires external definitions for .onReceive() and .onTxDone(onTxDone) routines by

LoRa.onReceive(onReceive);

  LoRa.onTxDone(onTxDone);

where the 2 functions onReceive() and onTxDone() must be written separately at the end of the source code of the software for station:

void onReceive(int packetSize) {

  String rx\_message = "";

  while (LoRa.available()) {

    rx\_message += (char)LoRa.read();

  }

}

void onTxDone() {

  Serial.println("TxDone\n");

  LoRa\_rxMode();

}

Since the Autonomous Wireless Agrometeorology Station is designed to only send data uplink to a server via a gateway, the onReceive(..) function does not handle any notifications or received messages to the main program. The rx\_message variable is put in the software merely to help clear the LoRa buffer should any messages arrive somehow.

Although the station is only designed to send sensor data, there requires 2 additional functions to be defined, 1 to set the LoRa module in receive mode, and the other to set the LoRa module in transmit mode:

// Set LoRa in receive mode

void LoRa\_rxMode() {

  LoRa.enableInvertIQ();   // active invert I and Q signals

  LoRa.receive();           // set receive mode

}

// Set LoRa in transmit mode

void LoRa\_txMode() {

  LoRa.idle();               // set standby mode

  LoRa. disableInvertIQ ();    // normal mode

}

Like onReceive() and onTxDone(), these 2 functions must be put at the end of the source code for the station. Since the SX1278 allows setting LoRa I and Q signals [42], the behaviours of a simple LoRa node could be mimicked, and the functions LoRa\_rxMode() and LoRa\_txMode() must handle the I and Q signals of the LoRa modules. For the node, the I and Q signals follows [72]:

* Transmit mode: Normal I and Q signals (.disableInvertIQ())
* Receive mode: Inverted I and Q signals (.enableInvertIQ())

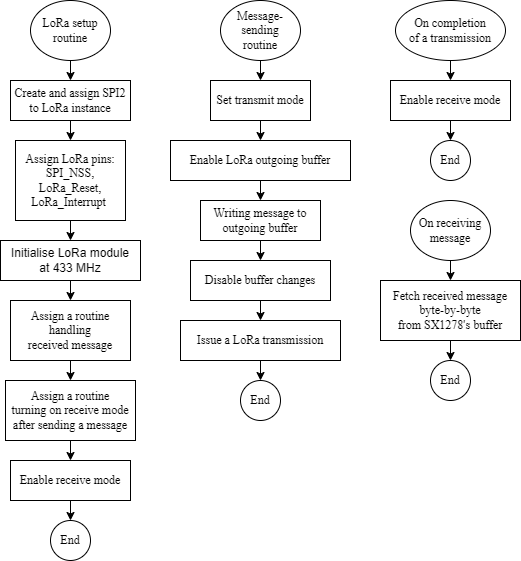


Figure 3‑24. SX1278 software setup and message-handling routines

## Other Modules

### DS3231SN

#### Hardware design

Since the DS3231SN pinout module contains all the necessary components for the RTC IC, there is no need for further circuitries. It does not matter whether the DS3231SN is powered at 3.3 V or 5 V, since the I2C pins of the STM32F103C8T6 microcontroller are 5-V tolerant [9]. However, if the RTC module is powered at 5 V, the minimum logic “HIGH” input is , which is higher than the logic level of the microcontroller. Furthermore, sharing the I2C bus is another device with the maximum input/output voltage of 3.6 V, the BME280 sensor [19], 5-V power supply is not ideal for integrating the DS3231SN with the system. Therefore, the RTC module is to be powered at 3.3 V, and the backup battery is chosen to be a non-rechargeable CR2032. The removal of either the 1N4148 or the 200-Ω resistor is not necessary in this setup since the difference between power supply and the battery voltage is not enough for the diode to conduct and damage the battery.

The SQW of the RTC module is connected to an interrupt input of the microcontroller under a signal named to provide the microcontroller with any RTC alarm events, while the 32K pin is wired to a header to allow checking the compensated 32768-Hz clock of the RTC device.

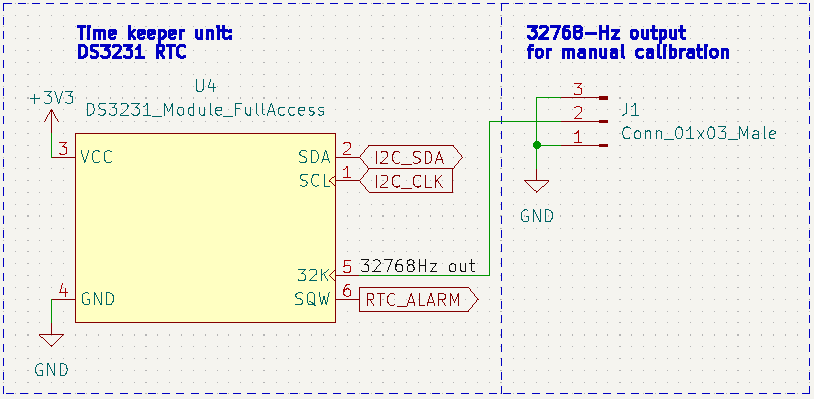


Figure 3‑25. Hardware setup with the DS3231SN RTC module

#### Software design

On the Arduino platform, communications with the DS3231 RTC module could be handled with the “DS3231” library by Andrew Wickert. The library itself is not a built-in, but could be installed via Library Manager of the Arduino IDE.

In order to use the library, it first needs to be included in the software:

#include "DS3231.h"

The library provides 2 classes, DS3231 to handle the settings of the DS3231 module, and DateTime to access date and time data. The objects for the 2 classes could be initialised before the set-up routine of the main software, or they could be temporarily called to access the RTC module. This software design follows the latter.

The first functionality of the library to be explored is setting the DS3231 up with a specific date and time. For example, if the date and time are desired to be 11 September 2023 at 9 sharp in the morning, the software calls:

DS3231 clock;     // Initiate a DS3231 object

  clock.setClockMode(false);  // Set clock mode to 24h

  clock.setYear(2023);        // Set year: 2023

  clock.setMonth(9);          // Set month: September

  clock.setDate(11);          // Set date: 11th

  clock.setHours(9);          // Set hour: 9

  clock.setMinute(0);         // Set minute: 0

  clock.setSecond(0);         // Set second: 0

After the clock object of the class DS3231 is initiated, any setting method could be called to set a specific field to the DS3231 chip without the need to call all the other setting methods.

In the software design for the Autonomous Wireless Agrometeorology Station, the system is expected to issue a global sensor data request once every 5 minutes, which could be helped by the DS3231 module. The alarm 2 of the RTC is to be initialised during the setup routine on the global scale, which goes off every minute, and the microcontroller then recognise the 5-minute windows every 5 rising edge on the RTC\_Alarm pin.

pinMode(RTC\_Alarm, INPUT\_PULLDOWN);

  DS3231 clock;

  clock.turnOffAlarm(2);

  clock.setA2Time(0, 0, 0, 0x70, false, false, false);   // Alarm every minute

  clock.turnOnAlarm(2);

  clock.checkIfAlarm(2);

  // Enable interrupt from alarm 2

  attachInterrupt(digitalPinToInterrupt(RTC\_Alarm), Alarm\_Callback, RISING);

Moreover, there is a specific device, the rain gauge, that needs to return data daily at 9 in the morning. Therefore, the alarm 1 of the RTC is dedicated for this purpose.

DS3231 clock;

  clock.turnOffAlarm(1);    // Disable alarms

  clock.setA1Time(1, 9, 0, 0, 0x8, false, false, false);    // Set up daily   
 // alarm at 9am

  clock.turnOnAlarm(1);     // Enable alarms

  clock.checkIfAlarm(1);    // Make sure there are no alarms present initially

The previously set-up alarm 2 already initiate the interrupt function of the RTC\_Alarm pin, so the function attachInterrupt() does not need to be called again. If the alarm settings for alarms 1 and 2 are put in the same scope, the line DS3231 clock; is to be omitted.

For requesting date and time data from the RTC, the procedure involves initiations of a DS3231 object, then a DateTime object. Afterward, each date time field is to be read separately.

DS3231 myRTC;

  DateTime now = myRTC.now();

  int dayValue = now.day();       // Day of month

  int monthValue = now.month();   // Month

  int yearValue = now.year();     // Year

  int hourValue = now.hour();       // Hour

  int minuteValue = now.minute();   // Minute

  int secondValue = now.second();   // Second

Earlier, it is mentioned that the alarms produce an external interrupt on the RTC\_Alarm pin, so an ISR must be defined at the end of the source code to handle these alarms. The custom function Alarm\_Callback acts as the ISR which checks the alarm type from the RTC for suitable global actions, by first initiating a DS3231 object

DS3231 alarm;

then check the alarms of the DS3231 module one by one with alarm.checkIfAlarm(1) and alarm.checkIfAlarm(1). These 2 lines return the conditional parameter true or false, which could be used with if clauses.

### microSD Card

#### Hardware design

Since the microSD card is interfaced with the STM32F103C8T6 microcontroller via a microSD card module, no further circuitry is required apart from the direction pin connections.

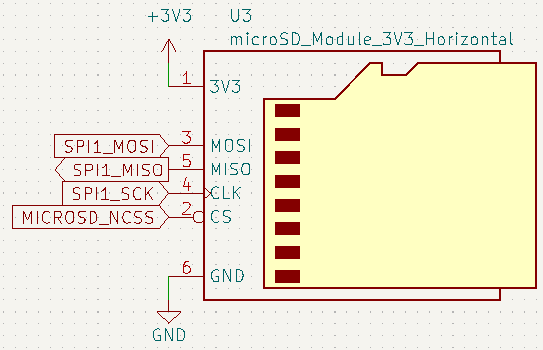


Figure 3‑26. Physical pin connections of the microSD card module with the microcontroller

#### Software design

On the Arduino framework, there is a built-in library SD.h that allows accessing and managing files on any type of Secure Digital (SD) card, including microSD cards. Therefore, no external library needs installing before the software piece for the microSD card module is contructed. On the global scale, after including the library declaring the slave select pin for the micro SD card, and initialising an file-managing object, the library is ready to be used.

// Include the built-in SD library

#include <SD.h>

// Declare slave select pin for SPI

uint32\_t CSPin = PA4;

// Initialise a file-handling object

File cardData;

Everytime the log file (namely STATION\_LOG.CSV) is to receive a new data line entry, the microcontroller simply attempts to open the microSD card and write the content of entry to the designated log file STATION\_LOG.CSV. If the file named STATION\_LOG.CSV does not exist, the .open(..) method creates a file that file by default. In case the microSD card cannot be accessed, or if there is a problem with the designated file to be written, the File class provides a mechanism which passes the value 0 to cardData and prevents any further complications

cardData = SD.open("STATION\_LOG.CSV", FILE\_WRITE);

    if (cardData) {

        cardData.println(entry);

        cardData.close();

    } else {

        // For debugging only

        Serial.println("Error accessing microSD card.");

    }

## System Powering

In theory, powering a system from 2 sources simultaneously is possible if the sources are independent and produce similar voltage outputs. However, in this off-grid system, the battery unit is dependent on the photovoltaic cells to be charged. Therefore, the main system can be powered by either the solar panel or the batteries at a time.

An ideal circuit of the powering unit for the sensor station should be able to switch the sources without introducing any downtime to the sensor station. The block diagram of a such circuit is as follows.

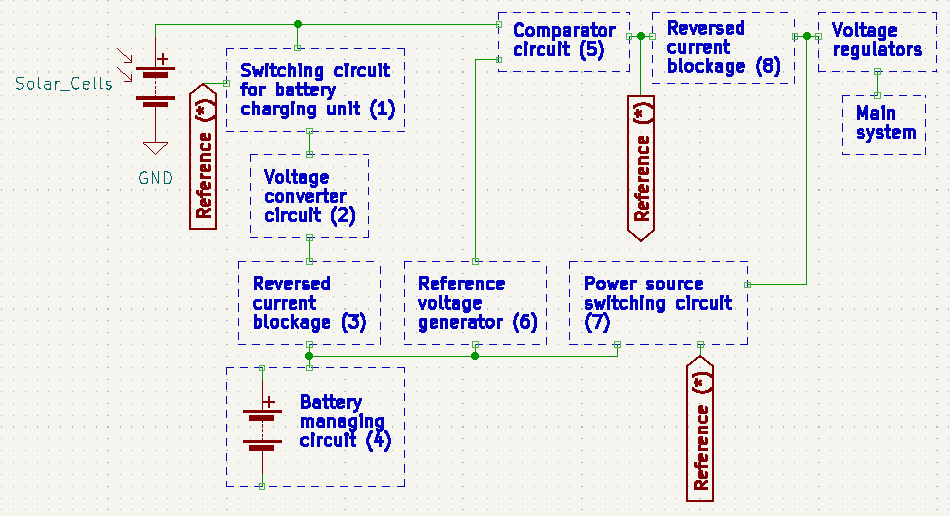


Figure 3‑27. Block diagram of the system powering circuitry

The expected operations of the circuit are divided into 3 phases:

* The solar panel charges the batteries and powers the main system simultaneously when it produces a voltage above the charging voltage of the batteries.
* The solar panel only powers the main system when it produces a voltage above the minimum input voltage of the voltage regulators, but below the charging voltage of the batteries.
* The solar panel is electrically disconnected when it outputs a voltage below the minimum input for the voltage regulators; the batteries take over the line and powers the main system instead.

The Reference (\*) is the voltage of the solar panel injected into blocks (1) and (7). In the first phase, Reference (\*) enables block (1), allowing a current flow into block (4) to charge the batteries. At the same time, Reference (\*) disables block (7) whose function is to create a bridge for the battery current in a later phase. In this phase, block (6), Reference voltage generator, has the input of solar panel through blocks (1), (2), and (3), and produces a voltage reference to the comparator circuit (5); the reference by block (6), however, is lower than the direct input of the solar panel to block (5) which results in current continuing to flow from the solar panel to the main system. A side note is that block (6) must always generate a voltage equal to the minimum input voltage of the voltage regulators.

In the second phase, despite the Reference (\*) enables block (1) for a current flow, block (2), Voltage converter circuit, expectedly outputs 0 V due to insufficient input voltage from the solar panel. As a result, the current from the solar panel only flows on 1 path and no longer flows toward block (4). Block (6) now has the input from the batteries, but its reference for block (5) remains lower than the solar panel voltage, so the path from the solar panel to the main system persists. In this phase, block (3) prevents the reversed current flow from the batteries to the voltage converter circuit (2), thus protection against any potential damages.

In the third phase, the reference by block (6) is greater than the input voltage of the solar panel to block (5), so block (5) disrupts the path from the solar panel to the main system. Expectedly, Reference (\*) drops to 0 V and disables block (5), which makes the solar panel totally isolated. As opposed to the first phase, block (7) is enabled due to the absence of Reference (\*) voltage and provides a current path from the batteries to the voltage regulators and the main system. In this phase, block (8) prevents Reference (\*) from having the voltage of the batteries to mimic its behaviours in the first and second phases.

Since this remains a concept, no circuitry has been built and tested. In order to realise the idea, the components need picking and studying carefully, due the sensitive nature of voltage comparison in the concept. For the time being, 4 things that are clear are:

* The main system is expected to run at 3.3 V, so the voltage regulators should have the input of at least 3.6 V for some components. In case of AMS1117-3.3, the typical input is 3.3 V + 1.1 V = 4.4 V [73]. This voltage needs to be the output of block (6), Reference voltage generator.
* The “Reversed current blockage” blocks (3) and (8) have to have low forward voltage drops as well as low reversed leakage currents for the maximum protection of other components/blocks.
* The concept works in 3 phases if the solar panel has the maximum voltage output higher than the charging voltage of the batteries. In turn, block (2), Voltage converter circuit, is essentially a buck converter.
* The switching circuits of blocks (1) and (7) need to be fast so as not to introduce any downtime to the main sensor station. Electrolytic capacitors shall be needed to compensate for the temporary power drop during switching periods.

## Gateway and Server

##### Hardware design

As mentioned in Section 2.6, a LoRa gateway is built upon a nodeMCU to push the read sensor data to an online database, ThingSpeak. In this project, there is no indepth hardware design for the gateway since it does not contain multiple parts other than the nodeMCU ESP8266 and a LoRa module SX1278. Since it is expected to positioned indoor near a Wi-Fi source, the gateway is to be powered by a 9-V wall adapter. The nodeMCU ESP8266 has an on-board voltage regulator for 3.3V, so the 9-V supply is fed directly to its Vin pin. The LoRa module, on the other hand, is powered via an LM317 circuit demonstrated in Section 3.5 for the station. The physical connections are depicted in Figure 3‑26. There have been attempts to try switching the LoRa\_RST and LoRa\_DIO0 pins to D3 and D4 pins on the nodeMCU, but these physical pins take part in the boot process of the ESP8266 microcontroller [74] and could not work on any test for LoRa.

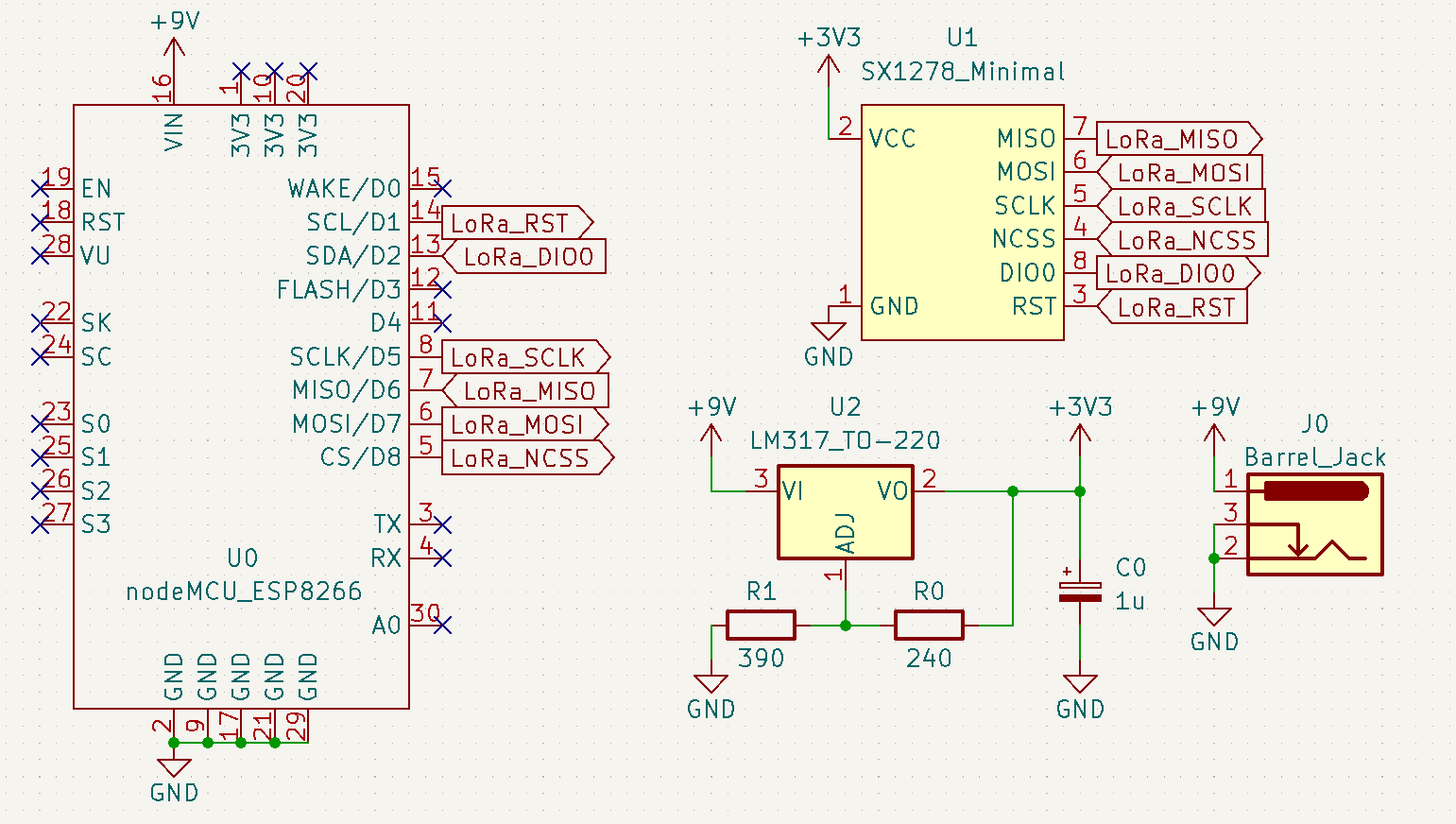


Figure 3‑28. Hardware setup for the gateway

##### Software design

Like the Autonomous Wireless Agrometeorology Station, the software for the gateway is developed on the Arduino platform. Although the ESP8266 is not an officially supported microcontroller, the board definitions could be added by appending the following to the Additional Boards Manager URLs in the Arduino IDE, then installing “esp8266” by ESP8266 Community from Boards Manager [75]:

https://arduino.esp8266.com/stable/package\_esp8266com\_index.json

On the Arduino IDE, after the board is chosen to be “NodeMCU 1.0 (ESP-12E Module)” (due to the “ESP8266MOD” marking and the built-in LED as shown in Figure 2‑20), the rest of the tool settings are left at default. The software for the ESP8266 as the gateway is then constructed upon the ESP8266WiFi library, which is shipped with the board definitions and firmware of the ESP8266 microcontroller line, and the ThingSpeak library, which could be found in the Library Manager as “ThingSpeak” by MathWorks. The LoRa library, which is installed to be used for the software of the station (node), is then imported for the gateway’s software as well.

#include <ESP8266WiFi.h>  // Wi-Fi connection handling library

#include "ThingSpeak.h"   // Library providing APIs for ThingSpeak access

#include <SPI.h>    // Built-in SPI library for LoRa

#include <LoRa.h>   // LoRa library handling SX1278

After the library come the setup parameters. The Wi-Fi network credentials must match the Wi-Fi router’s in order for the ESP8266 microcontroller to establish a connection. The channelNumber and writeAPIKey parameters could be found directly on the “API Keys” tab of the ThingSpeak channel for the project as “Channel ID” and on the “Key” box under “Write API Key” header respectively (Figure 3‑27). The hardware dependencies for LoRa module are just the necessary physical pins declaration accordingly to the hardware setup (Figure 3‑26). The LoRa setup parameters are initiated accordingly to the node’s; a side note to which is that only the frequency and syncWord need to match the node’s counterparts.

// Wi-Fi network credentials

const char\* ssid = "Station - test network";   // SSID

const char\* password = "testnetwork";          // Password

// ThingSpeak - Channel feed access parameters

unsigned long channelNumber = 2103173;

const char\* writeAPIKey = "TKJB5M4KKRH868E4";

// LoRa module hardware dependencies

const int csPin = D8;      // SPI NCSS pin for LoRa

const int resetPin = D1;   // LoRa reset

const int irqPin = D2;     // Interrupt by LoRa

// LoRa setup parameters

const long frequency = 433E6;   // LoRa frequency

int spreadingFactor = 12;       // LoRa spreading factor

long signalBandwidth = 500E3;   // LoRa signal bandwidth

int codingRate4 = 8;            // Denominator for LoRa coding rate

int syncWord = 0x92;            // Sync word for LoRa communication

Finally, in the software preparation, a WiFiClient instance must be declared for the WiFi object of ESP8266WiFi to initialise the Wi-Fi connection for ThingSpeak client.

// Initiate a WiFiClient instance

WiFiClient client;

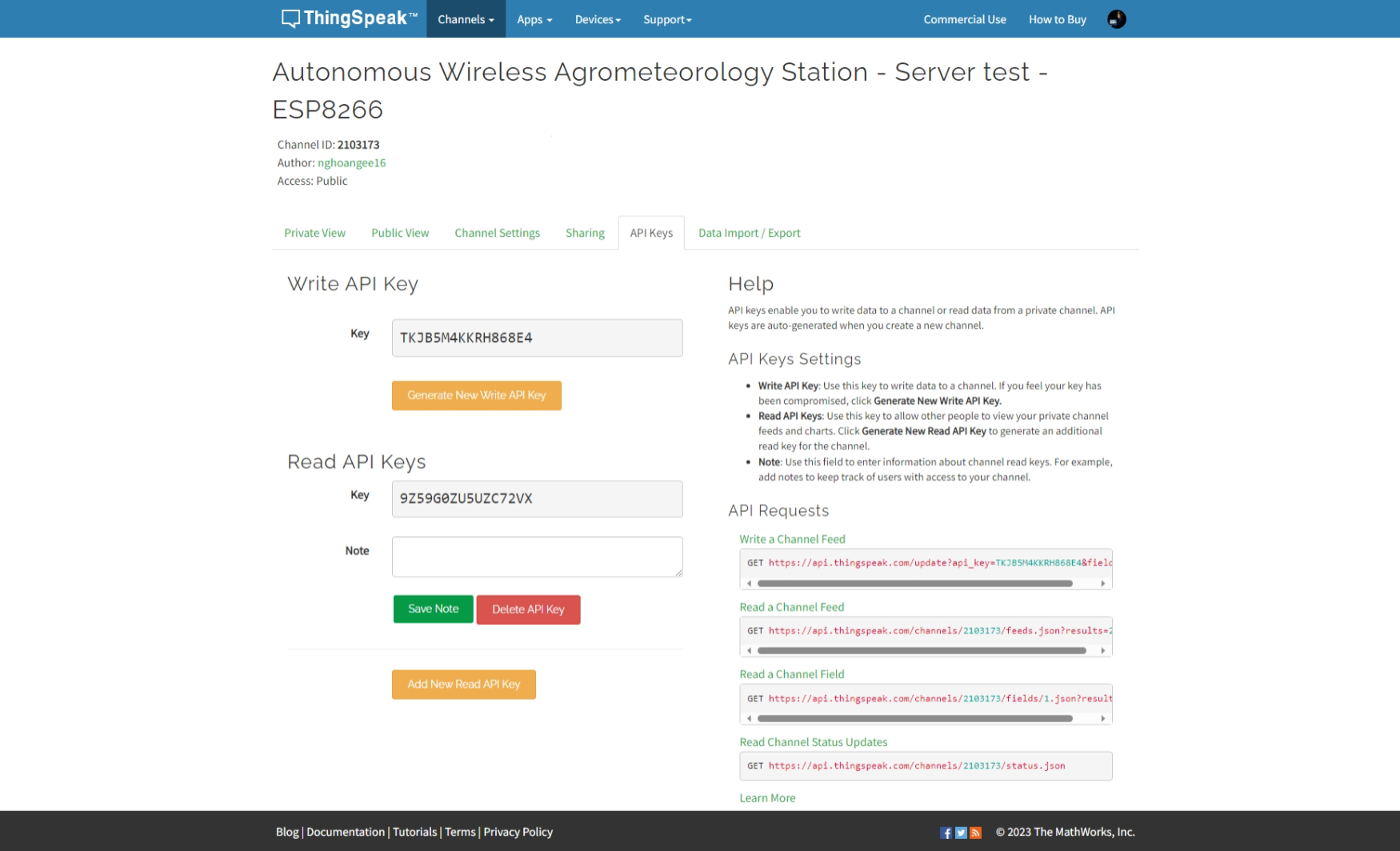


Figure 3‑29. ThingSpeak channel’s “API Keys” tab

The setup routine follows the flow and contains the functions shown in Figure 3‑28. It could be seen that if the initialisation of the LoRa module is successful, the ESP8266 is supposed to push the LoRa settings to the SX1278 module. This could be done by calling a user-defined function containing

LoRa.setSpreadingFactor(spreadingFactor);

  LoRa.setSignalBandwidth(signalBandwidth);

  LoRa.setCodingRate4(codingRate4);

  LoRa.setSyncWord(syncWord);

Then, the LoRa driver requires external definitions for .onReceive() and .onTxDone(onTxDone) routines by

LoRa.onReceive(onReceive);

  LoRa.onTxDone(onTxDone);

where the 2 functions onReceive() and onTxDone() must be written separately at the end of the source code of the software for gateway:

void onReceive(int packetSize) {

  rx\_message = "";

  while (LoRa.available()) {

    rx\_message += (char)LoRa.read();

  }

  nodeMessageAvailable = true;

}

void onTxDone() {

  Serial.println("TxDone\n");

  LoRa\_rxMode();

}

Within the onReceive() function, there is a global variable of the bool type, nodeMessageAvailable, which lets the microcontroller know when data is received from the node and perform suitable tasks in the loop() scope. There exists another global variable of String type, rx\_message, where the message from the node is stored upon arrival and to be processed by the ESP8266 microcontroller.

Although the gateway is only designed to receive sensor data from the node, there requires 2 additional functions to be defined, 1 to set the LoRa module in receive mode, and the other to set the LoRa module in transmit mode:

// Set LoRa in receive mode

void LoRa\_rxMode() {

  LoRa.disableInvertIQ();   // normal mode

  LoRa.receive();           // set receive mode

}

// Set LoRa in transmit mode

void LoRa\_txMode() {

  LoRa.idle();              // set standby mode

  LoRa.enableInvertIQ();    // active invert I and Q signals

}

Like onReceive() and onTxDone(), these 2 functions must be put at the end of the source code for the gateway. As mentioned in Section 3.3, the SX1278 LoRa modules does not turn the connected microcontrollers into real LoRaWAN nodes and gateways, but mimics the behaviours as simple ones, so the functions LoRa\_rxMode() and LoRa\_txMode() must handle the I and Q signals of the LoRa modules. For the gateway, the I and Q signals are the opposite of those for the node:

* Transmit mode: Inverted I and Q signals (.enableInvertIQ())
* Receive mode: Normal I and Q signals (.disableInvertIQ())

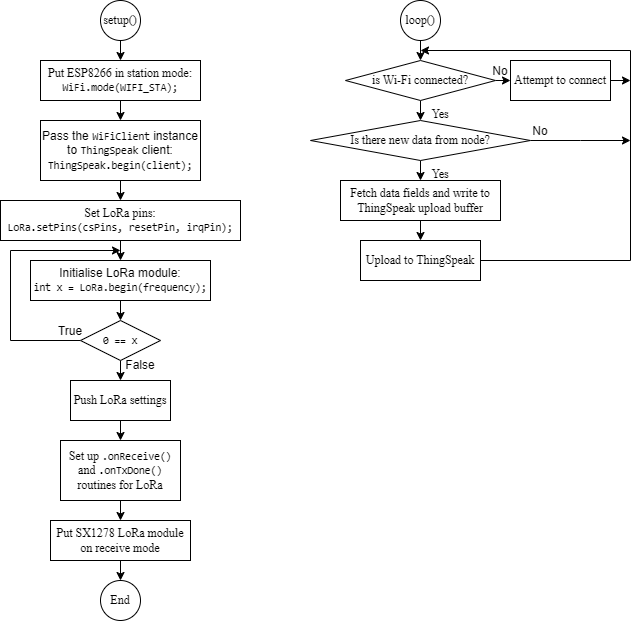


Figure 3‑30. Gateway setup and loop routines

After the setup routine, the software design for the gateway of the Autonomous Wireless Agrometeorology Station puts the ESP8266 microcontroller in a loops as depicted in Figure 3‑28. On each loop run, the ESP8266 microcontroller always check is its Wi-Fi connection to the router remains; if it is disconnected, the operation WiFi.status() != WL\_CONNECTED returns true and it shall make attempts to reconnect to the Wi-Fi network, since data cannot be pushed to ThingSpeak server without an Internet connection:

while (WiFi.status() != WL\_CONNECTED) {

      WiFi.begin(ssid, password);

      delay(5000);

    }

Then, if the aforementioned variable nodeMessageAvailable is true, since the data is always transmitted as String via LoRa, the nodeMessageAvailable variable is cleared and the String methods are used to extract data for each field of the ThingSpeak upload buffer. A side note is that the gateway shall always assume the node transmits data correctly if the transmission operation is successful, and the received data is in the format of:

YYYY-MM-DD HH:MM:SS+07,[1],[2],[3],[4],[5],[6],[7]

where

* YYYY is the year,
* MM is the month,
* DD is the day of month,
* “+07” is the timezone of Vietnam, UTC+07,
* [1] is the wind direction,
* [2] is the wind speed,
* [3] is the temperature at ground level read by the DS18B20,
* [4] is the temperature at 1 m above the ground read by the BME280,
* [5] is the relative humidity read by the BME280,
* [6] is the barometric pressure read by the BME280,
* and [7] is the rainfall data,

with “,” is the delimiter of the data fields. For example,   
“2023-09-11 09:00:00+07,22.5,4.44,29.1,28.8,75,1001.4,6.604” means:

* The date is 11 September 2023 at 9 in the morning, UTC+07 time.
* The wind speed is 4.44 m/s, blowing from the 22.5˚ direction.
* The temperature at the ground level read by the DS18B20 sensor is 29.1 ℃.
* The temperature at 1 m above the ground read by the BME280 sensor is 28.8 ℃.
* The relative humidity is estimated to be 75 %RH.
* The barometric pressure measured is 1001.4 hPa.
* The rainfall amount of the previous day is 6.604 mm.

On the point of view of ThingSpeak, there are a total of 7 data fields and a timestamp. Therefore, the timestamp is to be extracted and written to the ThingSpeak upload buffer first:

int delimiterIndex = rx\_message.indexOf(',');

  ThingSpeak.setCreatedAt(rx\_message.substring(0, delimiterIndex));

Then, a for loop runs 7 times to extract the entries sequentially and write to the respective buffer fields. Unlike the C++ programming language of the Arduino framework, ThingSpeak manages its database from index 1 for each entry, so the for loop runs with the fieldIndex parameter taking the initial value of 1, not 0, and the stop condition being fieldIndex <= 7 instead of fieldIndex < 7.

for (int fieldIndex = 1; fieldIndex <= 7; fieldIndex += 1) {

    float fieldValue;

    int startPoint = delimiterIndex + 1;

    if (fieldIndex < 7) {

      delimiterIndex = rx\_message.indexOf(',', startPoint);

      fieldValue = rx\_message.substring(startPoint, delimiterIndex).toFloat();

    } else {

      fieldValue = rx\_message.substring(startPoint).toFloat();

    }

    ThingSpeak.setField(fieldIndex, fieldValue);

  }

Finally, the upload buffer is pushed to ThingSpeak by a single line below, and the main loop routine starts over.

ThingSpeak.writeFields(channelNumber, writeAPIKey);

The ThingSpeak library provides the APIs with return codes, for example, a code of int type for the .setField(..) method. These return codes allow the determination of the status of each called operation for/with ThingSpeak and suitable subsequent actions to be taken if necessary on the software level. However, since the focus of this project is the sensor station, the gateway shall assume no issues further than the Wi-Fi connection availability and take no measures against failed communications with ThingSpeak. The such subject, on the other hand, does not remain overlooked and is reserved for further developments of this project.

# Experimental Characterisation

In this thesis, there shall not be a full integration test for the Autonomous Wireless Agrometeorology Station. Instead, the sensors from the SF-WS02 kit are tested separately since the reading methodologies are merely either interrupt-based or software-controlled activation of the microcontroller’s ADC module. The LoRa module has its own experiments as well for connectivity and range, while giving some insights into how integration with ThingSpeak performs. Finally, there is a test for the accuracy of BME280 and DS18B20 sensors, along with the determination if all the communication interfaces could be utilised in a single firmware for the STM32F103C8T6 microcontroller.

## Anemometer

The test for the 3-cup anemometer behaviour involves an MFP107 Axial Fan Module by TecQuipment. The fan module has protective grilles at both ends of the duct, so the anemometer is set up just outside the duct to catch the exiting air whose flow is assumed to be uniform. The slide-valve is opened to 100% for maximum air flow. The fan speed is increased manually via the control panel by a step of 200 revolutions per minute; the volume metric flow rate is measured by the fan module and recorded via software once per second. For each fan speed value, the STM32F103C8T6 microcontroller counts the number of pulses from the anemometer for 20 seconds and displays on the Serial Monitor at the end of each sampling window. The mean volume metric flow rate of the fan module and the pulse count by the microcontroller are then recorded manually to a spreadsheet to be processed later.



Figure 4‑1. Axial Fan Module MFP107 by TecQuipment [76]

Although the MFP107 Axial Fan Module does not monitor the air speed through the duct directly, its built-in sensors still read the volume metric flow rate, which could still be used to calculate the air speed by dividing it by the cross-section of the duct, whose diameter is 40cm. The test run result is as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Duct diameter** | **D = 40 cm** |  | **R = 9.2 cm** | **T = 20 s** |
| **Mean flow rate** *fmean***, (m3/s)** | **Machine wind speed** *Vm***, (m/s)** | **Pulse count *n*** | **Read wind speed** *Vs***, (m/s)** | **Factor =** *Vm***/***Vs* |
| 0.85 | 6.764085081 | 30 | 0.433539786 | 15.60199386 |
| 0.91 | 7.241549911 | 55 | 0.794822941 | 9.110896948 |
| 0.958 | 7.623521774 | 92 | 1.329522011 | 5.734032014 |
| 1.022 | 8.132817592 | 135 | 1.950929038 | 4.168689601 |
| 1.14 | 9.071831756 | 176 | 2.543433412 | 3.566765976 |
| 1.33 | 10.58380372 | 254 | 3.670636856 | 2.88336987 |
| 1.44 | 11.4591559 | 289 | 4.176433274 | 2.743766068 |
| 1.56 | 12.41408556 | 329 | 4.754486322 | 2.611025613 |
| 1.65 | 13.13028281 | 370 | 5.346990696 | 2.455639733 |
| 1.78 | 14.16478994 | 383 | 5.534857937 | 2.559196658 |
| 1.9 | 15.11971959 | 437 | 6.315229552 | 2.394167855 |
| 2.03 | 16.15422672 | 505 | 7.297919734 | 2.213538558 |
| 2.15 | 17.10915638 | 538 | 7.774813499 | 2.200587369 |

Table 4‑1. Anemometer test data with the Axial Fan Module MFP107

Although there appears to be a linear association between the read wind speed by the anemometer, this relationship remains inapplicable since the test is limited by the highest fan speed of the Axial Fan Module. However, the wind speed factors between the machine’s and sensor’s values by the read wind speed from the anemometer pose a more practical trend.

Figure 4‑2. Near-linear association   
between the anemometer’s and the Axial Fan Module’s wind speeds

Figure 4‑3. Wind speed factors by the read wind speed from the anemometer

It could be observed that at stronger wind, the wind speed factor approaches a certain value. Although the such value appears to be 2 from Figure 4‑3, without the exact model based on aerodynamics, it is uncertain how the anemometer factor really behaves. On the other hand, by the use of Curve Fitting Tool in Matlab, the mathematic model could be estimated. For the time being, the wind speed Vm by the Axial Fan Module is considered the absolute value, and the wind speed read by the anemometer Vs is derived by a factor F following , or if the factor is considered a function of Vs.

To keep it simple, the function is determined to be a rational function of the same degree on the numerator and the denominator:

For N = 1, the Curve Fitting Tool gives

This function, however, is undesirable because it produces a negative factor F as wind speed Vs approaches 0 from +, resulting in a negative value for speed, a non-negative quantity.

For N = 2, the Curve Fitting Tool gives

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

As the wind speed Vs approaches 0 from, the anemometer factor F remains positive, thus a valid value for the real wind speed Vm.

For higher values of N, the Curve Fitting result starts to show fluctuations (Figure 4‑6). The such issue could be avoided by using a rational function model whose numerator degree is different from the denominator degree. However, this solution complicates the software, and potentially introduces computational errors, so the model (3) is the most suitable function for the anemometer F to fix the anemometer behaviour as shown in Table 4‑1.

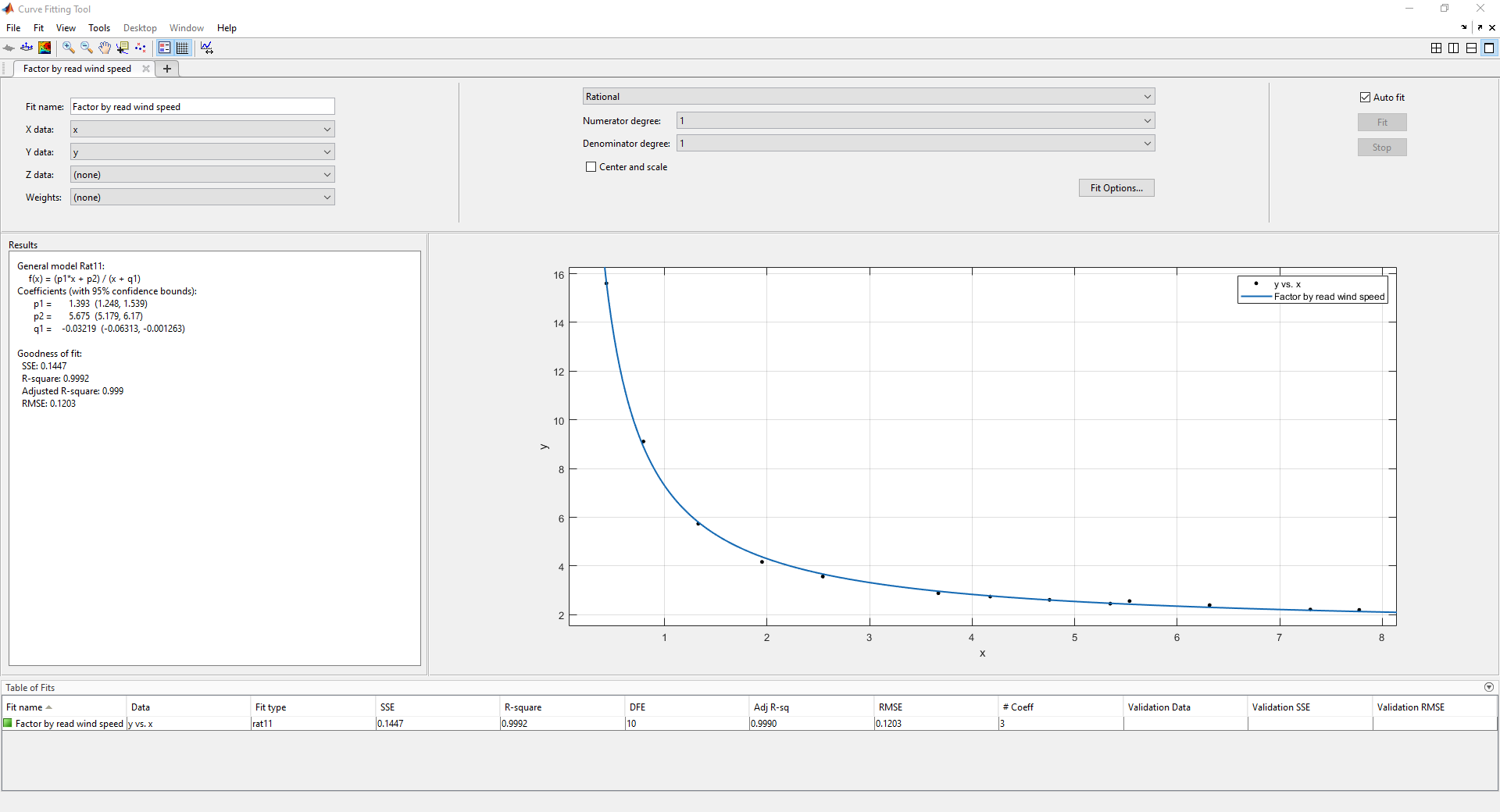


Figure 4‑4. Curve Fitting result for N = 1

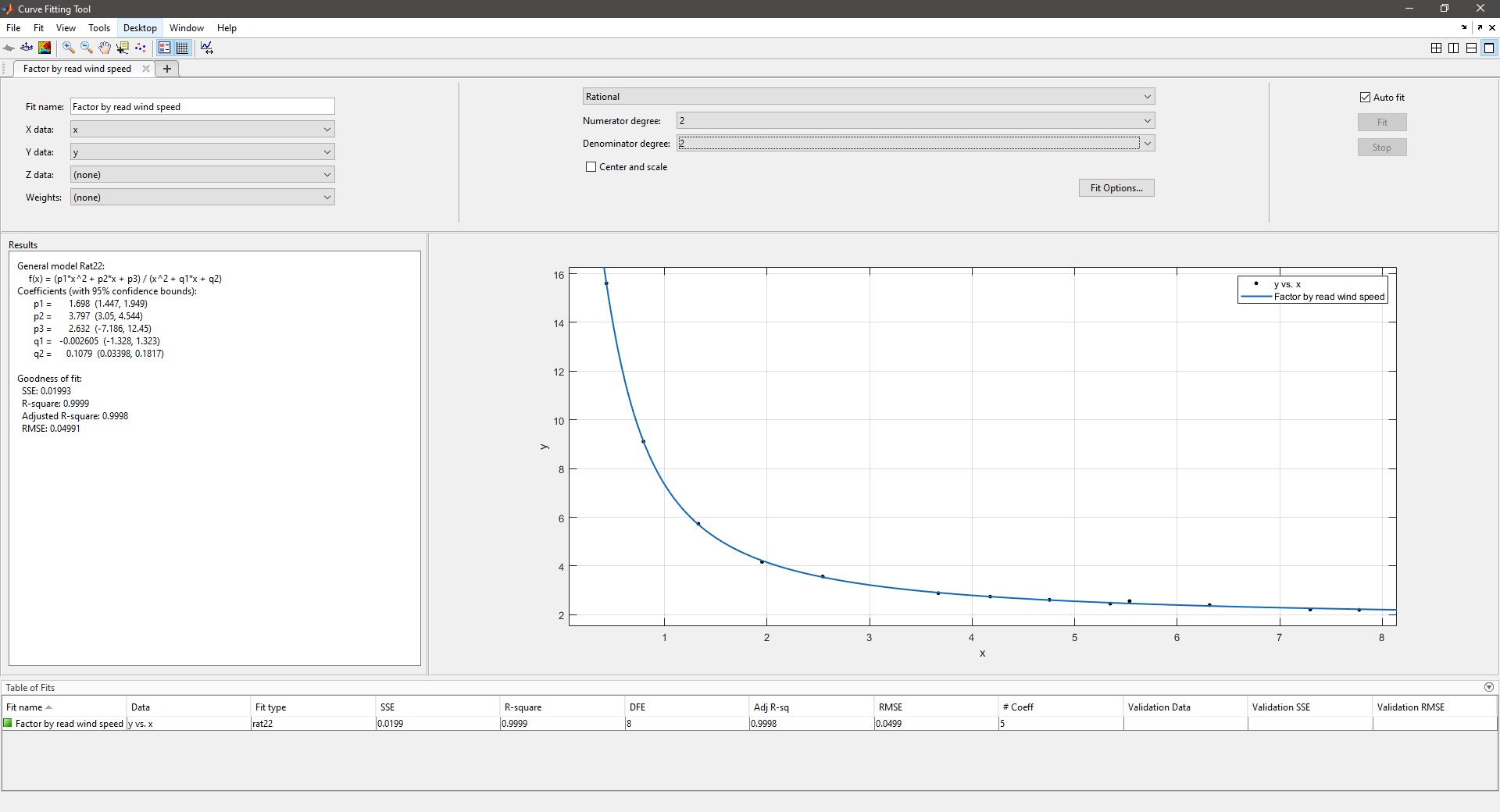


Figure 4‑5. Curve Fitting result for N = 2

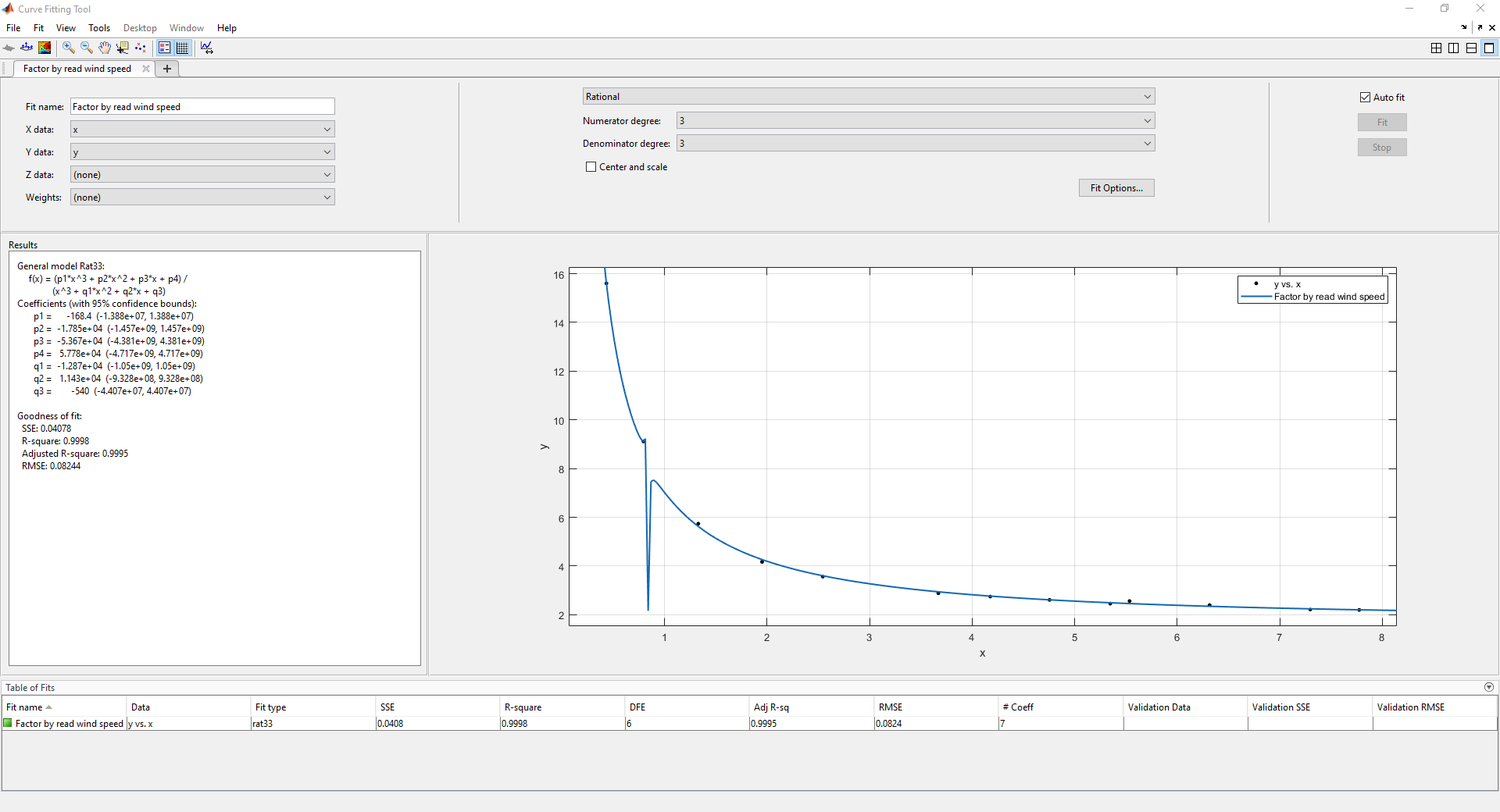


Figure 4‑6. Curve Fitting result for N = 3

## Wind Vane

The test for the wind vane is performed mostly on the stage of designing hardware for the microcontroller to read data from the sensor. The wind vane is first integrated with the pull-up resistor to form a voltage divider as described in Section 3.2.2.1. On each stage of the test, the sensor is held and turned manually to every of the 16 directions. Although there are only 4 directions marked on the body of the sensor, a voltmeter is put in use to determine the remaining 12 directions following the voltage outputs shown on the table of Figure 3‑12. At each direction, the microcontroller samples 20 ADC values and determine the direction as described in Section 3.2.2.2. The results are written to a CSV file on a microSD card and later processed on Microsoft Excel.

##### Reading unfiltered signals by the voltage divider

The wind vane test starts with reading the raw signals directly from the voltage regulator. For most of the positions of the rudder blade, the STM32F103C8T6 microcontroller could determine the expected directions. However, there exists bad readings when the microcontroller tries to detect the 0˚, 67.5˚, 225˚, and 337.5˚ directions, while it completely fails to read the 270˚, 292.5˚, and 315˚ directions.

Figure 4‑7. Number of correct and bad readings when sampling from the voltage divider

##### Reading signal outputs by the passive LPF

Upon putting a passive LPF with the cut-off frequency f-3dB = 11.587 kHz between the output of the voltage divider and the ADC input of the microcontroller, the test result greatly improves when the microcontroller only produces 4 erroneous readings on the 270˚ direction. However, upon inspection of the incorrect readings, it is discovered that the issue may have been caused by the computational errors of the microcontroller itself, not because of the filtered signal at the input of the ADC module. For example, there are instances (marked with \* in Table 4‑2) where the same raw ADC value is read (3957), the same Vin value is calculated (3.15 V), but the produced Rread values are different (95.3 kΩ and 91.9 kΩ), and in turn, the directions. Moreover, the expected Rread value for the 270˚ direction is 120 kΩ (Table 2‑1), which means the input signals are too attenuated and all the calculated Rread values by the microcontroller are too close to the lower value, 64.9 kΩ of the 315˚, to be distinguished properly. The such issue could be resolved by redefining the thresholds between 2 closest resistive values on the software level, but this solution remains open, so that the hardware level is further explored for better results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input direction: 270** | |  |  |
| **Read direction** | **Raw ADC** | **Vin** | **Rread** |
| 270 | 3955 | 3.15 | 94623.84 |
| 270 | 3955 | 3.15 | 93224.98 |
| 270 | 3955 | 3.15 | 93224.98 |
| 270 | 3958 | 3.15 | 93224.98 |
| 270 (\*) | 3957 | 3.15 | 95338.59 |
| 270 | 3955 | 3.15 | 94623.84 |
| 270 | 3953 | 3.15 | 93224.98 |
| 315 (\*) | 3957 | 3.15 | 91865.53 |
| 270 | 3953 | 3.15 | 94623.84 |
| 315 | 3956 | 3.15 | 91865.53 |
| 270 | 3953 | 3.15 | 93919.38 |
| 315 | 3956 | 3.15 | 91865.53 |
| 270 | 3953 | 3.15 | 93919.38 |
| 315 | 3956 | 3.15 | 91865.53 |
| 270 | 3954 | 3.15 | 93919.38 |
| 270 | 3956 | 3.15 | 92540.44 |
| 270 | 3957 | 3.15 | 93919.38 |
| 270 | 3955 | 3.15 | 94623.84 |
| 270 | 3956 | 3.15 | 93224.98 |
| 270 | 3955 | 3.15 | 93919.38 |

Table 4‑2. Output by the microcontroller for the 270˚ direction in the test with the passive LPF

Figure 4‑8. Number of correct and bad readings when sampling from the passive 11.587-kHz LPF

##### Reading signal outputs by an active LPF using CA3149EZ Op-Amp

Following the test stage with the passive LPF, the whole sampling process is redone after a voltage buffer is added between the LPF and the ADC input of the microcontroller, producing an active LPF with the cut-off frequency of f-3dB = 11.587 kHz on this next stage. The Op-Amp in use is a CA3140EZ.

Before the CA3140EZ Op-Amp is added to the wind vane test, it is put through a separate voltage buffer test. The general idea of hardware connections is as illistrated by Figure 4‑9. The signal generator in use produces a sine wave at 440 Hz with a 3.3-Vpp amplitude and 1.65-V offset for an input signal between 0 and 3.3 V. The power supply is set manually from 3.3 V, the voltage level of the main system of the Autonomous Wireless Agrometeorology Station. The goal of this mini test is to determine at which power supply VCC the Op-Amp produces the best result as a voltage buffer.

By observing the signals on an oscilloscope (Appendix B), it is determined that the CA3140EZ Op-Amp must be powered at 6 V for the main test with the wind vane. When the supply VCC is under 6V, the outputs of the CA3140EZ as a voltage buffer are either capped (from 3.3 V to 5.41 V) or overshoot (5.9 V). When the power supply reaches 6.3 V, the output waveforms starts having fluctiations. The CA3140EZ Op-Amp is then powered at 6 V and put back to the main test on the wind vane.

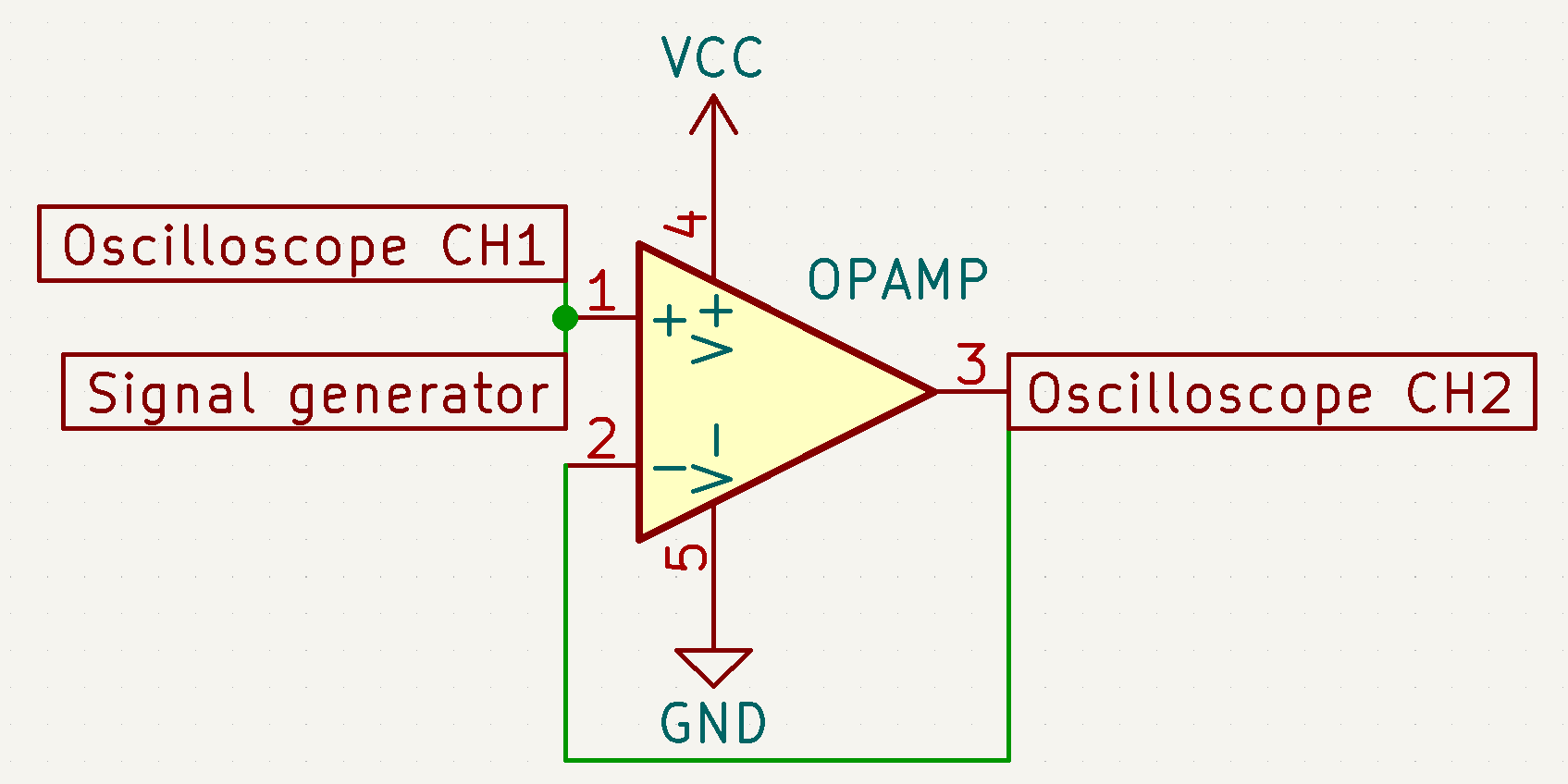


Figure 4‑9. Op-Amp setup for the voltage buffer test

It could be said that the result of this test is basically the same as the previous’. The bad readings appear when the microcontroller tries to detect the 67.5˚ direction, and a single bad value at the 112.5˚ direction. The difference between the 2 tests is that when the CSV files are checked, the Rread values from the active LPF are higher, while the passive LPF produces Rread lower than given values by [15] in Table 2‑1.

Figure 4‑10. Number of correct and bad readings when sampling from an active 11.587-kHz LPF using a CA3140EZ Op-Amp

##### Reading signal outputs by an active LPF using MCP6002-I/P

Although the active LPF with the CA3140EZ Op-Amp produces decently clean outputs, the supply voltage of 6 V for the Op-Amp is not ideal since the main system runs at 3.3 V. Therefore, an alternative must be explored. The test is first conducted with the setup like Figure 4‑9, but in this case, the VCC is kept at 3.3 V and the Op-Amp becomes the variable.

There are a total of 5 Op-Amp options put in test due to their availability: CA3140EZ, LM358P, LM393P, MCP6002-I/P, and UA741CP. In this test, the signal generator is configured to output a triangular waveform because it rises and falls linearly. The frequency of the signal is 250 mHz, the amplitude is 3.3 V, and the offset is 1.65 V for a signal going from 0 V to 3.3 V like the previous test. The results as shown in Appendix C makes MCP6002-I/P the only valid option as the Op-Amp for the voltage buffer part of the active LPF at the 3.3-V power supply. The MCP6002-I/P is a dual Op-Amp device, but only 1 unit is put in use, and the pins of the other are left unconnected.

Upon putting the MCP6002-I/P in test for the wind vane, the result retains the bad entries issue. The 247.5˚ direction sees 7 bad entries out of 20, while the microcontroller returns the correct direction for only 1 out of 20 readings of the 0˚ one. All the tests on the wind vane up to this point shows that the STM32F103C8T6 obtains erroneous data from its ADC module, so there arises a question why this happens. Another test attempt then takes place to determine a viable solution.

Figure 4‑11. Number of correct and bad readings when sampling from an active 11.587-kHz LPF using a MCP6002-I/P Op-Amp

##### The addition of a pull-down resistor between the latest active LPF design and the microcontroller

On every testing stage, a voltmeter is in use to make sure that the rudder blade of the wind vane is in the correct position to produce expected voltage signals at the ADC input pin of the microcontroller. If the input is closely monitor as such, there potentially is a problem with the ADC, or the hardware design is simply incomplete despite the good separate test results. When reviewed, the CSV files containing test results on the wind vane with both active LPF designs reveal that all entries are higher than their expected values. Therefore, it is determined that all the current of the output signals from the Op-Amp flowing into the ADC input pin of the microcontroller might the the cause. As a result, a pull-down resistor is added as shown in Figure 3‑14.

The process of picking a suitable resistor value starts with a 10-kΩ resistor. There is no reason for starting with this value because it is only to observe the behaviours of the microcontroller’s outputs for the wind vane direction. Since the result stays the same, the value of the resistor is determined to be lowered to further decrease the current flow into the ADC module. The available resistive values in this test are respectively 8.2kΩ, 6.8kΩ, 5.6kΩ, 4.7kΩ, and 3.3kΩ. The test ends when the pull-down resistor is 3.3-kΩ, and the microcontroller reads the directions as expected.

Figure 4‑12. Final wind vane test result from the exact circuitry as Figure 3‑14

## Rain Gauge

There is no precise measurement tool to use as a reference for testing the rain gauge, so the test is conducted based on the theory that 1 mm of rainfall is equal to 1 litre of water collected by a metre square of a horizontal area [77]. For an area of S m2, 1 mm of rainfall then equals S litres of water collected, so if an area of S m2 collects A litres of rain water, the amount of rain fall is mm.

Based on the dimensions of the orifice of the rain gauge (Figure 2‑10), it is calculated that the device has an area of 5368.54 mm2 to collect rain water when put in use. For a controlled test, water is poured onto the rain gauge instead. The amount of water is measured by a mini digital scale whose resolution is 0.01 g, then converted from weight to volume by the approximation .



Figure 4‑13. The mini digital scale used in rain gauge test

The test is conducted twice, once for a fixed amount of water, and once for different amount of water poured onto the rain gauge. The amount of “rainfall” is then calculated on the amount of water and the orifice size of the sensor, and the expected number of pulse counts is then obtained based on the given 0.3 mm of rainfall per tip [17]. The test results are as follows.

|  |  |  |  |
| --- | --- | --- | --- |
| **Orifice size (mm2)** | 5368.54 |  |  |
| **Amount of water (ml)** | **Supposed rainfall (mm)** | **Expected pulse counts** | **Real  pulse counts** |
| 14.80 | 2.756801663 | 9 | 6 |
| 14.85 | 2.766115182 | 9 | 6 |
| 14.87 | 2.76984059 | 9 | 6 |
| 14.82 | 2.760527071 | 9 | 6 |
| 14.83 | 2.762389775 | 9 | 6 |
| 14.81 | 2.758664367 | 9 | 6 |
| 14.83 | 2.762389775 | 9 | 6 |
| 14.85 | 2.766115182 | 9 | 6 |
| 14.82 | 2.760527071 | 9 | 6 |
| 14.82 | 2.760527071 | 9 | 6 |

Table 4‑3. Rain gauge test with a fixed amount of water

|  |  |  |  |
| --- | --- | --- | --- |
| **Orifice size (mm2)** | 5368.54 |  |  |
| **Amount of water (ml)** | **Supposed rainfall (mm)** | **Expected pulse counts** | **Real  pulse counts** |
| 50.38 | 9.384301877 | 31 | 20 |
| 104.76 | 19.51368529 | 65 | 39 |
| 144.80 | 26.97195141 | 90 | 56 |
| 186.16 | 34.67609443 | 116 | 66 |
| 233.83 | 43.55560357 | 145 | 81 |

Table 4‑4. Rain gauge test with different amounts of water

The first test run shows the consistency in pulse counting by the microcontroller for the approximately same amount of water poured onto the rain gauge, so it could be said that the circuitry of the tipping bucket produces clean signals for the microcontroller to detect (without undesired noise spikes). The number of pulse counts, however, does not match the expected values from the calculation with the product description of the manufacturer.

The second test run confirms that there is a shift in the real pulse counts and the expected values. Without a precise data reference, it is not possible to determine if the rain gauge is faulty, or the rainfall per tip specification is not actually correct. However, the former is assumed, and the software needs modifying accordingly to these findings.

By dividing the amounts of water to the real pulse counts, it is obtained that the buckets tip over once for about every 2.55 ml of water. That converts to rainfall per tip.

## Temperature Sensors – Hygrometer – Barometer

In this test, both the DS18B20 temperature sensor and the BME280 compound sensor have their readings recorded and compared to the data of an Extech 445815 Humidity Alert II Hygro-Thermometer unit due to its precision and availability at Vietnamese – German University. However, the hygro-thermometer unit does not provide a communication interface for data to be extracted directly; instead, this task is done with a USB2.0 UVC Webcam on a Raspberry Pi 4B single board computer. The USB2.0 UVC Webcam is available on icdayroi.com as “Camera USB UVC USB2.0 cho raspberry pi”.



Figure 4‑14. Extech 445815 Humidity Alert II Hygro-Thermometer unit used as reference

Apart from the sensors, this test also aims at testing the communication interfaces of the STM32F103C8T6 microcontroller when put together, the DS3231 RTC module, and the microSD card module. The STM32F103C8T6 microcontroller is set up with the included sensor units, a DS18B20, a DS3231 RTC module, and a microSD card module as described in the corresponding sections under Chapter 3. It is then powered via a 3.3V pins of the Raspberry Pi 4B, and communicates with the Pi computer via UART; pins PA9 (TX) and PA10 (RX) of the STM32 are wired to pins 10 (UART0 RX) and 8 (UART0 TX) of the Raspberry Pi 4B respectively.

On the Raspberry Pi 4B, the driver for the USB2.0 UVC Webcam needs installing before it could be used. This is done on the Terminal with:

sudo apt-get install fswebcam

The software on the Raspberry Pi 4B (Appendix E) is written in Python to capture a picture of the hygro-thermometer screen with the USB2.0 UVC Webcam whenever the STM32F103C8T6 sends a data line once every 5 minutes. The picture is first stored locally and then uploaded to Dropbox for remote access while the data via UART is written to a CSV file on the Raspberry Pi. The test is expected to run in 48 hours; however, due to a Internet connection for the Raspberry Pi 4B for a short period, the total run time is extended to 52 hours. Within this window, another CSV file is used to manually log the temperature and humidity from the pictures of the Humidity Alert II screen. At the end of the run period, the 2 CSV files are merged and the comparison of data is performed.

At the beginning, it was found that the microSD card module would halt the STM32F103C8T6 microcontroller when the default SPI was used in the same software with the I2C interface. The removal of the microSD card module then allowed the test to continue.

The software on the STM32F103C8T6 is firstly built with DS3231 alarm once every minute. When detecting the alarm, the microcontroller clears it and increment a global count variable. When the count variable reaches the value of 5, the microcontroller resets its value to 0, read data and time from the RTC module, and fetch the sensor data, then send to the Raspberry Pi. In fact, if left untouched, the STM32 microcontroller and the Raspberry Pi would run this test indefinitely since there is no stop condition on the software level.

The results of the test are as follows.

Figure 4‑15. Temperature readings from the compound test

Figure 4‑16. Humidity readings from the compound test

Figure 4‑17. Barometric pressure readings from the compound test

For the temperature sensing units, Figure 4‑15 shows a smooth temperature line by the BME280, while the DS18B20 returns a constant. It is unclear of the reason why the behaviour of the latter is as such, but that means each DS18B20 device must later be carefully checked before the assembly stage of any design. The BME280, on the other hand, proves to be able to detect changes in ambient temperature much like the reference, the Humidity Alert II device, but there is an offset. Upon checking the final log file, it is found that the difference of temperature readings between the BME280 and the Humidity Alert device ranges from 1.37 ℃ to 2.83 ℃, with a mean value of 1.99 ℃. Since the latter is included in this test as a precise reference, it is concluded that the software for the BME280 must introduce a temperature offset value of -1.99 ℃.

Like temperature, the humidity readings of the BME280 appear to contain an offset, apart from a single value where the sensor returns a “nan” (not a number) entry near the beginning. Since the incident does not repeat throughout the test, it could be assumed that there may have been a single case of data-exchanging error. For the rest of the test result on humidity, the juxtaposition with the reference from the Humidity Alert II device gives that the readings by the BME280 is from 0.5 %RH to 7.89 %RH higher, with an average of 4.46 %RH. As a result, there requires a humidity offset of -4.46 %RH within the software for the BME280.

Finally, there is no reference point to determine if the barometric pressure read by the BME280 is correct or not. However, the chart of Figure 4‑17 displays a repeated pattern among over 2 days of test, which means the BME280 does not read random values and could detect changes in atmospheric pressure like it does with ambient temperature and humidity. On the other hand, it could not be ruled out that the barometric readings of the BME280 contain an offset, since [19] specifies that both the humidity and pressure sensing units of the sensor take its read temperature as a parameter for the estimation of the environment sampling results.

## LoRa Connectivity – ThingSpeak

Although this test is on the LoRa connectivity and the communication with ThingSpeak, it roots from the temperature-reading results of the test in Section 4.4, where the DS18B20 in test does not return any values other than 31 ℃ throughout the 52-hour window.

The test is done with a nodeMCU-ESP8266 as a simple gateway, and an STM32F103C8T6 as a simle node. The communication between the 2 microcontrollers is established via 2 SX1278 LoRa modules. The STM32 microcontroller reads data from a BME280 compound sensor, and 3 DS18B20s with a pre-determined ROM codes on the same 1-wire bus. The data channel on ThingSpeak is set up as follows.

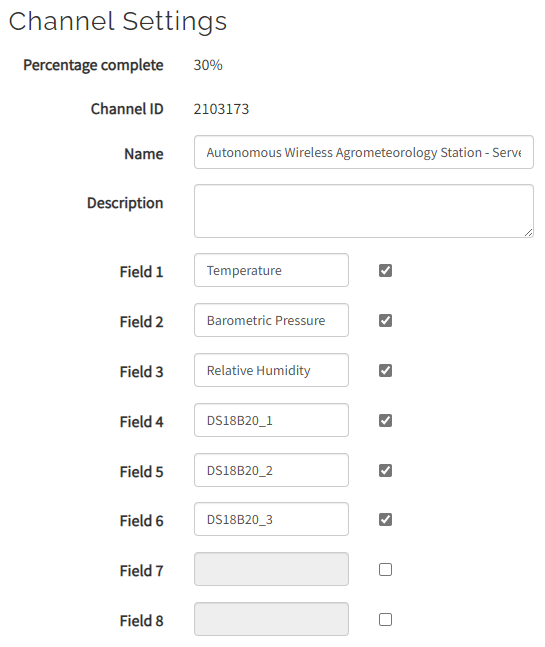


Figure 4‑18. Channel settings on ThingSpeak for LoRa connectivity and server test

In this test, the STM32 microcontroller also utilises a 1-minute alarm by a DS3231 RTC module, and it pushes sensor data as soon as there is an alarm. The test is set to run and stopped manually after 7 hours and 33 minutes.

The gateway, apart from handling its LoRa connections and uploading to ThingSpeak server, keeps a log on Serial Monitor, which gives that there exits no failed communication attempts between the LoRa modules and with ThingSpeak. On the webview of the ThingSpeak channel for this test, the results are displayed as follows.

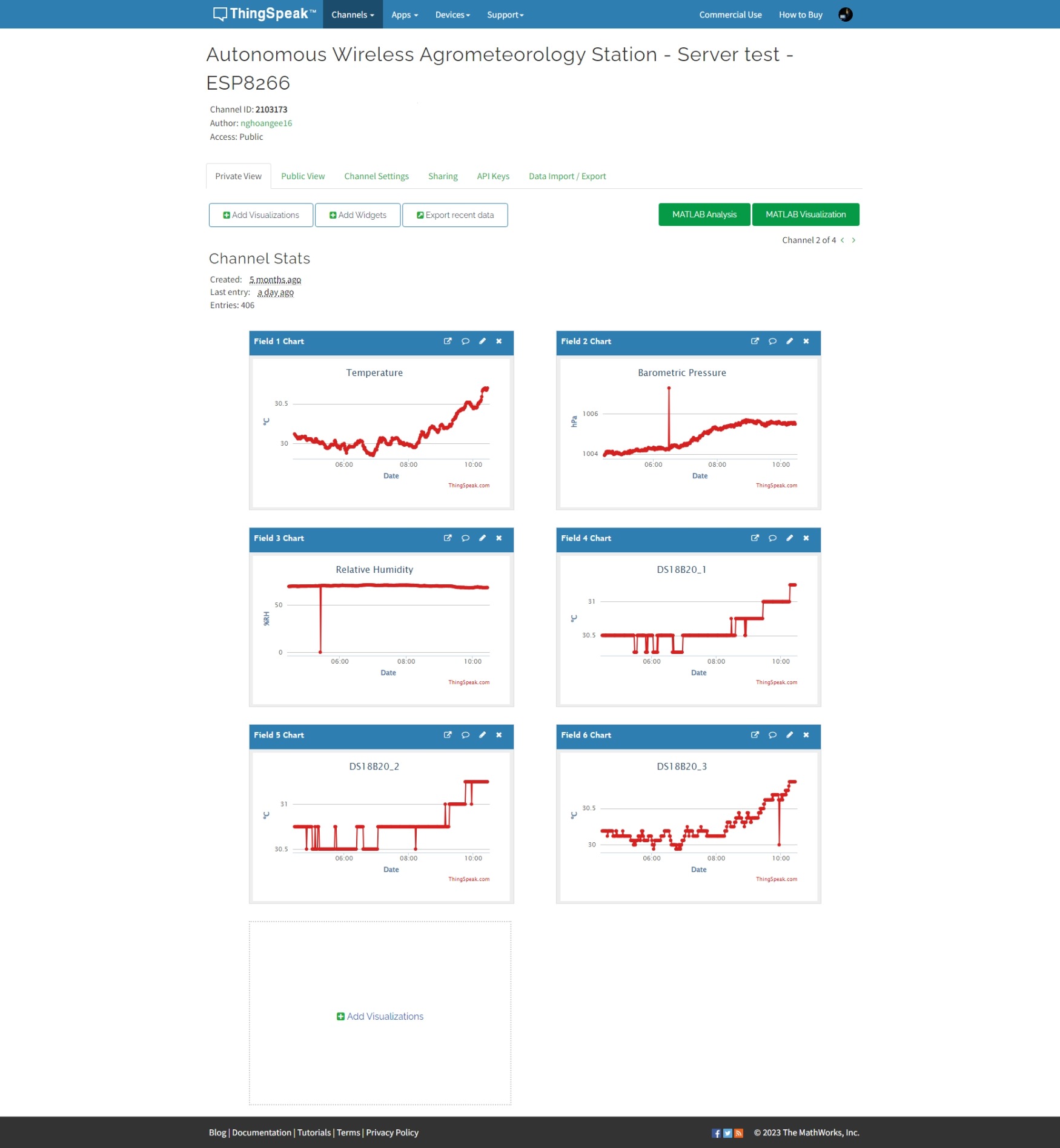


Figure 4‑19. Private view of the server test channel on ThingSpeak

It could be seen that ThingSpeak provides a private view and a public view for each data channel. If configured, the public view appears just the same as the private view. Moreover, ThingSpeak allows each chart to be modified to display the data as desired. For example, axes, line colour, background, number of data points in view, etc. of Field 1 Chart could be configured via the Field 1 Chart Option interface as depicted in Figure 4‑20.

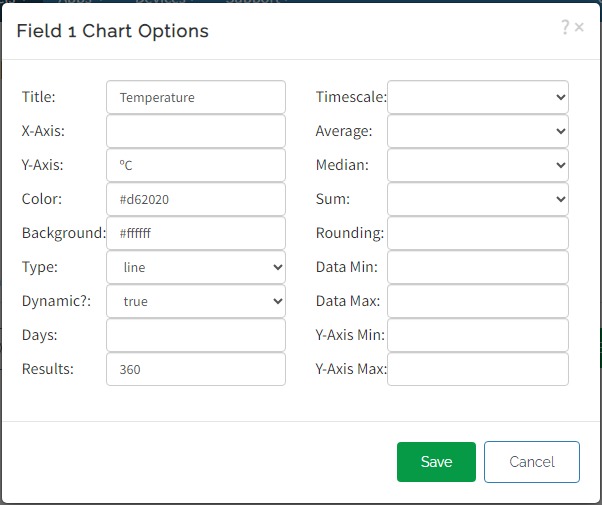


Figure 4‑20. Example of Chart Option window on ThingSpeak

On a side note, this test confirms that the DS18B20 device used in the test of Section 4.4 is faulty. The 3 DS18B20 devices used in this test show a capability of detecting changes in temperature much like the BME280 sensor, with the pre-determined resolutions of 0.25 ℃ for the first 2 DS18B20s and 0.0625 ℃ for the other. Moreover, the DS18B20 with the 0.0625 ℃ thermometer resolution reads nearly exactly the same as the BME280, so it could be assumed that the temperature offset error of the 2 sensors are the same, and the calibration value of -1.99 ℃ could be introduced to the software of the DS18B20.

## LoRa Range Test

The LoRa range test is performed to test the LoRa coverage along with the change of the received signal strength indicator (RSSI), signal-to-noise ratio (SNR), and frequency error of the SX1278 modules when put on the field. The experiment is performed with 2 SX1278 LoRa modules on an open field to ensure they could transfer data on a near line-of-sight. The node is run on an STM32F103C8T6 which receives GPS coordinates via Serial USB Terminal by Kai Morich on a phone; which are then sent uplink to the gateway via LoRa. The gateway is run on a nodeMCU-ESP8266, connected to ThingSpeak via Wi-Fi by another phone to upload the node’s GPS coordinates to determine the distances later on, as well as the RSSI, SNR, and frequency error of its received signals. Both the SX1278 modules in test are initialised with the following settings:

* LoRa frequency: 433 MHz
* Spreading factor: 12
* Signal bandwidth: 500 kHz
* Coding rate: 4/5
* Sync word: 0x92

Figure 4‑7 displays the locations on the Google Maps where the node transmits data to the server. The start point labelled “Start point – 0” is where the node is put next to the gateway, making the distance near 0 m. By using the distance measurement of Google Maps, the distances between the node and the gateway is determined as depicted in Table 4‑2. Although the walking path is a straight line, the markers for each location are not on the same path since GPS modules are rarely able to read the coordinations with a perfect precision.

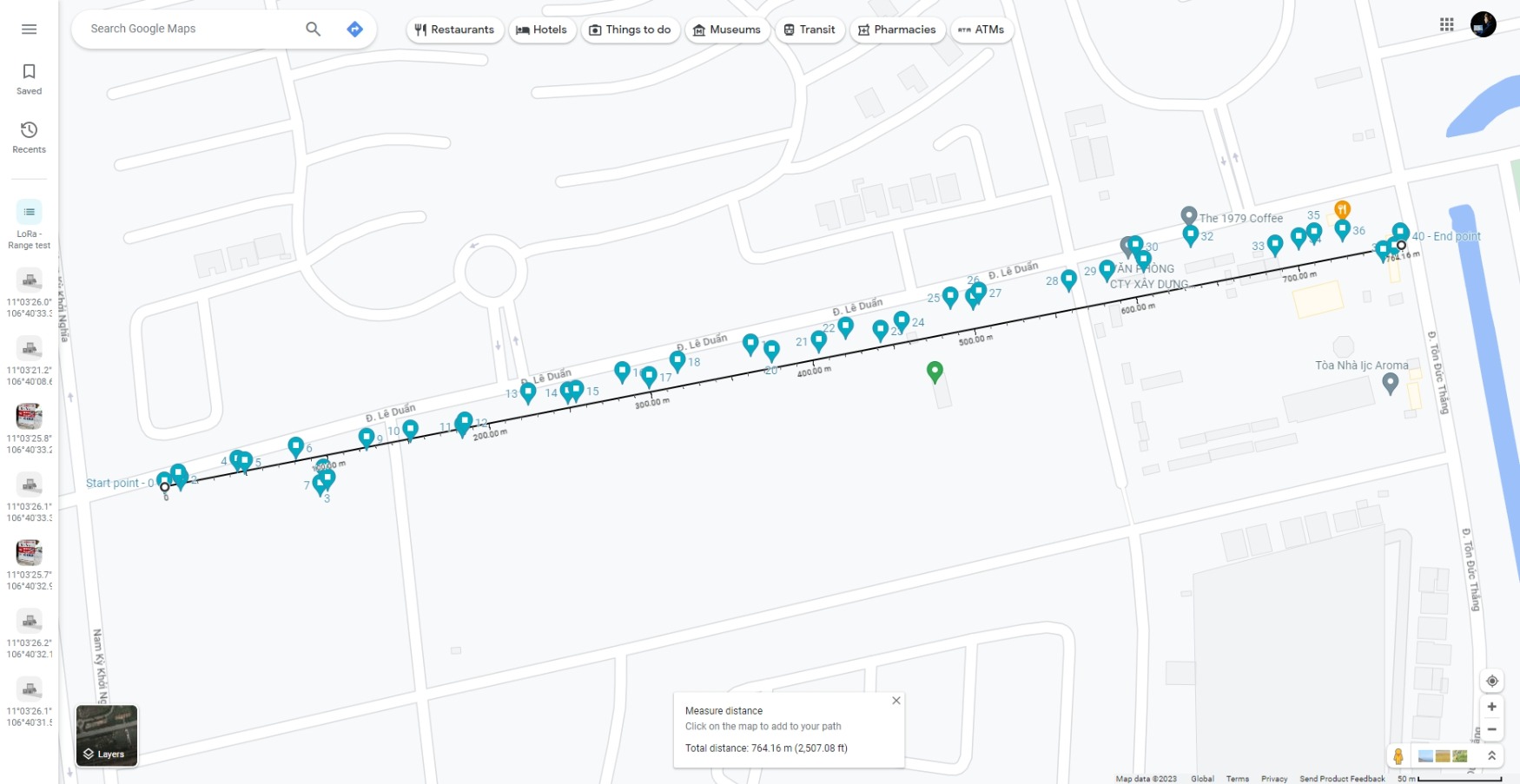


Figure 4‑21. LoRa Range Test – logged locations of LoRa transmissions by the node

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Index** | **Latitude** | **Longtitude** | **Distance (m)** | **RSSI (dBi)** | **SNR (dB)** | **Frequency error (Hz)** | **Index** | **Latitude** | **Longtitude** | **Distance (m)** | **RSSI (dBi)** | **SNR (dB)** | **Frequency error (Hz)** |
| 0 | 11.0559 | 106.669041 | 0 | -48 | 6.5 | -4669 | 23 | 11.0567 | 106.673019 | 443.71 | -110 | -6.75 | -4763 |
| 1 | 11.0559 | 106.669117 | 9.73 | -65 | 6.5 | -4686 | 24 | 11.0568 | 106.673138 | 457.72 | -110 | -8.25 | -4728 |
| 2 | 11.0559 | 106.669131 | 10.42 | -76 | 6.25 | -4744 | 25 | 11.0569 | 106.673409 | 489.78 | -111 | -11.75 | -4776 |
| 3 | 11.0559 | 106.669946 | 99.97 | -84 | 7 | -4774 | 26 | 11.0569 | 106.673534 | 502.91 | -111 | -9 | -4742 |
| 4 | 11.056 | 106.669446 | 46.6 | -89 | 6.75 | -4832 | 27 | 11.0569 | 106.673565 | 506.99 | -111 | -15.5 | -4780 |
| 5 | 11.056 | 106.669488 | 51.1 | -107 | -1.5 | -4851 | 28 | 11.057 | 106.674068 | 562.19 | -112 | -15.25 | -4734 |
| 6 | 11.0561 | 106.669772 | 83.38 | -88 | 6 | -4809 | 29 | 11.057 | 106.674277 | 585.82 | -111 | -17.75 | -4755 |
| 7 | 11.0559 | 106.669907 | 95.23 | -96 | 5.75 | -4870 | 30 | 11.0572 | 106.674437 | 606.01 | -109 | -14.5 | -4660 |
| 8 | 11.056 | 106.669924 | 97.39 | -91 | 6.25 | -4855 | 31 | 11.0571 | 106.674482 | 608.82 | -110 | -23.25 | -4763 |
| 9 | 11.0561 | 106.670162 | 125.93 | -99 | 3.5 | -4902 | 32 | 11.0572 | 106.674745 | 640.18 | -111 | -12.5 | -4612 |
| 10 | 11.0562 | 106.670407 | 153.16 | -101 | 4 | -4849 | 33 | 11.0572 | 106.675211 | 688.55 | -111 | -9 | -4579 |
| 11 | 11.0562 | 106.670696 | 183.89 | -102 | 3.25 | -4782 | 34 | 11.0572 | 106.675343 | 703.53 | -112 | -10 | -4545 |
| 12 | 11.0562 | 106.670708 | 185.61 | -104 | 1.25 | -4772 | 35 | 11.0572 | 106.675428 | 713.37 | -111 | -20.75 | -4545 |
| 13 | 11.0564 | 106.671063 | 227.27 | -105 | 1 | -4772 | 36 | 11.0573 | 106.675587 | 730.82 | -112 | -16.25 | -4591 |
| 14 | 11.0564 | 106.671284 | 250.92 | -109 | -3.75 | -4772 | 37 | 11.0571 | 106.6758 | 752.14 | -110 | -15.25 | -4577 |
| 15 | 11.0564 | 106.671329 | 255.98 | -104 | 1.25 | -4753 | 38 | 11.0572 | 106.6759 | 764.57 | -112 | -18 | -4579 |
| 16 | 11.0565 | 106.671586 | 286.04 | -107 | -1.5 | -4738 | 39 | 11.0572 | 106.6759 | 759.66 | -112 | -23.25 | -4532 |
| 17 | 11.0565 | 106.671733 | 300.79 | -108 | -3.25 | -4711 | 40 | 11.0572 | 106.675914 | 764.16 |  |  |  |
| 18 | 11.0565 | 106.671893 | 320.1 | -108 | -3.75 | -4656 | 41 | 11.0572 | 106.675914 | 764.16 |  |  |  |
| 19 | 11.0566 | 106.672299 | 365.68 | -110 | -13 | -4688 | 42 | 11.0572 | 106.675914 | 764.16 |  |  |  |
| 20 | 11.0566 | 106.672413 | 376.55 | -109 | -5.75 | -4721 | 43 | 11.0572 | 106.675914 | 764.16 |  |  |  |
| 21 | 11.0567 | 106.672675 | 405.79 | -111 | -12.5 | -4698 | 44 | 11.0572 | 106.675914 | 764.16 |  |  |  |
| 22 | 11.0567 | 106.672825 | 423.51 | -111 | -16.5 | -4721 | 45 | 11.0572 | 106.675914 | 764.16 |  |  |  |

Table 4‑5. LoRa range test result

The RSSI and SNR parameters of the received signals by the gateway are sent and charted on ThingSpeak. It could be observed that from the first to the tenth message by the node, the RSSI at the gateway drops the most drastically, which is -51 dBi in total after 125.93 m. Afterwards, it remains between -101 dBi and -112 dBi. The SNR also shows a generally downward slope as distance between the node and the gateway increases, with an addition of heavy fluctuations when the distance reaches 320.1 m.

|  |  |
| --- | --- |
|  |  |

Figure 4‑22. RSSI and SNR of the gateway displayed on ThingSpeak

Beyond 759.66 m, the communication between the node and the gateway could no longer be established. It is then determined that the communication range of SX1278 modules at the 500-kHz bandwidth is about 760 m. Since LoRa gains coverage by lowering the signal bandwidth [78], the maxium distance over which the SX1278 modules could communicate is expected to be much higher with proper settings.

Last but not least, this test on LoRa coverage initially aims at testing both uplink and downlink transmissions. However, as the SX1278 modules are put on the field, the downlink transmission (from gateway to node) is severed for unknown reason(s), thus only uplink results available in this experiment. Since the design for the Autonomous Wireless Agrometeorology Station focuses on uploading the data from the station (node) to a server via a gateway, the performance on the downlink connection does not bear heavy weights. Due to the lack of testing equipment and/or methodologies, there does not exist any findings on the data-exchanging rate and transmission success rate either.

# Conclusion and Further Development

The design of an Autonomous Wireless Agrometeorology Station introduces a new territory toward improvements of agricultural production via monitoring the meteorological factors, including wind parameters, precipitations, temperature, humidity, and atmospheric pressure. The available results suggest that the sensor outputs could become useful after calibrations, and remote data access in real time is possible with LoRa communication and ThingSpeak.

However, the system powering design remains basic and needs improvements, while a data backup solution is missing due to the invalidation of the microSD card in use. Therefore, the further development of the project is suggested as below:

* Redesign the system powering block by adding load balancing.
* Add a data backup solution. The microSD card module is suggested to remain, but another microcontroller (i.e. STM8S103F3P6) is to be put between the main one and the microSD card module as a bridge. The bridge is to receive data to be logged via UART then write those data on the microSD card for the main microcontroller.
* Evaluation of the anemometer to complete the wind speed model. Due to the aerodynamic nature of the device, this development may need collaborations with other engineering fields.
* Further evaluation of the rain gauge to determine the fault of the device on top of the product specifications by the manufacterer.
* Further evaluations of the digital sensors.
* Further improvements of communications via LoRa.
* Integration of all modules on a single prototype.

# APPENDIX A. HAL and DMA Configuration Functions for ADC

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \*\*\* HAL settings generated by STM32CubeMX \*\*\*

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* @brief ADC MSP Initialization

\* This function configures the hardware resources used in this example

\* @param hadc: ADC handle pointer

\* @retval None

\*/

extern "C" void HAL\_ADC\_MspInit(ADC\_HandleTypeDef\* hadc) {

  GPIO\_InitTypeDef GPIO\_InitStruct = {0};

  if (hadc->Instance == ADC1) {

    /\* Peripheral clock enable \*/

    \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE();

    \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

    /\*\*ADC1 GPIO Configuration

    PB1     ------> ADC1\_IN9

    \*/

    GPIO\_InitStruct.Pin = GPIO\_PIN\_1;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

    HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

    /\* ADC1 DMA Init \*/

    /\* ADC1 Init \*/

    hdma\_adc1.Instance = DMA1\_Channel1;

    hdma\_adc1.Init.Direction = DMA\_PERIPH\_TO\_MEMORY;

    hdma\_adc1.Init.PeriphInc = DMA\_PINC\_DISABLE;

    hdma\_adc1.Init.MemInc = DMA\_MINC\_ENABLE;

    hdma\_adc1.Init.PeriphDataAlignment = DMA\_PDATAALIGN\_HALFWORD;

    hdma\_adc1.Init.MemDataAlignment = DMA\_MDATAALIGN\_HALFWORD;

    hdma\_adc1.Init.Mode = DMA\_NORMAL;

    hdma\_adc1.Init.Priority = DMA\_PRIORITY\_HIGH;

    if (HAL\_DMA\_Init(&hdma\_adc1) != HAL\_OK) {

      while (1);

    }

    \_\_HAL\_LINKDMA(hadc,DMA\_Handle,hdma\_adc1);

  }

}

/\*\*

\* @brief ADC MSP De-Initialization

\* This function freeze the hardware resources used in this example

\* @param hadc: ADC handle pointer

\* @retval None

\*/

extern "C" void HAL\_ADC\_MspDeInit(ADC\_HandleTypeDef\* hadc) {

  if(hadc->Instance==ADC1) {

    /\* Peripheral clock disable \*/

    \_\_HAL\_RCC\_ADC1\_CLK\_DISABLE();

    /\*\*ADC1 GPIO Configuration

    PB1     ------> ADC1\_IN9

    \*/

    HAL\_GPIO\_DeInit(GPIOB, GPIO\_PIN\_1);

    /\* ADC1 DMA DeInit \*/

    HAL\_DMA\_DeInit(hadc->DMA\_Handle);

  }

}

/\*\*

  \* @brief This function handles DMA1 channel1 global interrupt.

  \*/

extern "C" void DMA1\_Channel1\_IRQHandler(void) {

  HAL\_DMA\_IRQHandler(&hdma\_adc1);

}

/\*\*

  \* @brief ADC1 Initialization Function

  \* @param None

  \* @retval None

  \*/

static void MX\_ADC1\_Init(ADC\_INPUT\_TYPE input\_type) {

  ADC\_ChannelConfTypeDef sConfig = {0};

  /\* USER CODE BEGIN ADC1\_Init 1 \*/

  if (!first\_run) {

    if (HAL\_ADC\_DeInit(&hadc1) != HAL\_OK)

    {

      while (1);

    }

  }

  else {

    first\_run = false;

  }

  /\* USER CODE END ADC1\_Init 1 \*/

  /\* USER CODE BEGIN ADC1\_Init 2 \*/

  if (INTERNAL\_REFERENCE\_VOLTAGE == input\_type) {

    /\*\* Common config

    \*/

    hadc1.Instance = ADC1;

    hadc1.Init.ScanConvMode = ADC\_SCAN\_DISABLE;

    hadc1.Init.ContinuousConvMode = ENABLE;

    hadc1.Init.DiscontinuousConvMode = DISABLE;

    hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

    hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

    hadc1.Init.NbrOfConversion = 1;

    if (HAL\_ADC\_Init(&hadc1) != HAL\_OK) {

      while (1);

    }

    /\*\* Configure Regular Channel

    \*/

    sConfig.Channel = ADC\_CHANNEL\_VREFINT;

    sConfig.Rank = ADC\_REGULAR\_RANK\_1;

    sConfig.SamplingTime = ADC\_SAMPLETIME\_7CYCLES\_5;

    if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK) {

      while (1);

    }

  }

  else if (EXTERNAL\_INPUT\_SIGNAL == input\_type) {

    /\*\* Common config

    \*/

    hadc1.Instance = ADC1;

    hadc1.Init.ScanConvMode = ADC\_SCAN\_DISABLE;

    hadc1.Init.ContinuousConvMode = ENABLE;

    hadc1.Init.DiscontinuousConvMode = DISABLE;

    hadc1.Init.ExternalTrigConv = ADC\_SOFTWARE\_START;

    hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

    hadc1.Init.NbrOfConversion = 1;

    if (HAL\_ADC\_Init(&hadc1) != HAL\_OK) {

      while (1);

    }

    /\*\* Configure Regular Channel

    \*/

    sConfig.Channel = ADC\_CHANNEL\_9;

    sConfig.Rank = ADC\_REGULAR\_RANK\_1;

    sConfig.SamplingTime = ADC\_SAMPLETIME\_7CYCLES\_5;

    if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK) {

      while (1);

    }

  }

  else {

    // Do nothing

  }

/\* USER CODE END ADC1\_Init 2 \*/

}

/\*\*

  \* Enable DMA controller clock

  \*/

static void MX\_DMA\_Init(void) {

  /\* DMA controller clock enable \*/

  \_\_HAL\_RCC\_DMA1\_CLK\_ENABLE();

  /\* DMA interrupt init \*/

  /\* DMA1\_Channel1\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(DMA1\_Channel1\_IRQn, 0, 0);

  HAL\_NVIC\_EnableIRQ(DMA1\_Channel1\_IRQn);

}

# APPENDIX B. CA3140EZ Voltage Buffer Test Result

|  |  |
| --- | --- |
|  |  |
| VCC = 3.3 V | VCC = 5V |
|  |  |
|  |  |
| VCC = 5.41 V | VCC = 5.9 V |
|  |  |
|  |  |
| VCC = 6 V | VCC = 6.3 V |

# APPENDIX C. Op-Amp Pick-up Test for Voltage Buffer at 3.3-V Power Supply

|  |  |
| --- | --- |
|  |  |
| CA3140EZ | LM358P |
|  |  |
|  |  |
| LM393P | UA741CP |
|  |  |
|  | |
| MCP6002-I/P | |

# APPENDIX D. Wind Vane Test Result for Circuitry of Figure 3‑14

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input direction:  0 | | Voltmeter: 2.9949 V | | Input direction: 22.5 | | Voltmeter: 2.1995 V | | Input direction:  45 | | Voltmeter: 2.3538 V | | Input direction: 67.5 | | Voltmeter: 0.7084 V | |
| **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** |
| 0 | 3741 | 2.99 | 35644 | 22.5 | 2740 | 2.19 | 6680.4 | 45 | 2933 | 2.34 | 8339.5 | 67.5 | 867 | 0.69 | 885.04 |
| 0 | 3738 | 2.99 | 34874 | 22.5 | 2737 | 2.19 | 6673.1 | 45 | 2933 | 2.34 | 8329.5 | 67.5 | 864 | 0.69 | 886.34 |
| 0 | 3739 | 2.98 | 34553 | 22.5 | 2739 | 2.19 | 6651 | 45 | 2930 | 2.34 | 8329.5 | 67.5 | 873 | 0.69 | 882.45 |
| 0 | 3741 | 2.99 | 34659 | 22.5 | 2734 | 2.19 | 6665.7 | 45 | 2929 | 2.34 | 8299.6 | 67.5 | 863 | 0.7 | 894.13 |
| 0 | 3736 | 2.99 | 34874 | 22.5 | 2735 | 2.18 | 6629.1 | 45 | 2931 | 2.34 | 8289.6 | 67.5 | 861 | 0.69 | 881.16 |
| 0 | 3734 | 2.98 | 34342 | 22.5 | 2739 | 2.18 | 6636.4 | 45 | 2933 | 2.34 | 8309.5 | 67.5 | 865 | 0.69 | 878.57 |
| 0 | 3736 | 2.98 | 34134 | 22.5 | 2735 | 2.19 | 6665.7 | 45 | 2934 | 2.34 | 8329.5 | 67.5 | 863 | 0.69 | 883.75 |
| 0 | 3739 | 2.98 | 34342 | 22.5 | 2738 | 2.18 | 6636.4 | 45 | 2933 | 2.34 | 8339.5 | 67.5 | 866 | 0.69 | 881.16 |
| 0 | 3741 | 2.99 | 34659 | 22.5 | 2739 | 2.19 | 6658.4 | 45 | 2933 | 2.34 | 8329.5 | 67.5 | 877 | 0.69 | 885.04 |
| 0 | 3741 | 2.99 | 34874 | 22.5 | 2735 | 2.19 | 6665.7 | 45 | 2929 | 2.34 | 8329.5 | 67.5 | 863 | 0.7 | 899.35 |
| 0 | 3741 | 2.99 | 34874 | 22.5 | 2745 | 2.18 | 6636.4 | 45 | 2931 | 2.34 | 8289.6 | 67.5 | 869 | 0.69 | 881.16 |
| 0 | 3736 | 2.99 | 34874 | 22.5 | 2734 | 2.19 | 6710 | 45 | 2933 | 2.34 | 8309.5 | 67.5 | 864 | 0.69 | 888.93 |
| 0 | 3738 | 2.98 | 34342 | 22.5 | 2737 | 2.18 | 6629.1 | 45 | 2931 | 2.34 | 8329.5 | 67.5 | 865 | 0.69 | 882.45 |
| 0 | 3738 | 2.98 | 34553 | 22.5 | 2738 | 2.19 | 6651 | 45 | 2934 | 2.34 | 8309.5 | 67.5 | 866 | 0.69 | 883.75 |
| 0 | 3738 | 2.98 | 34553 | 22.5 | 2739 | 2.19 | 6658.4 | 45 | 2934 | 2.34 | 8339.5 | 67.5 | 867 | 0.69 | 885.04 |
| 0 | 3737 | 2.98 | 34553 | 22.5 | 2726 | 2.19 | 6665.7 | 45 | 2936 | 2.34 | 8339.5 | 67.5 | 865 | 0.69 | 886.34 |
| 0 | 3739 | 2.98 | 34447 | 22.5 | 2737 | 2.18 | 6571.1 | 45 | 2936 | 2.34 | 8359.6 | 67.5 | 863 | 0.69 | 883.75 |
| 0 | 3737 | 2.99 | 34659 | 22.5 | 2739 | 2.19 | 6651 | 45 | 2931 | 2.34 | 8359.6 | 67.5 | 864 | 0.69 | 881.16 |
| 0 | 3736 | 2.98 | 34447 | 22.5 | 2736 | 2.19 | 6665.7 | 45 | 2930 | 2.34 | 8309.5 | 67.5 | 865 | 0.69 | 882.45 |
| 0 | 3736 | 2.98 | 34342 | 22.5 | 2739 | 2.18 | 6643.7 | 45 | 2937 | 2.34 | 8299.6 | 67.5 | 866 | 0.69 | 883.75 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input direction:  90 | | Voltmeter: 0.7741 V | | Input direction:  112.5 | | Voltmeter: 0.5747 V | | Input direction: 135 | | Voltmeter: 1.3235 V | | Input direction:  157.5 | | Voltmeter: 0.9908 V | |
| **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** |
| 90 | 947 | 0.76 | 1000.9 | 112.5 | 694 | 0.56 | 676.9 | 135 | 1650 | 1.32 | 2220.2 | 157.5 | 1220 | 0.98 | 1408.5 |
| 90 | 957 | 0.76 | 992.73 | 112.5 | 698 | 0.55 | 673.39 | 135 | 1644 | 1.32 | 2227 | 157.5 | 1229 | 0.97 | 1400.4 |
| 90 | 950 | 0.76 | 1006.4 | 112.5 | 691 | 0.56 | 678.07 | 135 | 1645 | 1.31 | 2213.5 | 157.5 | 1224 | 0.98 | 1415.1 |
| 90 | 953 | 0.76 | 996.82 | 112.5 | 693 | 0.55 | 669.89 | 135 | 1645 | 1.31 | 2215.7 | 157.5 | 1224 | 0.98 | 1406.9 |
| 90 | 949 | 0.76 | 1000.9 | 112.5 | 697 | 0.55 | 672.22 | 135 | 1646 | 1.31 | 2215.7 | 157.5 | 1225 | 0.98 | 1406.9 |
| 90 | 947 | 0.76 | 995.45 | 112.5 | 696 | 0.56 | 676.9 | 135 | 1650 | 1.31 | 2218 | 157.5 | 1221 | 0.98 | 1408.5 |
| 90 | 946 | 0.76 | 992.73 | 112.5 | 694 | 0.56 | 675.73 | 135 | 1640 | 1.32 | 2227 | 157.5 | 1226 | 0.98 | 1402 |
| 90 | 947 | 0.76 | 991.36 | 112.5 | 695 | 0.55 | 673.39 | 135 | 1646 | 1.31 | 2204.5 | 157.5 | 1226 | 0.98 | 1410.2 |
| 90 | 947 | 0.76 | 992.73 | 112.5 | 702 | 0.55 | 674.56 | 135 | 1649 | 1.31 | 2218 | 157.5 | 1224 | 0.98 | 1410.2 |
| 90 | 946 | 0.76 | 992.73 | 112.5 | 696 | 0.56 | 682.76 | 135 | 1648 | 1.32 | 2224.7 | 157.5 | 1232 | 0.98 | 1406.9 |
| 90 | 947 | 0.76 | 991.36 | 112.5 | 694 | 0.56 | 675.73 | 135 | 1648 | 1.32 | 2222.5 | 157.5 | 1226 | 0.98 | 1420.1 |
| 90 | 948 | 0.76 | 992.73 | 112.5 | 696 | 0.55 | 673.39 | 135 | 1647 | 1.32 | 2222.5 | 157.5 | 1223 | 0.98 | 1410.2 |
| 90 | 948 | 0.76 | 994.09 | 112.5 | 692 | 0.56 | 675.73 | 135 | 1645 | 1.32 | 2220.2 | 157.5 | 1226 | 0.98 | 1405.3 |
| 90 | 947 | 0.76 | 994.09 | 112.5 | 712 | 0.55 | 671.05 | 135 | 1645 | 1.31 | 2215.7 | 157.5 | 1225 | 0.98 | 1410.2 |
| 90 | 947 | 0.76 | 992.73 | 112.5 | 693 | 0.57 | 694.53 | 135 | 1644 | 1.31 | 2215.7 | 157.5 | 1225 | 0.98 | 1408.5 |
| 90 | 949 | 0.76 | 992.73 | 112.5 | 696 | 0.55 | 672.22 | 135 | 1646 | 1.31 | 2213.5 | 157.5 | 1225 | 0.98 | 1408.5 |
| 90 | 950 | 0.76 | 995.45 | 112.5 | 695 | 0.56 | 675.73 | 135 | 1650 | 1.31 | 2218 | 157.5 | 1225 | 0.98 | 1408.5 |
| 90 | 949 | 0.76 | 996.82 | 112.5 | 696 | 0.55 | 674.56 | 135 | 1647 | 1.32 | 2227 | 157.5 | 1222 | 0.98 | 1408.5 |
| 90 | 963 | 0.76 | 995.45 | 112.5 | 699 | 0.56 | 675.73 | 135 | 1645 | 1.32 | 2220.2 | 157.5 | 1226 | 0.98 | 1403.6 |
| 90 | 948 | 0.77 | 1014.7 | 112.5 | 696 | 0.56 | 679.24 | 135 | 1646 | 1.31 | 2215.7 | 157.5 | 1224 | 0.98 | 1410.2 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input direction:  180 | | Voltmeter: 1.7911 V | | Input direction:  202.5 | | Voltmeter: 1.6124 V | | Input direction:  225 | | Voltmeter: 2.7332 V | | Input direction:  247.5 | | Voltmeter: 2.6726 V | |
| **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** |
| 180 | 2233 | 1.79 | 4012.5 | 202.5 | 1988 | 1.6 | 3165.8 | 225 | 3400 | 2.72 | 16228 | 247.5 | 3323 | 2.66 | 14365 |
| 180 | 2232 | 1.78 | 3957.5 | 202.5 | 2010 | 1.59 | 3113.6 | 225 | 3408 | 2.72 | 16144 | 247.5 | 3324 | 2.65 | 14205 |
| 180 | 2228 | 1.78 | 3953.6 | 202.5 | 2008 | 1.61 | 3181.3 | 225 | 3400 | 2.72 | 16370 | 247.5 | 3325 | 2.65 | 14227 |
| 180 | 2232 | 1.78 | 3938.1 | 202.5 | 2008 | 1.6 | 3175.1 | 225 | 3402 | 2.72 | 16144 | 247.5 | 3323 | 2.66 | 14250 |
| 180 | 2230 | 1.78 | 3953.6 | 202.5 | 2006 | 1.6 | 3175.1 | 225 | 3403 | 2.72 | 16200 | 247.5 | 3323 | 2.65 | 14205 |
| 180 | 2227 | 1.78 | 3945.8 | 202.5 | 2008 | 1.6 | 3168.9 | 225 | 3400 | 2.72 | 16228 | 247.5 | 3325 | 2.65 | 14205 |
| 180 | 2230 | 1.78 | 3934.2 | 202.5 | 2008 | 1.6 | 3175.1 | 225 | 3401 | 2.72 | 16144 | 247.5 | 3325 | 2.66 | 14250 |
| 180 | 2227 | 1.78 | 3945.8 | 202.5 | 2008 | 1.6 | 3175.1 | 225 | 3406 | 2.72 | 16172 | 247.5 | 3327 | 2.66 | 14250 |
| 180 | 2230 | 1.78 | 3934.2 | 202.5 | 2009 | 1.6 | 3175.1 | 225 | 3397 | 2.72 | 16313 | 247.5 | 3327 | 2.66 | 14296 |
| 180 | 2232 | 1.78 | 3945.8 | 202.5 | 2005 | 1.6 | 3178.2 | 225 | 3402 | 2.71 | 16060 | 247.5 | 3323 | 2.66 | 14296 |
| 180 | 2234 | 1.78 | 3953.6 | 202.5 | 2014 | 1.6 | 3165.8 | 225 | 3402 | 2.72 | 16200 | 247.5 | 3326 | 2.65 | 14205 |
| 180 | 2229 | 1.78 | 3961.4 | 202.5 | 2006 | 1.61 | 3193.8 | 225 | 3402 | 2.72 | 16200 | 247.5 | 3326 | 2.66 | 14273 |
| 180 | 2233 | 1.78 | 3942 | 202.5 | 2008 | 1.6 | 3168.9 | 225 | 3400 | 2.72 | 16200 | 247.5 | 3327 | 2.66 | 14273 |
| 180 | 2228 | 1.78 | 3957.5 | 202.5 | 2009 | 1.6 | 3175.1 | 225 | 3400 | 2.72 | 16144 | 247.5 | 3325 | 2.66 | 14296 |
| 180 | 2230 | 1.78 | 3938.1 | 202.5 | 2006 | 1.6 | 3178.2 | 225 | 3402 | 2.72 | 16144 | 247.5 | 3328 | 2.66 | 14250 |
| 180 | 2226 | 1.78 | 3945.8 | 202.5 | 2006 | 1.6 | 3168.9 | 225 | 3402 | 2.72 | 16200 | 247.5 | 3326 | 2.66 | 14319 |
| 180 | 2229 | 1.78 | 3930.3 | 202.5 | 2006 | 1.6 | 3168.9 | 225 | 3400 | 2.72 | 16200 | 247.5 | 3328 | 2.66 | 14273 |
| 180 | 2229 | 1.78 | 3942 | 202.5 | 2004 | 1.6 | 3168.9 | 225 | 3402 | 2.72 | 16144 | 247.5 | 3320 | 2.66 | 14319 |
| 180 | 2229 | 1.78 | 3942 | 202.5 | 2007 | 1.6 | 3162.7 | 225 | 3400 | 2.72 | 16200 | 247.5 | 3343 | 2.65 | 14137 |
| 180 | 2228 | 1.78 | 3942 | 202.5 | 2004 | 1.6 | 3172 | 225 | 3400 | 2.72 | 16144 | 247.5 | 3325 | 2.67 | 14670 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input direction:  270 | | Voltmeter: 3.2039 V | | Input direction:  292.5 | | Voltmeter: 3.0544 V | | Input direction:  315 | | Voltmeter: 3.1338 V | | Input direction:  337.5 | | Voltmeter: 2.8647 V | |
| **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** | **Read direction** | **Raw ADC** | **Vin** | **Rread** |
| 270 | 3987 | 3.19 | 131835 | 292.5 | 3813 | 3.05 | 45135 | 315 | 3904 | 3.12 | 68580 | 337.5 | 3571 | 2.85 | 22294 |
| 270 | 3987 | 3.18 | 121825 | 292.5 | 3808 | 3.04 | 44620 | 315 | 3906 | 3.12 | 67451 | 337.5 | 3563 | 2.85 | 22489 |
| 270 | 3992 | 3.18 | 121825 | 292.5 | 3811 | 3.04 | 43785 | 315 | 3899 | 3.12 | 68200 | 337.5 | 3568 | 2.85 | 22101 |
| 270 | 3988 | 3.19 | 127899 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3909 | 3.11 | 65646 | 337.5 | 3566 | 2.85 | 22342 |
| 270 | 3987 | 3.18 | 122994 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3904 | 3.12 | 69353 | 337.5 | 3571 | 2.85 | 22245 |
| 270 | 3990 | 3.18 | 121825 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3906 | 3.12 | 67451 | 337.5 | 3568 | 2.85 | 22489 |
| 270 | 3987 | 3.19 | 125400 | 292.5 | 3813 | 3.04 | 44283 | 315 | 3904 | 3.12 | 68200 | 337.5 | 3566 | 2.85 | 22342 |
| 270 | 3994 | 3.18 | 121825 | 292.5 | 3811 | 3.04 | 44620 | 315 | 3906 | 3.12 | 67451 | 337.5 | 3567 | 2.85 | 22245 |
| 270 | 3990 | 3.19 | 130497 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3905 | 3.12 | 68200 | 337.5 | 3566 | 2.85 | 22294 |
| 270 | 3992 | 3.19 | 125400 | 292.5 | 3802 | 3.04 | 44283 | 315 | 3907 | 3.12 | 67824 | 337.5 | 3568 | 2.85 | 22245 |
| 270 | 3993 | 3.19 | 127899 | 292.5 | 3813 | 3.04 | 42821 | 315 | 3907 | 3.12 | 68580 | 337.5 | 3557 | 2.85 | 22342 |
| 270 | 3987 | 3.19 | 129185 | 292.5 | 3816 | 3.04 | 44620 | 315 | 3907 | 3.12 | 68580 | 337.5 | 3567 | 2.84 | 21818 |
| 270 | 3994 | 3.18 | 121825 | 292.5 | 3810 | 3.05 | 45135 | 315 | 3904 | 3.12 | 68580 | 337.5 | 3565 | 2.85 | 22294 |
| 270 | 3987 | 3.19 | 130497 | 292.5 | 3811 | 3.04 | 44116 | 315 | 3906 | 3.12 | 67451 | 337.5 | 3566 | 2.85 | 22197 |
| 270 | 3987 | 3.18 | 121825 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3906 | 3.12 | 68200 | 337.5 | 3563 | 2.85 | 22245 |
| 270 | 3984 | 3.18 | 121825 | 292.5 | 3803 | 3.04 | 44283 | 315 | 3906 | 3.12 | 68200 | 337.5 | 3563 | 2.85 | 22101 |
| 270 | 3989 | 3.18 | 118443 | 292.5 | 3817 | 3.04 | 42979 | 315 | 3907 | 3.12 | 68200 | 337.5 | 3565 | 2.85 | 22101 |
| 270 | 3990 | 3.19 | 124186 | 292.5 | 3811 | 3.05 | 45310 | 315 | 3906 | 3.12 | 68580 | 337.5 | 3568 | 2.85 | 22197 |
| 270 | 3990 | 3.19 | 125400 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3906 | 3.12 | 68200 | 337.5 | 3563 | 2.85 | 22342 |
| 270 | 3987 | 3.19 | 125400 | 292.5 | 3811 | 3.04 | 44283 | 315 | 3904 | 3.12 | 68200 | 337.5 | 3563 | 2.85 | 22101 |

# APPENDIX E. Python Program on Raspberry Pi 4B

import os

import time

import serial

import dropbox

import dropbox.files

# Serial handler(s)

ser = serial.Serial("/dev/ttyS0", 9600)

serial\_input\_available = False

received\_string = ''

# Camera handler(s)

webcamBaseCommand = "fswebcam -r 640x480 "

fileDirectory = "/home/pi/Desktop/WebcamLogger/entries"

fileNameIndex = 0

fileNameExtension = ".jpg"

# Dropbox handler(s)

fileForDropbox = ''

fileOnDropbox = ''

APP\_KEY = 'c4zjrcgqwz9ym02'

APP\_SECRET = '9oy93mpg5gre7tz'

REFRESH\_TOKEN = '3RoiSnQVa4wAAAAAAAAAAZaNqTksHYuLQ6dxacvw4kBB3JFZ2ujUH-zwENNDmaG\_'

dbc = dropbox.Dropbox(

    app\_key=APP\_KEY,

    app\_secret=APP\_SECRET,

    oauth2\_refresh\_token=REFRESH\_TOKEN

)

# Taking a picture with a designated name

def webcamCapture():

    global fileForDropbox, fileOnDropbox, fileNameIndex

    if fileNameIndex < 10:

        fileOnDropbox = "/000"

    elif fileNameIndex < 100:

        fileOnDropbox = "/00"

    elif fileNameIndex < 1000:

        fileOnDropbox = "/0"

    else:

        fileOnDropbox = "/"

    fileOnDropbox += f"{fileNameIndex}"

    fileOnDropbox += fileNameExtension

    fileForDropbox = fileDirectory + fileOnDropbox

    webcamCaptureCommand = webcamBaseCommand + fileForDropbox

    try:

        os.system(webcamCaptureCommand)

    except:

        webcamCapture()

# Upload to Dropbox

def dropbox\_captureUpload():

    global fileForDropbox, fileOnDropbox

    with open(fileForDropbox, 'rb') as f:

        data = f.read()

        dbc.files\_upload(data, fileOnDropbox, mode=dropbox.files.WriteMode('overwrite'))

# Monitor and handle serial events

def UART\_watch():

    global received\_string, serial\_input\_available, fileNameIndex

    rx\_data = ser.readline()

    rx\_data = rx\_data.strip()

    received\_string = rx\_data.decode("utf-8")

    dataPoint\_Index = received\_string.split(',')[0]

    if dataPoint\_Index.isnumeric():

        fileNameIndex = int(dataPoint\_Index)

        serial\_input\_available = True

    else:

        with open('logger.csv', 'a') as f:

            f.write(received\_string)

            f.write('\n')

            f.close()

def main():

    global serial\_input\_available

    while True:

        UART\_watch()

        if serial\_input\_available:

            webcamCapture()

            try:

                dropbox\_captureUpload()

            except:

                print("Network issue.")

            with open('logger.csv', 'a') as f:

                f.write(received\_string)

                f.write(',' + fileOnDropbox[1:])

                f.write('\n')

                f.close()

if \_\_name\_\_ == '\_\_main\_\_':

    main()

# APPENDIX F. LoRa Range Test – Node’s Log

14:20:12.848 USB device detected

14:20:22.573 Connected to CP210x device

14:23:19.989 Input: 11.055887,106.669041

14:23:20.017 Index: 0

14:23:20.026 Latitude = 11.055887; Longtitude = 106.669041

14:23:20.448 Coordinations sent.

14:23:59.072 Input: 11.055929,106.669117

14:23:59.100 Index: 1

14:23:59.109 Latitude = 11.055929; Longtitude = 106.669117

14:23:59.531 Coordinations sent.

14:24:33.659 Input: 11.055904,106.669131

14:24:33.688 Index: 2

14:24:33.696 Latitude = 11.055904; Longtitude = 106.669131

14:24:34.119 Coordinations sent.

14:25:03.464 Input: 11.055904,106.669946

14:25:03.491 Index: 3

14:25:03.501 Latitude = 11.055904; Longtitude = 106.669946

14:25:03.924 Coordinations sent.

14:25:37.438 Input: 11.056005,106.669446

14:25:37.466 Index: 4

14:25:37.475 Latitude = 11.056005; Longtitude = 106.669446

14:25:37.898 Coordinations sent.

14:26:11.659 Input: 11.055999,106.669488

14:26:11.688 Index: 5

14:26:11.696 Latitude = 11.055999; Longtitude = 106.669488

14:26:12.118 Coordinations sent.

14:26:42.179 Input: 11.056077,106.669772

14:26:42.208 Index: 6

14:26:42.216 Latitude = 11.056077; Longtitude = 106.669772

14:26:42.638 Coordinations sent.

14:27:19.506 Input: 11.055875,106.669907

14:27:19.534 Index: 7

14:27:19.542 Latitude = 11.055875; Longtitude = 106.669907

14:27:19.965 Coordinations sent.

14:27:54.840 Input: 11.055959,106.669924

14:27:54.868 Index: 8

14:27:54.877 Latitude = 11.055959; Longtitude = 106.669924

14:27:55.300 Coordinations sent.

14:28:41.034 Input: 11.056128,106.670162

14:28:41.062 Index: 9

14:28:41.071 Latitude = 11.056128; Longtitude = 106.670162

14:28:41.493 Coordinations sent.

14:29:19.124 Input: 11.056172,106.670407

14:29:19.151 Index: 10

14:29:19.160 Latitude = 11.056172; Longtitude = 106.670407

14:29:19.583 Coordinations sent.

14:29:56.104 Input: 11.056193,106.670696

14:29:56.132 Index: 11

14:29:56.142 Latitude = 11.056193; Longtitude = 106.670696

14:29:56.564 Coordinations sent.

14:30:31.986 Input: 11.056214,106.670708

14:30:32.014 Index: 12

14:30:32.023 Latitude = 11.056214; Longtitude = 106.67070

14:30:32.445 Coordinations sent.

14:31:11.523 Input: 11.056376,106.671063

14:31:11.551 Index: 13

14:31:11.560 Latitude = 11.056376; Longtitude = 106.671063

14:31:11.982 Coordinations sent.

14:31:49.314 Input: 11.05638,106.671284

14:31:49.341 Index: 14

14:31:49.350 Latitude = 11.05638; Longtitude = 106.671284

14:31:49.706 Coordinations sent.

14:32:29.950 Input: 11.056388,106.671329

14:32:29.978 Index: 15

14:32:29.987 Latitude = 11.056388; Longtitude = 106.671329

14:32:30.409 Coordinations sent.

14:33:02.989 Input: 11.056491,106.671586

14:33:03.017 Index: 16

14:33:03.025 Latitude = 11.056491; Longtitude = 106.671586

14:33:03.448 Coordinations sent.

14:33:39.486 Input: 11.056461,106.671733

14:33:39.514 Index: 17

14:33:39.522 Latitude = 11.056461; Longtitude = 106.671733

14:33:39.945 Coordinations sent.

14:34:09.441 Input: 11.056548,106.671893

14:34:09.468 Index: 18

14:34:09.477 Latitude = 11.056548; Longtitude = 106.671893

14:34:09.900 Coordinations sent.

14:34:48.494 Input: 11.056641,106.672299

14:34:48.521 Index: 19

14:34:48.530 Latitude = 11.056641; Longtitude = 106.672299

14:34:48.953 Coordinations sent.

14:35:19.346 Input: 11.056602,106.672413

14:35:19.374 Index: 20

14:35:19.383 Latitude = 11.056602; Longtitude = 106.672413

14:35:19.805 Coordinations sent.

14:35:59.694 Input: 11.056656,106.672675

14:35:59.723 Index: 21

14:35:59.731 Latitude = 11.056656; Longtitude = 106.672675

14:36:00.154 Coordinations sent.

14:36:35.079 Input: 11.056731,106.672825

14:36:35.108 Index: 22

14:36:35.117 Latitude = 11.056731; Longtitude = 106.672825

14:36:35.539 Coordinations sent.

14:37:10.961 Input: 11.056715,106.673019

14:37:10.989 Index: 23

14:37:10.999 Latitude = 11.056715; Longtitude = 106.673019

14:37:11.421 Coordinations sent.

14:37:45.747 Input: 11.056764,106.673138

14:37:45.775 Index: 24

14:37:45.784 Latitude = 11.056764; Longtitude = 106.673138

14:37:46.206 Coordinations sent.

14:38:22.112 Input: 11.056897,106.673409

14:38:22.141 Index: 25

14:38:22.151 Latitude = 11.056897; Longtitude = 106.673409

14:38:22.572 Coordinations sent.

14:39:00.236 Input: 11.056891,106.673534

14:39:00.264 Index: 26

14:39:00.273 Latitude = 11.056891; Longtitude = 106.673534

14:39:00.695 Coordinations sent.

14:39:30.772 Input: 11.056922,106.673565

14:39:30.801 Index: 27

14:39:30.810 Latitude = 11.056922; Longtitude = 106.673565

14:39:31.232 Coordinations sent.

14:40:10.938 Input: 11.056991,106.674068

14:40:10.967 Index: 28

14:40:10.975 Latitude = 11.056991; Longtitude = 106.674068

14:40:11.398 Coordinations sent.

14:40:50.857 Input: 11.057044,106.674277

14:40:50.884 Index: 29

14:40:50.893 Latitude = 11.057044; Longtitude = 106.674277

14:40:51.316 Coordinations sent.

14:41:30.659 Input: 11.057174,106.674437

14:41:30.687 Index: 30

14:41:30.695 Latitude = 11.057174; Longtitude = 106.674437

14:41:31.118 Coordinations sent.

14:42:06.691 Input: 11.057097,106.674482

14:42:06.719 Index: 31

14:42:06.728 Latitude = 11.057097; Longtitude = 106.674482

14:42:07.150 Coordinations sent.

14:42:46.277 Input: 11.057232,106.674745

14:42:46.305 Index: 32

14:42:46.314 Latitude = 11.057232; Longtitude = 106.674745

14:42:46.736 Coordinations sent.

14:43:27.521 Input: 11.057179,106.675211

14:43:27.550 Index: 33

14:43:27.559 Latitude = 11.057179; Longtitude = 106.675211

14:43:27.982 Coordinations sent.

14:44:03.636 Input: 11.057219,106.675343

14:44:03.664 Index: 34

14:44:03.673 Latitude = 11.057219; Longtitude = 106.675343

14:44:04.094 Coordinations sent.

14:44:35.186 Input: 11.057246,106.675428

14:44:35.214 Index: 35

14:44:35.223 Latitude = 11.057246; Longtitude = 106.675428

14:44:35.645 Coordinations sent.

14:45:09.276 Input: 11.057264,106.675587

14:45:09.303 Index: 36

14:45:09.312 Latitude = 11.057264; Longtitude = 106.675587

14:45:09.735 Coordinations sent.

14:45:42.666 Input: 11.057148,106.675811

14:45:42.693 Index: 37

14:45:42.703 Latitude = 11.057148; Longtitude = 106.675811

14:45:43.125 Coordinations sent.

14:46:24.195 Input: 11.057244,106.675908

14:46:24.223 Index: 38

14:46:24.232 Latitude = 11.057244; Longtitude = 106.675908

14:46:24.654 Coordinations sent.

14:47:08.745 Input: 11.057171,106.675876

14:47:08.773 Index: 39

14:47:08.783 Latitude = 11.057171; Longtitude = 106.675876

14:47:09.205 Coordinations sent.

14:47:57.281 Input: 11.057233,106.675914

14:47:57.309 Index: 40

14:47:57.318 Latitude = 11.057233; Longtitude = 106.675914

14:47:57.740 Coordinations sent.

14:48:16.773 Input: 11.057233,106.675914

14:48:16.802 Index: 41

14:48:16.810 Latitude = 11.057233; Longtitude = 106.675914

14:48:17.233 Coordinations sent.

14:48:26.918 Input: 11.057233,106.675914

14:48:26.946 Index: 42

14:48:26.956 Latitude = 11.057233; Longtitude = 106.675914

14:48:27.378 Coordinations sent.

14:48:41.547 Input: 11.057233,106.675914

14:48:41.575 Index: 43

14:48:41.583 Latitude = 11.057233; Longtitude = 106.675914

14:48:42.006 Coordinations sent.

14:48:55.262 Input: 11.057233,106.675914

14:48:55.289 Index: 44

14:48:55.299 Latitude = 11.057233; Longtitude = 106.675914

14:48:55.722 Coordinations sent.

14:49:01.740 Input: 11.057233,106.675914

14:49:01.768 Index: 45

14:49:01.778 Latitude = 11.057233; Longtitude = 106.675914

14:49:02.200 Coordinations sent.

14:49:52.150 Input: 11.057233,106.675914

14:49:52.178 Index: 46

14:49:52.187 Latitude = 11.057233; Longtitude = 106.675914

14:49:52.609 Coordinations sent.

14:50:51.511 Disconnected

# APPENDIX G. LoRa Range Test – Gateway’s Log

14:20:48.097 LoRa init succeeded.

14:21:37.660 Attempting to connect

14:21:47.645 Connected.

14:21:47.645

14:23:19.484 (Latitude, Longtitude) = (11.055887,106.669041)

14:23:19.516 RSSI = -48 dBi

14:23:19.550 SNR = 6.50 dB

14:23:19.550 Frequency error = -4669 Hz

14:23:21.245 TxDone

14:23:21.245

14:23:58.566 (Latitude, Longtitude) = (11.055929,106.669117)

14:23:58.599 RSSI = -65 dBi

14:23:58.632 SNR = 6.50 dB

14:23:58.632 Frequency error = -4686 Hz

14:24:00.186 TxDone

14:24:00.186

14:24:33.154 (Latitude, Longtitude) = (11.055904,106.669131)

14:24:33.188 RSSI = -76 dBi

14:24:33.221 SNR = 6.25 dB

14:24:33.221 Frequency error = -4744 Hz

14:24:34.768 TxDone

14:24:34.768

14:25:02.958 (Latitude, Longtitude) = (11.055904,106.669946)

14:25:02.991 RSSI = -84 dBi

14:25:03.025 SNR = 7.00 dB

14:25:03.025 Frequency error = -4774 Hz

14:25:04.566 TxDone

14:25:04.566

14:25:36.932 (Latitude, Longtitude) = (11.056005,106.669446)

14:25:36.965 RSSI = -89 dBi

14:25:36.999 SNR = 6.75 dB

14:25:36.999 Frequency error = -4832 Hz

14:25:38.357 TxDone

14:25:38.357

14:26:11.153 (Latitude, Longtitude) = (11.055999,106.669488)

14:26:11.187 RSSI = -107 dBi

14:26:11.220 SNR = -1.50 dB

14:26:11.220 Frequency error = -4851 Hz

14:26:12.664 TxDone

14:26:12.664

14:26:41.674 (Latitude, Longtitude) = (11.056077,106.669772)

14:26:41.707 RSSI = -88 dBi

14:26:41.740 SNR = 6.00 dB

14:26:41.740 Frequency error = -4809 Hz

14:26:43.382 TxDone

14:26:43.382

14:27:19.000 (Latitude, Longtitude) = (11.055875,106.669907)

14:27:19.033 RSSI = -96 dBi

14:27:19.067 SNR = 5.75 dB

14:27:19.067 Frequency error = -4870 Hz

14:27:20.499 TxDone

14:27:20.499

14:27:54.334 (Latitude, Longtitude) = (11.055959,106.669924)

14:27:54.368 RSSI = -91 dBi

14:27:54.401 SNR = 6.25 dB

14:27:54.401 Frequency error = -4855 Hz

14:27:55.696 TxDone

14:27:55.696

14:28:40.528 (Latitude, Longtitude) = (11.056128,106.670162)

14:28:40.562 RSSI = -99 dBi

14:28:40.595 SNR = 3.50 dB

14:28:40.595 Frequency error = -4902 Hz

14:28:42.413 TxDone

14:28:42.413

14:29:18.618 (Latitude, Longtitude) = (11.056172,106.670407)

14:29:18.651 RSSI = -101 dBi

14:29:18.685 SNR = 4.00 dB

14:29:18.685 Frequency error = -4849 Hz

14:29:20.256 TxDone

14:29:20.256

14:29:55.599 (Latitude, Longtitude) = (11.056193,106.670696)

14:29:55.632 RSSI = -102 dBi

14:29:55.665 SNR = 3.25 dB

14:29:55.665 Frequency error = -4782 Hz

14:29:57.324 TxDone

14:29:57.324

14:30:31.480 (Latitude, Longtitude) = (11.056214,106.670708)

14:30:31.514 RSSI = -104 dBi

14:30:31.547 SNR = 1.25 dB

14:30:31.547 Frequency error = -4772 Hz

14:30:33.164 TxDone

14:30:33.164

14:31:11.017 (Latitude, Longtitude) = (11.056376,106.671063)

14:31:11.051 RSSI = -105 dBi

14:31:11.084 SNR = 1.00 dB

14:31:11.084 Frequency error = -4772 Hz

14:31:12.588 TxDone

14:31:12.588

14:31:48.741 (Latitude, Longtitude) = (11.05638,106.671284)

14:31:48.775 RSSI = -109 dBi

14:31:48.808 SNR = -3.75 dB

14:31:48.808 Frequency error = -4772 Hz

14:31:50.066 TxDone

14:31:50.066

14:32:29.444 (Latitude, Longtitude) = (11.056388,106.671329)

14:32:29.478 RSSI = -104 dBi

14:32:29.511 SNR = 1.25 dB

14:32:29.511 Frequency error = -4753 Hz

14:32:30.923 TxDone

14:32:30.923

14:33:02.484 (Latitude, Longtitude) = (11.056491,106.671586)

14:33:02.516 RSSI = -107 dBi

14:33:02.549 SNR = -1.50 dB

14:33:02.549 Frequency error = -4738 Hz

14:33:03.998 TxDone

14:33:03.998

14:33:38.981 (Latitude, Longtitude) = (11.056461,106.671733)

14:33:39.013 RSSI = -108 dBi

14:33:39.046 SNR = -3.25 dB

14:33:39.046 Frequency error = -4711 Hz

14:33:40.656 TxDone

14:33:40.656

14:34:08.934 (Latitude, Longtitude) = (11.056548,106.671893)

14:34:08.968 RSSI = -108 dBi

14:34:09.001 SNR = -3.75 dB

14:34:09.001 Frequency error = -4656 Hz

14:34:10.454 TxDone

14:34:10.454

14:34:47.989 (Latitude, Longtitude) = (11.056641,106.672299)

14:34:48.021 RSSI = -110 dBi

14:34:48.054 SNR = -13.00 dB

14:34:48.054 Frequency error = -4688 Hz

14:34:49.570 TxDone

14:34:49.570

14:35:18.841 (Latitude, Longtitude) = (11.056602,106.672413)

14:35:18.873 RSSI = -109 dBi

14:35:18.907 SNR = -5.75 dB

14:35:18.907 Frequency error = -4721 Hz

14:35:20.290 TxDone

14:35:20.290

14:35:59.189 (Latitude, Longtitude) = (11.056656,106.672675)

14:35:59.222 RSSI = -111 dBi

14:35:59.255 SNR = -12.50 dB

14:35:59.255 Frequency error = -4698 Hz

14:36:00.636 TxDone

14:36:00.636

14:36:34.574 (Latitude, Longtitude) = (11.056731,106.672825)

14:36:34.607 RSSI = -111 dBi

14:36:34.640 SNR = -16.50 dB

14:36:34.640 Frequency error = -4721 Hz

14:36:36.064 TxDone

14:36:36.064

14:37:10.455 (Latitude, Longtitude) = (11.056715,106.673019)

14:37:10.489 RSSI = -110 dBi

14:37:10.522 SNR = -6.75 dB

14:37:10.522 Frequency error = -4763 Hz

14:37:11.835 TxDone

14:37:11.835

14:37:45.241 (Latitude, Longtitude) = (11.056764,106.673138)

14:37:45.274 RSSI = -110 dBi

14:37:45.308 SNR = -8.25 dB

14:37:45.308 Frequency error = -4728 Hz

14:37:47.035 TxDone

14:37:47.035

14:38:21.607 (Latitude, Longtitude) = (11.056897,106.673409)

14:38:21.641 RSSI = -111 dBi

14:38:21.674 SNR = -11.75 dB

14:38:21.674 Frequency error = -4776 Hz

14:38:22.903 TxDone

14:38:22.903

14:38:59.730 (Latitude, Longtitude) = (11.056891,106.673534)

14:38:59.763 RSSI = -111 dBi

14:38:59.797 SNR = -9.00 dB

14:38:59.797 Frequency error = -4742 Hz

14:39:01.370 TxDone

14:39:01.370

14:39:30.267 (Latitude, Longtitude) = (11.056922,106.673565)

14:39:30.301 RSSI = -111 dBi

14:39:30.334 SNR = -15.50 dB

14:39:30.334 Frequency error = -4780 Hz

14:39:31.783 TxDone

14:39:31.783

14:40:10.433 (Latitude, Longtitude) = (11.056991,106.674068)

14:40:10.466 RSSI = -112 dBi

14:40:10.500 SNR = -15.25 dB

14:40:10.500 Frequency error = -4734 Hz

14:40:11.706 TxDone

14:40:11.706

14:40:50.351 (Latitude, Longtitude) = (11.057044,106.674277)

14:40:50.384 RSSI = -111 dBi

14:40:50.418 SNR = -17.75 dB

14:40:50.418 Frequency error = -4755 Hz

14:40:51.988 TxDone

14:40:51.988

14:41:30.154 (Latitude, Longtitude) = (11.057174,106.674437)

14:41:30.187 RSSI = -109 dBi

14:41:30.220 SNR = -14.50 dB

14:41:30.220 Frequency error = -4660 Hz

14:41:31.692 TxDone

14:41:31.692

14:42:06.186 (Latitude, Longtitude) = (11.057097,106.674482)

14:42:06.218 RSSI = -110 dBi

14:42:06.251 SNR = -23.25 dB

14:42:06.251 Frequency error = -4763 Hz

14:42:07.532 TxDone

14:42:07.532

14:42:45.772 (Latitude, Longtitude) = (11.057232,106.674745)

14:42:45.805 RSSI = -111 dBi

14:42:45.839 SNR = -12.50 dB

14:42:45.839 Frequency error = -4612 Hz

14:42:47.263 TxDone

14:42:47.263

14:43:27.017 (Latitude, Longtitude) = (11.057179,106.675211)

14:43:27.050 RSSI = -111 dBi

14:43:27.083 SNR = -9.00 dB

14:43:27.083 Frequency error = -4579 Hz

14:43:28.835 TxDone

14:43:28.835

14:44:03.131 (Latitude, Longtitude) = (11.057219,106.675343)

14:44:03.164 RSSI = -112 dBi

14:44:03.197 SNR = -10.00 dB

14:44:03.197 Frequency error = -4545 Hz

14:44:04.675 TxDone

14:44:04.675

14:44:34.681 (Latitude, Longtitude) = (11.057246,106.67õ428)

14:44:34.713 RSSI = -111 dBi

14:44:34.747 SNR = -20.75 dB

14:44:34.747 Frequency error = -4545 Hz

14:44:36.009 TxDone

14:44:36.009

14:45:08.770 (Latitude, Longtitude) = (11.057264,106.675587)

14:45:08.803 RSSI = -112 dBi

14:45:08.836 SNR = -16.25 dB

14:45:08.836 Frequency error = -4591 Hz

14:45:11.235 TxDone

14:45:11.235

14:45:42.161 (Latitude, Longtitude) = (11.057148,106.675811)

14:45:42.194 RSSI = -110 dBi

14:45:42.227 SNR = -15.25 dB

14:45:42.227 Frequency error = -4577 Hz

14:45:46.153 TxDone

14:45:46.153

14:46:23.689 (Latitude, Longtitude) = (11.057244,106.675908)

14:46:23.722 RSSI = -112 dBi

14:46:23.756 SNR = -18.00 dB

14:46:23.756 Frequency error = -4579 Hz

14:46:25.425 TxDone

14:46:25.425

14:49:01.235 (Latitude, Longtitude) = (11.057171,106.675876)

14:49:01.268 RSSI = -112 dBi

14:49:01.302 SNR = -23.25 dB

14:49:01.302 Frequency error = -4532 Hz

14:49:02.555 TxDone

14:49:02.555

# REFERENCES

[1] H. N. Do, “Conceptual Design of an Autonomous Wireless Agrometeorology Station,” 2023.

[2] E. Fahad, “Microcontroller Selection Criteria for your project, 8 bit Vs 16 bit Microcontroller,” *Electronic Clinic*, 2021. https://www.electroniclinic.com/microcontroller-selection-criteria-for-your-project-8-bit-vs-16-bit-microcontroller/.

[3] J. Koon, “Key factors to consider when choosing a microcontroller,” *Micrcontroller Tips*, 2019. https://www.microcontrollertips.com/key-factors-consider-choosing-microcontroller/.

[4] EntcEngg Team, “Criteria for Choosing a Microcontroller.” 2017, [Online]. Available: https://www.entcengg.com/criteria-choosing-microcontroller/.

[5] Robocraze, “How to choose a microcontroller,” *Robocraze*, 2021. https://robocraze.com/blogs/post/selecting-a-microcontroller-for-your-projects.

[6] Texas Instruments, “Benefits and Issues on Migration of 5-V and 3.3-V Logic to Lower-Voltage Supplies,” 1999. [Online]. Available: https://www.ti.com/lit/an/sdaa011a/sdaa011a.pdf.

[7] STM32-base, “Blue Pill,” *STM32-base*. https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html.

[8] STMicroelectronics, “STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs,” no. February 2021. 2021.

[9] STMicroelectronics, “STM32F103x8, STM32F103xB.” pp. 1–116, 2022.

[10] M. Cardinali, “The importance of weather data in agriculture,” *Agricolus*, 2022. https://www.agricolus.com/en/the-importance-of-weather-data-in-agriculture/.

[11] B. Gardiner, P. Berry, and B. Moulia, “Review: Wind impacts on plant growth, mechanics and damage,” *Plant Sci.*, vol. 245, pp. 94–118, Apr. 2016, doi: 10.1016/j.plantsci.2016.01.006.

[12] E. de Langre, “Effects of Wind on Plants,” *Annu. Rev. Fluid Mech.*, vol. 40, no. 1, pp. 141–168, Jan. 2008, doi: 10.1146/annurev.fluid.40.111406.102135.

[13] Misol Electronics, “misol SP-WS02 Spare part (outdoor unit) for Professional Wireless Weather Station.” http://www.misolweather.com/index.php?m=home&c=View&a=index&aid=60.

[14] S. Pindado, J. Cubas, and F. Sorribes-Palmer, “The Cup Anemometer, a Fundamental Meteorological Instrument for the Wind Energy Industry. Research at the IDR/UPM Institute,” *Sensors*, vol. 14, no. 11, pp. 21418–21452, Nov. 2014, doi: 10.3390/s141121418.

[15] Argent Data System, “Weather Sensor Assembly p / n 80422.” 2014, [Online]. Available: https://www.argentdata.com/files/80422\_datasheet.pdf.

[16] T. Reed, “Multimeter Measurements Explained,” *Electronic Design*, 2020.

[17] Misol Electronics, “Professional Weather Station (Wind and Air Pressure) - Operational Manual.” pp. 1–24.

[18] Bosch Sensortec GmbH, “Humidity sensor BME280,” *Bosch Sensortec GmbH*. https://www.bosch-sensortec.com/products/environmental-sensors/humidity-sensors-bme280/#description.

[19] Bosch Sensortec GmbH, “BME280 Environmental Sensor,” *Bosch Sensortec GmbH*. Jan. 2022.

[20] Linh Kiện 888, “Cảm Biến Áp Suất BME280 3.3V.” https://linhkien888.vn/cam-bien-ap-suat-bme280-3-3v.

[21] C. Petrich, N. P. Dang, I. Sæther, Ø. Kleven, and M. O’Sadnick, “A Note on Remote Temperature Measurements with DS18B20 Digital,” 2020, [Online]. Available: https://hdl.handle.net/11250/2716073.

[22] Maxim Integrated, “DS18B20.” 2019, [Online]. Available: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[23] M. Fezari and A. Al Dahoud, *Exploring One-wire Temperature sensor “DS18B20” with Microcontrollers*. 2019.

[24] S. Budijono and Felita, “Smart Temperature Monitoring System Using ESP32 and DS18B20,” *IOP Conf. Ser. Earth Environ. Sci.*, vol. 794, no. 1, p. 012125, Jul. 2021, doi: 10.1088/1755-1315/794/1/012125.

[25] Widodo and E. A. Stiyawan, “Design of Total Dissolve Solid (Tds) Measuring Using Conductivity Sensor and Temperature Sensor Ds18B20,” *BEST J. Appl. Electr. Sci. Technol.*, vol. 2, no. 1, pp. 25–29, 2020, doi: 10.36456/best.vol2.no1.2583.

[26] Process Parameters Ltd, “HOW TO CALIBRATE PT100 TEMPERATURE SENSOR,” *Process Parameters Ltd*. https://www.processparameters.co.uk/how-to-calibrate-pt100/.

[27] Thegioiic, “DS18B20-1M Digital Temperature Sensor.” https://www.thegioiic.com/ds18b20-day-cam-bien-nhiet-do-1m.

[28] Thegioiic, “AM2315 Temperature And Humidity Module.” https://www.thegioiic.com/am2315-cam-bien-nhiet-do-do-am.

[29] B. Linke, “Reading and Writing 1-Wire® Devices Through Serial Interfaces,” 2009.

[30] Texas Instruments, “Implementing 1-Wire Enumeration for TMP1826 With TM4C129x Microcontrollers.” 2018, [Online]. Available: https://www.ti.com/lit/an/spma057d/spma057d.pdf.

[31] Speedtest® by Ookla®, “Speedtest Global Index,” *Speedtest® by Ookla®*. https://www.speedtest.net/global-index/vietnam.

[32] B. Mitchell, “What is the Range of a Typical Wi-Fi Network?,” *Lifewire*. 2020, [Online]. Available: https://www.lifewire.com/range-of-typical-wifi-network-816564%0Ahttp://compnetworking.about.com/cs/wirelessproducts/f/wifirange.htm.

[33] J. G. Sponås, “Things You Should Know About Bluetooth Range,” *Get Connected Blog*, 2023. https://blog.nordicsemi.com/getconnected/things-you-should-know-about-bluetooth-range.

[34] A. Lavric and V. Popa, “Performance Evaluation of LoRaWAN Communication Scalability in Large-Scale Wireless Sensor Networks,” *Wirel. Commun. Mob. Comput.*, vol. 2018, pp. 1–9, Jun. 2018, doi: 10.1155/2018/6730719.

[35] A. Ambanloc, “What Communication Protocol to Use and When,” *NeuronicWorks*. https://neuronicworks.com/blog/wireless-communication-protocols/.

[36] Voler Systems, “Wireless Communication Choices for IoT Designs,” *Voler Systems*. https://www.volersystems.com/blog/wireless-communication-choices-for-iot-designs.

[37] EPB, “How Far Will Your Wi-Fi Signal Reach?,” *EPB.com*. https://epb.com/get-connected/gig-internet/how-far-will-your-wi-fi-signal-reach/.

[38] Tessie, “HC-06 vs. HC-05 Bluetooth Module: What is the difference between HC-06 and HC-05?,” *Utmel Electronics*, 2021. https://www.utmel.com/components/hc-06-vs-hc-05-bluetooth-module-what-is-the-difference-between-hc-06-and-hc-05?id=889.

[39] A. Wang, “Comparison between LoRa and other wireless technologies,” *MOKOLoRa*, 2021. https://www.mokolora.com/lora-and-wireless-technologies/.

[40] Thegioiic, “Zigbee Modules.” https://www.thegioiic.com/product/mach-zigbee.

[41] Thegioiic, “RF Transceivers.” https://www.thegioiic.com/product/mach-thu-phat-rf.

[42] Semtech, “Semtech SX1276,” *Wirel. Sens. Prod.*, no. May, 2020, [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-core/sx1276.

[43] HeyPete, “Major differences between the DS3231 and DS3231M RTC chips.” 2017, [Online]. Available: https://blog.heypete.com/2017/09/05/major-differences-between-the-ds3231-and-ds3231m-rtc-chips/.

[44] HeyPete, “DS3231 Drift Results (5 months).” 2018, [Online]. Available: https://blog.heypete.com/2018/02/04/ds3231-drift-results-5-months/.

[45] Maxim Integrated, “DS3231.” 2015.

[46] Thegioiic, “Mạch Đọc Thẻ Nhớ MicroSD Mini Giao Tiếp SPI.” https://www.thegioiic.com/mach-doc-the-nho-microsd-mini-giao-tiep-spi.

[47] Omega Engineering, “Sources of Noise and Some Typical Solutions,” *Omega Engineering*. https://www.omega.co.uk/techref/das/noise.html.

[48] J. Heath, “Electrical noise can come from anywhere,” *Analog IC Tips*, 2021. https://www.analogictips.com/electrical-noise-can-come-from-anywhere/.

[49] V. Muthukrishnan, “Cutoff Frequency: What is it? Formula And How To Find it,” *Electrical4U*, 2022. https://www.electrical4u.com/cutoff-frequency/.

[50] Thegioiic, “Tấm Năng Lượng Mặt Trời 5W 18V.” https://www.thegioiic.com/tam-nang-luong-mat-troi-5w-18v.

[51] Điện tử Minh Nguyên, “[F123.7] Mạch sạc bảo vệ pin 18650 2 cell 20A.” https://chipn24.com/f123-7-mach-sac-bao-ve-pin-18650-2-cell-20a-id2589.html.

[52] STMicroelectronics, “Arduino\_Core\_STM32.” [Online]. Available: https://github.com/stm32duino/Arduino\_Core\_STM32.

[53] SparkFun, “Pull-up Resistors,” *SparkFun*. https://learn.sparkfun.com/tutorials/pull-up-resistors/all (accessed Jul. 26, 2023).

[54] J. Christoffersen, “Switch Bounce and How to Deal with It,” *All About Circuits*, 2015. https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/.

[55] Texas Instruments, “Debounce a Switch.” 2020.

[56] Raspberry Pi Foundation, “Build your own weather station.” https://projects.raspberrypi.org/en/projects/build-your-own-weather-station/0.

[57] Windy.app, “Where wind gusts come from,” *Windy.app Meteorological Textbook*. https://windy.app/textbook/where-wind-gusts-come-from.html.

[58] J. W. Nilsson and S. A. Riedel, “The Step Response of RL and RC Circuits,” in *Electric Circuits*, 10th ed., Pearson Education, Inc., 2015, pp. 224–231.

[59] Electronics Tutorials Team, “RC Charging Circuit.” https://www.electronics-tutorials.ws/rc/rc\_1.html.

[60] Z. Peterson, “ADC Sampling Rate and Layout for Mixed Signal Boards,” *Altium Limited*, 2020. https://resources.altium.com/p/adc-sampling-rate-and-layout-mixed-signal-boards.

[61] N. Abbas, “SPI vs I2C vs UART: In-Depth Comparison,” *Wevolver*, 2023. https://www.wevolver.com/article/spi-vs-i2c-vs-uart-in-depth-comparison.

[62] L. Fried, “Adafruit\_BME280\_Library.” [Online]. Available: https://github.com/adafruit/Adafruit\_BME280\_Library.

[63] I. Analog Devices, “How to Power the Extended Features of 1-Wire® Devices,” *Analog Devices, Inc.* https://www.analog.com/en/technical-articles/how-to-power-the-extended-features-of-1wire-devices.html.

[64] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, “Long-range communications in unlicensed bands: the rising stars in the IoT and smart city scenarios,” *IEEE Wirel. Commun.*, vol. 23, no. 5, pp. 60–67, Oct. 2016, doi: 10.1109/MWC.2016.7721743.

[65] The Things Network, “What are LoRa and LoRaWAN?,” *The Things Network*. https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/.

[66] V. Q. Tran, “Evaluate The Effectiveness Of Lora Network For Data Collection,” 2022.

[67] R. P. Centelles, F. Freitag, R. Meseguer, and L. Navarro, “Beyond the Star of Stars: An Introduction to Multihop and Mesh for LoRa and LoRaWAN,” *IEEE Pervasive Comput.*, vol. 20, no. 2, pp. 63–72, Apr. 2021, doi: 10.1109/MPRV.2021.3063443.

[68] D. Kjendal, “LoRa-Alliance Regional Parameters Overview,” *J. ICT Stand.*, vol. 9, no. 1, pp. 35–46, Apr. 2021, doi: 10.13052/jicts2245-800X.914.

[69] The Things Network, “Frequency Plans by Country,” *The Things Network*. https://www.thethingsnetwork.org/docs/lorawan/frequencies-by-country/.

[70] LORIOT, “LoRaWAN Network Server v7.x (Osprey) - Global Frequency Plans,” *LOIOT*. https://docs.loriot.io/display/LNS/Global+Frequency+Plans.

[71] Heltec Automation, “LoRaWAN Frequency Plans,” *Heltec Automation*. https://docs.heltec.org/general/lorawan\_frequency\_plans.html.

[72] S. Mistry, “Arduino LoRa.” GitHub, [Online]. Available: https://github.com/sandeepmistry/arduino-LoRa.

[73] I. Advanced Monolithic Systems, “AMS1117 1A LDO Voltage Regulator,” *Datasheet*. 2009.

[74] S. Santos, “ESP8266 Pinout Reference: Which GPIO pins should you use?,” *Random Nerd Tutorials*, 2019. https://randomnerdtutorials.com/esp8266-pinout-reference-gpios/.

[75] ESP8266 Community, “Arduino core for ESP8266 WiFi chip.” https://github.com/esp8266/Arduino#readme.

[76] AYVA Educational Solutions, “TecQuipment Axial Fan Module – MFP107,” *AYVA Educational Solutions*. https://www.tecquipment.com/axial-fan-module.

[77] S. Bhattacharjee, “Explained: What exactly is 1 mm of rainfall?,” *CNBC TV18*, Jul. 23, 2021.

[78] M. Zachmann, “The Best LoRa Settings for Range and Reliability,” *Medium.com*, 2018. https://medium.com/home-wireless/testing-lora-radios-with-the-limesdr-mini-part-2-37fa481217ff.