School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Development of an IoT system for Environmental Monitoring

Louis Li

Supervisor: James Quilty

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

The Greater Wellington Regional Council has been exploring ideas to update their environmental data loggers. In 2020, a proof-of-concept, low-cost, open-source Internet of Things solution was accepted by our client. This year, he is looking for a prototype with more capabilities and has a higher level of completeness to show to his team leader. This report looks at developing a prototype device capable of reading from sensors and uploading data to the internet via Low Power Wide Area Network technology.

Contents

Chapter 1

Introduction and Background

1.1 Introduction

The Greater Wellington Regional Council (GWRC) is a legislative mandate to monitor the Wellington region's air, land, and water resources. The data of interest is rainfall, river levels, water temperature, flow speed, and other environmental data. This data is required to report to individuals, businesses and governments adhering to the Resource Management Act. Without a doubt, environmental monitoring is essential to promote citizens health and safety. For example, the city council can raise a flood alert when the water level exceeds the warning level [1]. As a result, companies and citizens can get prepared to avoid damage as much as possible.

To acquire a broader range of data, GWRC needs to set up more sites meaning more data loggers are required. Our client, who is the leader of the Hydrology team, considered that the current data logger provided by HyQuest is no longer suitable for serving the local government needs. He thinks the current data logger is expensive and power-hungry. Moreover, he worried about getting lock-in the closed ecosystem of HyQuest and could only use hardware and software from a single vendor. In 2020 Victoria University of Wellington students undertook an Honours project using state-of-art technology to develop a proof-of-concept Internet of Things data logging. This data logger is open-source, small-scale, low-powered and low-cost. Given the final result, our client gets convinced that the idea is worth pushing. Therefore, he is demanding a prototype with more capabilities and has a higher level of completeness this year.

1.1.1 Objective

The proof-of-concept data logger has fulfilled many early requirements, such as collecting data from SDI-12 sensors, logging data into an SD card, and a localhost server for device configuration. However, several requirements are still unmet. The objective of this project is to keep developing and improving the 2020 proof-of-concept Internet of Things data logger and fulfil essential hardware requirements that are yet to be met:

- transmit data via Low Power Wide Area Network
- ability to log data from tipping bucket rain gauge
- produced and maintained with limited soldering/electronics experience
- battery powered with a lifespan of 3 months

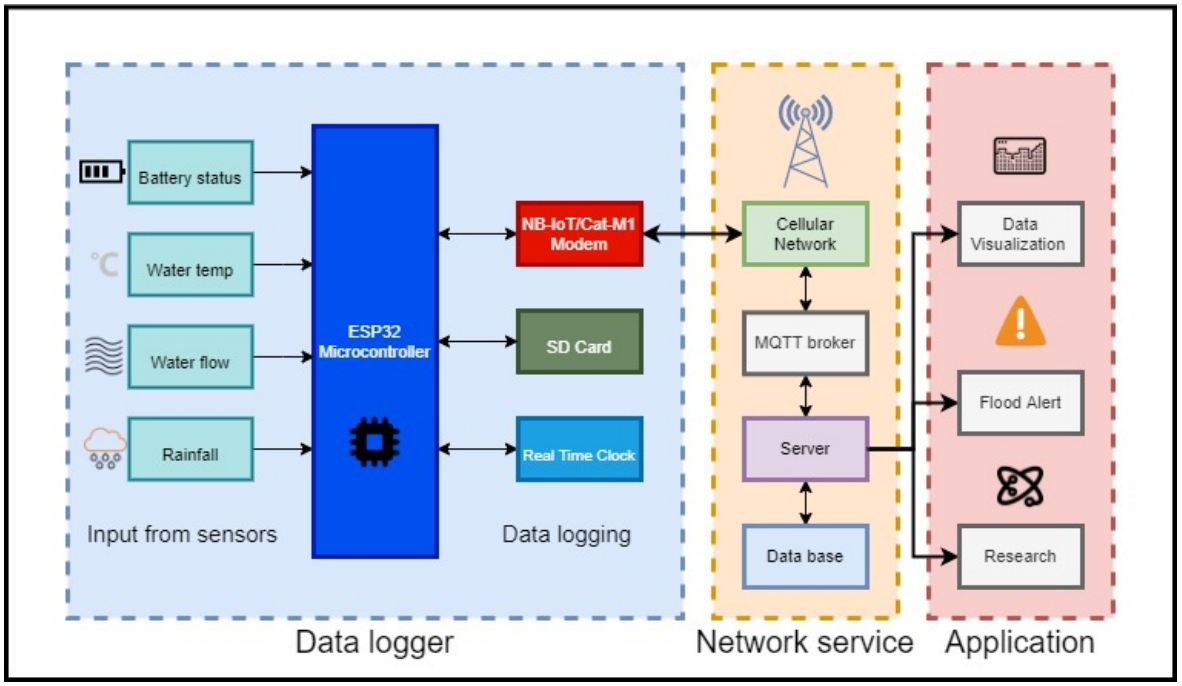

Figure 1.1: Functional block diagram of the proposed system.

1.1.2 Overview

In order to reduce the cost of maintenance and facilitate remote environmental monitoring in real-time, a system base on ESP32 microcontroller and NB-IoT/Cat-M1 was proposed. The Block diagram of the proposed system is presented in Figure 1.1. The data logger monitors water temperature, water flow, rainfall data as well as battery status. Any data being read will save a local copy to an SD card as backup. A Real-Time Clock is used for calibrating the time on the microcontroller to ensure accurate time-stamp. Usually, time calibration is subject to network time get by modem, and Real-Time Clock is useful for the offline device. The data package will be published to the MQTT broker by the modem via NB-IoT/Cat-M1 network when the data package is ready. A server on a computer will subscribe to the MQTT broker to get the telemetry data. After that, the data will be stored in the database and processed for applications such as data visualization, publish flood alerts and research.

1.2 Background

1.2.1 HyQuest Solutions Data Loggers

HyQuest is a company based in Australia and New Zealand with 40 years of experience providing solutions for environmental monitoring [2]. GWRC uses services from HyQuest and purchases products from them for many years, including iRIS data logger to store data and transmit the telemetry data over the air, SDI-12 sensors for river level monitoring, tipping buck rain gauge for rainfall, and more. The HyQuest loggers works fine so far for GWRC. However, our client, the GWRC Hydrology team leader, considered that the current data logger provided by HyQuest is no longer suitable for serving the local government needs.

The primary defect of the HyQuest data logger is not open source. Therefore, GWRC cannot customize the functionality of the data logger but depends on what HyQuest offers. This means GWRC is lock-in the HyQuest's ecosystem. What's more, there is no schematic

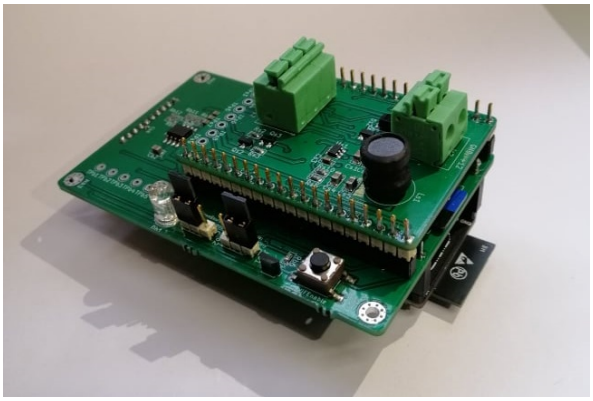

Figure 1.2: Data logger Rev 2.0 from early development.

as a reference to self-repair and maintenance if anything goes wrong. Several solutions have been proposed in the literature indicating Low Power Wide Area technology is more suitable for environment monitoring [3, 4]. However, HyQuest is not keeping up to date with the latest technology and refine their product but keep using their power-hungry 3G and 4G technology [5, 6].

1.2.2 Early development

Ben and Jolen [7, 8] proposed a low-cost open-source data logger for environmental monitoring based on NB-IoT. This idea was to replace the HyQuest's data logger with the proposed device with NB-IoT capability. This data logger is integrated with SDI-12 sensor sensing for water level, water temperature sensing, data processing and transmission of environmental data to the Azure cloud platform. For data collection and system tasks scheduling, the ESP32 microcontroller is used. This microcontroller has a lot of computational power and low price. The firmware being used on the microcontroller is microPython, a lightweight version of python design to run on microcontrollers enabling execute the code in real-time, which is very handy during development. The assembled data logger is shown in Figure 1.2. However, this data logger is incomplete as the proposed NB-IoT Network has some flaws. Also, the data logger is missing the capability to log data from the rain gauge.

My work is a continuation of the previous work. The goal is to complete the foundation of the project and evaluation of the system's performance in power consumption, system stability, data accuracy and user experience.

Chapter 2

Design

2.1 Design Requirements

The data logger must be capable of performing the following features:

1. Ability to use Low Power Wide Area Network (LPWAN) technologies to transmit data to the internet.
2. Ability to log data from the tipping bucket rain gauge.
3. Must be produced and maintained with limited soldering/electronics experience.
4. Must be small enough to put into a tipping bucket rain gauge.
5. Able to remain powered by a battery for at least three months.
6. Ability to monitor battery's status and send out a message when it gets low.

In addition, our client has specified the requirement on data quality:

1. The time stamping for the data must be accurate and integrated.
2. The data logger can either show time-stamp for each tip or show tips in 1-minute intervals.

2.2 Internet Connection Design

The top priority of this project specified in requirement 1 is sending telemetry data over the Low Power Wide Area Network technologies. This technology is appropriate for this project because it is marketed as low power consumption, wide covering range, and low-cost application. In order to connect the data logger to this network, we need to select an appropriate modem that meets the technical requirements. Also, a reliable network operator and a suitable protocol for data transitioning.

2.2.1 Modem Selection

In order to get a stable network connection, selecting a reliable modem is very important. The modems being considered are Ublox Sara R410M, Quectel BG96, and Telit ME910C1-AU. These modems are considered because they support the LPWAN bandwidth in New Zealand, which is 700MHz.

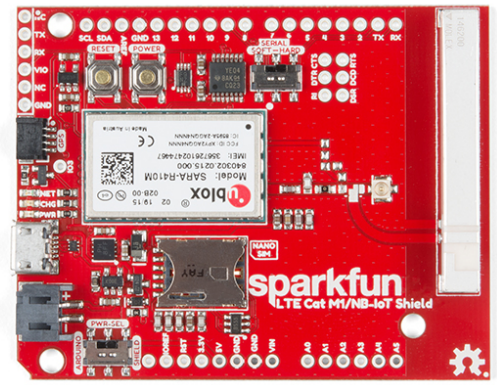

Figure 2.1: SparkFun LTE CAT M1/NB-IoT Shield - SARA-R4.[11]

Telit ME910C1-AU

The modem from Telit was selected for the data logger last year. Ben and Jolon choose this modem because it is cheaper than the other two modems, and it comes with a compact breakout board so they can easily put the modem on the PCB. However, they cannot get network communication working as expected due to hardware flaws and time constraints. Jolon points out that some function in the documentation is yet to be available unless firmware update in the future. Overall, continue using the Telit was considered being risky, and I decided to try another modem.

Ublox SARA R410M

As a prototype, the priority is ensuring the basic functionality and minimize any risk. The chosen modem is the Ublox SARA R410M. This modem matches the expectation in many perspectives. First of all, this modem is compatible with NB-IoT and CAT-M1 network and low powered, so it meets the technical requirement [9]. Secondly, this modem was deployed on the Spark fun's LTE CAT M1/NB-IoT Shield for Arduino as shown in Figure 2.1. Since Arduino is a big open-source community with a large user base, many users have used and evaluated this product. So, it is easier to get supports and troubleshoot if there is any problem. Thirdly, unlike Telit modem requires a 3.8V power supply, this modem is powered by 3.3V, which is the same voltage level as the microcontroller, therefore is no need for another voltage regulator. What's more, this modem has good documentation and well instruction on typical applications. Last but not least, it was suggested by an experienced engineer who working at local industry[10].

2.2.2 Network Operator Selection

Three network operators offer NB-IoT or Cat-M1 services in New Zealand. They are Vodafone, Spark and 2degrees. We can choose one of the operators and apply for a prepaid SIM card from them. However, not all areas are covered by each operator. For example, in some regions, Vodafone users are able to search for signals, but Spark users cannot. This issue will

	Data	SMS
Hologram	0.6/*device*/*month*+0.4/MB	$0.19/message
Vodafone	3/*device*/*month*+0.10/MB	$0.17/message

Table 2.1: Cost of Low Power Wide Area Network data and SMS.

significantly affect the data logger's usability as GWRC plans to put these devices around the Wellington region.

The option we considered is using a Hologram SIM card. Hologram IoT SIM card provides worldwide cellular connection, and they have a reliable multi-carrier network to achieves this. Therefore, we are not limited to one of the network operators but connected to whoever has the strongest signal. What's more, the Hologram SIM card is straightforward to set up and have a reasonable price point for data.

Table 2.1 shows the cost of data and SMS of Vodafone and Hologram. Vodafone charges less for data but high initial cost, where Hologram is the opposite. Therefore, Hologram' plan is more suitable for low data applications.

2.2.3 Telemetry Data Transmission Selection

NB-IoT/Cat-M1

In the Low Power Wide Area category, NB-IoT and Cat-M1 networks are being considered. These two protocols are very similar, but the critical difference is NB-IoT has a lower transmission speed than Cat-M1. In exchange, NB-IoT consumes less power. From a technical perspective, NB-IoT is more suitable for our application; however, Cat-M1 was chosen because investment and infrastructure have lean on Cat-M1 therefore, this technology is more mature.

TCP to Hologram Dashboard

Hologram has a dashboard for monitoring messages sent from all devices and is able to send messages to a specific device[12]. Also, the Hologram dashboard allows the user to set a route for each device to interpret the data. For example, users can reformat the data and save it to a SQL database. Hologram dashboard uses TCP/IP protocol. It uses Secure Socket Layer (SSL) or Transport Layer Security (TLS) to send data securely. Sockets are a ubiquitous interposes communication tool for sending and receiving data across various networking protocols. However, only Hologram SIM card users can access the Hologram dashboard. It is a risk because it will lock us into the Hologram ecosystem and lose the opportunity to switch to other operators.

2.2.4 SMS

An SMS message is also available but will not be considered because it is not cost-efficient. An SMS message has a very steep price, $0.19 per message, and we need to send over a hundred messages every day, and each message is only less than a hundred characters.

MQTT

Finally, the MQTT protocol was selected because it is a lightweight IoT messaging protocol. The term lightweight means less code is required to be executed, therefore lower power consumption. How it works is by having devices that publish ”topics” to a ”broker”. The

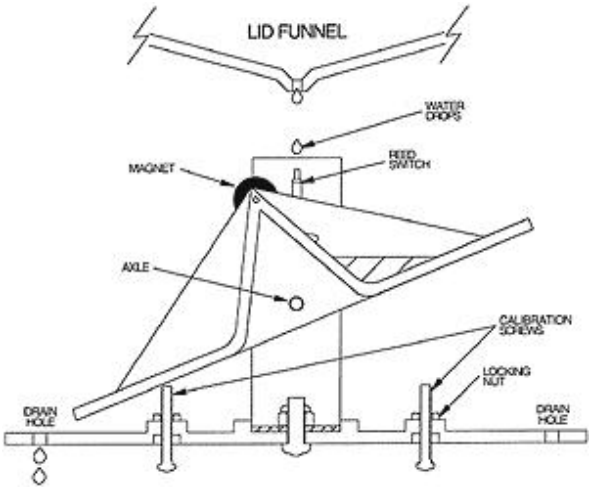

Figure 2.2: Tipping bucket rain gauge setup.

topics are a form of identification for the data being transmitted. Another end device can then subscribe to a specific topic through the broker, and it will receive any data published to the topic. The packets sent using MQTT are very small, so the modem would be transmitting for less time, which would result in less power consumption and less cost. The SARA R410M modem has a built-in MQTT client module, allowing the broker to set up a secure MQTT connection.

2.3 Rain Gauge Counter Design

Requirement 2 states that the data logger should be able to read data from the HyQuest tipping bucket rain gauge. One typical setup of the rain gauge is shown in Figure 2.2. When one side of the bucket is full, it will tip the bucket, and the magnet will pass the reed switch. Consequently, the reed switch will be open for a very short amount of time and then close. There are two challenges in logging data from the rain gauge: switch bounce and count all tips.

2.3.1 Switch Debounce

Like all switches, the reed switch will bounce. That is, the switch changes back and forth between open and closes before settling down to the final state. This phenomenon will seriously decrease the accuracy of the rainfall data. The considered options are on the software and hardware aspect.

Software debounce

Because switch bounces only sustain several thousandths of a second, the most intuitive method is to delay after detecting the first reading. Usually, 20ms is sufficient for most cases. However, this method is only suitable for programable microcontrollers. For any integrated circuits, there is nothing they can do to fix the issue.

Hardware debounce

The switch bounce can be seen as a high-frequency signal. In order to eliminate the high-frequency part of the signal, we can apply a low-pass filter. There are many types of low-pass filters to choose from, such as Butterworth, Chebyshev and Elliptic. However, for simplicity, I choose to use a resistor-capacitor network to resolve the problem. After filtering, the transition of the state has been smoothened. However, it is not good to feed this signal directly to the input of a downstream digital logic function, which wouldn't appreciate seeing a signal that dawdles in the undefined region between "good" logic 0 and logic 1 values. Instead, this signal can feed to the voltage comparator to get a clean pulse signal.

2.3.2 Counter

A counter is essential in order to meet the requirement of connecting to the HyQuest rain gauge. The current tipping bucket rain gauge has the ability to detect rain intensity of 700mm per hour. Since the bucket has a capacity of 1.0 mm, the data logger should at least be able to count pulses with 5-second intervals.

ESP32 Counter

The esp32 microcontroller has built-in a large variety of hardware sensors. This includes a hardware pulse counter. Since the hardware counter has a built-in hardware filter so, it can filter out switch bounce. However, the hardware counter will be disabled under deep sleep mode, which means the device needs to be in active mode all the time to count the tipping bucket rain gauge. Therefore, this method is very energy-intensive and will reduce the longevity of the battery seriously. Another potential solution is using the ESP32 external interrupt feature to wake the device when the bucket tips. When the device wakes up because of an interrupt, it will increment the count in the memory and log the current time. If the device is already awake, set up an interrupt routing for the signal as usual. This solution is very energy efficient as it only triggers when there is rain. Since no ICs required so this design is cheaper. However, due to the immaturity of Micro Python, it cannot distinguish which GPIO cause the interrupt and perform a different task when it wakes up. This is a problem because we have already use GPIO interrupt to wake up the device from a deep sleep and enter configure mode. Therefore, this method cannot be implemented at the moment.

External Counter

The microcontroller should remain in a deep sleep during regular operation to meet the requirement of low-power. In order to keep the count while the microcontroller is in a deep sleep, we consider using an external counter circuit. The external counter can operate independently from the microcontroller, and they are relatively low power as they are not doing any work but keeping the voltage level in logic gates.

The limitation of the external counter is unable to time-stamp for each tip, but instead, the number of tips in a given interval. However, if this interval is too large, there will be only big spikes read from the data logger and not very helpful for researching or proacting. This is why requirement 8 ask for a 1-minute sampling interval, but the trade-off is it will shorten the battery longevity as the microcontroller wakes up more often.

The external counter design was selected as it has a good balance between longevity on battery power and data continuity.

Figure 2.3: HyQuest data logeer inside a tipping bucket rain gauge [13].

2.4 PCB DESIGN

Many requirements need to be considered for the new PCB design. First of all, the new PCB design is essential to meet the requirement of supporting connection to the modem (requirement 1) and rain gauge (requirement 2). Also, it has to fulfil earlier requirements, such as connecting the SDI-12 sensor. What's more, considerations needed to be made around manufacturing and user experience to meet requirements 3, 4.

2.4.1 PCB Size

GWRC wants to insulate the PCB with a waterproof housing and then fit it inside the HyQuest's tipping bucket rain gauge (requirement 4), as illustrated in Figure 2.3. Although housing that interfaces with the rain gauge are not covered in this project, it is still worthwhile to keep the maximum space available in mine and consider it when designing the PCB. My estimated size to constrain the devices is $100 \times 60 \times 60$ mm3.

2.4.2 Baseboard Design

Unified

The last revision data logger uses a modular design with a baseboard and an expansion board, each focusing on different functionality. The benefit of the modular design is easy to replace the expansion board with a new design if there are any mistakes. However, the downside of this design is it uses two ten pins headers to attach the expansion board to the baseboard, which makes the device bulky and hard to assemble.

After two revisions, the functionality on the expansion board has been finalized. Therefore, I decided to unify two boards into one PCB. Figure 2.4 shows the assembled PCB rev 3.0. Since pins and headers are now removed, components can get closer to each other, which means shorter trace. The benefit of using short paths is that it will help reduce EMI risk from antennas and trace proximity. Another benefit is all the user interfaces such as buttons, screw terminals, LED indicators, SD card sockets can be accessed on one side. This design improves the user experience because the waterproof case only allows users to open the lead for configuration.

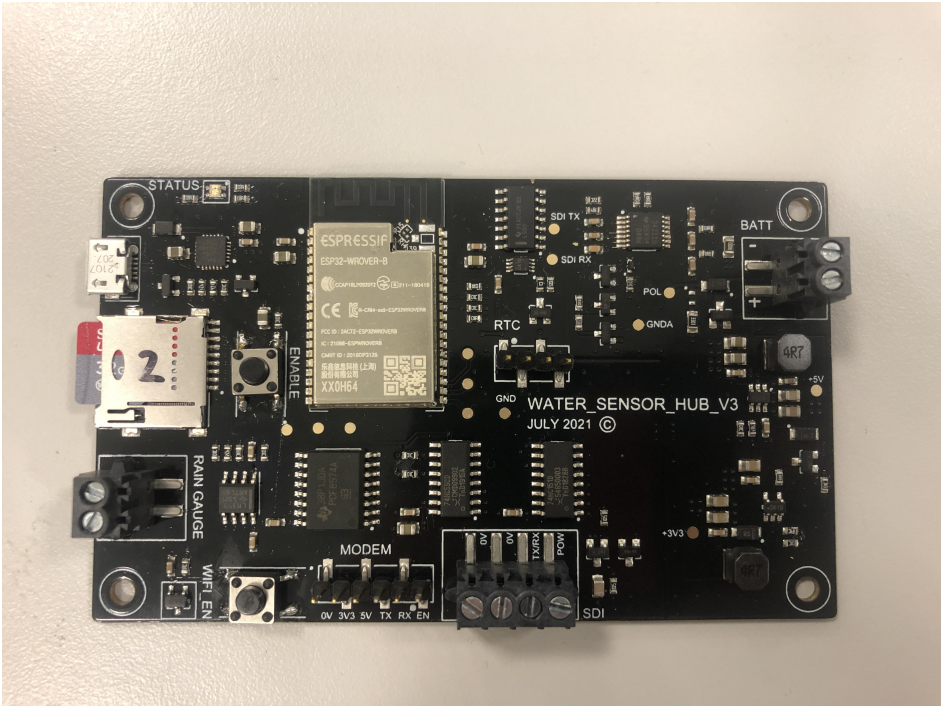

Figure 2.4: Rev 3.0 Data logger.

Component selection

Using surface mount components as much as possible is essential to meet requirement 3. This design decision was made last year, and it's motivated by minimizing the manual steps required to assemble the boards since some PCB manufacturers offer a surface-mounting service as part of manufacture. The only downside is they are more challenging to get replaced. However, with considerable time and investment in equipment, replacing a broken board with an assembled spare one is more cost-effective than repairing it.

2.4.3 Modem connection selection

There are two options being considered to connect the PCB and the modem. Either use a terminal connector and wires to connect the Spark fun's LTE shield as a breakout board or surface mounts the SARA R410 modem and auxiliary circuits on the PCB. Regarding the cost, the unit price for the LTE shield will be slightly higher than the surface mount; however, if PCB is not working at some stage, it is almost impossible to reuse the surface mount modem. The modem circuit layout is very complicated on the PCB design aspect, and if there are any mistakes in the layout, the modem may not work. Since the modem's functionality is the top priority in this project, and surface mount is considered risky, I decided to use the off-the-shelf LTE shield from spark fun directly to meet design requirement 1.

2.4.4 POWER SYSTEM DESIGN

Power consumption

GWRC wants the data loggers to be able to monitor environmental data across the Wellington region. Therefore the data logger must be powered by a battery. In addition, they expect the data logger to be able to remain powered for three months (requirement 5) so that they

won't need to maintain the data logger frequently, thus reducing labouring costs. What's more, the low-power provides feasibility to utilizing the existing low-cost, self-sufficient solar system for the data logger. T he current battery being used is deep-cycle lead-acid batteries with a capacity of either 18 or 32 Ah, which is unlikely to change in the immediate future. These types of batteries are designed to be discharged almost completely. So, it is reasonable to assume that we can use all of the capacity. Given the required battery lifetime and battery capacity, the baseline average power consumption required to design was calculated by the expression I = Q/t = 14.6mA.

Voltage regulation

The data logger needs to support the SDI-12 sensor, and the SDI-12 sensor needs to be powered by 9V to 16V power source. Therefore a 12V battery was selected for SDI-12 sensor voltage compatibility. Since the communication of the SDI-12 sensor uses a 5V logic level, a 12V to 5V buck converter was implemented in the previous revision. As mentioned in section 3.4.2, the new PCB design decided to use surface mount components, including the ESP32 microcontroller. However, ESP32 is powered by 3.3V, which means a 5V to 3.3V regulator is required.

The high efficiency of the voltage regulator is essential to meet the requirement of low power consumption. Two types of voltage regulators – switching and linear regulators are being considered.

Switching regulator

Buck converter is one of the switching regulators. It converts high voltage to low voltage by changing the duty cycle of the Pulse Width Modulation signal. Buck convert is very efficient. Theoretically, buck converter has efficiency as high as 99%. But in practice, it is around 85% due to imperfection. However, the downside of the buck converter is the size of the circuit is generally more prominent than the linear regulator due to the inductor being used. What's more, the buck converter has a ripple current therefore relatively noisy.

Linear regulator

The Low Dropout Regulator is one of the linear regulators. They are not as efficient as switching regulators because they convert the voltage drop over the regulator into heat. However, when the voltage difference is small, for example, 5V to 3.3V and under light load, the efficiency of the buck converter and LDR is about the same. In exchange, LDR is smaller, cheaper, quieter, and simpler to implement.

Overall, the more efficient buck converter was selected because meeting the requirement of low power consumption has a higher priority than considering size, cost and PCB complexity.

2.4.5 Dual power source switching

The data logger has a commonly used USB port for programming and debugging to meet design criteria 5. The USB port comes with a 5V power supply that can power the microcontroller and any other electronics. However, in regular operation, the battery will also be supplying the 5V from the regulator. There will be two 5V power sources powering the circuit simultaneously when the user tries to programming or debugging the device when the device is connected to the battery. This is a problem because the current can flow from higher to lower voltage and may cause damage.

Single source

One option being considered is not connecting the 5V from USB to the device so, the device has only one power source. However, the disadvantage is the device must connect to a 12V power source to work even the user only wants to program the device or to develop the low voltage component. Therefore, this design is slightly inconvenient for the developer.

Isolation

The chosen design isolates the USB power supply and only use the battery when the battery is connected. When the battery is not connected, the PCB can be powered by a USB port. The isolation process must be performed automatically rather than using switches or jumpers. The advantage is this design provides flexibility on the power source and improve ease of use.

2.4.6 Battery monitoring

Our client most values data's quality which reflected in requirement 7, 8. Also, requirement 6 directly show that the system must be able to monitor batteries state. This is because, in order to ensure data integrity and measurement requirements, the device must operate continuously. If the battery is over-discharged, it may cause the data logger to fail to log data and even shut down, which cause data losses. Therefore, the data logger must notify the GWRC staff if the battery needs to recharge or be replaced.

Voltage method

Previous revision uses the voltage method to monitor the battery's State of Charge. They use a voltage divider to scale down the 12V from the battery within 3.3V so the microcontroller can use ADC to read the battery's voltage. This method has limitations on the ease of use as the user needs to optimize the Analog to Digital Converter for each device. Also, measure the battery's voltage under load cannot epitomize the State of Charge accurately.

Estimation

Since the device is doing the same task repletely, it is easy to estimate the remaining charge on the battery and when to replace/recharge them. However, the battery's actual capacity will be lower than the design capacity depending on the cycling time. For example, after 200 deep cycles, the capacity of the lead-acid battery will have 80

Battery Fuel Gauge

One commonly used solution for reachable battery monitoring in the industry is using a battery fuel gauge[14]. As shown in Figure 2.5.the BQ34Z100EVM battery fuel gauge module from TEXAS Instruments has accurate, real-time battery voltage and State of Charge readings. The technology being used is the proprietary Impedance Tracking algorithm[15]. This algorithm measures voltages and calibrates with current, temperature, and battery characteristics to assess the battery's State of Charge and State of Health . What's more, after running one calibration cycle to the battery, this measurement result can have an error within 1%. The calibration cycle only needs to do once, and it will generate a standard calibration file that can be applied to other devices.

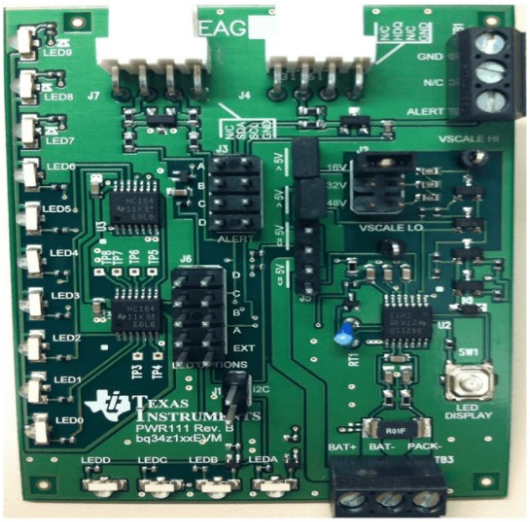

Figure 2.5: Bq34z100EVM 1s to 16s Impedance Track Fuel Gauge Battery Evaluation Module[16].

Chapter 3

Implementation

3.1 System Overview

The detailed functional block diagram of the system is shown in Figure 3.1.

3.2 Modem To Internet Connection

The initial design for the data logger is to send data to the Azure IoT hub via MQTT. Unfortunately, this didn't work out. The issue is Ublox have set in their firmware to limit the username and password to 30 characters. However, the client ID and the Shared Access Signatures used by the IoT hub exceeds this limit which means the device can not get certified. This can only be changed if a new firmware update fixes this issue. There is an alternative that can send MQTT messages over TCP sockets. The basic idea is to send a package with certification information and payloads over TCP protocol to a web server. Then this web server will publish the MQTT message to the Azure IoT hub as a bridge. Nevertheless, this method will add more layers of complexity to the project, and TCP is relatively power-hungry. Therefore, we decided to abandon the IoT hub but send MQTT message over to the Mosquitto MQTT broker.

Currently, we are sending a message to the test server of Mosquitto (test.mosquitto.org) under a topic of our choice, which is "test/environmentMonitoring/data". Another PC will be connected to the MQTT broker and subscribe to this topic. The PC will receive the telemetry data when the data logger publishes a message to the topic. However, this is not good for practical use as lack of security. The implementation is only focused on the MQTT functionality.

3.2.1 External Counter

Hardware Debounce

As mentioned in section 3.3.1, the signal generated by close or open the reed switch is very noisy, and the choose solution is to use hardware to filter out the noise and then generate a clean pulse signal for the counter. Figure 3.2 shows the pulse generator circuit. R1 and C1 form a simple resistor-capacitor filter that smoothens any high-frequency signal. This signal will pass to inverting terminal of the LM393 voltage comparator. A voltage divider set R2 and R3 will divide the 3.3 volts into 1 volt as a reference voltage. This reference voltage is connected to the non-inverting terminal and compare with the inverting terminal. If the signal is higher than the reference voltage, the output on the voltage comparator will

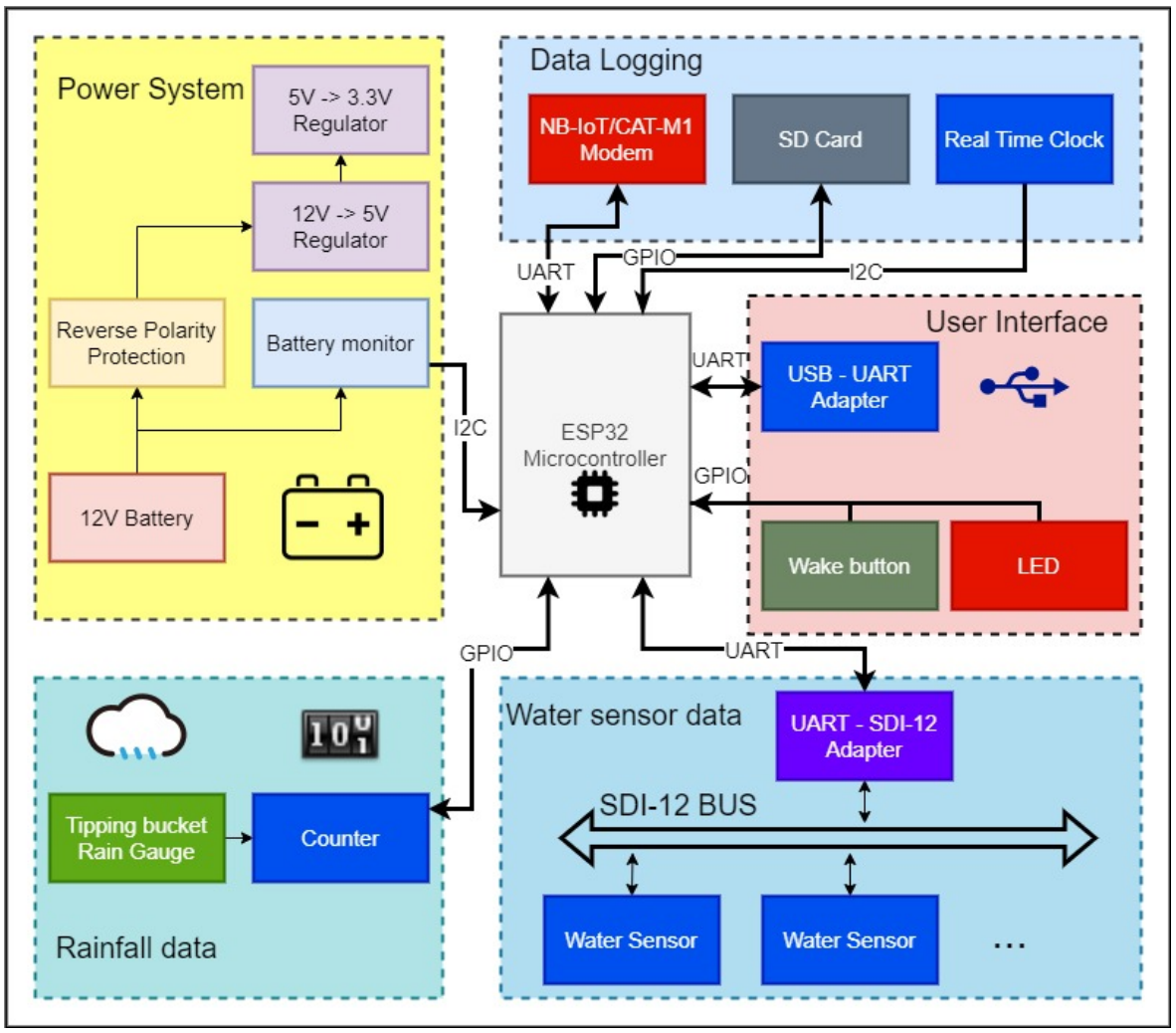

Figure 3.1: Detailed Functional Block Diagram of the system.

be LOW, and if the signal is lower than the reference voltage, the output on the voltage comparator will be HIGH.

Count Rainfalls

Figure 3.3 shows the functional block diagram of the counter circuit. A 74HC590 8 bits binary counter was selected. This counter is low-cost, low energy consumption and most importantly, very robust. When the pulse signal generated from upstream feeds into the clock pin of the Counter IC, the 8 parallel output will start to toggle. The less significant bit will toggle for two clock pulse, and the second bit will toggle for four clock pulse and so on.

Read Count

The counter has an 8-bits parallel output which means 8 GPIOs are needed to read the count. However, after assigning GPIOs to peripherals like the SDI-12 interface, the SD card, the LEDs and the modem, there are not enough GPIOs for reading the 8-parallel output from the counter directly. Therefore, a 74HC151 3 to 8 bits multiplexer is used to read the parallel output into serial output, and a PCF8574 IO expander is used to control the output of the Multiplexer and the Counter.

Memory

The 8-bits binary counter can only count up to 255 and then overflow. This amount of memory is very small compared to the microcontroller. But it is sufficient because the microcontroller can reset the counter after reading the count. If the count is being read in a 1-minute interval specified in the requirement and the pulse signal has a maximum 5 seconds interval, then the maximum count required would be 12. Even the interval increases to 10 minutes, the count needed is only 120, which is still within the counter's maximum count.

3.3 Power System

3.3.1 Power switching

Figure implemented the design in section 3.5.1. This design uses a P-Channel MOSFET. When both battery and USB supply appear, the MOSFET will turn off, thus blocking the USB current and ensuring only the battery is powering the data logger. The MOSFET will be on when the battery is not connected, allowing current flow from the USB.

3.3.2 Regulator

A 12V battery is selected for powering the SDI-12, and a 12V to 5V buck converter has been implemented in the previous PCB. There is no prominent flaw in the buck converter circuit except they got the inductor value slightly wrong. According to the datasheet, for an output voltage of 5V, the inductance of the inductor should be 4.7 µH rather than 3.3µH.

The ESP32 microcontroller operates at 3.3V. Since we decided to surface mount the microcontroller directly on the PCB, we are no longer being forced to use the low-efficiency linear regulator on the development board. Instead, a more efficient buck converter mentioned in section 3.5.2 was used. The requirement of the buck converter is taking input of 5V and step down to 3.3V. Since this converter is used mainly to power the microcontroller and the peak current should be over 240mA.

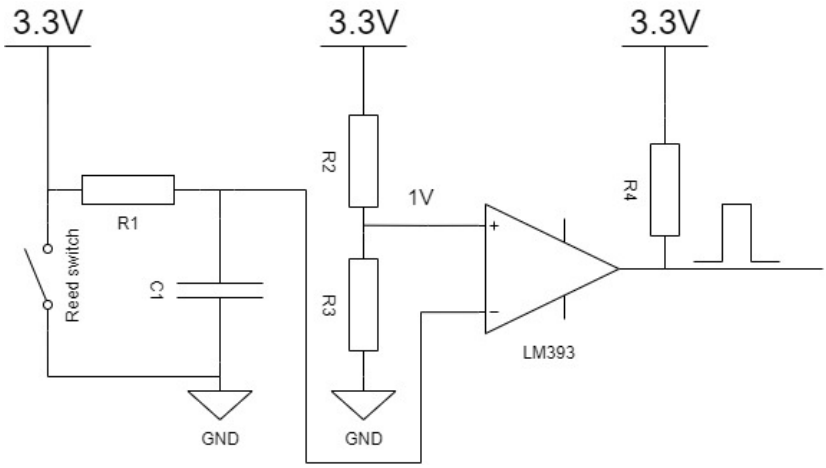

Figure 3.2: Pulse generator.

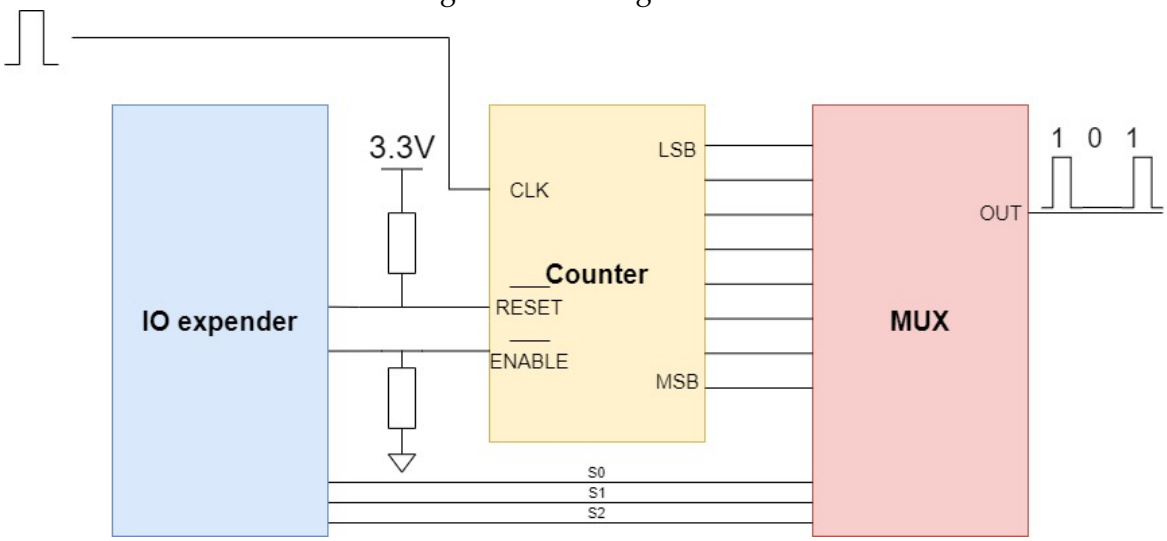

Figure 3.3: Rain guage counter circuit.

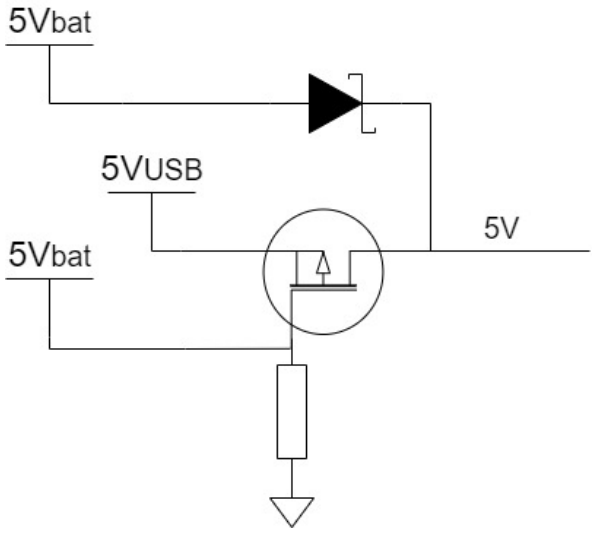

Figure 3.4: Power isolation circuit.

	Deep-sleep	Active	Transmission
ESP32	10µA	20mA	160mA
SARA R410M	8µA	9mA	60mA

Table 3.1: Power consumption of microcontroller and the modem.

An LM2831 bucket converter was selected to meet the requirement. This buck converter support an input range of 3V to 5.5V and an output range0.6V to 4.5V. What's more, it can supply up to 1.5A and has typical 92

3.3.3 Battery Monitoring

Instead of using an expensive and bulky evaluation board, I decided to rebuild the monitoring circuit on the PCB directly. However, this means I can't use Texas Instruments' software but communicate with the IC with the I2C interface. Texas Instrument provides good documentation on the BQ34Z110 battery fuel gauge IC and an example circuit in the datasheet. Rather than blind copy the example circuit, it requires a serval change for my application. First of all, to ensure measuring the voltage correctly when the battery is at peak voltage, the input voltage to the IC should be 0.9V. Therefore, we need a voltage divider circuit. The resistance can be calculated by:

$$R = \frac{16.5K(Vmax - 0.9V)}{0.9V} = 295.2k = 300k \tag{3.1}$$

So, our two resistor value is 16.5k and 300K. After that, remove all unnecessary circuits, such as the thermostat and LED indicator.

3.4 Embedded System Design

The embedded system must be programmed to ensure accurate time-stamp and the effectiveness of collecting and transmitting the data. More importantly, the embedded system is essential to switch the power of the component in order to reduce energy waste in non-active mode and achieve the requirement on the device's life span (requirement 5).

3.4.1 Scheduler

The current embedded system stays mostly in deep sleep mode, waking at user-defined intervals for measurement and transmission, as shown in Figure 3.5. This super loop architecture is essential to meet the requirement on low-power and data continuity. The power consumption of the data logger varies because the microcontroller and the modem have variable power consumption depends on tasks being performed in a different mode. Generally, it can be divided into three modes: deep-sleep, active and transmission mode.

Table 3.1 shows the power consumption of the microcontroller and the modem in each state. One can perceive that both microcontroller and modem consume much less power in deep-sleep compared to other states. Therefore, expand the duration of deep sleep is the most effective way to reduce the average power consumption. However, the system needs to ensure the data is real-time and, therefore, must balance power consumption and real-time operation.

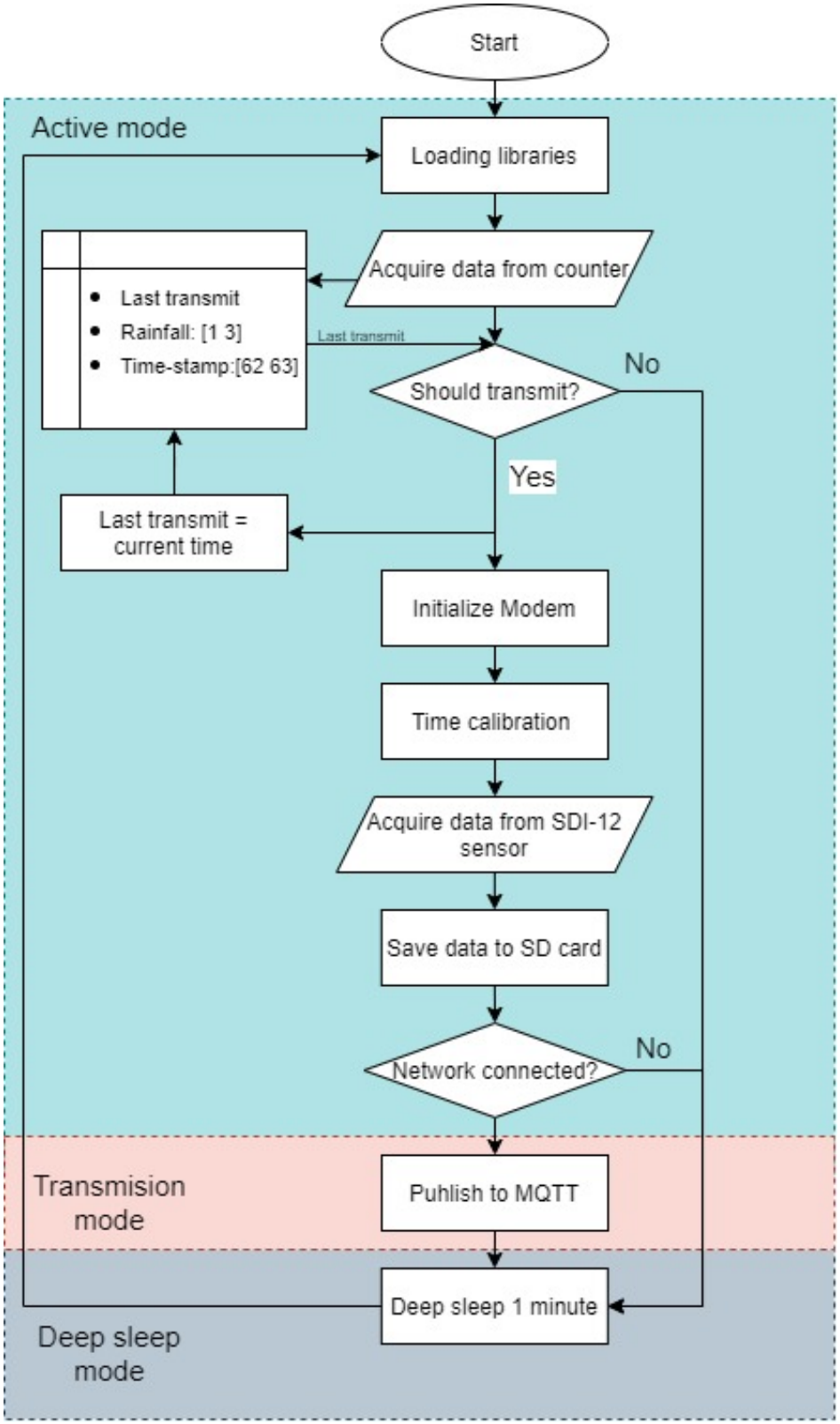

Figure 3.5: The wokrflow of the system.

Should transmit

As mentioned in 3.32, the rainfall is required to be sampled in a 1-minute interval so the data logger is scheduled in 1-minute deep sleep at the end of each cycle. However, the data logger also needs to transmit the data in user-specified intervals, such as 10 minutes. Therefore, I implemented a "should transmit" method. This call will extract the last transmit time stored in the system memory and subtracted it by the current time to calculate the time difference. After that, compare this time difference with the send interval. If the time difference is more than the send interval, update the last send with the current time and return True to transmit data at this wake-up. If the time difference is less than the send interval, save the data in the SD card as a buffer.

The rainfall data was stored into two lists—one for the count of tips and the other for the time-stamp. Each count has a corresponding time-stamp. The time-stamp is an integer that represents seconds from the year 2000. However, it can be converted back to Coordinated Universal Time format.

Power switching

It is essential to optimize for low-power in the deep-sleep mode because the data logger will be operating in this mode most of the time. The SDI-12 sensor has a power relay implemented in early development to be powered down when unused. Some peripheral components such as counter and battery monitoring IC cannot be considered as they are designed to remain powered. The only design consideration is on the modem. The modem has a port used as a switch to turn on/off the modem by the microcontroller. As mentioned before, putting the modem in deep-sleep mode can save significant power; however, like most smartphones, the modem needs some time to turn on and connect to the network operator. Sometimes, the modem needs to be reset and waiting for serval times to establish the network connection. In the worst case, the network connection fails after many attempts. The long search and connection time will lead to increased power consumption and affect the real-time performance of data which is considered a risk to the system's stability.

Nevertheless, the chosen solution is to turn off the modem in a deep sleep by GPIO. First of all, this design reduces a significant amount of power. Secondly, it allows the microcontroller to perform a hardware reset to the modem, which is more effective when the modem is not responding to the command. The trade-off is that we have to consider various error states and develop recovering mechanism in the modem driver.

3.4.2 Modem Driver

The SARA R410M modem communicates with the microcontroller by UART interface. How it works is the microcontroller needs to send AT command to the modem. After receiving the command, the modem will perform a specific function, and return the response in a buffer. The modem has built-in extensive predefined functions documented in AT command manual [17]. For example, AT+CCLK? will return network time. The main focus of the modem driver is to convert AT command set into functions so that the main program will be more concise and make it easier for developers[18].

Power

The modem has a port used to turn on/off the modem by connecting to the microcontroller's GPIO. When this GPIO is set to low output with a pull-down resistor, the modem will turn on after 3 seconds. To turn off the modem, configure the GPIO to input and set to low.

Initialiser

The initializer must be able to do two things –configures a UART port for communication and network registration. Also, the initializer is required to implement a failsafe and recovery mechanism to ensure the system's reliability.

Configure a UART port is done by using the UART library from micropython. If the modem does not respond, the initialize will perform a power off and on cycle. Network registration is done automatically after modem boot up if the profile has been configured. How to configure the profile will be discussed in the next section. The network registration might fail for some reason, such as heavy traffic on the cellular tower or the modem going into an error state during the registration process. In this case, the initializer will perform a soft reboot on the modem. If there is no response or no network after three recovering cycles, the data logger will continue logging data without transmitting data at this wake-up.

Provisioning

The modem needs to be provisioned before being used. The provisioning process consists of baud rate configuration and network registration.

Baud rate The SARA R410M modem supports six baud rates , and we are using the default value which is 115200. However, since Sparkfun's LTE shield is built for the Arduino board, it was set to 9600. Therefore, we need to change it to our desired baud rate. It requires three steps. First, configure the UART to match the baud rate set on the modem. After that, send a command to set the baud rate to 115200. Finally, configure the UART back to 115200.

Manually doing this baud rate configuration in the command line is unpleasant, and we need to consider a situation where the modem is set to other baud rates. Therefore, an auto baud rate method was implemented. This method will perform the same steps, but the only difference is it will loop through all possible baud rates in step 1 to find a match. Therefore, it is guaranteed to find the baud rate of the modem and set it to our desired one.

Network Registration There are there network profile parameters that need to be configured in order to register to Mobile Network Operator automatically.

- Mobile Network Operator profile number: 19 for Vodaphone
- Access Point Name: "hologram" for Hologram SIM card and "iot" for Vodaphone SIM card
- Operator selection: Each operator is represented by a unique code which is a combination of mobile country codes and mobile network codes. For example, "530 01" represents Vodaphone.

Send MQTT Message

As mentioned in section 3.2.2, the SARA R410M modem has a built-in MQTT client. The modem driver has implemented connect or disconnect from MQTT broker, publish or subscribe message by AT commands. The modem driver also implemented a method to configure the user name, password and ip address of the MQTT broker.

3.4.3 Counter driver

As mentioned in section 4.2.1, the rain gauge counter circuit consists of an IO expander, counter and Multiplexer. The counter driver is required to configure the logic level of GPIOs to control and read count from the counter circuit. The counter is initialized by setting the enable pin to low. Reading the count will need three steps. First of all, disable the counter because the count will be destroy and unable to recover if the rain gauge tips during the reading process. After that, create a zero variable to store the count being read and change the three select pin S0 to S2 of the Multiplexer in sequence to read all eight parallel output from the counter. For example, If the first bit is high, it means there is one then add 1 to count. Finally, reset the counter by setting the reset pin on the counter to HIGH and input a short pulse to the clock pin.

Chapter 4

Evaluation

4.1 Power Consumption

Power consumption is one of the most critical factors to evaluate if the design is fulfilling the requirement. For the power measurement, we need to know the current flow through the data logger. The experimental setup is shown in Figure 4.1. It shows the setup of a resistor is connected in series between the battery and the data logger, and an Oscilloscope measures the voltage across the resistor. According to Ohms law, the current can be calculated by I = U/R.

The power consumption of the data logger varies on different operating modes. Broadly, this can be divided into two modes: transmitting mode and deep-sleep mode. The power consumption for transmitting mode as shown in Figure 4.2 It starts with the microcontroller initialization marked as state 1, then starts booting the modem and registration process with the network operator as shown in state 2. In this state, we can observe that the modem took 3 seconds to boot up, and there are energy spikes when the modem communicating with the cellular tower. In state 3, power up the SDI-12 sensor and read data. This state only lasts for 4 seconds, and as soon as the data being collected, the SDI-12 sensor will be power off. The collected data were processed in state 4 and saved to the SD card. State 5 is the transmission state to send the MQTT message. We can see that the modem first sends a request to connect the MQTT broker and then goes into energy-saving mode while waiting for a response. Once received a response, the modem will wake up and send the message out. State 6 is the deep-sleep mode, where the modem gets switched off and the microcontroller going into deep-sleep mode. After 1 minute, the data logger will wake up and read the count from the counter. As shown in Figure, this process took 10 seconds to complete. Since no transmission is scheduled at this wake-up, the data logger is going into deep sleep again.

Based on the data produced from the power consumption analysis, the current and dura-

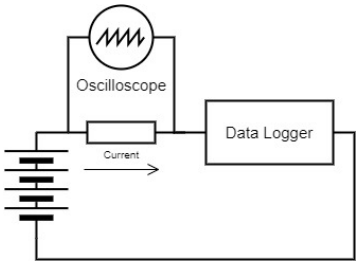

Figure 4.1: Experiment setup for energy measurement using an oscilloscope.

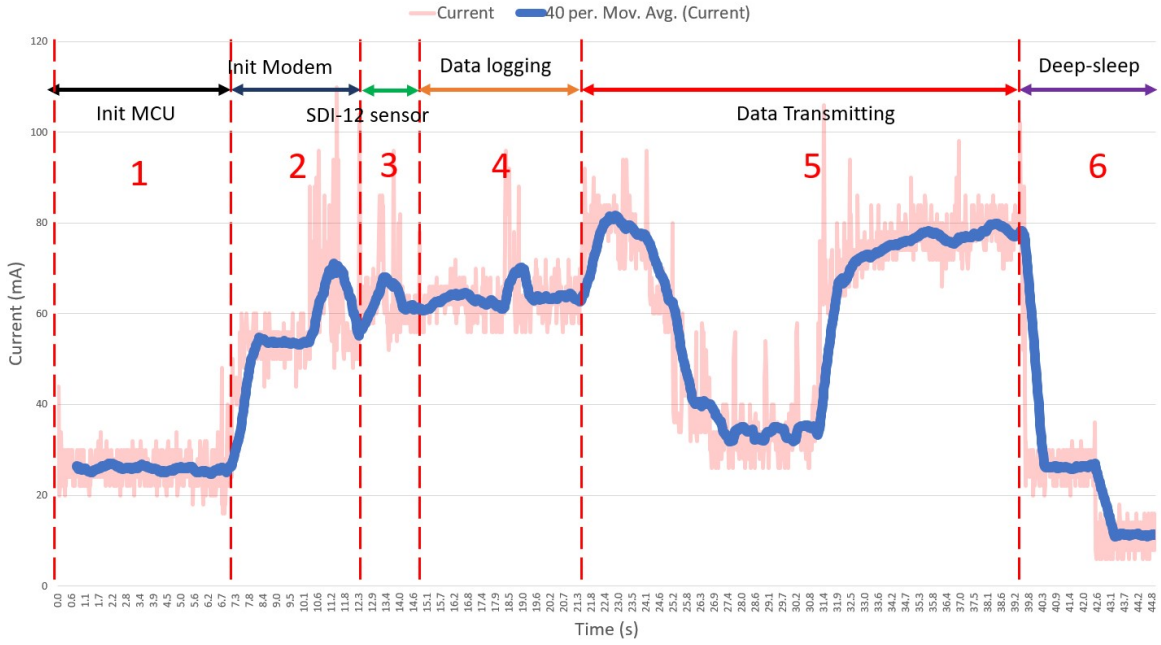

Figure 4.2: Scope view of collecting and transmitting data. (The red line is the current measured in 50 samples/second, and the blue line is the running average of 40 samples.)

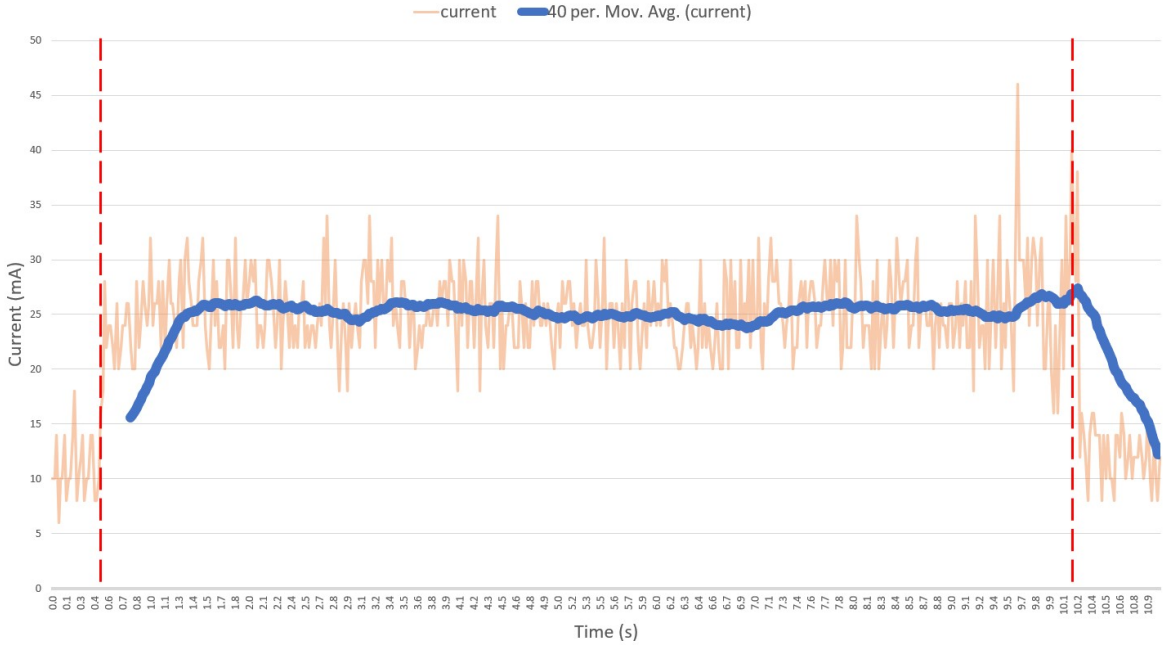

Figure 4.3: Scope view of collecting rainfall data. (The orange line is the current measured in 50 samples/second, and the blue line is the running average of 40 samples.))

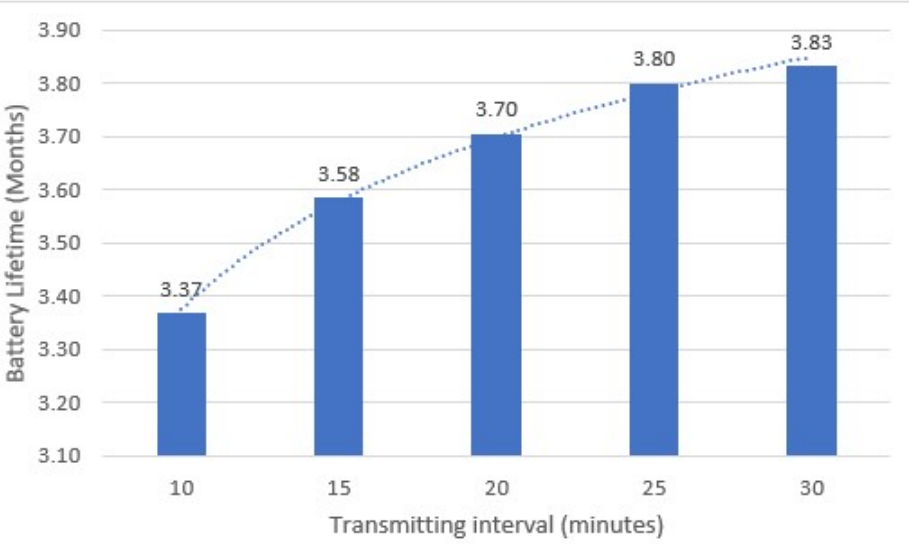

Figure 4.4: Battery life time estimation.

State	Process	Duration (S)	Avg. Current (mA)
1	Microcontroller Init	7.5	23
2	Modem Init	4.5	45
3	SDI-12 sensor data	4	60
4	Data Logging	8.5	60
5	Data Transmitting	20	59
6	Deep-sleep	60*N	8.7
7	Rainfall data	10*N	23

Table 4.1: The current and duration for each state in one cycle.

tion of each state as shown in Table 4.1. As you can see, there is a variable N which represent the transmitting interval in a minute. Assume that the transmitting interval is set to 10 minutes; therefore, the duration of Deep-sleep mode and reading rainfall data is 600 seconds and 100 seconds correspondingly. The average current in one cycle can be calculated as 13.2mA.

4.1.1 Battery Lifetime

Based on the current being calculated in the previous section, we can estimate the theoretical lifetime for the data logger running on the battery. Figure 4.4 shows the estimated lifetime base on different transmitting intervals. Consider 10 minutes as a standard transmitting interval, and it has 3.37 months of lifetime which is beyond the required 3 months lifetime. Also, if we increase the transmitting interval, the lifetime will expand, but the transmitting interval has a logarithmic relationship with a lifetime which means, further increase in the transmission interval won't have a significant improvement on the battery's lifetime.

4.2 Size

The size of the data logger needs to be constrained by dimensions of 100 × 60 × 60 mm3. These dimensions include the data logger's size with the modem. The length and width

Parts	PCB board	Other Components	PCB Assembly	Modem	Total
Price (NZ$)	22.76	84.61	8.64	100	180.01

Table 4.2: Total cost for each data logger

of the data logger are fully dependent on the PCB dimensions, as this is the largest board. The board's dimensions are 100 × 60 mm2. The height of the data logger with the modem stacking on top is 20 mm. Therefore, the size of the data logger is within the requirement specified.

4.3 Cost

Table 4.2 shows the total cost for each data logger is under NZ$ 300.

4.4 Rev 3.0 Data logger review

The PCB we are making is a third revision of the data logger. It was first sent to manufacturing in early August, and after 30 days of waiting, it finally arrived in early September. The PCB was made up entirely of surface mount components and was assembled by the PCB manufacturer. The quality of the board was terrific as well as the assembly. The board is designed to be plug and play when it comes down the production line, and no hand-on is needed. However, these boards come with seral issues.

The footprint for N-MOSFET didn't match the symbol of FDN361BN. This issue was corrected by selecting a suitable footprint. However, the temporary solution is to rotate the MOSFET by 120 degrees and the footprint matches again.

Another mistake being made is using input-only GPIO for output applications. Not all GPIOs in ESP32 are capable to configured as input and output. In particular, GPIO 36, 39, 34, 34 are input-only pin. This issue was fixed by assigning an appropriate GPIO to the pin with output capability. The temporary solution is to use a jump wire to reroute the wrong pin to the IO expander since there is unused GPIO leftover.

The last issue is that the device can not enter bootloader mode, so we cannot program the microcontroller. The problem was later be identified to be an incorrect state on the strapping pin.

A strapping pin is used to configure different behaviour when the device boot up. To enter bootloader mode, GPIO2 must be low or floating. If GPIO 2 were HIGH when GPIO 0 is LOW, the system would enter an error state. Unfortunately, GPIO 2 was connected to the output of the Multiplexer and it was HIGH when the data logger powered up. The solution was to attach a pull-down resistor to the Multiplexer's output, and this pin will not hold HIGH.

After reworking the PCB, it can perform the designed functionality.

Chapter 5

Conclution

5.1 Conclusion

The goal of this project was to keep developing and improving the early proof-of-concept Internet of Things data logger. This early revision data can read data from SDI-12 sensors, save it to an SD card, and transmit it to Azure. Unfortunately, because Ben and Jolon didn't pick the suitable modem, the data logger is missing the functionality of transmitting data via Low Power Wide Area Network, which is very important to an Internet of Things project.

Now, this imperfection has been fixed by choosing the SARA R410M modem. With the celebrated design of the modem driver, this modem can but is not limited to publish MQTT messages via the Cat-M1 network, which will provide a tool for GWRC to gain knowledge of the Low Power Wide Area Network. In addition, the data logger has developed a counter to read data from the tipping bucket rain gauge. This design is not only low-power but also provides high-quality data. What's more, it adds versatility to the data logger.

More importantly, a PCB made up entirely of surface mount components was produced. The PCB design and component selection always pursuing maximum efficiency, and as a result, the power consumption at deep-sleep is only 8.7mA is the foundation of achieving the requirement of a 3-months life time. Based on the low-power and highly autonomous embedded system being developed, the data logger will be able to test in the field.

5.2 Future Work

5.2.1 Rain gauge counter

Since the tipping bucket rain gauge can only tip at most 12 tips per minute, using full eight bits from the counter is unnecessary. In the future, consider removing the Multiplexer but using the IO expander only. This IO expander has eight programmable input and output ports. Use two ports for reset and enable/disable; use the rest of the six ports to read parallel output from the counter. This configuration will minimize IC being used, therefore more efficient.

5.2.2 Fix PCB design

The current PCB has a few mistakes been made. We have already finished the 4.0 PCB design[19]. However, due to the current lead time of PCB production take more than 30 days, we won't be able to receive and evaluate the updated version of the PCB before the project's end date. Beyond fixing the mistakes, the new PCB should put more consideration on user experience affected by the housing. For example, since the edge of the PCB will be

surrounded by the housing, assess the SD card socket and USB port will be difficult. Consider reorientated the port direction inward or towards the user for a better user interface.

Bibliography

[1] G. W. R. Council, "Environmental Monitoring and Research." http://graphs.gw.govt.nz/. Accessed: 2021-05-20.

[2] HyQuest, "Precision Instrumentation, Data Loggers, Internet of Things and Environmental Monitoring." https://www.hyquestsolutions.com/company/about-us. Accessed: 2021-09-20.

[3] S. Ritesh Kumar, "Energy Consumption Analysis of LPWAN Technologies and Lifetime Estimation for IoT Application," *Sensors*, vol. 20, July 2020.

[4] N. Khan, K. S. Khattak, S. Ullah, and Z. Khan, "A low-cost iot based system for environmental monitoring," in *2019 International Conference on Frontiers of Information Technology (FIT)*, pp. 173–1735, 2019.

[5] HyQuest Solutions, *iRIS 150FX Datalogger*, Apr. 2015. v. 1.40.

[6] HyQuest Solutions, *iRIS 350FX Datalogger*, Sept. 2018. v. 1.70.

[7] J. Behrent, "Hardware final report." https://gitlab.ecs.vuw.ac.nz/QuiltyGroup/Research/ENGR489/Environmental-Monitoring-2020/engr489-project/-/tree/master/docs/reports/final-report/hardware, May 2020. Accessed: 2021-05-22.

[8] B. Secker, "Development of an IoT System for Environmental Monitoring," 2020.

[9] ublox, "Sara-r4 series." https://www.u-blox.com/sites/default/files/SARA-R4_DataSheet_UBX-16024152.pdf. Accessed: 2020-05-24.

[10] "Discuession with nick thompson." https://gitlab.ecs.vuw.ac.nz/QuiltyGroup/Research/ENGR489/Environmental-Monitoring-2020/engr489-project/-/issues/397, May 2021. Accessed: 2021-05-22.

[11] SparkFun, "Sparkfun lte cat m1/nb-iot shield - sara-r4." https://www.sparkfun.com/products/14997. Accessed: 2020-05-24.

[12] Hologram, "Hologram officil website." https://www.hologram.io/. Accessed: 2021-09-20.

[13] HyQuest Solutions, *ML-IoT/W WiFi Data Logger*, Sept. 2018.

[14] R. Ashish, "Design and Development of a Real-Time Monitoring System for Multiple Lead–Acid Batteries Based on Internet of Things," *Future Internet*, vol. 9, June 2017.

[15] T. INSTRUMENTS, "Bq34z110 datasheet." https://www.ti.com/lit/ds/symlink/bq34z110.pdf?ts=1621939028873&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FBQ34Z110. Accessed: 2020-05-24.

[16] T. INSTRUMENTS, "Bq34z100evm." https://www.ti.com/tool/BQ34Z100EVM. Accessed: 2020-09-24.

[17] Ublox, *SARA-R4/N4 series - AT Commands Manual*, June 2018.

[18] L. Louis, "Drivers." https://gitlab.ecs.vuw.ac.nz/QuiltyGroup/Research/ENGR489/Environmental-Monitoring-2020/engr489-project/-/tree/master/device_rev_3/src/drivers, 2021.

[19] L. Louis, "Rev4.0 pcb." https://gitlab.ecs.vuw.ac.nz/QuiltyGroup/Research/ENGR489/Environmental-Monitoring-2020/engr489-project/-/tree/426-rev-4-0-schematic/hardware/Water%20Sensor%20Hub%20V4, 2021.