VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

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

Jolon Behrent

Supervisor: James Quilty

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

Abstract

The Greater Wellington Regional Council currently uses data loggers to monitor the environment. These loggers and accompanying software are provided by a single supplier which effectively locks the Council into using them for all monitoring. The Council wants to develop a low-cost, open-source Internet of Things solution with a connection to a cloud platform. This report looks at the development of a successful proof-of-concept device capable of reading from sensors and transmitting the data to Azure.

Contents

Figures

Chapter 1

Introduction and Background

1.1 Introduction

The Greater Wellington Regional Council (GWRC) monitors the air, land, and water resources of the Wellington region. Their main focus is monitoring the rainfall levels, water levels, groundwater quality, and air quality. The environment is monitored for many reasons including providing environmental information to the community [1], observing the impact of things on the environment, and adhering to the requirements of the Resource Management Act [2].

Internally, the environmental monitoring data is used by the Flood Protection, Environmental Regulation, Resource Consents, and Water Supply teams. These groups need more data, which means more data loggers are required. The problem is that the data loggers from GWRC's current supplier, HyQuest, are expensive, power-hungry, and also lock them in to only using loggers and software in HyQuest's ecosystem. As a result, GWRC wants to switch to an open-source Internet of Things (IoT) alternative for data logging. This will be beneficial for them as it means they have more control over what the device is capable of, and the price could be heavily reduced.

1.1.1 Objective

The objective of this project is to develop a low-cost, low-powered, IoT device with fully open-source hardware. Although the device should be cheaper than the HyQuest devices, it should not compromise sensor accuracy and device reliability. Many requirements need to be met for this device to be considered complete [3], the most important hardware requirements are:

- cost between $200 and $300 per device
- ability to communicate with SDI-12 sensors
- ability to monitor rain gauges
- battery powered with a lifespan of 3 months
- small enough to fit inside a tipping bucket rain gauge
- local data storage on SD card
- can be produced and maintained with limited soldering/electronics experience
- connection to cloud server over a low power wide area network, such as the Vodafone Narrowband Internet of Things (NB-IoT) network

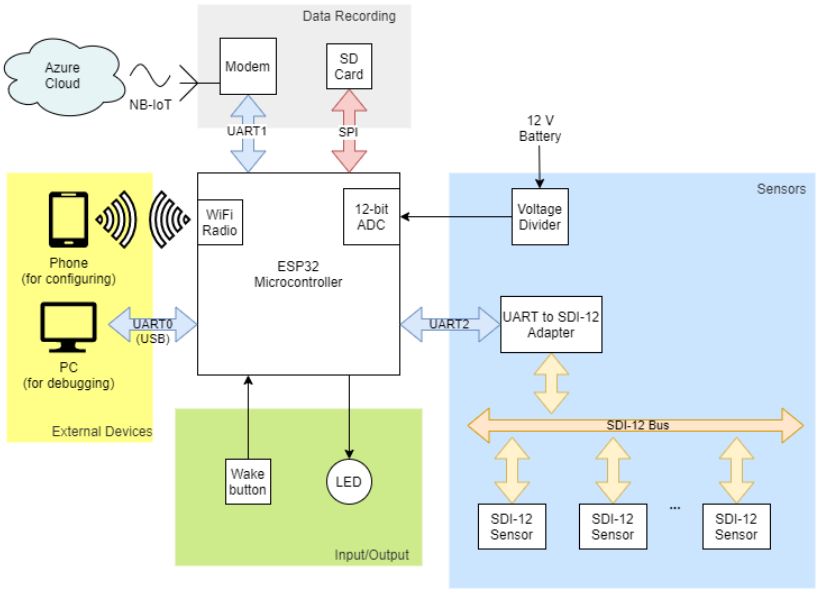


Figure 1.1: Block diagram overview of the entire system.

1.1.2 Overview

The system designed for this project is at a functional proof-of-concept stage. An outline of the system is shown in Figure 1.1. The diagram shows the main parts of the system:

- ESP32, the central processing unit of the data logger.
- Sensors, which can record data from the SDI-12 sensors or get the battery level.
- Data recording, which sends measurements to the SD card and the cloud.
- Input/output, which provides feedback to the user and allows them to wake the device and enable the WiFi.
- External devices, which are used for setting up the device.

1.2 Background

1.2.1 HyQuest Solutions Data Loggers

The iRIS data loggers are intended to be used for rainfall measurement, river level monitoring, wind measurement, and more. These devices can support a range of sensors, including but not limited to SDI-12 sensors and rain gauges. SDI-12 refers to the communication protocol used between the data logger and the sensor. Section 1.2.2 explains this protocol in more detail; in short, it is a unique protocol designed specifically for low-power environmental monitoring. HyQuest also manufactures tipping bucket rain gauges. These work by having a small bucket fill up with rain water. When the bucket reaches capacity, it tips and toggles the output voltage. The output is used by the data logger to determine how many times the rain gauge has tipped which gives an indication of the amount of rainfall. These two sensors types, although they are compatible with HyQuest's proprietary loggers, are not proprietary themselves, so could interface with any device provided it had the appropriate hardware. Both iRIS devices have a small LCD screen and buttons to allow a user to log in, view the system status, and enable a wired connection to a computer [4, 5].

The HyQuest loggers currently work well for GWRC; however, they fall short in a few areas that are important to GWRC.

- They use power-hungry 3G and 4G technologies for telemetry [4, 5]. This could be better done with low power wide area technologies, such as Narrowband IoT.
- They are expensive, according to GWRC.
- The types of monitoring GWRC can do is dependent on what products HyQuest offers. It is not possible for GWRC to customise the functionality of the HyQuest products.
- If HyQuest makes major changes to the products they sell, GWRC has to change too.
- They are difficult to self-repair and maintain if anything goes wrong.

1.2.2 SDI-12

The purpose of this section is to provide some background information about the SDI-12 protocol. Serial Digital Interface at 1200 baud (SDI-12) is a communications standard for low-powered data recorders. It is specifically designed to be used for collecting environmental data, and GWRC currently uses these sensors for the bulk of their environmental monitoring. It is similar to the I^2C protocol in that it uses only two wires and supports multiple addressable slave sensors connected to one master device. Where it differs from I^2C is that rather than using a clock line, it sends data at an agreed baud rate, similar to the Universal Asynchronous Receiver/Transmitter (UART) serial protocol [6].

Electrical Interface

The SDI-12 protocol uses 2–3 wires to provide 12 V power and communicate with the sensor. In some cases, the 12 V line can be removed and the device can be charged up over the data line. The data line communicates using half-duplex serial with inverted logic. The three states of the data line are shown in Table 1.1 [6].

Condition	Binary State	Voltage Range
marking	1	−0.5–1.0 V
spacing	0	3.5–5.5 V
transition	undefined	1.0–3.5 V

Table 1.1: SDI-12 logic levels

Communication

The SDI-12 protocol specifies that data recorders and sensors communicate via the exchange of printable ASCII characters (with some exceptions) over the data line. "Printable ASCII characters" refers to those between `<space>` and `~`. As printable ASCII characters do not use the most significant bit, one "byte" of data is represented by 7 bits. The byte frame for SDI-12 is 1 start bit, 7 data bits with the least significant bit first, 1 even parity bit, and 1 stop bit [6].

As mentioned at the start of this section, SDI-12 sensors are addressable. To communicate with a specific sensor, the command must begin with the sensor address which can be in the range of `0`–`9`. Some sensors also support addressing in the ranges of `a`–`z` and `A`–`Z`.

SDI-12 sensors must be awake to send commands to them. The exact timing parameters of waking a sensor are out of the scope of this report (more information in the SDI-12 specification [6]), however, it involves sending a period of spacing (a break) followed by a period of marking (a mark). This wakes up all sensors attached to the device, but sensors that are not addressed will return to sleep after receiving the first command. The current draw varies between sensors, though it can be as low as 0.5 mA in sleep mode and 6 mA in active mode [7]. For every additional sensor, the power consumption increases, but this only has a notable effect during the time following the break being sent until a sensor is addressed.

Existing Implementation

The TBS01A module supports bi-directional conversion of data into the SDI-12 format and back. This means that it converts the SDI-12 byte frame into the default UART byte frame: 1 start bit, 8 data bits, no parity, and 1 stop bit at a speed other than 1200 baud. The module also automatically handles the sending of breaks, marks, and any other SDI-12 specific messages [8].

The module is designed to be soldered directly to a circuit board. The module is shown in Figure 1.2.

Figure 1.2: The TBS01A module
Source: [8]

As with many existing implementations of SDI-12, how it is implemented is unclear. The TBS01A module uses the "ultra-lower-power" STM8L151 from STMicroelectronics which is configured in some way to perform all the necessary conversions. The STM8L151 does not have dedicated hardware for SDI-12, so this will be implemented in software.

Chapter 2

Design

Some of the design work done for this project had not been published as existing designs before. In particular, this relates to the design of the SDI-12 communication circuit which was designed based purely on the specification for SDI-12. This chapter compares design decisions that were important in the development of the data logger.

2.1 Design Constraints

The data loggers we developed have some overarching design constraints which influenced decisions made during the development of several different parts of the project. These constraints are listed here and will be referenced in later sections.

2.1.1 Power Consumption

The Greater Wellington Regional Council wants the data loggers to be able to remain powered for a period of over three months, with a preference for lasting closer to six months. They currently use deep-cycle lead-acid batteries with a capacity of either 18 or 32 Ah (ampere hours). These types of batteries are designed to be discharged almost completely without damaging the battery. As such, close to the entire capacity of the battery can be used.

Using this information and assuming that the entire capacity *is* used, we can find the maximum average current draw the data logger can have (in amps) with the following expression.

$$I = \frac{Q}{t} \tag{2.1}$$

where Q is the capacity in Ah and t is the time in hours. Table 2.1 shows the current draw for the batteries over both a three- and six-month period.

I decided to constrain the power consumption to the value for an 18 Ah battery over a period of three months. This means that GWRC can use any battery and be certain that it will last for a minimum of three months. As a result, all design decisions must be made in

Capacity \ Time	3 months	6 months
18 Ah	8.2 mA	4.1 mA
32 Ah	14.6 mA	7.3 mA

Table 2.1: The maximum average current draw for batteries over 3- and 6-month periods.

the context of being limited to drawing an average current of just 7.3 mA. To put this into context, the HyQuest Solutions data loggers that GWRC currently uses have an operating current of 7 mA in the lowest power mode [9].

2.1.2 Size

There was no firm specification for the size of the data logger, however, a constraint suggested by GWRC was that the logger be capable of fitting inside a tipping bucket rain gauge, which is another product from HyQuest. Although time constraints meant we were unable to implement interfacing with a rain gauge, this is still a good guideline to follow. These have a diameter of 200 mm and a height of over 300 mm [10]. The majority of the height is open but only about one-third of the diameter is free space. To have an estimated size to constrain the design, I decided the maximum dimension constraint will be $150 \times 60 \times 60\,\text{mm}^3$, which means the data logger would be small enough to fit inside the rain gauge with plenty of remaining space. These dimensions will influence the size of the PCB design as well as any components selected.

2.1.3 Cost

The maximum GWRC is willing to spend per data logger is $300. This should cover all components, PCB manufacturing and assembly, and any other additional costs which are incurred when manufacturing a data logger. During development, the cost of the data logger was measured based on the cost of one device, rather than taking bulk discounts on components into account. In most cases, this meant that when selecting a component, once all other hard constraints of the component had been met, the part with the minimum cost was selected.

2.2 Microcontroller Selection

Early in the project, we set out the requirements for the microcontroller, which specified the features which were required and the ones that would be useful to have. The following features (listed in order of importance) are what were considered when selecting a microcontroller.

1. **Required** A sufficient number of GPIO pins for all external connections. This was originally estimated as 20, but was later found to be 21.
2. **Required** Interrupt support to allow the device to be woken by internal and external interrupts.
3. **Required** UART driver for communication with SDI-12 sensors and any other peripherals.
4. **Required** Meeting the overarching hardware requirements (Section 2.1).
5. **Optional** WiFi module for allowing wireless connection to the device from a laptop or phone.
6. **Optional** MicroSD card slot to allow data logging to an SD card.
7. **Optional** NB-IoT mode for transmitting data over the air.

Some microcontrollers do have all or most of those features, however, this typically resulted in them being either too expensive, too large, or too power-hungry. The optional features are listed as such because it is not required that the feature be present on the microcontroller, as long as it is possible to implement the feature with external components.

The microcontrollers were narrowed down by their ability to be programmed in Python/ MicroPython. The reason behind this decision was that a Python-based language would allow for more rapid prototyping, which would be beneficial for a project with a relatively short development period.

Taking the required features into account reduced the large pool of microcontrollers being considered to either the Espressif ESP32-DevkitC or the MicroPython pyboard.

Pyboard The MicroPython pyboard is a microcontroller development board that is manufactured with MicroPython firmware. It uses a low-powered STM32F405RG ARM microprocessor which draws 7 mA during normal operation [11]. It can enter a deep sleep mode to conserve power. The board has 24 GPIO pins available and some of which support interrupts. The board has five UART drivers meaning there are sufficient ways to communicate with the sensors and with an external modem. There is an on-board SD card slot and the device supports WiFi.

The biggest downside of the board is its cost. They typically cost upwards of $70. This accounts for almost one-quarter of the budget for one device.

ESP32 Development Board The Espressif ESP32 is a low-power and low-cost development board. It can be flashed to support MicroPython firmware which gives it almost exactly the same functionality as the pyboard. The ESP32 has 22 GPIO pins, which is enough to support all the required functionality while leaving one pin reserved for future use. The board has three UART drivers, which is enough for the SDI-12 sensors and an external modem. It supports both WiFi and Bluetooth, although only WiFi is needed for this project.

The power consumption of the microprocessor is much lower than the pyboard, coming to only 4 mA during normal operation and dropping to 5 µA when in deep sleep mode. The board costs only $20, which makes the overall device $50 less expensive than if using the pyboard.

The ESP32 also has one major benefit over the pyboard in terms of future development. As GWRC has expressed interest in having support for HyQuest rain gauges as a stretch goal, the ability to collect frequent data without waking the device is important to consider. In heavy rain, the rain gauge will be sending pulses up to one per second, which could mean the device is fully powered for a significant amount of time, which would result in the battery draining faster than expected. The ESP32 has an additional processor called the Ultra Low Power coprocessor. This processor remains active even when the device is in deep sleep. It can access some of the GPIO pins, read the state of them, and record data into RAM to be accessed when the logger wakes. Since the pyboard does not have a low-powered coprocessor, it would not be worth considering it for rain gauge purposes. Although the rain gauge connection was not implemented at the end of the project, this means it will be possible to include this functionality in the future. As a result, the ESP32 was chosen as the microcontroller.

2.3 Modem Selection

The modem used for this project needed to be able to communicate over Vodafone's Narrowband Internet of Things (NB-IoT) network. As such, our selection of modems was heav-

ily dependent on what Vodafone had tested and approved for use on their network. Using a different modem would be possible, but it would mean Vodafone would not offer their support if issues arose. The modems supported by Vodafone at the time we made a decision were: Ublox Sara R410M, Quectel BG96, and Telit ME910C1-AU.

The only requirements we had for the modem was that it should be able to communicate over a protocol supported by the microcontroller, which could be UART, SPI, or I^2C. As the modem was going to be one of the most expensive components, the client had a preference for a modem that could be removed from a faulty data logger and placed in a functional one.

There were no clear benefits with using any particular modem as they all offered very similar features. The ME910C1-AU modem was the lowest price of them all and also was available in a form factor with pins (rather than solder pads) so could be inserted and removed from headers that could be soldered to the data logger. Due to the form factor, the modem can only communicate over UART, however, this was sufficient for sending commands to the modem.

2.4 Capacitor Selection

Capacitors are crucial components in the power circuit, used with the voltage regulators, and on direct current (DC) voltage lines. As capacitors act as short circuits to alternating current (AC) and open circuits to DC, they are best placed around the inputs to regulators in order to prevent changes in the input voltage. Without them, any major fluctuations could cause damage to the regulator. They are placed on the regulator outputs to stabilise the voltage being fed into other components. Capacitors should also be used near the power inputs to other components to protect against brownouts, which typically occur when a component tries to draw too much current before the regulator can respond and provide more.

There are many different types of capacitors with different properties. In this situation, the type of capacitor doesn't matter, as it only needs to meet the capacitance, voltage rating, and equivalent series resistance (ESR) requirements. Some capacitors achieve these better than others, which will be discussed here. Specifically, I considered aluminium electrolytic, tantalum electrolytic, polymer electrolytic, and ceramic capacitors.

Capacitance The capacitance of a capacitor is a measure of its ability to store charge. A higher capacitance means that more charge is stored for a given voltage. A high capacitance is beneficial in the previously mentioned power circuitry as means that if a component draws too much current, the capacitor can supply some charge without the voltage dropping. Aluminium electrolytic capacitors typically have the greatest capacitance of the ones mentioned above [12].

Voltage Rating The voltage rating of a capacitor is the maximum voltage that it can have applied across it. If this voltage is exceeded the capacitor can be damaged, which can be dangerous if aluminium or tantalum capacitors are selected as they will leak or explode. Regardless, the voltage rating that is chosen should be greater than the maximum expected voltage by about 50% in most cases. Ceramic capacitors can be made with much higher voltage ratings that any of the other types [12].

Equivalent Series Resistance The equivalent series resistance (ESR) is the resistance to alternating currents passing through the capacitor. As the capacitors are being used to filter out voltage fluctuations by sending it to ground rather than the circuit, having more AC resistance is detrimental as it means that it will be more difficult to send the

variations to ground. Ceramic and polymer capacitors typically have low ESR which means they result in greater stability [13].

2.5 Regulator Selection

GWRC plans to power the data loggers with 12 V lead-acid batteries. As the all of the electronics, with the exception of the SDI-12 sensors, will operate at a lower voltage, the circuit will need to include some regulators to manage the voltage and current appropriately without using significant amounts of power. Communication with the SDI-12 sensors operates at 5 V (Section 1.2.2)—the highest voltage level used in the electronics—so it would be logical to use a 5 V regulator as an input to the other regulators to avoid having multiple regulators converting from 12 V to the desired voltage. The only other regulators required for this design are for the modem and the microcontroller, as almost every other component will be powered from the 3.3 V output on the microcontroller.

Most components are relatively low-powered, with the exception on the modem, which can draw up to 600 mA when transmitting data [14]. Both the main, 5 V regulator and the modem regulator must be able to support that. If not, the modem may not be able to transmit data reliably. The main regulator must also be able to support the potential current draw of the other components, which could be an additional 240 mA or more if the microcontroller was transmitting over WiFi [15]. To account for the worst case scenario, the regulators have been over-specified so that they must be capable of supply a minimum of 1500 mA.

The main regulator has the potential to have a high amount of current passing though it. Therefore, it is imperative that it has a high efficiency to avoid needless power draw. Buck converters—a type of switching regulator—can have peak efficiencies of up to 99 %. This means that for every milliamp supplied by the battery, the data logger would get 0.99 mA. As the efficiency increases so does the price, so I decided that the minimum efficiency the design needed to meet was just 85 %. These regulators are also capable of converting voltage to current, so a 12 V and 5 mA input could to dropped to a 5 V and 12 mA output. In this case, the input current is 40 % of the output current.

As the modem regulator is going to be taking the 5 V output of the modem and dropping it to a lower voltage, it is simpler and cheaper to just use a linear regulator. In particular, a low dropout regulator would be more likely to be able to reliably regulate the output even when the input voltage is relatively low. Linear regulators are not as efficient as switching regulators because they convert the voltage drop over the regulator into heat. However, as the modem is only going to be on periodically, a linear regulator is much more suitable.

Another important factor I need to consider in the selection of both modems is the quiescent current. This is the passive current that flows through the regulator even when no load is connected. To prevent excessive power consumption while the device is asleep, I must select a regulator with a suitably low quiescent current. I decided the maximum current would be 500 µA for the main regulator, and 100 µA for the modem regulator. Any current drawn by the modem regulator will need to pass through the main regulator, hence the lower allowable quiescent current.

2.6 PCB Design

As a project stretch goal, GWRC had asked for the data loggers to support connecting to HyQuest rain gauges, which operate by sending a pulse to the microcontroller to indicate that a certain quantity of rain had been collected. As the main goal was to get a logger functioning with SDI-12 sensors, considerations needed to be made around the printed circuit

board (PCB) design to facilitate producing rain gauge compatible boards (which are not necessarily compatible with SDI-12 sensors) some time after the completion of this project. The designs we considered were: a modular design, with SDI-12 circuitry on a separate board; two separate boards, one for connecting to rain gauges and one for connecting to SDI-12 sensors; or one data logger that can connect to both rain gauges and SDI-12 sensors.

Unified The unified design would involve a single PCB which holds all of the hardware required for both SDI-12 and rain gauges. This design is good because it means that only one PCB needs to be manufactured, which can reduce costs. This design has several issues though. The first issue being that a single board design will be larger in terms of length and width, however, it is unlikely that it would not meet the size requirements. Having one system also means that GWRC will have to pay for a full board, even if they aren't using all the features of it. Finally, this means that the rain gauge would no longer be a stretch goal, as it would need to be implemented on the main board from the start. Overall, the downsides of the unified board design far outweighed any benefits received from it.

Independent The independent design meant that two separate board would be created, one for SDI-12 and one for the rain gauge, if time permitted. Each PCB could be made smaller or even shaped in a particular way more easily, which would mean they could fit in to a particular housing more easily. It would also make the development of the software simpler, leading to faster development, as the code would only need to be written with a particular functionality in mind. The biggest downside of this design is that it requires multiple devices at one site if both rainfall and SDI-12 measurements are being taken. The biggest cost item for each device is the modem, which would need to be present on both boards and would increase the overall cost dramatically.

Modular The modular design would have a base board which supports the rain gauge functionality by default, as it requires only a small amount of additional hardware. The base board would hold the microcontroller and modem, but would also have headers to attach an expansion board to the top. The expansion board would have all the hardware required for SDI-12 functionality. This would mean that GWRC only needs to purchase a base board for rain gauge functionality, and an SDI-12 expansion board if they want to be able to attach additional sensors. This design would also open up opportunities to develop expansion boards for other sensor types, which is something GWRC was considering, without needing to modify the base board. The only downsides to this were that the software would need to be written to support different sensor types and the cost would increase slightly due to additional components needed to support the expansion boards, such as headers.

In the end, the modular design was selected. It has far greater extensibility than the other designs, allowing GWRC to develop it further in the future to support additional expansion boards. The modular design is slightly more expensive but offers more benefits and does not have the same issues as the other options.

2.7 Enabling Local Wireless Connection

One of the requirements for this project was to allow a user to access the device locally via a wireless connection, i.e. WiFi or Bluetooth. WiFi was selected for various reasons, including ease of development and speed. The purpose of this section is not to look at which communication method was selected, rather it will focus on how this is enabled.

Since one of the most important constraints of the data logger is that it needs to be low-powered, it would be detrimental to the battery life if the WiFi was always enabled. Instead, the data logger needs to offer a user the ability to enable WiFi when they are nearby. The options we considered were reed switches and hall effect sensors which can be triggered from a short distance with a magnet.

2.7.1 Reed Switches

Reed switches are made up of two small pieces of metal inside a glass casing which move to close a circuit when in the presence of a magnetic field. The range of a reed switch is heavily dependent on the strength of the magnet, so they typically only work from within a range of a few centimetres. As they involve moving parts, they are prone to switch bounce, which occurs when the pieces of metal bounce against each other during switching, resulting in several pulses being detected [16]. However, this is simple to filter out either with hardware or software.

2.7.2 Hall Effect Sensors

Hall effect sensors contain a thin sheet of conductive material which allows a voltage to be produced on the output when subjected to a magnetic field. Depending on the construction of the sensor, it will output an analogue signal proportional to the intensity of the magnetic field or a binary signal which is dependent on whether the magnetic field is above some threshold voltage [17]. Hall effect sensors are not susceptible to switch bounce so no additional circuitry or software is needed.

Using only a magnetic field sensing element for enabling the WiFi access point on the device would be problematic as it requires the user to bring a magnet with them (or leave a magnet at the site) to enable it. To account for this, a non-latching push button was also added to the logger to provide some redundancy if the hall effect sensor fails or the user otherwise cannot activate the sensor.

2.8 SDI-12 Design

The SDI-12 protocol is very similar to the commonly-used UART protocol. As practically all common microcontrollers contain a built-in UART driver, it seemed the most sensible to use the existing driver and build upon it either by modifying the software or adding additional hardware to convert between protocols. Two requirements must be met to design a working UART to SDI-12 converter: voltage conversion and data line union.

2.8.1 Voltage Conversion

The general-purpose I/O (GPIO) pins on the microcontroller all operate at 3.3 V [15], while the logic level required for reliable SDI-12 communication is 5 V. As the microcontroller will be both sending and receiving data, there needs to be circuitry that is capable of converting voltage from 3.3 V up to 5 V for transmitting data, and from 5 V to 3.3 V for receiving. Implementing this is simple, however, due to the inherent complexity involved with reliably converting between SDI-12 and UART, it would be beneficial if this could be implemented with as few parts as possible to minimise cost. It is also important that the current draw of the conversion circuits is limited especially if the circuit is in an active state by default,

as this could cause the battery to drain even when the logger isn't communicating with the sensor.

Additionally, the SDI-12 protocol communicates with inverted logic levels [6]. This differs from most implementations of UART which only support high voltages representing ones and low voltages representing zeros. In the case of inverted logic levels, a high voltage represents a zero instead. An inversion step must be done somewhere between the data logger and the sensors to set the logic levels for successful communication between the devices.

2.8.2 Data Line Union

SDI-12 sensors communicate over a single data line [6], while UART is full-duplex meaning it can communicate in both directions at the same time. This means the data logger, which has separate lines for transmitting and receiving, needs to be adapted to communicate over a single line. An ideal solution would result in none of the transmitted commands from the microcontroller being fed back to the receiving line, otherwise, this would incur a minor computational cost to remove unwanted data.

2.9 Voltage Protection

In order to provide extra safety to the devices, voltage protection needs to be added to the battery input. The requirements stated that the device should have overvoltage and reverse polarity protection. The purpose of voltage protection is to prevent excessively high voltages from reaching the rest of the circuit, as this could cause significant damage to the data logger. Reverse polarity protection grounds the entire circuit if the battery is connected the wrong way as this could also cause damage.

A typical reverse polarity protection circuit uses a Zener diode—which drops a fixed voltage over it, known as the "Zener voltage" unless the input voltage is less than this voltage—with a resistor and a MOSFET. Once the input voltage exceeds the Zener voltage, the MOSFET will be enabled and the circuit will open. This prevents any voltage that is not the correct polarity from passing through. If the MOSFET can withstand the input voltage, it is possible to remove the diode and resistor completely.

Overvoltage protection circuits can be relatively complex, but in the interest of making the data loggers inexpensive, a simpler overvoltage circuit can be used instead. This circuit can use just a single Zener diode which would have a Zener voltage not much more than 14–16 V as this will set the maximum input voltage. There is a small resistor before the diode to slow the current if the voltage exceeds the Zener voltage.

Chapter 3

Implementation

3.1 Circuit Design

There were many decisions that needed to be made in the design of this circuit. These are:

- Power switching, to enable and disable SDI-12 sensors and the battery monitor
- SDI-12 logic, to communicate with the sensors
- Voltage protection, to prevent damage if an incorrect power source is attached
- Battery monitoring, to record the voltage of the battery
- Regulators and capacitors, for managing voltages while taking into account the power considerations

3.1.1 Power Switching

To conserve power, a few parts of the circuit can be switched off and only turned on when they are needed. This was most important for parts that can draw particularly high current, specifically the voltage divider for battery level monitoring and the SDI-12 sensors, which increases linearly with the number of sensors attached. There are several possible approaches to switching, but the ones compared here are relays, BJTs, and MOSFETs.

Relays are relatively large switching devices which use a magnetic coil to open and close a switch. However, as with any magnetic coil, they require current to create a magnetic field. This means that a relatively substantial amount of current, in the order of milliamps, will be used to close the switch. They also create electrical noise, which could interfere with any wireless communication.

Bipolar junction transistors (BJTs) are current-controlled devices, which means they allow a current to flow past them when a current is applied as the control signal. Again, as power consumption is a concern, using a device that relies on current to switch on will have a negative impact on power consumption.

Metal-oxide-semiconductor field-effect transistors (MOSFETs) are similar to BJTs, except that they are voltage-controlled rather than current-controlled and consequently have significantly lower current draw in operation. One issue with MOSFETs is that more current will be drawn if the input voltage is different from the control voltage, even when the control signal switches it off. I fixed this issue by implementing the circuit in Figure 3.1, which allows the control voltage (3.3 V) to be different to the input voltage (12 V) without affecting functionality. In that circuit, setting the control signal to low will disable the first MOSFET,

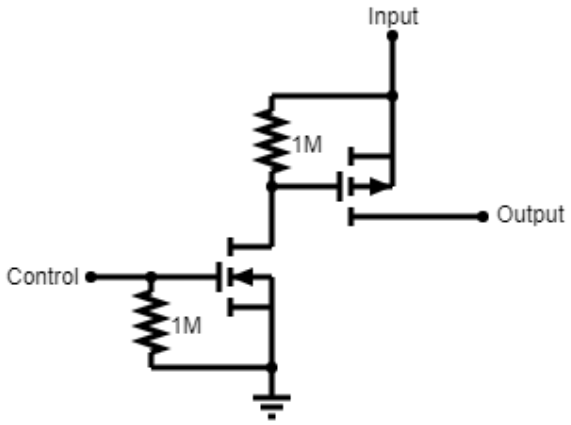


Figure 3.1: Circuit diagram of MOSFET switch

which will cause the second MOSFET to open. This allows the input voltage to propagate to the output. By using large resistors, the passive current draw is reduced to less than 20 µA.

3.1.2 SDI-12 Hardware

Although there are plenty of off-the-shelf solutions for converting common communication protocols to SDI-12, these usually come at a very steep price, typically costing more than $100 [18], which is one-third of the budget per device. Most other implementations are not open-source, so these could not be recreated for this project.

The SDI-12 protocol communicates over a single wire and sends inverted bits at a rate of 1200 bits per second (Section 1.2.2). Due to the similarities of the SDI-12 protocol and UART, I chose to design a circuit capable of adapting between UART and SDI-12. This meant that we could use the ESP32's dedicated UART hardware, rather than implementing the SDI-12 protocol in software. The circuit also needs to be able to convert the 3.3 V signals from the ESP32 to 5 V signals that the SDI-12 sensors accept, and vice versa.

Potential Solutions

To begin, I will look at solutions that I had considered during development with a short evaluation of each solution.

Resistor Bridging One of the most difficult parts of the system to implement was combining the transmitting and receiving lines so the device could communicate over a single wire. A simple way to achieve this would be to connect the transmit and receive lines with a resistor. Unfortunately, this did not work, and the data received was typically garbled.

RS-485 Transceiver Another option—suggested in an online forum [19]—was to use an integrated circuit used for communications using the RS-485 protocol. RS-485 is a common protocol that is even more closely related to SDI-12 than UART is. This chip works by accepted UART signals and converting them to RS-485. The RS-485 signals are then even simpler to convert to SDI-12, needing only a small number of additional components.

IN_1	IN_2	AND	Tri-state
0	0	0	Z
0	1	0	Z
1	0	0	0
1	1	1	1

Table 3.1: Truth table for AND gate and tri-state buffer. IN_1 is being used as the enable input (`Dir`) on the tri-state buffer, while IN_2 represents either the transmitted or received data. The state listed as Z is known as high-impedance and refers to an open circuit.

Although the circuit was implemented in the same way as in [19], I could not establish working communication with the sensor. During testing, I found that the output voltage going into the SDI-12 sensors was noisy and was typically around 2.3 V when transmitting a high voltage, much lower than the 3.5–5.5 V range required by the sensors. With more development work, it may have been possible to fix any issues with it, but the RS-485 transceiver chip cost $10—about $\frac{1}{30}$ of the budget per device—so I chose to pursue cheaper and potentially more reliable options.

Chosen Solution

One of the benefits of the RS-485 transceiver approach is that it can disconnect the receiving line when the microcontroller is transmitting. This means that the device does not receive any redundant data that it then needs to parse out, which would increase the chance of errors if not parsed properly. To achieve the same operation, I chose to use two tri-state buffers, which pass the input signal through to the output when the buffer is enabled but completely disconnects the circuit when disabled. This differs from an AND gate, which would pull the output to ground and block communication between the logger and the sensors. The truth tables of a tri-state buffer and an AND gate are compared in Table 3.1.

It would be possible to implement the circuit with a single tri-state buffer on the transmitting line [20], which would simply prevent the transmit line from interfering with the received signal when it is in a passive state. Adding an additional buffer on the receiving line with an inverted enable input means the receiver will always be blocked while transmitting, which means none of the transmitted data will be received.

As mentioned in Section 1.2.2, the device must also be able to send a break signal—at least 12 ms of spacing (see Table 1.1)—to wake the device. The dedicated UART hardware does not make it possible to send a continuous voltage level, so I needed to develop an additional circuit to handle that. I observed that when the device is not transmitting, the transmit pin defaults to a high voltage (3.3 V), so this could be used in conjunction with a NAND gate and a control signal to send a break. Table 3.2 shows the signal being transmitted to the SDI-12 sensor. When the transmitting line (Tx) is in its default state (1), the output can be controlled by toggling the force out (FOut) pin between 0 and 1. When FOut is a high voltage, the output is equal to the inverse of the transmit line. Otherwise, it forces a high voltage on the transmit line, which represents a break. This is intentional, as the SDI-12 protocol requires inverted logic levels. The NAND gate I selected is an SN74LS00 as it is capable of accepting "high" input voltages as low as 2 V, while also outputting high voltages above 3.5 V[21]. The SN74LVC2G241 tri-state buffer can further increase this voltage to 5 V [22], which allows the transmitting line to shift from 3.3 V up to 5 V without any dedicated voltage shifting circuitry. On the receive side, the MOSFET and resistor are used to reduce the 5 V data to 3.3 V to be compatible with the microcontroller. The full implementation of this circuit is shown in Figure 3.2.

Tx	FOut	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.2: Truth table for a NAND gate.

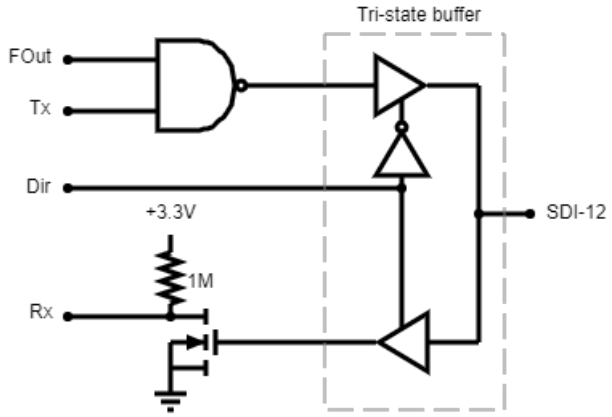


Figure 3.2: Full implementation of the SDI-12 circuit. The tristate buffers are used as a complementary pair to prevent transmitted signals from also being received.

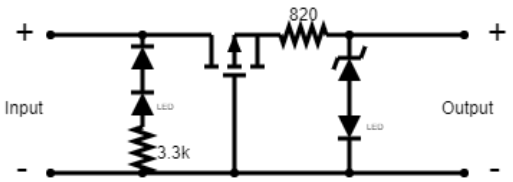


Figure 3.3: Protection circuit including LEDs for user feedback

3.1.3 Voltage Protection

The voltage protection from Section 2.9 was implemented in the final circuit design. Perhaps one of the most significant issues is the lack of indication to the user when the protection has been triggered. This would be useful as it would let users know when something is wrong.

To provide this information, one LED was added to each circuit. For the reverse polarity protection, a simple LED circuit protected by a diode was added just before the protection MOSFET. In any situation where the input voltage is reversed, the LED will turn on, indicating that the battery is improperly connected. In the case of the overvoltage protection, an LED was added after the Zener diode. Since the voltage drop over an LED is proportional to the current through it, as the input voltage begins to exceed the Zener voltage, the output voltage will switch from increasing linearly to increasing logarithmically. As expected, the output voltage increases slowly in the logarithmic region, so the input can safely exceed 20 V without risking damage to the circuit. The full circuit, including LEDs, is shown in Figure 3.3.

3.1.4 Battery Monitoring

GWRC wants the loggers to return information about the battery's voltage so they know when they should replace it. This is simple to achieve with one of the many built-in analogue to digital converters (ADCs) available on the ESP32 microcontrollers. The battery monitoring is a two-step process that involves the user configuring the ADC on any new data loggers before the devices can read the voltage properly.

Initial Setup/Provisioning

A linear ADC would be simple to calibrate, and it would involve taking only two measurements and determining the rate of change of the ADC output against the voltage. Unfortunately, the ADCs on ESP32s are notorious for being non-linear [23], meaning the response deviates as the input voltage increases linearly, as shown in Figure 3.4. After the voltage is scaled up to 12 V, this error is significant enough that the battery voltage can be inaccurate by over 1 V. As such, the ADCs typically require some extra calibration to track the voltage accurately. There were two possible ways I considered for calibrating the ADC. The first was to develop a regression model, and the second was to use a lookup table. Whichever option was chosen would need to be set up for every new device, as the ADC response can vary dramatically between different ESP32s. This means that it would be beneficial for GWRC if it could be generated by the microcontroller, rather than requiring them to generate it manually.

The most important thing to note when it comes to linearising the ADC is that the digital to analogue converters (DAC) on the ESP32 *are* linear. This means the DAC response can

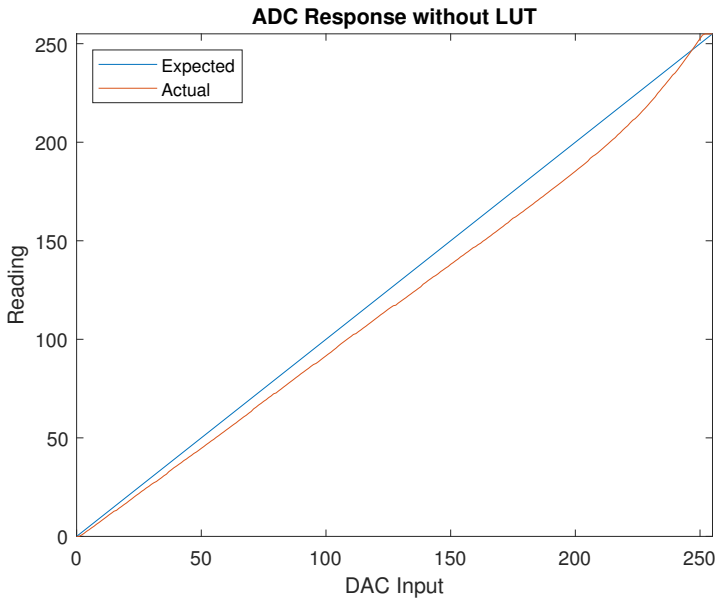


Figure 3.4: ADC response compared to the expected response. The ADC inputs have been scaled to match the DAC outputs.

be modelled with a straight line equation, so only two voltage measurements need to be taken. By connecting the DAC to the ADC during the initial setup stage, it can act as a linear reference. With this information, the DAC values can be converted to a voltage with very high accuracy.

Regression If regression were chosen for linearising the ADC input, a general model would first need to be designed that could represent any ADC response curve, as in, it would need to be *at least* a second-order model. The key problem here is that it could be higher than a second-order model, which would increase the computational complexity for generating the model if more accuracy was desired. It is also possible that the ADC could change suddenly for a small input voltage change. This would be even more difficult to model with regression and could even require treating the response as a piecewise function and developing a model for each segment.

Lookup Table The lookup table (LUT) approach does not have the same issues as using regression. It can model the response better because it does not need to generate a model; instead, it works by mapping ADC values to DAC values (and, therefore, voltages because the DAC is linear). Unlike regression, the complexity of the ADC response curve does not increase the complexity of the LUT. One problem with both approaches, which affects the LUT more than the regression, is that the resolution of the DAC is just 8-bits (256 unique values) while the ADC is 12-bits (4096 values). This means that there are 3,840 voltage levels that are unaccounted for in the LUT. To fix this issue, I designed the LUT so that it interpolates between known readings to generate the missing readings. This results in a mapping made up of hundreds of linearised regions, which reduces accuracy, whereas the regression model could be generated without the need for interpolating. The entire LUT also uses around 70 kB of internal storage to store all the required information, whereas the regression model would only need a few bytes to store the regression coefficients. Regardless, when stored on an SD card of several gigabytes, either value is insignificant. Ultimately, the lookup table

was chosen for this design as it offered more protection against sudden changes in the ADC response.

Typical Use

With some linearisation implemented, it was still possible for me to improve the precision of the ADC. This was achieved in two ways. The first was to add a small capacitor to the input of the ADC. This capacitor was selected to be small enough that it did not have a large enough time constant to require the ADC to wait for the voltage to settle, but large enough that it would filter out minor fluctuations in the battery level. A 0.1 µF capacitor placed close to the ADC input pin was selected for this purpose. The second way that the accuracy was improved was by taking the average of several readings. By default, the ADC software driver that I wrote takes 100 readings.

As the maximum input voltage to any general-purpose IO pin on the ESP32 is limited to 3.3 V [15], the battery voltage—which is typically around 12 V—must be dropped to an acceptable level to avoid damaging the microcontroller. As shown in Figure 3.4, the ADC repeats several of the same values at high voltages. To account for this, I decided not to use the full range of the ADC, to avoid reading the repeated values. Therefore, I designed a voltage divider that could reduce voltages up to 20 V to 3.3 V. The equation for a voltage divider is shown in Equation 3.1. The resistors that I chose were $R_1 = 15\,000\,\text{k}\Omega$ and $R_2 = 2700\,\text{k}\Omega$. This meant that 12 V would be dropped to 1.83 V at the ADC input.

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot V_{in} \tag{3.1}$$

3.1.5 Regulators

The project's requirements specified that a 12 V deep-cycle lead-acid battery should power the device. However, all of the electronics operate at either 3.3 V, 3.8 V or 5 V. More specifically, the SDI-12 communication circuitry operates at 5 V, the modem operates at 3.8 V, and everything else, such as the SD card and microcontroller, operates at 3.3 V. Figure 3.5 shows a breakdown of the voltage levels for the different parts of the system. As shown here, a single regulator is used to convert the 12 V input down to 5 V, which is then used by all other components.

The selected microcontroller has its own on-board linear regulator for connecting 5 V power directly to it and dropping the voltage to 3.3 V. Being a linear regulator will be subject to some losses; however, the 3.3 V circuitry has a relatively low current draw so these losses will be small—approximately equal to $1.7 \cdot I$, where I is the current and 1.7 is the voltage dropped by the regulator. The microcontroller has a 3.3 V output pin, which can be used to power any other low-current logic circuits that operate at that voltage. The presence of an on-board regulator means that only two regulators (3.8 V and 5 volt) need to be selected for the data logger.

Main Regulator The regulator must be capable of taking voltages in the range of 10–15 V and supplying over 1.5 mA with an efficiency greater than 85 % (Section 2.5). A TPS562208 linear regulator was selected to achieve this. This regulator can accept input voltages up to 17 volt and convert them to 5 V. It is capable of supplying up to 2 A with at least 92 % efficiency. It has a maximum standby current of 750 µA, which is slightly more than specified in the design constraints. I allowed this constraint to be ignored because it meant the data logger did not need to use a more expensive regulator at the cost of only a small amount of battery life.

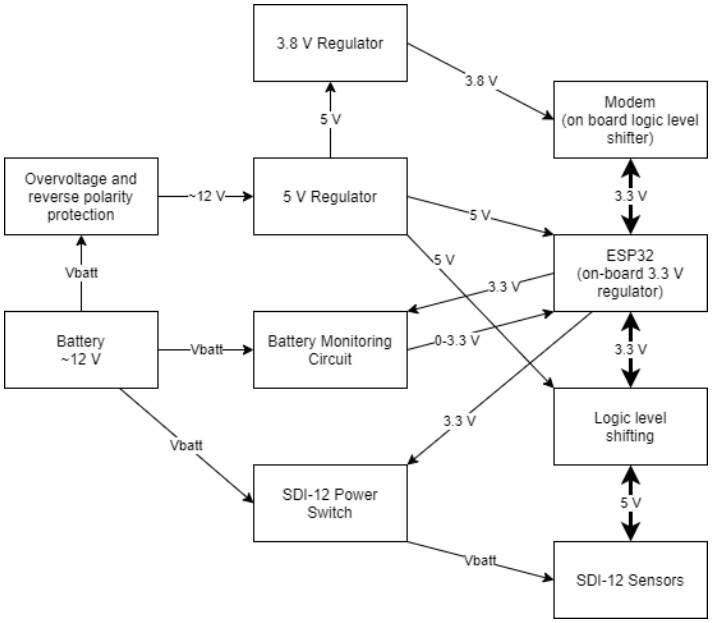


Figure 3.5: The voltage levels of the power lines (thin) and the data lines (thick) between components.

Modem Regulator The modem has a nominal input voltage of 3.8 V, although it is possible to power it at voltages of 3.2–4.5 V. This means it could be powered by a 3.3 V supply, assuming all design suggestions from the modem manufacturer, Telit, were adhered to [14]. In either case, a separate 3.3 V regulator would be needed as the one on the ESP32 would not be able to supply enough current for the modem. Therefore, I decided that a 3.8 V regulator would be the best to ensure the modem operates correctly.

As mentioned in Section 2.3, the modem regulator needs to be able to supply a minimum of 600 mA and have a quiescent current of less than 100 mA. The TPS7A7001 was selected for this. It is capable of supplying up to 2 A and has a maximum dropout of 380 mV, which means the output voltage will never droop. Most regulators do not come with a 3.8 V variant, so this must be configured manually with a voltage divider connected to the feedback pin. The feedback voltage is a constant—0.5 V for this regulator—so the voltage divider must be tuned appropriately to convert the output voltage of 3.8 V to the correct feedback voltage. Equation 3.1 shows how to select the resistors for the voltage divider. Using $V_{out} = 0.5\,\text{V}$ and $V_{in} = 3.8\,\text{V}$, I found that the ratio of the voltage divider needs to be $\frac{5}{38}$.

3.1.6 Capacitor Selection

As seen in Figure 3.6, all capacitors that were considered in Section 2.4 (ceramic, aluminium, tantalum, and polymer) exist within the range of nanofarads to microfarads, though sometimes on the edges of a particular region. It is unlikely that the input voltage to the data logger will ever exceed 16 V, so the voltage rating of any capacitor does not need to be any higher than 25 V (although most of the capacitors I selected are less than this, where appropriate). In order to improve stability, the capacitors need to have low equivalent series resistance (ESR). Aluminium and tantalum electrolytic capacitors typically have comparatively high ESR. Although it is unlikely that the ESR of these capacitors would cause issues, it was safer to select a polymer or ceramic capacitor to reduce the chance of having issues. The size of the capacitors is also important. Having capacitors that are too large will increase

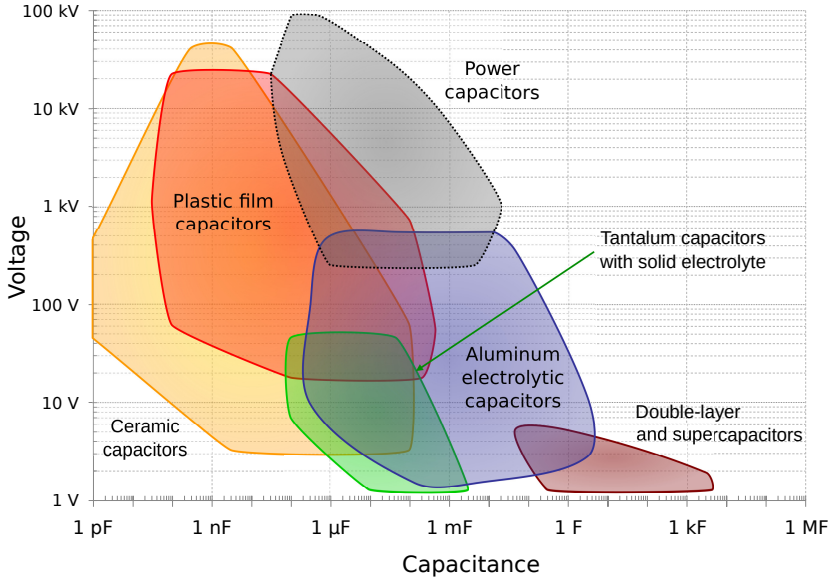


Figure 3.6: Capacitance and voltage rating of capacitor types
Source: [24]

the size of the data logger. Therefore, the ideal capacitors to use for the devices are ceramic capacitors, which can often be the smallest components on the board.

3.2 Modem to Azure Connection

The ME910C1 modem from Telit offers an extensive command set, and Azure IoT Central provides several methods of connecting to it, which meant there were multiple potential methods for connecting to IoT Central. The possible ways of connecting were with HTTPS, MQTT, or an application binary provided by Telit.

HTTPS HTTPS is a secure method of Internet communication that uses Secure Socket Layer (SSL) or Transport Layer Security (TLS) to send data securely. The data transmitted over HTTPS typically consists of a header and content [25]. The modem has a set of commands capable of sending HTTPS requests to a specific port on a server. To use HTTPS with Azure, the HTTP header must contain the symmetric key for that device. The key is over 100 characters long, and the modem does not allow custom headers to exceed 100 characters [26], which meant it was impossible to use the HTTPS commands the modem.

There is another way to send HTTPS requests with the modem, and it is achieved by manually opening up a port to communicate over. While the HTTPS command set abstracted away a lot of the setup, using sockets requires much more work to configure. However, with some work, they allow sending HTTPS data. The entire HTTP header must be created manually, but its length can be up to 1500 characters long [26].

MQTT MQTT is a lightweight IoT messaging protocol which works by having devices which publish "topics" to a "broker". The topics are a form of identification for the data being transmitted. Another device can then subscribe to a specific topic through the broker and it will receive any data published to the topic [27]. The packets sent using MQTT are very small, so the modem would be transmitting for less time, which would result in less

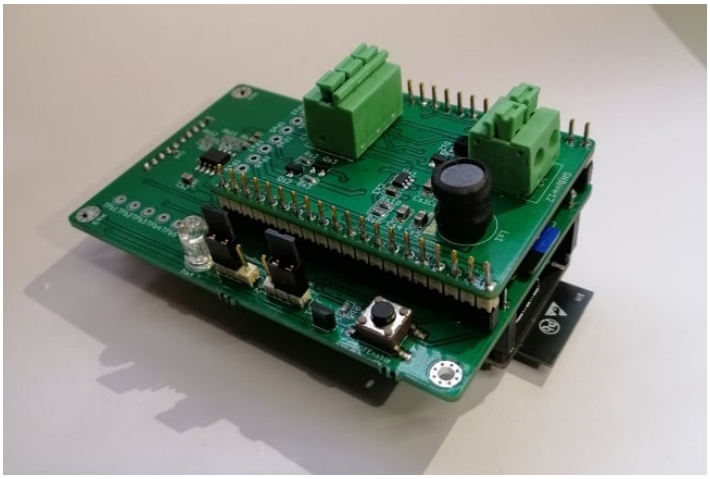


Figure 3.7: The base board with microcontroller, modem, and expansion board attached.

power consumption. The modem has commands which allow setting up a secure MQTT connection to a broker.

Application Binary Telit provides an application binary designed for connecting to Azure. The binary includes additional commands which make establishing a connection and sending data to IoT Central much easier. Due to the simplicity of this option, this was used for the implementation.

Unfortunately, due to the way IoT Central works, some of the information needed to connect to it can change without notice. The modem needs to request the most up-to-date information from Azure to account for this. This must be done using either MQTT or HTTPS.

3.3 PCB Design

Section 2.6 covered that the printed circuit board (PCB) has been designed to have two separate boards: a *base board* and an *expansion board*. The base board holds the main hardware required for the device, such as the modem and microcontroller. The expansion board contains all the circuitry for connecting to a battery and interfacing with SDI-12 sensors. The full board is shown in Figure 3.7. Both boards have a large ground plane, which helps with counteracting minor electromagnetic interference.

3.3.1 Base Board Design

Headers The base board needs to be able to have the modem, microcontroller, and the expansion board all connected to it, while also adhering to the size constraints set out in Section 2.1.2. The connections have been made possible by using female socket headers. The SDI-12 expansion boards' headers have been placed on the opposite side to the microcontroller and modem. The reasoning behind this was that the board could either have a set of headers on the same side as the microcontroller, which are tall enough that the expansion board does not touch the microcontroller, or it could be placed on the opposite side with average-sized headers. As the tall headers are less common and more expensive than the shorter variants, the expansion board was placed on the opposite side using the short headers.

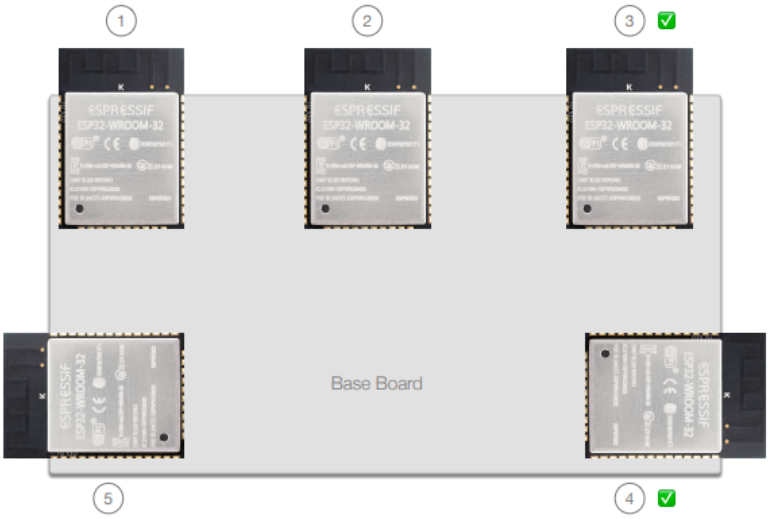


Figure 3.8: The possible positions to place the microcontroller, with the optimal positions being marked by a tick.

Source: [28]

Mounting The board has mounting holes on three of its corners. The fourth corner is occupied by the headers for the expansion board and microcontroller. It would be detrimental to the performance of the WiFi on the ESP32 if it were moved to another position, as shown in Figure 3.8, so it would not be possible to have four mounting holes. Regardless, having three holes is sufficient to hold the data logger in place in a custom housing that could be developed in the future.

User Interaction In some situations, the data logger may need users to interact with it. The parts that act as inputs and outputs for a general user are found on the side of the board and are shown in Figure 3.9. On the right is the button and hall effect sensor, which are used to enable the data logger's configure mode. The hall effect sensor is placed as close to the edge as possible to avoid potential electrical interference from other components and to make it easier to get a magnet close to it to activate it.

In the middle of the board, there are two sets of headers with jumpers on them. The right-most jumper is used to select between the connecting the microcontroller's analogue to digital converter (ADC) to the battery voltage divider output or the digital to analogue converter (DAC). The jumper should be moved to the DAC position when calibrating the device and in the voltage divider position during regular operation. The left-most jumper is used to select the voltage source for the ADC. It allows the user to choose between the power from the base board or the expansion board. As there is no need for a base board power supply due to a lack of rain gauge circuitry, the jumper does not need to be moved.

Finally, there is the LED, which is used to provide feedback to the user about the current state of the device. It is placed close to the components mentioned above so the user can do everything they need to without needing to adjust the board.

Servicing and Maintenance When it comes to troubleshooting the device, this has been made easier with test points. These small holes make it easy for voltages to be tested at specific points in the circuit. These are especially useful where the point would not usually be easily accessible, for example, when under a surface-mounted component. The base

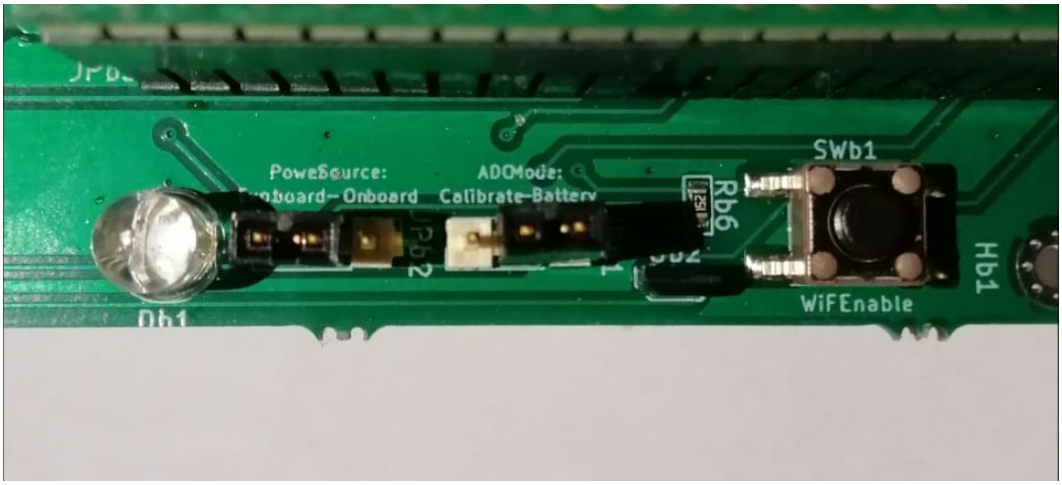


Figure 3.9: Area of base board designed for user interaction.

board has a total of five test points. These are used to check the voltage on the ADC, the state of the WiFi enabling button or hall effect sensor, the output of the modem 3.8 V regulator, the voltage of the power supply into the microcontroller, and ground.

There is also a position for another jumper near the modem. This jumper sits between the regulated 5 V supply from the expansion board and the 5 V power pin on the microcontroller. The jumper short circuits a resistor that is placed between these two points. The purpose of the resistor is to drop cancel out any voltage differences when both a USB and a 12 V supply are connected simultaneously. Having this mode is useful during development and debugging. When the device is not connected to a computer via USB, the jumper should be placed on the header to prevent any voltage from being dropped over the resistor.

SD Card The SD card holder is placed directly underneath the expansion board. This location was selected to be easy to access while also being protected from any accidental damage when moving the data logger.

3.3.2 Expansion Board Design

External Connections When not connected to anything, the expansion board is useless. In order to have any functionality, it must be connected to the base board, a 12 V power source, and at least one SDI-12 sensor. The connection to the base board is achieved with short male headers that can fit into the female sockets on the base board. The power and SDI-12 connection is made with push terminals. These are spring-loaded terminals that allow wires to be inserted when pushed down, and then they tightly clamp onto the wire when released.

Test Points The SDI-12 expansion board has lots of small, complex circuitry. To make it possible to troubleshoot any issues, the board has nine test points on it. These test points are connected to the UART transmit and receive lines, the SDI-12 data line, the SDI-12 power enable line, the reverse polarity protection circuit, and the voltage divider enabling circuit. There are also test points for the 12 V, 5 V, and ground lines.

3.4 Hardware Drivers

The main program can manage the entire system by stepping through all the tasks it needs to complete [29]; however, it does not know how to interface with the hardware. The purpose of the drivers is to allow the main program to interface with the hardware without knowing exactly how it works. The main parts of the system that the microcontroller needs to communicate with are the SDI-12 circuit, the SD card, the voltage monitoring circuit, the modem, and the diagnostic LED.

3.4.1 SDI-12 Driver

The SDI-12 driver simplifies communication with the SDI-12 hardware. The driver provides five public functions: an initialiser, two functions for turning the sensor on or off, a function to read data from the sensor, and a function to send any command to the sensor.

Initialiser The initialiser function sets up the connections to the external hardware and creates an asynchronous lock (which will be explained later). It creates a connection to the dedicated UART driver using MicroPython's UART library and makes connections to the "direction", "enable", and "force out" pins with the Pin library. The direction and force out pins were shown in Figure 3.2. The enable pin is used to connect power to the sensors using a switch from Section 3.1.1.

Turn On/Off These functions toggle the enable pin, which will either connect or disconnect the 12 V power to the sensors. The function to power the sensors also returns the system time when the power was connected. This is used to determine if a sensor has had enough time to boot up fully.

Read Data This function is used for taking measurements from the sensors. The function accepts information about the sensor it is going to communicate with. It begins by checking that enough time has passed since the turn-on function was called for the particular sensor being read or waits otherwise. To avoid blocking the rest of the system, it waits asynchronously, which means it surrenders CPU control to other processes before trying again. It then sends a measure command to the sensor, which tells the sensor to take a measurement. The sensor does not return the measured data; instead, it tells the driver how long to wait before requesting it. The driver asynchronously waits for this time before sending the command to request data. During this time, it enables the lock, which will queue up any other sensor readings to avoid bus contention, which would happen when the microcontroller tries to read from another sensor before the first one has responded.

After the time has passed, it sends commands to retrieve the data. If the sensor can take multiple readings, the driver may need to make several requests to receive all of the data. The driver will parse all the sensor data and then remove any data except what the user had requested when configuring the data logger. While it is reading from the sensors, the driver periodically gives up control to run other tasks. This function will return all of the data the user requested from a particular sensor.

Send Command The web app for the data loggers offers users the ability to send any command to the sensor. To achieve this, the driver has a function that can take any command string as an input. To verify that the command is valid, it checks that it begins with an address, which must be either an alphanumeric character or a question mark. The command

must also end with an exclamation mark, which is used as a termination character in SDI-12. This function does not verify the response as the function for reading data does. Instead, it will return the raw response, which can be displayed in the monitor on the web app.

3.4.2 Battery Driver

Lookup Table Setup

The first part of the battery system, which is directly related to the driver, is the provisioning step. This step allows the user to configure the lookup table (LUT) on the device and is done when setting up the device for the first time. It requires the user to disconnect the analogue to digital converter (ADC) from the battery and connect it to the DAC (through the use of a jumper). The provisioning step guides them through what they need to do through the command-line interface over a USB connection. It will ask them to record at least two voltages to calibrate the slope of the ADC. It will ask the user to enter the values of the resistors in the voltage divider. By default, these are set to the same values as mentioned in Section 3.1.4—15 000 kΩ and 2700 kΩ.

After this step, the microcontroller automatically generates the LUT by recording the ADC readings at a range of DAC outputs. As the DAC cannot output quite as many values as the ADC can take as an input, the missing values are found by interpolating the known values. The ESP32 does not have enough memory to generate the full LUT and store it in RAM before saving it to a file. Instead, each time it interpolates between a range, it adds those values to the LUT file.

The textual representation of the values could vary in length, depending on how many decimal places they have. This would mean that when trying to retrieve a value from the LUT, the driver would need to search through the entire file until it reaches the desired reading. This would have a time complexity of $\mathcal{O}(n)$. When $n = 4096$, these operations begin to take a noticeable amount of time on an ESP32. This issue can be countered by converting the numbers to their binary representation, which occupies four bytes. The time complexity is now $\mathcal{O}(1)$, as the battery driver can jump directly to the correct value.

Reading Voltages

The battery driver I wrote provides just two functions. There is an initialiser and a function to get the level of the battery.

Initialiser The initialiser function sets up the ADC, the DAC, and the pin to enable the battery voltage divider. By default, the ADC is set to a low resolution and only accepts very low voltage inputs. This function reconfigures it to have a 12-bit resolution and accept voltages up to 3.3 V. The enable pin is set to off by default as this disconnects the battery from the voltage divider, which reduces the current draw to almost zero.

Get Voltage This function has two main steps: taking an ADC reading and converting it to a voltage. The driver begins by setting the enable pin, which connects the battery to the ADC. It waits five milliseconds before taking a reading to give the capacitor near the ADC some time to rise to the battery. It then takes the average of 100 ADC readings rounds it to the nearest integer. The LUT file is opened, and the driver jumps to the correct position in the file to reads the next four bytes. For instance, if the ADC reads 2053, the driver will jump to the 8212th byte (2053×4) in the file and read the following bytes. It converts this binary value to a float, and this number is then converted to a voltage using the voltage and resistor configuration values the user entered during the setup stage.

3.4.3 SD Card Driver

MicroPython already has an existing SD card driver which allows the microcontroller to easily connect to an SD card and add it to the filesystem. Rather than reinventing the wheel, the SD card driver I developed expands on the existing driver's current functionality to provide functions for easily saving files directly to the SD card and checking if the SD card is present. The SD card is used for several things, including the lookup table, device configuration, data logging, and system logs. These are all stored on the SD card to allow the user to modify them on a computer if needed, except for the lookup table, which is stored on the SD card to avoid storing mutable data in the internal flash memory. The SD card driver developed for this project also has a function to request the remaining capacity, in bytes, of the SD card.

Data Logging When data is read from the sensors, or the battery's voltage is requested, the device sends this data over the air to be logged in Azure. As a form of redundancy, GWRC requested that the data also be logged to a CSV file on an SD card. This means that data can be recovered from the file later if any is dropped when being sent.

Whenever the microcontroller wants to log data, it can send it to a logging function on the SD card. This function accepts any number of readings and the current time in milliseconds since the beginning of the year 2000. The driver formats the time to an ISO 8601 compliant format[1]. Any data is saved to a new line in the main data logging CSV file. The same data is also saved to a daily logging file. The daily files exist for redundancy if the main file gets corrupted. If any of the CSV files do not already exist, they will be created with the first line containing column names.

System Logs All parts of the data logger developed by us for the project generate system log information describing what task is currently being done. For example, when a sensor reads data, the SDI-12 driver would log "Reading data from sensor at address 1". Usually, this data is logged to the terminal, so it can only be viewed if a computer is connected to it. However, we needed to show real-time log information to the user while also saving it to the SD card. To achieve this, the SD card driver was programmed to set up a log file when it is initialised. The logging output is then duplicated and printed both to the terminal and the logging file. All logging information has the time appended to it unless the current year is before 2020, indicating that the time has not been synced yet.

3.4.4 Modem Driver

The ME910C1 modem works by having commands sent to it over UART and performing the action corresponding with the command. The modem has an extensive command set—over 300 commands [26]—but the driver only needs to know the small subset of these required to establish wireless communication over the Narrowband Internet of Things (NB-IoT) network. The modem driver has functions to initialise it, power it, get the network time, and connect and send data to Azure IoT Central.

Initialiser and Power The initialiser sets up the hardware pins required to communicate with the modem. This configures one UART driver with a baud rate of 115200 bits per second and sets up the wake pin, which keeps the modem turned off by default. It also configures a lock. The lock works similarly to the one in the SDI-12 driver in that it prevents concurrent access to the UART data lines.

[1] Under ISO 8601, the date 31/12/2020 at 1:30pm would be expressed as *2020-12-31T13:30:00+12:00* [30]. The characters after and including the + represent the timezone offset in hours and minutes from UTC.

The modem regulator must be enabled before the modem can wake up. When the microcontroller calls one of the functions to turn the modem on or off, the power pin will disable the modem regulator, connecting or disconnecting power from the modem. When the modem needs to be turned off, the driver first sends a shutdown command to the modem. Once the modem has disconnected from the network and switched to sleep mode, it returns a `OK` response. Upon receiving this response, the regulator will be turned off.

Network Time To provide accurate timestamping of data recordings, the data logger needs to set the internal system time. The only way it can do this is through the modem. Fortunately, the modem's command set comes with a clock command, which returns the time provided by the network. The date and time returned are provided in UTC time, meaning it will be 12 hours (or 13 hours during daylight saving time) behind the current time in New Zealand. The command also returns the timezone offset and indicates if the timezone offset accounts for daylight savings time. Regular expressions are used to parse the time and extract all the relevant information. The UTC timestamp is converted to local time by adding the timezone offset minus any daylight savings offset. The driver returns the local date and time, ignoring daylight savings time.

Send Telemetry All the data collected by the data logger is sent to Azure IoT Central. To send telemetry to IoT Central, the driver needs to be able to first establish a connection to Azure's Device Provisioning Service (DPS). The data logger must connect to this every time before it can transmit data in case any security information is out of date. DPS returns a response with the most up-to-date information. With this information, the modem can attempt to establish a connection with IoT Central. Fortunately, there are applications available for the modems, provided by the manufacturer, which abstract some of the work away from sending data to Azure. The commands provided by the application were used, which immensely reduced the work done by the driver.

Unfortunately, the modem did not end up being completed to the point that it could send data reliably to IoT Central. This will be discussed more in-depth in the Section 4.6.

3.4.5 LED Driver

There needed to be a way to provide information to the user about the current status of the data logger. For example, it would be useful for them to know if the logger was currently on, as that would indicate that they need to wait for the device to safely shut itself down before they replace the battery or remove the SD card. We decided that an RGB LED was the best way to achieve this, as it provides several distinct colours that could be used to indicate various things. The driver provides an interface between the three pins on the LED and the microcontroller. It has several functions that allow the microcontroller to initialise the LED, turn it on or off, and set the colour.

Initialiser and Power The initialiser sets up the LED and defines an array of integer colour values. The initial colour is set to nothing, so the LED will not light up even when turned on. The LED's default state is off, so it does not draw current when not providing any meaningful diagnostics. When the LED is turned on, it will be set to whichever colour the microcontroller set. When it is turned off, it remembers the colour it was set to, so it does not need to be set again when turning it back on. The LED also has a toggle function, which inverts the current state of the LED.

Set Colour The LED driver contains two separate functions for setting the colour. One of them sets the colour with an RGB colour code, and the other uses a hexadecimal colour code. Unfortunately, the LED's full functionality has not yet been implemented, but only one of these functions will be used when it is. There are also functions to return the current colour in either RGB or hexadecimal.

Chapter 4

Evaluation

4.1 Power Consumption

The initial requirements for the power consumption set out at the beginning of the project required the device to draw less than 7.3 mA on average over a three- to six-month period. It is a certainty that the device will draw more than the average current when active, so it needs to draw much less current while the device is asleep.

4.1.1 SDI-12 Expansion Board

When the SDI-12 expansion board is powered, it is expected that it will draw some amount of quiescent current through inactive components. When powering the expansion board (without the base board), the only active components will be the regulator and the logic circuits for SDI-12 communication. The SDI-12 sensors will be unpowered by default. Due to the operation of buck converters (explained in Section 2.5), the current drawn from the battery is around 40 % the current drawn from the logic circuit.

I tested the current draw of the board at different voltages in the range of 11–13 V. The results are given in Table 4.1. The results show that regardless of the input voltage, the SDI-12 expansion board will draw more than the maximum allowable current while doing nothing.

It is not clear which component is drawing such high current, as the only components that could be drawing current have low quiescent current. However, it is clear that the selected regulator—a TPS562208—was not fit for purpose and may be contributing to the low power efficiency. The datasheet for the regulator provides graphs of the regulator efficiency against the output current. When the regulator operates at low currents, the efficiency can be less than 20 %, which means that there are plenty of internal losses. The TPS562201 is a related regulator which has very similar operating properties. The most crucial difference between this regulator and the TPS562208 is that it has consistently higher efficiency, at up to 80 % even at low currents. The difference between these two regulators is illustrated in Figure 4.1. By selecting this regulator instead, the current draw could potentially be reduced by a factor of at least three. Until then, the current draw means that it is unlikely that the

Voltage (V)	11.0	11.5	12.0	12.5	13.0
Current (mA)	12.50	12.70	13.05	13.22	13.39

Table 4.1: The current draw of the SDI-12 expansion board at different voltages. Measurements were recorded 5 seconds after powering.

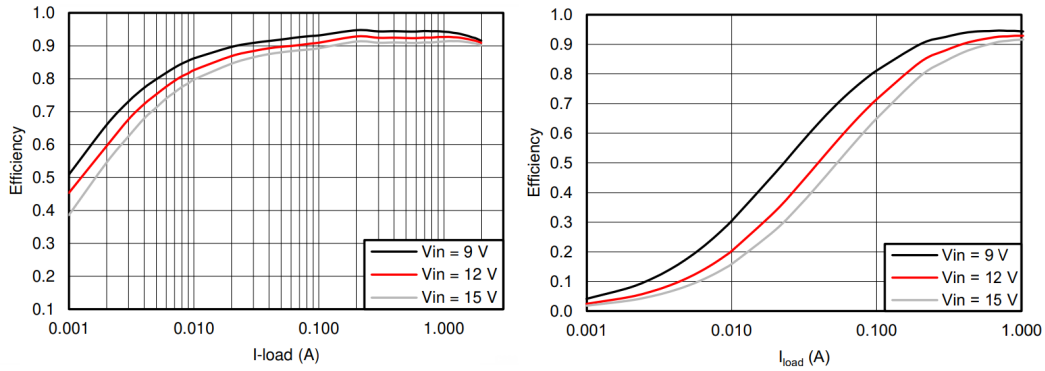


Figure 4.1: The efficiency of the TPS562201 (left) compared to the TPS562208 (right).

data logger will be able to be powered for much more than 1.5 months on a single 32 Ah battery.

4.1.2 Microcontroller

When the data logger is not taking measurements and transmitting data, it is in deep-sleep mode. The purpose of deep-sleep is to put the device into a very low-power state, which can have a current draw of as low as 150 µA [15]. This value represents the processor's current draw and does not account for any peripherals soldered to the ESP32 development kit. During testing, I found that the current draw during deep-sleep was 3.985 mA. This is significantly higher than expected, although it is still within the constraints for power consumption. After removing the on-board LED, the current dropped to 2.443 mA. If we wanted to reduce the microcontroller's current draw further, we could consider having the microprocessor (without the rest of the development kit) soldered directly to the board.

When the data logger is awake and sitting idle, the current draw is much higher than in deep-sleep. However, it is possible to alter the processor clock frequency to slow down the device but conserve power. This can be a bit of a trade-off, as it could be a fast device that completes tasks quickly while consuming a lot of current, or a slow one that must be powered for longer but has a lower current consumption. The ESP32 can operate at clock speeds of 20, 40, 80, 160, and 240 MHz [15], so the microcontroller's current draw was tested at these speeds. The results are shown in Figure 4.2. This shows that the worst frequency to operate at might be 80 MHz as it draws significantly more current than the other options.

The next step is to run a simple benchmark test to check the computation speed against the current draw to indicate which clock speed provides the best performance to current draw ratio. The benchmark test involved iterating through a for loop and recording the time taken to finish. As both 20 MHz and 40 MHz operate at very low speeds, they frequently run into unexpected issues. Therefore, I decided not to analyse them here. As shown in Figure 4.3, the time taken to process tasks is not linearly related to the current drawn. It would be best to select the 160 MHz clock speed as that has a low current draw without taking a long time to process. Unfortunately, this would not change the microcontroller's efficiency as this is the default clock speed.

4.2 Size

The size of the data logger needs to be constrained by dimensions of $150 \times 60 \times 60\,\text{mm}^3$. These dimensions include the entire data logger's size with the expansion board and micro-

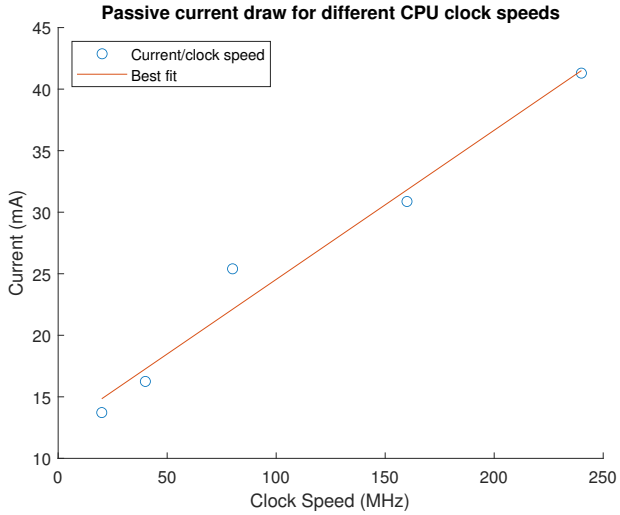


Figure 4.2: Plot of current draw against CPU clock frequency. The red line is the line of best fit, so any values below this line are more efficient than average.

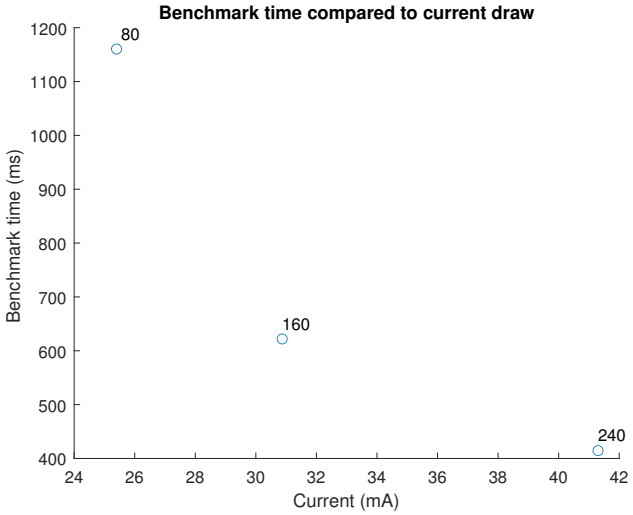


Figure 4.3: The current draw and time taken to complete a benchmark test for each clock speed (in MHz).

Part(s)	ESP32	Modem	PCB Assembly	Other Components	Total
Price (NZD)	18.18	73.00	53.30	48.20	144.68

Table 4.2: Breakdown of the cost of one device. The PCB Assembly also includes the manufacturing of the PCB and the cost of parts.

controller connected to the base board. The length and width of the data logger are fully dependent on the base board's dimensions, as this is the largest board. The board's dimensions are $50.8 \times 81.28\,\text{mm}^2$. The height of the board is 41 mm. Therefore, the board's overall dimensions are $81.28 \times 50.8 \times 41\,\text{mm}^3$, which is within the constraints specified.

4.3 Cost

GWRC expected the cost of one device to be between $200 and $300. Table 4.2 gives the cost of groups of components. Many of these prices will be subject to bulk discounts, so the actual device may be even cheaper. This cost does not account for the price of SD cards, a cellular network contract, batteries, or housing. Regardless, the cost is still within budget, meaning this is far more affordable than the equipment GWRC currently uses.

4.4 Client Feedback

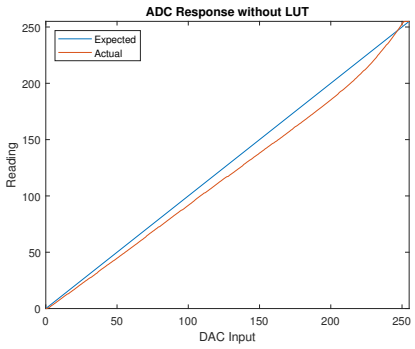
Ease of Use The client expressed that he was happy with the design of the device and that it was easy to access everything he needed to. The device's size was good, although he would have preferred if the board was slightly thinner. He said that setting up the lookup table was made easy with the command-line interface instructions. The client's main issue with the device was that the placement of the modem and the microcontroller made it difficult to plug a USB cable in the device for uploading code and debugging.

Ease of Manufacture The initial requirements of the project asked that the data loggers were designed to make it easy for the GWRC team to solder them. The reason was that they wanted to be able to fix the boards themselves if something breaks, and they were not aware that the board could also be assembled for a reasonable price during PCB manufacturing. However, after producing a near-fully assembled board within budget, the client stated that this was the better option.

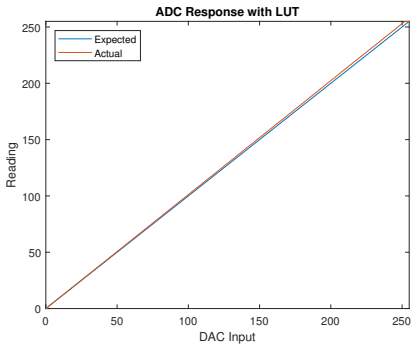
4.5 Voltage Monitoring Accuracy

GWRC needs to have accurate voltage measurements to know when they need to go to a site to replace a battery. A lookup table was implemented, which is used to counteract the non-linear characteristics of the ESP32's analogue to digital converter. The non-linearity of the ADC is shown again in Figure 4.4a. For the lookup table to be successful, we would expect to see the curve become linear with a gradient of 1; as in, it should track the DAC output. Figure 4.4b shows the new response of the ADC. This is a major improvement over the original ADC, and the time taken to convert to the corrected value is practically negligible.

In testing, I found that the voltage read by the device is typically slightly lower than the actual battery voltage. When at the normal voltage levels (around 12 V), the error is no more than 40 mV. Although this means that the battery monitoring is slightly incorrect, this can



(a) Original ADC response.



(b) Lookup table corrected ADC response.

Figure 4.4: Comparison of ADC response with and without a lookup table.

be beneficial as it encourages users to replace the battery slightly earlier. The client has also stated that an error of up to 150 mV would be acceptable too, so this error is allowable.

4.6 Modem Reliability

Many attempts were made to get the modems sending data to Azure properly. The first attempt was to send data using the Azure application binary provided by Telit. This binary worked great for sending data to IoT Central and was very reliable. The only issue with this application is that it does not perform the device provisioning step that is required before connecting to IoT Central. Instead, this was performed manually during testing. The provisioning step could be done using either MQTT or HTTPS. I first attempted to use MQTT as it is the most lightweight option, before moving on to implementing it with HTTPS, and then sockets. Unfortunately, none of these worked as expected.

Ultimately, we decided that it would not be possible to make progress on the modem. During the attempts to implement the modem, it failed to perform as specified in the documentation. When contacted about it, Telit support confirmed that some of the commands present in the documentation described functionality that has yet to be implemented on the device. As a final resort, the device was reworked to connect to a WiFi hotspot and use it as a bridge to send data to Azure.

4.7 Real Time Clock Accuracy

The real-time clock (RTC) in the ESP32 is controlled by an internal RC (resistor-capacitor) oscillator circuit [15]. RC oscillators are designed to keep track of time but can be prone to clock drift over long periods. A test was performed to verify that the internal RC oscillator would be appropriate for the device's timekeeping. This involved setting the RTC to the current time and then printing what the RTC *thinks* the current time is for eight hours.

The testing I performed was not extensive; however, it provides enough information to understand how much the RTC drifts. At the start, the RTC time was the same as the actual time, though, after the first 1.5 hours, the RTC had lost one full second. After 5 hours, the RTC time caught up, and the time was correct again. At 7 and 8 hours, the RTC had fallen behind by about half a second. This shows that the RTC does not just drift in one direction, so it is unlikely that it will be able to drift very far from the time that it is set to. In any case, the RTC is updated every time the device boots up, so there is little-to-no chance that the RTC will drift by more than a few hundred milliseconds under regular operation.

4.8 SDI-12 Reliability

The SDI-12 logic circuitry conforms to the guidelines set out in the SDI-12 specification [6]. This means that it should be able to interface correctly with any SDI-12 sensors, not just the ones used during development. After producing the data logger, the client was able to test it with additional sensors and confirmed that the device worked on all of them. Overall, four different sensors worked without requiring any extra work to get them set up, so it is safe to assume that the data logger will work with any SDI-12 sensor.

4.9 Voltage Protection

Reverse Polarity Protection The reverse polarity protection works as expected. It does not interfere with the input when the battery is connected correctly but will successfully cut power from the battery if connected in reverse. Additionally, when reverse polarity is applied, the LED illuminates itself to indicate to the user that the device is not being provided power.

Overvoltage Protection When tested independently, the overvoltage protection worked successfully and would start attenuating the output voltage past the cutoff point. These tests were performed with a large resistive load on the circuit's output, which meant that the small resistor placed in series with the load was negligible. However, the regulator can pass high amounts of current, which, under Ohm's law ($V = IR$), means that the regulator's equivalent resistance could be small. In this situation, the small resistor is now comparatively large, which means a significant voltage is dropped over it, leaving less voltage available for the rest of the device. The result of this is that the device is unable to power up.

This issue was fixed by removing the overvoltage protection entirely. Ultimately, the protection is not entirely required because the regulator accepts input voltages up to 17 V, and it is unlikely that the battery voltage will exceed this. The only other parts of the circuit are connected directly after the protection circuit are the battery monitoring circuit and the SDI-12 sensor power line, both of which are turned off by default, so the voltage will not ever be able to pass through. In future, the battery monitoring line could be used for software overvoltage protection, in which the user is alerted through the RGB LED when the input voltage is greater than 15–16 V.

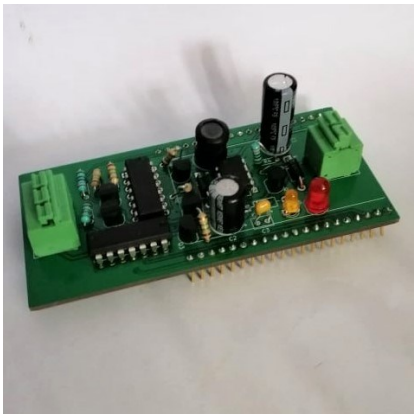
4.10 PCB Revisions

4.10.1 Version 1 PCB

The first versions of the base board and expansion board PCBs were produced in early August and were made up entirely of through-hole components, except for the SD card reader. These boards can be seen in Figure 4.5. These boards had several issues, which are briefly discussed below.

Size These boards were physically large due to the use of through-hole components, and the SDI-12 expansion board was longer than the base board. The board had no mounting holes, so it would have been difficult to mount it inside a case.

(a) First version of base board.



(b) First version of expansion board.

Figure 4.5: First version of the PCB.

Component Selection Many of the components that we needed were only available in surface-mount footprints, so in some cases, components that did not have all of the desired characteristics were used. For example, there are many high efficiency, surface mount buck converters available that met the project's requirements. However, it is much harder to find through-hole regulators with the same properties.

Compenent Placement The placement of some of the components was not optimal either. For example, some of the jumpers were placed too close to other components, making them near impossible to move without using tools. The RGB LED was positioned such that it was under the expansion board, making it more difficult to see.

Functionality Some of the circuits on the version 1 boards did not function properly. In particular, the overvoltage (refer to Section 4.9) and the battery monitoring circuits did not function. The battery monitoring circuit was non-functional because a MOSFET switch was placed with the wrong footprint. Both of these issues were fixed in the version 2 board.

Prodction Time By ordering a board that used only surface mount components, the manufacture of the PCB was faster and cheaper as we could order the components separately and solder them ourselves. Unfortunately, soldering each pair of boards took 2–3 hours, which meant it could take several days to prepare only ten of them.

4.10.2 Version 2 PCB

Shortly after evaluating the original PCB, I began work on designing the second version. This PCB began production in late August. This board tried to fix many of the first one's flaws, and using surface-mount components helped achieve this. The new board is shown in Figure 4.6.

Size These boards are much smaller than the version 1 boards, with the use of surface mount components being a key reason for this. The boards also have mounting holes to make it easier to mount it inside a case.

Figure 4.6: Second version of the PCB. These were purchased as a single board that could be snapped apart. No through-hole components have been soldered in this image.

Component Selection By using surface mounted components, practically all the parts needed were available. As we wanted the boards to be produced quickly, we could only use components available in the PCB manufacturer's part library. Therefore, some components, such as the headers and inductor, have to be purchased separately as through-hole components to be soldered on by GWRC.

Compenent Placement The placement of some of the components was greatly improved. The components are now more closely grouped based on which part of the circuit they make up. The jumpers and LED have been made more accessible by moving them closer to the board's edge.

Functionality The entire circuit correctly implements the required functionality. The only issues experienced thus far have been caused by user error. The only issue is that the board is not very power efficient, which means it will not last long in the field.

Prodction Time Although the production time was notably longer when the PCBs were being manufactured, the final things that needed to be soldered could be done within minutes. This means that the client needs to do less work to get a data logger set up.

Chapter 5

Conclusions

5.1 Conclusion

The goal of this project was to develop an end-to-end data logging system that is capable of reading data from SDI-12 sensors, saving it to an SD card, and transmitting it to Azure. The device is able to do this, however, due to difficulties with the modems, the device can only communicate with Azure via a WiFi hotspot.

The client gave us plenty of positive feedback on the device. Although it will be difficult to place these devices at normal monitoring sites, they are still usable and have proven that they can perform all the tasks that the Greater Wellington Regional Council wants them to do.

We have provided GWRC will some knowledge of how the data loggers work, which means there is the potential of these being further developed by GWRC or another regional council in the future to support proper wireless communication over a low-powered wide area network, without using the same modem as used in this project. This proof-of-concept design shows that the data loggers that GWRC wanted were possible to produce with almost all of the required functionality while also meeting their budgetary constraints.

5.2 Future Work

There is still some work that could be done to further improve the end product. The following sections explain what could still be done, and provides some insight into how these tasks could be achieved.

5.2.1 Set Up Modem

One of the most important parts of the system was the modem. Unfortunately, due to time constraints and issues with the modem that were out of our control, it was not possible to get the modems set up so they were communicating with Azure. From the testing that was performed, it seems that the issue was due to incomplete firmware on the modems which did not have all of the functionality that was needed. To develop further, it would be better to use one of the other modems that Vodafone currently supports. These are the Ublox Sara R410M or the Quectel BG96.

Alternatively, more research could be done into alternative ways of connecting to low-powered wide area networks in the Wellington region. One suggested path to look into is using Hologram SIM cards instead. These may allow a wider range of modems to be used, rather than being restricted to the ones that Vodafone approved. They allow devices

to communicate over Cat-M1—a protocol very similar to NB-IoT—and can connect to either Vodafone or Spark's networks, depending on which is strongest. The Hologram SIMs also provide a dashboard that shows the location of all the devices, which could be very useful for revisiting sites when replacing the battery.

5.2.2 Fix Current Draw

The device currently draws too much current for it to meet the requirement of staying powered by one battery for over three months becaus the device currently draws over 15 mA when idle. Although the exact cause of this is uncertain, it seems likely that it is due to the 5 V regulator on the SDI-12 expansion board and the peripherals soldered on to the ESP32 development kit. To fix this issue, the regulator should be changed to a TPS562201 and the ESP32 development kit could be replaced with the ESP32 chip soldered directly to the board. There may be other causes for the high current draw, but these would need to be investigated further.

5.2.3 Rain Gauge

One of the stretch goals for this project was to design the device to read pulses from a tipping bucket rain gauge. The data logger was designed with the rain gauge in mind, so none of the components will need to be changed. The only change required is redesigning the base board to have terminals to connect to the external rain gauges. The code for this needs to be written for the ESP32's ultra low power (ULP) coprocessor, which means it will all be written in assembly. The data logger should be able to read pulses from the rain gauge and store these in the ULP RAM until the devices wakes up. Once the device wakes, it will save the data to the SD card.

When designing the rain gauge device, it should also be designed as a standalone device, for when there is no need for SDI-12 sensors. It should be able to be powered by a lower voltage battery, such as a 3.7 V LiPo battery.

5.2.4 Housing

One of the initial requirements for the project, which was later dropped from the scope, was to design a housing for the data loggers. This housing needs to be water resistant while also being able to connect to external sensors.

5.2.5 Radar Sensors

The client expressed interest in developing a device that could interface with pulsed coherent radar sensors which could be used at some of the sites. This would be implemented as it's own expansion board, similar to how there is a separate SDI-12 expansion board. Being an expansion board, the device software would only need to be modified to support a different type of sensor, but the use of the modem and similar parts of the system will remain the same. The expansion boards are not stackable, so a data logger will only be able to support either the SDI-12 *or* the radar sensor expansion board.

5.2.6 Offline Device

Some of the sites that GWRC wants to monitor do not have sufficient network coverage to send data to the cloud. This means the device should just save data to an SD card. The biggest issue with this sort of device is having accurate timekeeping, which could only be set

when a user visits the site. Testing showed that the real time clock doesn't drift significantly, but over a three month period without being updated to the network time, it is inevitable that the clock will be off by a small but noticeable amount. A potential solution to this would be to use an external real time clock that can keep the time more accurately than the one built into the board.

Bibliography

[1] G. W. R. Council, "Environmental Monitoring and Research." http://graphs.gw.govt.nz/. Accessed: 2020-06-02.

[2] G. W. R. Council, "Environmental science." https://www.gw.govt.nz/environmental-science/, 2020. Accessed: 2020-06-02.

[3] J. Behrent and B. Secker, "Requirements." https://gitlab.ecs.vuw.ac.nz/QuiltyGroup/Research/ENGR489/Environmental-Monitoring-2020/engr489-project/-/blob/master/docs/requirements/requirements.md, May 2020. Accessed: 2020-05-30.

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

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

[6] SDI-12 Support Group, River Heights, Utah, *SDI-12 A Serial-Digital Interface Standard for Microprocessor-Based Sensors*, January 2019.

[7] HyQuest Solutions, *Vented Hydrostatic Pressure Sensor*, Oct. 2019.

[8] Tekbox, "TBS01A SDI-12/UART interface module." https://www.tekbox.com/product/TBS01A_Datasheet.pdf. Accessed: 2020-05-29.

[9] HyQuest, "iRIS 270 Wireless IP-Capable Datalogger."

[10] "Model TB3 Tipping Bucket Rain Gauge | HyQuest Solutions."

[11] "MicroPython pyboard v1.1 with headers."

[12] KEMET, "What are Aluminum Polymer Capacitors? – KEMET and Mouser Electronics," Oct. 2017.

[13] J. Falin and J. Cummings, "ESR, Stability, and the LDO Regulator," tech. rep., Texas Instruments Incorporated, Feb. 2020.

[14] Telit, *ME910C1 HW User Guide*, 13 ed., June 2020.

[15] Espressif Systems, *ESP32 Series Datasheet*, 2020.

[16] "What is a Reed Switch? | Madison Company."

[17] Honeywell, "Hall Effect Sensing and Application," tech. rep.

[18] "UART to SDI-12 Interface Slave Module TBS05A."

[19] AntonieValente, "SDI-12." https://forum.pycom.io/topic/1924/sdi-12/8, Dec. 2017. Accessed: 2020-05-29.

[20] Daycounter, "Sdi-12 bus interface." https://www.daycounter.com/Circuits/SDI-12/SDI-12-Interface.phtml, 2019. Accessed: 2020-05-29.

[21] Texas Instruments, *SN74LS00 Quadruple 2-Input Positive-NAND Gates*, May 2017.

[22] Texas Instruments, *SN74LVC2G241 Dual Buffer and Driver with 3-State Outputs*, Jan. 2019.

[23] josmunpav, "Inconsistent values when using "analogRead()" · Issue #92 · espressif/arduino-esp32," Dec. 2016.

[24] Elcap, "English: Capacitance ranges vs voltage ranges of different capacitor types," July 2012.

[25] E. Rescorla, "HTTP Over TLS," May 2020.

[26] Telit, *ME910C1 AT Commands Reference Guide*, 12 ed., May 2020.

[27] Oasis, *MQTT Version 3.1.1*, Oct. 2014.

[28] Espressif Systems, *ESP32 Hardware Design Guidelines*, 2020.

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

[30] "Date and time – Representations for information interchange," tech. rep., International Organization for Standardization, Feb. 2019.