

This exemplar report was submitted by a student and is not reflective of the official views of the University of Auckland. The content and opinions expressed are solely those of the author.

Practical Work Portal

Report Year 2024

- ★ Student Report Progress
- 📅 Student Hours Completed
- 👤 Student Assigned Markers
- 📄 Marker Payment Tracker
- 📁 Work Registration Expenses
- ⚙️ System Configuration

Employment at Bermondsey Electronics Ltd

[Back to progress report](#)

Report Submitted : 2024-03-20

PASSED

Assigned Marks : 25 / 25 (Grade : A)

Marker : [REDACTED]

Hours Verified : 432

Student : [REDACTED]

Specialisation : Software Engineering

Student Group : [REDACTED]

Uploaded Report

[REDACTED]

Scoring Rubric

Section One	Passed Section If you pass Section One you still need to pass Section Two.			Failed Section If you fail Section One you will be asked to resubmit the report.			5/5
Section Two	Passed Section If you pass Section Two you will still need to pass Section Three			Failed Section If you fail Section Two you will be asked to revise and resubmit the report.			5/5
Section Three Part A : Structure of the Report	Excellent	Above Average	Satisfactory	Room for Improvement	Below Average	No Marks	5/5
Section Three Part B : Quality of the Report	Excellent	Above Average	Satisfactory	Room for Improvement	Below Average	No Marks	5/5
Section Three Part C : Presentation of the Report.	Excellent	Above Average	Satisfactory	Room for Improvement	Below Average	No Marks	5/5

Existing Admin Verified hours

No additional time added.

Administrator Add Verified Hours

Administrator Details

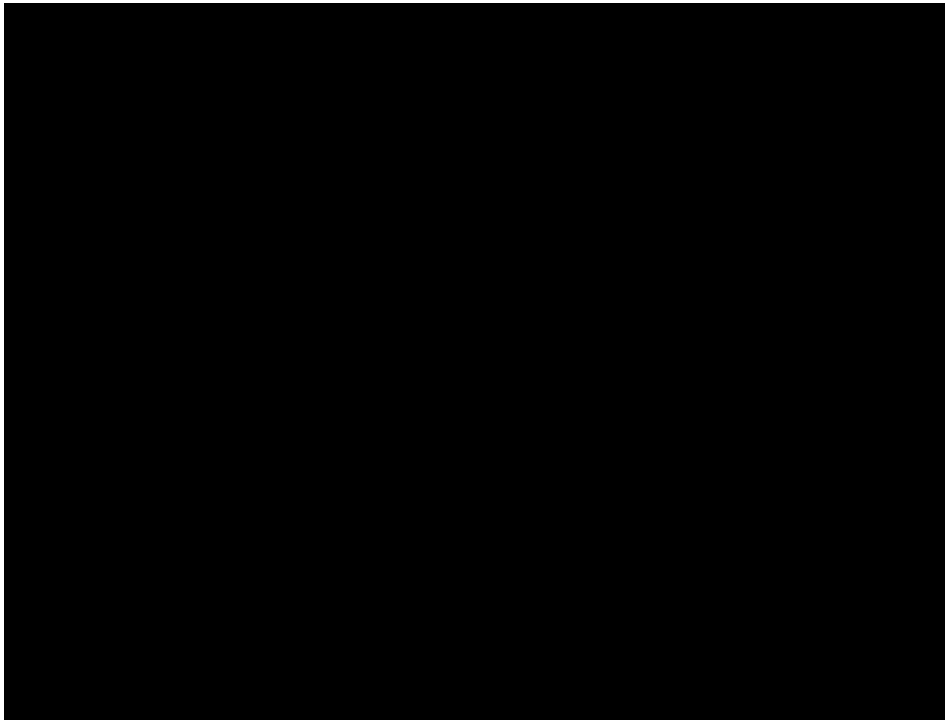
👤	[REDACTED]
☎️	[REDACTED]
✉️	[REDACTED]

Verified Hours

Hours Type



Practical Work Report



Software Engineering

Department of Electrical, Computer, and Software Engineering

Bermondsey Electronics Ltd.



United Kingdom

Dates of the Work Period

13th of November 2023 – 2nd of February 2024

Date of Report

19th of February 2024



Summary

Bermondsey Electronics Ltd. is a small, embedded software consulting company based in London, United Kingdom. Specialising in various sectors, including aerospace, automotive, and fitness, the company brings in customers from all over London, the United Kingdom, and the world. The company's primary focus is the design and verification of electronic systems (including software). Automated testing, predominantly through their in-house product BELIEVE, is a crucial factor in customer satisfaction.

I worked with [REDACTED] [REDACTED] and [REDACTED] [REDACTED] throughout my internship and completed six projects. From working with serial communication devices with RS232 and I²C to wireless communication devices with Bluetooth Low Energy, I developed a solid understanding of embedded software communication.

Apart from the software skills I gained, I also deepened my knowledge of both the processes involved in running a company and working with customers directly. Completing QMS activities was a significant element of team meetings. As it was the beginning of 2024 while I was there, the company also conducted its annual Year-In-Review. Critical processes have stemmed from these activities. For example, I was tasked with creating an Integration Test Plan at one point during the internship. The requirement to complete this document for new customers directly resulted from the Year-In-Review activities from previous years.

Living in London and working at Bermondsey Electronics for two and a half months was an incredible experience. I am immensely grateful to everyone who helped me realise this opportunity.



Acknowledgements

Bermondsey Electronics is a small, embedded firmware and software product specialist company run by [REDACTED] [REDACTED] [REDACTED] [REDACTED] was my manager.

I would like to thank [REDACTED] [REDACTED] for his trust in me. [REDACTED] sat directly opposite me in the office and was always willing to answer my questions – in great depth. He has taken on a number of interns over the years, allowing him to refine his stance on interns. [REDACTED] believes interns should do real work. That’s precisely what I got to do.

I would also like to extend my gratitude to [REDACTED] [REDACTED], who led me through my internship journey at Bermondsey Electronics. Always only a Microsoft Teams message away, [REDACTED] gave me the courage to learn a great deal throughout my internship. Whenever I found myself unable to progress, we would meet over Microsoft Teams, and he would give me hints so I could continue. Since I was working on real company projects, [REDACTED] often did not know the solution. He opened paths I could take to try and solve the problem myself, and I found this extremely valuable.

Near the end of my internship, [REDACTED] [REDACTED] was hired. I would like to acknowledge [REDACTED] for her friendliness and openness to working together. We ended up teaming up for my final project at Bermondsey Electronics and she generously offered me technical advice while I helped explain the company processes to her.

The internship at Bermondsey Electronics was such an amazing experience, and that would not have been the case without these amazing people. I can’t thank [REDACTED] [REDACTED], and [REDACTED] enough.



Table of Contents

SUMMARY..... 2

ACKNOWLEDGEMENTS..... 3

TABLE OF CONTENTS 4

TABLE OF FIGURES..... 4

1 INTRODUCTION 5

2 COMPANY INFORMATION 6

 2.1 OFFICE LAYOUT 6

 2.2 STAFF ORGANISATION STRUCTURE 6

 2.3 TECHNICAL FACILITIES AND AMENITIES 6

3 WORK EXPERIENCE 8

 3.1 LUTRON UVC LIGHT METER 8

 3.2 AMS UV SENSOR..... 11

 3.3 BLE OUTPUT BANDWIDTH..... 12

 3.4 DEVICE INFORMATION SERVICE DLL 14

 3.5 128-BIT UUIDS IN BELIEVE..... 14

 3.6 RFID STICK SCANNER FOR FARM ANIMALS 17

4 REFLECTIVE APPRAISAL..... 18

 4.1 IMPRESSIONS OF BERMONDSEY ELECTRONICS LTD. 18

 4.2 MY GROWTH AS A SOFTWARE ENGINEER..... 18

5 CONCLUSIONS 20

Table of Figures

FIGURE 1: BERMONDSEY ELECTRONICS LTD. LOGO. 5

FIGURE 2: UVC LED WITH HEAT SINK. 9

FIGURE 3: SENSOR PCB WITH I²C CONNECTIONS ON THE TOP, CONVERSION CHIP PCB WITH MICRO-USB ON THE BOTTOM. 11

FIGURE 4: THE SAME BANDWIDTH OUTPUT IN 3 UNITS, TAKEN FROM THE BELIEVE LOG CONSOLE. 14

FIGURE 5: SCREENSHOT OF BELIEVE LOG CONSOLE SHOWING THE SUCCESSFUL IMPLEMENTATION OF THE DIS DLL..... 14

FIGURE 6: PERIPHERAL BLE DEVICE TRANSMITTING MICROPHONE AND IMU DATA. 16



1 Introduction

Bermondsey Electronics Ltd. was established in 2017 by [REDACTED]. After years of being a PCB salesperson, selling to companies all around the United Kingdom, [REDACTED] decided to start his own embedded firmware and software company in the heart of Bermondsey, London. [REDACTED]'s company employs a small group of incredibly talented engineers with a wealth of experience. Potential customers reach out to Bermondsey Electronics with projects. While I was working at the internship, these projects ranged from medical device firmware to fitness machine software.



**Bermondsey
Electronics^{LTD}**

Figure 1: Bermondsey Electronics Ltd. logo.

This report covers the layout of the company office, the structure of the company, the facilities the company provides, and, of course, my work experience. Finally, my thoughts on the company, what I've learned, and how I've grown as a software engineer are discussed. Throughout the projects, I delved into software tools including but not limited to C#, JavaScript, Python, WinForms, Visual Studio, and Visual Studio Code. Hardware tools comprised of digital logic analysers, PCBs, dev kits, nRF BLE technology, power supplies, digital multimeters (both desktop and handheld), and more. As an Intern Software Developer, I thoroughly enjoyed my two and a half months at Bermondsey Electronics over the London winter.

2 Company Information

I noticed that a large part of Bermondsey Electronics was organisation. [REDACTED] [REDACTED] ensures everything, from hiring and firing to talking to potential customers, has a specific procedure. It is clear that the procedures are all implemented for a precise reason. [REDACTED] explained that almost all company processes have risen from one of two things – either by trial and error or by constantly looking for ways to improve efficiency.

2.1 Office Layout

[REDACTED]
[REDACTED]
[REDACTED]
[REDACTED]

The medium-sized office has large floor-to-ceiling windows on two of the four walls. It was, of course, winter in London over the New Zealand summer, which meant short and cloudy days most of the time. On occasions when the sun was shining, the windows gave plenty of natural light during daylight hours.

Starting from the door travelling clockwise around the room, there is:

- a soldering station,
- large racks for storing customer products and various company technologies,
- [REDACTED] [REDACTED]'s desk,
- my desk,
- a large storage unit of drawers containing smaller company technologies,
- [REDACTED] [REDACTED]'s desk,
- a spare desk,
- smaller drawers containing relevant items from/for specific customers.

2.2 Staff Organisation Structure

The company is very small, with only three employees. The employee structure is both rigorous and flexible. [REDACTED] [REDACTED] the founder and managing director of Bermondsey Electronics, sits at the top of the ladder. [REDACTED] [REDACTED], who started working at Bermondsey Electronics only a couple of months after the company was created, is just below [REDACTED] [REDACTED] works on projects of a similar calibre to [REDACTED]'s.

Near the completion of my internship, [REDACTED] and [REDACTED] hired another employee, [REDACTED] [REDACTED]. [REDACTED] lives in Wales, so she was working entirely remotely. As she was commencing her work at Bermondsey Electronics, she was delegated smaller tasks to introduce her to company policies and procedures, just as I had done 2 months earlier. I have every confidence that she will now be working at a similar level to [REDACTED] and [REDACTED].

2.3 Technical Facilities and Amenities

As the *Office Layout* section outlines, Bermondsey Electronics has one office space. The facilities like bathrooms, a kitchen, eateries, etc. are all located outside the office. The

space inside the office is designed for maximum productivity with minimal distractions, so the placement of the facilities external to the office is desirable.

The shared kitchen consists of a fridge where employees can store food and drinks, a microwave, a kettle, a sink, and some dishwashing equipment. Bermondsey Electronics supplied relevant utensils when necessary. Making coffee/tea during the day for mini breaks was encouraged. I don't tend to drink coffee or tea, so ██████ very kindly brought in some hot chocolate powder specially.

On rare occasions, everyone in the office would go out for lunch. This happened twice while I was working there. The first time was for the end-of-year Christmas lunch in Covent Garden. The second time was my last day at the internship, and we went to Borough Market. These were special lunches that I remember fondly. On a normal day, I would walk 10 minutes around the corner to the centre of Bermondsey and grab lunch from a selection of small eateries (e.g. the bakery, Turkish supermarket, or baguette café). I was also known to like my Tesco Meal Deals.

For technical facilities, the office had plenty of power sockets for the plethora of electronics that needed power. Everyone was given a large desk with a Herman Miller office chair for maximum comfort when sitting for long periods of time. We were given a monitor, USB hub, Windows PC, keyboard, and mouse. The keyboard was difficult to use to begin with because it turns out the UK keyboard layout is different from the US keyboard layout we use in New Zealand. Nevertheless, I stuck with it and can now use either keyboard.

██████ was happy to purchase any hardware or software that was either necessary or that would increase efficiency. Every Monday morning, we had a weekly meeting with all employees where this could be brought up. Also included in the meeting was a discussion on what everyone had been working on the previous week, some QMS (quality management system) tasks or a code review, a discussion on any new projects from new customers if appropriate, and deciding how much time we were going to spend on specific projects for the week ahead.

3 Work Experience

I accomplished a variety of software engineering tasks throughout the internship. I was paid for twelve weeks of work but took one week of holiday over Christmas. I was present at work for a total of eleven weeks. Each week was 40 hours, except for the week of the 1st of January 2024, which was 32 hours. This gave me a total of 432 hours.

██████████ was on leave for my first two days at Bermondsey Electronics. ██████████ organised my onboarding and inductions. My first day consisted of form-filling and reading through the company's background. This work was done remotely, as ██████████ often worked from home, and I had yet to gain personal access to the office.

For my second day at the internship, I went into the office and met ██████████ there. He explained important information specific to the office. This included the location of useful items, health and safety briefings, and how to lock up at the end of the day if I was the last person to leave. I was often the last person to leave the office, so this information was crucial. The Windows PC they offered me for the duration of the internship was corrupt, so I spent the entire day diagnosing, fixing, and eventually setting up the PC.

For the remainder of the internship, I worked on company projects for Bermondsey Electronics or their customers. These are comprehensively outlined in the sections below. Some projects were completed simultaneously, and some were completed with other projects in between. A lot of the projects are for customers of Bermondsey Electronics. For confidentiality, the customers' names will not be included. This will not interfere with the depth at which the projects are discussed and explained.

It is important to note that Bermondsey Electronics has an in-house product called BELleVE (Bermondsey Electronics Limited Integration Verification Engine). This application allows the set-up and operation of automatic tests between hardware and software. Single-threaded scripts are written in JavaScript, and through the use of custom DLLs, methods are called to run certain tasks on certain devices. The integration of DLLs ensures that the possibilities of testing in BELleVE are limitless. Although the BELleVE scripts are single-threaded, DLLs may be multithreaded. A DLL can be written to perform any task with any device.

3.1 Lutron UVC Light Meter

This project was originally going to be the entire project for the internship. One of Bermondsey Electronics' customers is creating a handheld device in which UVC light is emitted to treat corneal infections (infections at the eye's surface). For the internship to go ahead, I was required to find a UVC light meter that could measure the intensity of the UVC output from an LED. The UVC light meter had to be both accurate and accessible through a Windows PC. The accessibility requirement is usually achieved through the availability of public drivers. However, no such devices appeared to be actively available on the market. The chosen Lutron UVC light meter was accurate but had no public drivers. It did, however, have a 16-byte output stream that was continuously sent through an RS232 serial interface. This was enough to secure the internship, thankfully.

Firstly, I needed to be able to receive communication from the device. The RS232 output was a 16-byte data stream. To utilise BELleVE, I used C# and the `SerialPort` class. I created a simple GUI application using WinForms, which updates a text field when a new stream



of data is received from the light meter. This was a good starter project to familiarise myself with C#, Visual Studio, serial communication, and Bermondsey Electronics' git processes.

Once I was able to receive data from the light meter continuously, I was handed the ams UV Sensor to work with. I didn't have much of a chance to complete anything meaningful with it before we realised we didn't have any UVC LEDs to test the serial communication. The company performing the electronics engineering for the customer had access to the UVC LEDs, so we requested that one be sent to us. This didn't work out, unfortunately.

Due to the unavailability of UVC LEDs to test the light meter and light sensor, this project was put on hold, and I moved on to working on the project outlined in the *BLE Output Bandwidth* section, followed by the project outlined in the *Device Information Service DLL* section. After a few weeks, I returned to this project to write a DLL to interface with BELieVE.

The Lutron light meter continuously transmits 16-byte packets through the RS232 protocol. The light meter cannot receive data, so it is impossible to tell the light meter to send through a data packet. There are two ways to get the most up-to-date data from the light meter. The first is to do what was done previously – format the data transmitted from the device every time a new packet is sent and store the result without displaying it. The result is instant when the method to return the light meter output is called because we just read the latest value stored. The second method is to wait for a whole data packet to be received, format that one data packet only, and return the result. This is much more efficient as the computer only formats the incoming data packets when necessary. It is also much slower, however, as the whole process must be completed before the result of the light meter output can be returned when the method is called.

I chose to implement the second method mentioned above due to the drastic efficiency gains. Later, I would find out this was the wrong decision. I do believe implementing the solution the way I did the first time was good practice.

The DLL included error handling to warn testers if the light meter was not in the correct mode (it transmitted temperature sensor data if in the wrong mode on the device), if the device was not connected (timeout), and more. Sections of the 16-byte data stream helped identify these errors, so this wasn't too difficult to implement.

The UVC LEDs still hadn't shown up at this point, so I moved on to a new project outlined in the *128-bit UUIDs in BELieVE* section. After another few weeks had passed by, a UVC LED arrived, and I resumed work on this project. The LED gets hot very quickly when current is passed through it, so we attached it to a heat sink with a thermal sticker, as shown in *Figure 2*. The LED was connected to a power supply. A K-type thermocouple was pressed against the LED backboard, close to the LED. Connecting this to a desktop multimeter allowed temperature measurements to be pulled during LED operation in BELieVE.

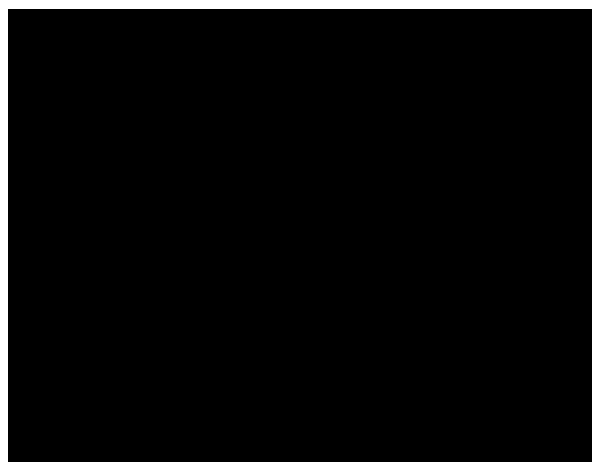


Figure 2: UVC LED with heat sink.



The power supply can be controlled using BELIEVE, so my next task was to create a script in BELIEVE to turn on the LED for a certain length of time, cut power to the LED if it exceeded 50 degrees Celsius, and measure the LED output intensity using the UVC DLL I had created. This is where the problem with timing arose. Since the JavaScript BELIEVE scripts are single-threaded, checking the temperature of the LED had to be done at a specified interval. There was no way to continuously check the temperature in a different thread and create an interrupt to abort on the main thread if the temperature was exceeded. The time it takes to check the temperature can vary, however, so it was impossible to power the LED for an exact length of time. Unfortunately, this problem cannot be solved entirely due to the constraints. To overcome the worst of the problem, the average time it takes to perform one temperature check was recorded. This changes slightly with every test run, but it was close enough to provide a rough constant.

More timing issues were also present, as I was using the second method to return the light meter output. The efficiency was improved, but it was impossible to measure the exact amount of time it would take to receive one 16-byte data packet, format it, and return the result. If the light meter was in the middle of sending a data packet when the method was called, the formatter would have to wait for the next data packet to be sent to start formatting. This changed drastically with each re-run, so a nifty constant wouldn't cut it. The only solution was to refactor the entire DLL to continuously update the current output value stored as new data packets arrived. It is important to note that while the BELIEVE scripts must be single-threaded, the C# DLLs may be multithreaded. This change ensured that the time between calling the method and returning the light meter output was negligible.

The final methods in the DLL perform the following processes:

- Connect to the Lutron UVC Light Meter.
- Disconnect from the Lutron UVC Light Meter.
- Get the light meter output at that instant.
- Get the mode of the light meter at that instant (i.e. UVC or temperature).
- Set the number of digits expected from the device (default is the maximum number of digits, so no data is lost accidentally).
- Check whether a timeout has occurred.


The next task in this project was to create a BELIEVE script to test the UVC LED intensity in a production line. This required many assumptions, such as the device containing the UVC LED having a custom-designed jig to hold the device and depress the trigger, and the UVC LED being activated for the duration of the trigger depression. I used pop-up windows in BELIEVE to inform the production line testers when to place the UVC device in calibration mode and to warn them if the UVC LED was on for too long.

A significant part of any software project is documentation. This project was no different. I had many comments throughout my code. [REDACTED] and [REDACTED] showed me how to write proper comments in the code reviews. I was originally explaining what lines of code did. Bermondsey Electronics wants comments that explain why you've implemented something a certain way instead. I also created a readme file for the DLL repository, which explained many nuances of the project in great depth.



3.2 ams UV Sensor

This project was similar to the one outlined in the *Lutron UVC Light Meter* section in the way that other projects were completed during this one. There were a few forced breaks when certain items didn't arrive or more pressing issues needed to be addressed.

The ams UV Sensor will be included in the customer's UVC device to measure the intensity of the UVC LED from within. During operation, the sensor should measure the intensity and stop the LED immediately if the intensity is above or below a certain threshold. My role was to create documentation for the sensor so that it will be easier for  to implement the logic later when that stage of the UVC device development begins.

Bermondsey Electronics also required a file with data from the Lutron UVC Light Meter and the ams UV Sensor to compare the results. Every LED (UVC or not) will have a slightly different ratio of output intensity to current, even for the same wavelength from the same manufacturer. The calibration of the UVC LED is designed to ensure the correct amount of current is being passed through the LED to achieve identical UVC intensities between all UVC devices. The calibration will be conducted using the Lutron UVC Light Meter at the factory. The device must capture that intensity using the ams UV Sensor when in the customer's hands. This is why the comparison between the two sensors is necessary.

I wrote two BELIEVE scripts and one Python script to achieve the previously stated goal of creating a CSV comparing the two sensors. The BELIEVE scripts were designed to test the LED under conditions similar to how they will be calibrated in the UVC device. The LED had power driven through it for 15 seconds at the lowest current accepted according to the datasheet. The LED would then have a 60-second cool-down period. The current would be increased slightly, and the cycle would continue until we reached the maximum current according to the datasheet. This was the same for the measurements with both the Lutron UVC Light Meter and the ams UV Sensor. The UVC DLL I had created much earlier in the internship greatly aided the BELIEVE script with the Lutron light meter. The ams sensor was slightly more complicated due to the unavailability of drivers for BELIEVE.

The ams UV sensor PCB (top PCB in *Figure 3*) has capabilities for measuring the intensity of light across the entire ultraviolet (UV) spectrum. This includes UVA, UVB, and UVC. The PCB also has a temperature sensor. A second PCB with a conversion chip (bottom PCB in *Figure 3*) is provided to connect the sensor PCB to a Windows PC. A ribbon cable attaches the two PCBs, and a Micro-USB port is supplied. This is great for experimenting with the sensor but not for utilising its capabilities from within the UVC device. The PCB with the sensors receives and transmits data through

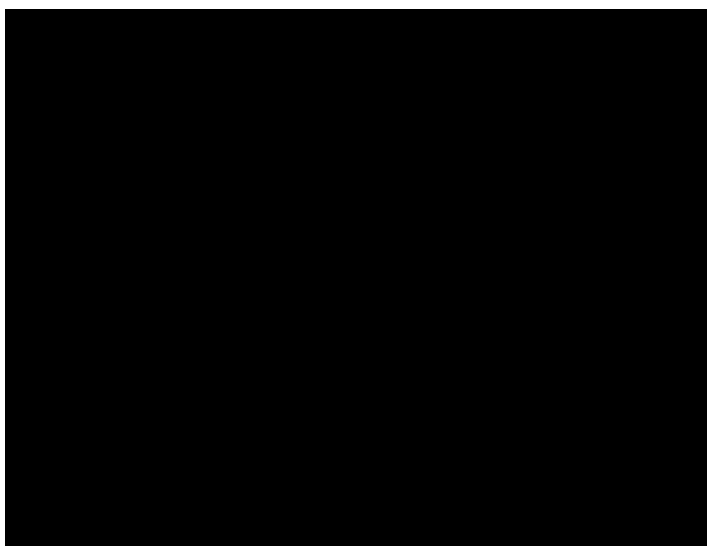


Figure 3: Sensor PCB with I²C connections on the top, conversion chip PCB with Micro-USB on the bottom.

the I²C protocol, so I recorded all packet transmissions using a DSLogic Plus logic analyser and the DSView desktop application.

The ams sensor came with a desktop application called EvalSW to extract and display the data from the UVA, UVB, and UVC sensors, as well as the onboard temperature sensor. This is transmitted through the Micro-USB connection on the second PCB described earlier. By starting data collection in DSView, starting the measurement in EvalSW, and starting the ams script in BELIEVE, DSView will record all the data the ams sensor receives from EvalSW for set-up, and all the data it transmits to EvalSW from detecting the UVC light from the LED managed by BELIEVE. The BELIEVE script generates a CSV file with the data, and DSView generates a CSV file with the ams sensor data packets.

The Python script was my final flourish at the internship (along with 20 pages of documentation to support it). I made sure it was packed with useful features to help later. The script takes in three CSV files. The first CSV file is the data from BELIEVE. The second CSV file is the data packets output from DSView. The third CSV file is a conversions table taken from the datasheet of the ams UV Sensor. It is possible to provide each file in the console during runtime or specify the paths beforehand for ease of use/repeatability.

The script starts by extracting the operations from the DSView output CSV file. The file is messy, and this section creates an intermediary CSV file with the data formatted nicely. Every byte that is received starts on a new line, so the script then collects the data into another intermediary CSV file where each line is one full data packet. We can then scan through each line in the latest CSV file and find when the intensity value is the greatest in each cycle of the LED being turned on. The script also extracts the set-up data packets to display the current measurement mode, clock frequency, integration time, etc., so can use the same set-up parameters further down the track.

When the script finds the greatest intensity value stored in the register for each cycle, it uses the conversions CSV file mentioned previously to convert the raw value into an actual light intensity with a unit of mW/cm^2 , the same as the Lutron UVC Light Meter. This line is added to the final output CSV of the Python script. The default behaviour is to then delete the intermediary CSV files permanently, but this can be changed very easily by commenting one line out within the script.

The script and the method to repeat this experiment were documented extremely thoroughly for Bermondsey Electronics. The 20-page document explains the equipment used, the Lutron UVC Light Meter test, the ams UV Sensor test, how to combine their results, and what the results may signify. Documentation is so important, and I am glad I will help in the future by having documented everything so thoroughly.

3.3 BLE Output Bandwidth

When the *Lutron UVC Light Meter* project and the *ams UV Sensor* project were put on hold, I started work obtaining the output bandwidth of certain characteristics broadcast by BLE devices. Firstly, I had to understand how BLE actually works. I spent a whole morning researching and completing mini quizzes online to gain an understanding of BLE.

BLE, or Bluetooth Low Energy, is a wireless communication protocol designed by Bluetooth SIG Inc. to transmit and receive small amounts of data between two devices with very little power usage. This is specifically designed to target devices where the preservation of battery life is critical. I had to learn new terminology as well. For example,

a characteristic stores the value of a certain property that will be transmitted through BLE, and a service is a collection of characteristics. Each service and characteristic follows the Attribute Protocol (ATT) format with a representative UUID. A list of all the assigned service UUIDs can be found [here](#), and a list of all the assigned characteristic UUIDs can be found [here](#).

The device I was working with was a rower machine. This has all sorts of services and characteristics. For example, it has a Fitness Machine service (UUID: 0x1826) with a Rower Data characteristic (UUID: 0x2AD1) and a Supported Resistance Level Range characteristic (UUID: 0x2AD6), among others. The Windows PC is the central device scanning for and receiving from the peripheral device. The rower machine is the peripheral device transmitting data to the central device.

The property of the characteristic defines the method in which the data transfer will occur. There are multiple ways in which a central device can read data from a peripheral device, but the most common are 'Notify' and 'Read'. The Rower Data characteristic contains the data for the current power, distance, time since exercise started, etc. and this is, of course, continuously updating. The property of the characteristic is set to 'Notify' so that the peripheral device notifies the central device when an update is sent over the air. This is the first method. The Supported Resistance Level Range characteristic contains one value set for that machine and does not change. The central device, therefore, does not need to know when this value changes because it will not change. The property of the characteristic is set to 'Read' so that the central device can fetch the value stored in the characteristic efficiently. This is the second method.

The bandwidth of a BLE characteristic only makes sense when the property is set to 'Notify', or the first way described above. This knowledge is what I needed to focus on calculating how many bits the central device received from the peripheral device, in how much time. This gives a unit of bits per second, also known as bandwidth. I created a script in BELIEVE to perform this calculation. The data from the rower machine comes through as a string of bytes. Every two characters is one byte (i.e. 0xFF is one byte). The time from when receiving notifications is started to when it is stopped is measured in milliseconds.

We can use dimensional analysis to work out the conversion equation to get bits per second from bytes per millisecond:

$$\frac{\text{bits}}{\text{second}} = \frac{\text{bytes}}{\text{millisecond}} * 8 \frac{\text{bits}}{\text{byte}} * 1000 \frac{\text{milliseconds}}{\text{second}}$$

This successfully reads the bandwidth from a notification characteristic. When working with 128-bit UUIDs, as outlined in the *128-bit UUIDs in BELIEVE* section, calculating the bandwidth was important. The bandwidth from this 128-bit characteristic was much greater than the fitness machine characteristic calculated previously. This required the units of bytes per second. This was an even simpler calculation:

$$\frac{\text{bytes}}{\text{second}} = \frac{\text{bytes}}{\text{millisecond}} * 1000 \frac{\text{milliseconds}}{\text{second}}$$

With these scripts ready in BELIEVE, this project was completed. The output in BELIEVE is shown in *Figure 4*. It was useful to include kilobytes per second due to the large number. *KB/s* is also a very common unit for bandwidth.

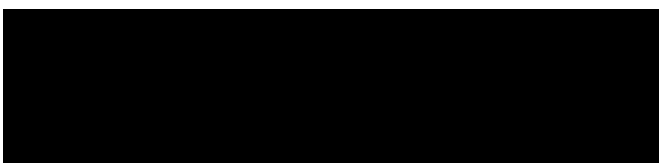


Figure 4: The same bandwidth output in 3 units, taken from the BELIEVE log console.

3.4 Device Information Service DLL

As discussed in the *BLE Output Bandwidth* section, services and characteristics form the underlying structure of BLE transmissions. This project was to create a DLL allowing a developer to write a script in BELIEVE to call methods that retrieve the characteristics within the Device Information service (UUID: 0x180A). The characteristics are:

- Manufacturer Name String (UUID: 0x2A29)
- Model Number String (UUID: 0x2A24)
- Serial Number String (UUID: 0x2A25)
- Hardware Revision String (UUID: 0x2A27)
- Firmware Revision String (UUID: 0x2A26)
- Software Revision String (UUID: 0x2A28)
- System ID (UUID: 0x2A23)
- IEEE 11073-20601 Regulatory Certification Data List (UUID: 0x2A2A)
- PnP ID (UUID: 0x2A50)

This project was a time-filler while waiting for a UVC LED to arrive. This basic grunt work was handy for the company but not terribly difficult. It had an underlying significance, however, allowing me to explore the fundamentals of Bluetooth wireless communication. The formation and implementation of DLLs in BELIEVE was also a significant learning objective achieved through this project. *Figure 5* shows that the value stored in each characteristic is what we expect (i.e. what was set on the peripheral BLE device).

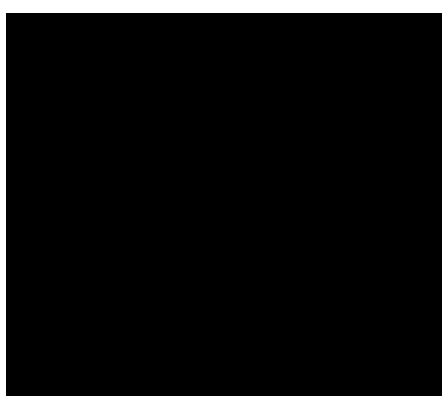


Figure 5: Screenshot of BELIEVE log console showing the successful implementation of the DIS DLL.

3.5 128-bit UUIDs in BELIEVE

BELIEVE has many features due to the ever-evolving DLLs that can be passed into it. As new features are needed, DLLs can be created or extended to cater to these requirements. The feature set initially included the ability to read characteristics with a 16-bit UUID. This was useful for the rower machine because the services and characteristics were standard across numerous rower machines. Another one of Bermondsey Electronics' customers was using a device that did not have standard services and characteristics. This called for the implementation of 128-bit UUIDs.

As stated in the *BLE Output Bandwidth* section, Bluetooth LE uses UUIDs to access services and characteristics. The standard UUID is 16 bits, but Bluetooth SIG Inc. must define these beforehand. These are expensive to purchase. The more common approach to defining custom services and characteristics is to use 128-bit UUIDs. These can be generated using an online tool such as uuidgenerator.net. The problem arises when

attempting to read from a characteristic with a 128-bit UUID in BELIEVE. Including this functionality within BELIEVE was desirable so that customers' custom services and characteristics can be automatically tested.

The BLE DLL uses the `pc-ble-driver-py` Python library to interface with an nRF53 dongle from Nordic Semiconductors. This allows the Windows PC to be a central BLE device. When BELIEVE is installed, a certain version of `pc-ble-driver-py` is installed from the online repository. Unfortunately, three of the included methods limit a certain feature that we require. To solve this issue, I patched these methods. The program uses the patched versions by overwriting those specific methods whenever they are called. This was well documented for future developers.

To extend the capabilities of the library, I needed to edit the wrapper (for us denoted as 'driver.py'). The first problem was that the UUIDs were integers. This worked well for 16-bit UUIDs, because integers can store up to 64 bits. This would not work for 128-bit UUIDs, however. I, therefore, had to convert all UUIDs to strings. The 16-bit UUIDs could be given as integers or strings to ensure backwards compatibility. The 128-bit UUIDs had to be given as type `string`. Python is rather ambiguous when object types are of importance, so I was forced to write new methods to handle the `string` UUIDs.

Error handling was another problem. There is a wrapper which handles the errors automatically, but this was only the case for the first call to a method in `pc-ble-driver-py`. Any subsequent calls to methods in the library would be ignored, as either `SUCCESS` or an error had been returned. To combat this, error handling was achieved within each new driver.py method itself. I also added new debugging tools for developers to make future improvements easier to implement.

Sometimes, customers want added security in their products to reduce the chances of being 'hacked'. This was a requirement for the device with 128-bit UUIDs for custom characteristics. After gaining a much deeper understanding of certain BLE processes, I implemented the function of connecting to a peripheral device with authentication. The device had the simplest form of authentication available to BLE devices, so there was no bonding, no Man in the Middle protection, no LE Secure Connection pairing, no Out of Bound data, and no keypress notifications. These options would be easy enough to change on our end, as these are just Boolean values. The library does the rest. The authentication process was a difficult challenge to overcome, so I was pleased when the Windows PC successfully connected to the peripheral device.

The most difficult part of the project was understanding how `pc-ble-driver-py` recognises 128-bit UUIDs. Generally, a different UUID would signify a different characteristic. In this library, however, this is not always the case. Every UUID has a base UUID. Please note that the following explanation will require me to swap between bits and bytes. A reminder that $128 \text{ bits} = 16 \text{ bytes}$ and $16 \text{ bits} = 2 \text{ bytes}$. The base UUID is the entire 16-byte/128-bit UUID, with the exception of bytes 2 and 3. The 16-byte/128-bit UUID `6258xxxx-231C-4311-A51B-C7EFCBF1BF6D` is an example of a base UUID, where `xxxx` denotes bytes 2 and 3. We are left with a 2-byte/16-bit ID (bytes 2 and 3, i.e. `xxxx`) to retrieve custom characteristics. This is great because we are now treating our custom services and characteristics like standard ones from Bluetooth. There is virtually no documentation on this subject from the developers of `pc-ble-driver-py`, so this was incredibly difficult to figure out.




Before I figured out this crucial information, Bermondsey Electronics was incrementing the last digit of the 128-bit UUID (i.e. the `D`), rather than the last digit in the first group of 8 digits in the 128-bit UUID (i.e. the last `x`). For every test script in BELLeVE, only one base UUID is permitted, and if multiple base UUIDs are provided, it will only use the first one. Although not our intention, we were effectively changing the base UUID for every characteristic but leaving the 16-bit ID the same. Therefore, the library was fetching the same characteristic every time, even though the 128-bit UUID was changing.

Thankfully, I realised this mistake in the format of our custom 128-bit UUIDs during the internship. The code in `driver.py` was rectified to split a 128-bit UUID into a base UUID and a 16-bit ID. Using these two values, `pc-ble-driver-py` successfully retrieved data from two different custom characteristics in the same test script. The only catch was that the UUIDs must follow the rules set out above (i.e., have the same base UUID).

Documentation, again, played a significant role in this project. I fashioned a readme file containing all the information for Bermondsey Electronics to set up and use custom UUIDs in BELLeVE in the future. In-line comments, header comments, XML documentation comments (C#), and Docstrings (Python) were all extensively included within the code base. This will aid the company in maintenance and development.

This was an exceptionally demanding but satisfying project to complete successfully. *Figure 6* shows the peripheral BLE device dev kit from which the Windows PC received data. The two custom characteristics were analogue microphones and an inertial measurement unit (IMU).

Through BELLeVE, I could then create a test script to extract the microphone and IMU data. The data are then exported to a CSV file primed for analysis. This is very useful for , who needed to automatically test the data from the microphone and IMU are credible and reliable.

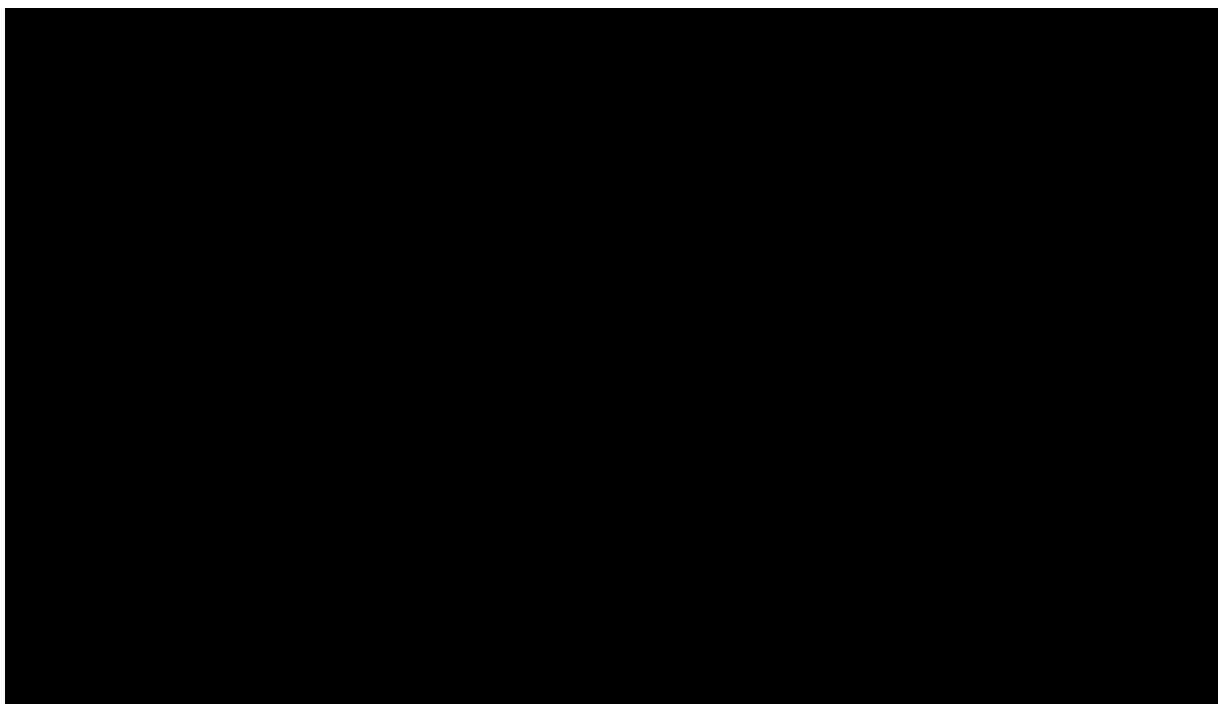


Figure 6: Peripheral BLE device transmitting microphone and IMU data.

3.6 RFID Stick Scanner for Farm Animals

A couple of weeks before the end of my internship, Bermondsey Electronics received a new customer. The job was to develop the firmware for a device that scans RFID tags attached to farm animals for tracking. Since this was near the end of the internship, my role was not programming. Instead, I was tasked with creating the Integration Test Plan. This is a crucial document that lists all expected outcomes to the customer from Bermondsey Electronics so no misunderstandings of requirements can arise.

The necessity of this document was made clear, as the company processes state that this is the one document which may save the company in case of a lawsuit for not completing what was initially discussed between Bermondsey Electronics and their customer. I aimed to be as detailed and thorough as possible so as to uphold the criticality of this document.

Firstly, I created an outline for the Integration Test Plan. This was a basic skeleton structure of the document to ensure I was heading in the right direction. After receiving feedback from █████, I proceeded to write the draft. This version was discussed with the customer for feedback, and I made the necessary additions and changes. The final version had 11 tests, each with sub-tests. Once every test achieves 100%, Bermondsey Electronics has finished its job with this customer.

██████ was using this document to design the product firmware as my internship was coming to an end. The Integration Test Plan helped ensure that the customer requires every component developed and that every component developed functions correctly.



4 Reflective Appraisal

4.1 Impressions of Bermondsey Electronics Ltd.

Bermondsey Electronics was a well-organised company, successfully completing many customer projects all year round. The small nature of the company is why customers choose to develop their products with Bermondsey Electronics, so growing the company is not the main goal. The goal is instead to make customers happier with the software solutions. A large portion of this can be accomplished through automatic testing, so the company has invested a lot of time and money into purchasing, using, and developing technologies to this effect.

The largest product the company has developed for itself (i.e. not for a customer) is the Bermondsey Electronics Limited Integration Verification Engine, or BELLeVE. This product allows Bermondsey Electronics to generate revenue without relying solely on selling their time-based services. BELLeVE is a very powerful tool, so I was working with this to complete many of my internship projects. Customers can be assured that the software products they are paying for have been thoroughly tested, making Bermondsey Electronics more reliable as a company.

It is also clear, however, that sometimes [REDACTED] takes on more than he and [REDACTED] can handle. This is evident in both [REDACTED]'s and [REDACTED]'s stress levels. Of course, I was able to help with this during my internship. The decision to hire [REDACTED] near the end of my internship likely arose from the realisation that more work needed to be done in less time, and my departure from the company wouldn't support that effort.

Overall, Bermondsey Electronics Ltd. is a high-functioning company that completes the work for its customers efficiently and effectively. A great productivity-focused company culture ensures customers are kept up-to-date during development and satisfied at project completion.

4.2 My Growth as a Software Engineer

Working with everyone in the company was an immensely fulfilling experience. As mentioned previously, Bermondsey Electronics is a very small company. Having only [REDACTED] and [REDACTED] for most of my time there, I learned a lot of specific knowledge from both. There were many tools I used throughout the internship that were completely new to me. Completing this internship after Part II and before Part III meant that significant software topics like databases, operating systems, machine learning, and networking had not been covered yet.

Serial communication was a large part of the internship. I had not previously explored this branch of software and electronics in detail, so there was a steep learning curve. Using RS232 and I²C communication protocols during the internship will greatly aid my future professional career, as understanding these technologies is very useful in a software engineering context.

Bluetooth Low Energy was also new to me. Understanding the fundamental principles of wireless communication is a significant objective. Hours of self-research led to the BLE project's successful completion. This skill will absolutely come in handy in the future as devices move closer and closer to purely relying on wireless communication technology.

One course that I did take in Part II was Software Quality Assurance. This included the theory and practice of testing software. Arguably, the largest component of the internship was testing. My previous knowledge was handy as a base-level foundation that could be built on top of during my time at Bermondsey Electronics. I delved much deeper into automated testing with both software and hardware. The large majority of automated testing was completed with BELLeVE due to its exceptionally flexible interface. I found creating drivers for certain hardware products to access them through BELLeVE to be particularly rewarding.

Of course, I acquired many other skills during the internship. The importance of teamwork and delegation were among some of the lessons I learned. As engineering students at the University of Auckland, we are taught to work in teams to a high level. In the workplace, this is brought to an even higher level. Everyone is responsible for certain parts of a project, and letting down the team is not taken lightly. The reputation of the company rides on everyone in the team, and this includes interns. If someone is struggling to understand or implement a certain element, this should be brought up in a meeting as soon as possible. Customers usually have strict time schedules, so it is crucial to ensure the output is what they expect, when they expect. This necessity is vastly different to the individual grades that teamwork influences within the context of university courses.

From this internship experience, I feel to have gained an immense wealth of knowledge. It has helped me to grow as a software engineer and as an individual. I could not expect anything more from an internship, so this was an exceedingly successful few months in London with Bermondsey Electronics.

5 Conclusions

Interning at Bermondsey Electronics Ltd. was an immensely fulfilling experience. I am very lucky to have had this opportunity, not only for the amazing internship experience but also for the ability to undertake the internship in London. London is an amazing city, and I had such a blast working, hanging out with family and new people, and, of course, exploring many of the city's tourist attractions.

The internship itself could have been only one project. I could have only been working on internal processes for the company and never touched a project for a customer. █████ and █████ could have never explained unfamiliar concepts to me. I am extremely fortunate that none of these occurred. I was able to learn so many software engineering skills that will directly or indirectly affect my future university studies, internships, and graduate roles.

Working with █████ █████, and eventually █████, was thoroughly enjoyable. Although █████ worked from home a lot of the time and I was never able to meet █████ in person, they were always only a Teams message or call away. █████ was fantastic at showing me how to set up and use certain equipment, explain difficult concepts to me, and uphold maximum productivity during working hours. All the employees are very highly educated and talented embedded software engineers, so it was a treat to discuss important software engineering topics with them in great depth.

Social events were great fun, with lunches, walks, and Friday evening drinks around London. Due to customer demands for strict schedules, social conversations were kept to a minimum during working hours. This is different from university, where there are always people to talk with in between lectures. I found it to be great for productivity. I achieved more at the internship during the day than I do at university some days, solely due to the absence of constant distractions. Fortunately, I enjoy being productive, so only having the occasional social event was a good fit for me.

Overall, Bermondsey Electronics Ltd. was an amazing place to work. The small nature of the company interweaved with the demanding projects undertaken, allowed me to expand my software engineering horizon further than I could have imagined. The two and a half months in London were some of the best two and half months of my life, and I couldn't ask for anything more.