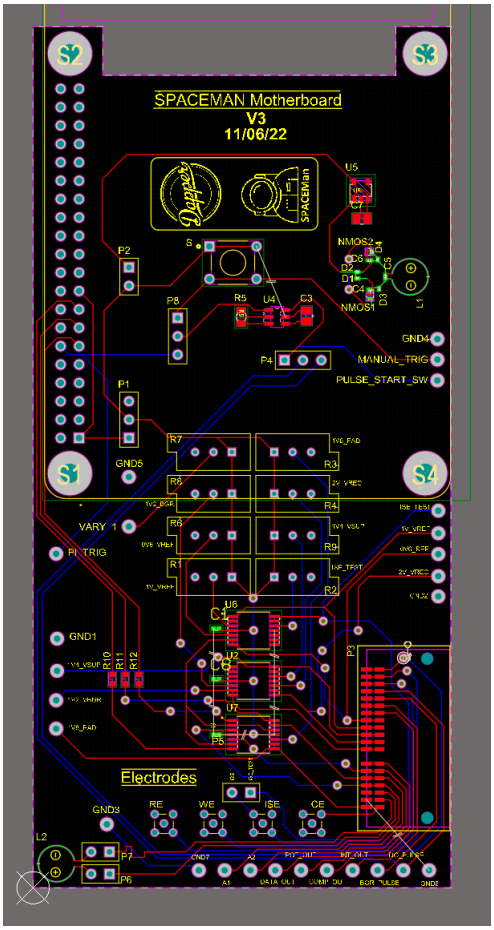
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2022



Project Title:	**Electrochemical Sensing Platform for SPACEMan**
Student:	**Yuting Xu**
CID:	**01518870**
Course:	**4T**
Project Supervisor:	**Professor Pantelis Georgiou**
Second Marker:	**Professor Christofer Toumazou**

Plagiarism Statement

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

Acknowledgements

Mr Daryl Ma for providing the context for this project. Without his work on SPACEMan, this project would not have existed. I would also like to thank his continuous guidance throughout the entire project.

Professor Pantelis Georgiou for his guidance and supervision.

Mr Lei Kuang for sharing insights into his publications on FPGA-based readout systems.

Mr Vic Boddy for helping out with the soldering of the SPACEMan motherboard and providing great technical advice.

Ms Isabella Breslin for her help with the testing with the wireless power module and consolidation of the communication protocol.

My family for their non-conditional support throughout the entire degree, especially in these challenging times.

My girlfriend **Jiayi Zhu** for the non-stop support.

Abstract

This paper presents an integrated platform for a wireless electrochemical SoC. The platform consists of a GUI running on a Raspberry Pi, a motherboard and a cartridge. The SoC is wire-bonded to the cartridge and plugged in to the motherboard, which is used to interface with the Raspberry Pi. This platform is capable of performing data acquisition using GPIO pins and displaying measured data. It also has IoT capabilities in the form of cloud data storage, and remote control.

Contents

List of Figures

List of Tables

Abbreviations

ADC - **Analogue**-to-**Digital** **C**onverter
AMP - **Amp**erometry
APP - **APP**lication
CBIT - **C**entre for **B**io-**I**nspired **T**echnology
CMOS - **C**omplementary **M**etal-**O**xide **S**emiconductor
DAC - **D**igital-to-**A**nalogue **C**onverter
DAPPER - **D**ual **A**mperometric and **P**otentiometric **P**ower **E**fficient instrumentation
DDR SDRAM - **D**ouble **D**ata **R**ate **S**ynchronous **D**ynamic **R**andom-**A**ccess **M**emory
DFF - **D** **f**lip-**f**lop
DMA - **D**irect **M**emory **A**ccess
FPGA - **F**ield-**P**rogrammable **G**ate **A**rray
FPS - **F**rames **P**er **S**econd
GPIO - **G**eneral-**P**urpose **I**nput/**O**utput
GUI - **G**raphical **U**ser **I**nterface
I2C - **I**nter-**I**ntegrated **C**ircuit
IC - **I**ntegrated **C**ircuit
IO - **I**nput/**O**utput
ISFET - **I**on-**s**ensitive **f**ield-**e**ffect **t**ransistors
IoT - **I**nternet of **T**hings
LDO - **L**ow-**D**ropout **R**egulator
LSK - **L**oad-**S**hift **K**eying
ISE - **I**on-**S**elective **E**lectrode
MCU - **M**icro **C**ontroller **U**nit
MUX - **Mu**ltiple**X**er
PC - **P**ersonal **C**omputer
PCIe - **P**eripheral **C**omponent **I**nterconnect **E**xpress
PoC - **P**oint-**o**f-**C**are
POT - **Pot**entiometry
SC - **S**witched-**C**apacitor
SPACEMan - **S**imultaneous **P**otentiometric and **A**mperometri**C** m**E**asure**M**ents
SoC - **S**ystem-**o**n-**C**hip
SPI - **S**erial **P**eripheral **I**nterface
UI - **U**ser **I**nterface

Chapter 1

Introduction

1.1 Project background

Point-of-care (PoC) healthcare has been rising in popularity in the last decades partly due to its ability to provide healthcare to regions that have limited healthcare capacity [9]. An emerging trend of PoC chemical sensing instrumentation is the use of ion-sensitive field-effect transistors (ISFETs), with numerous researches conducted to exploit the potential of such CMOS devices. ISFETs are pH-sensitive devices that can be used in ion imaging, DNA detection, vital physiological parameters measurement and etc. [11]. This however, does not mean that ISFET designs are without challenges; drift, induced noise, trapped charge, temperature variations and capacitive attenuation all have major impact on the performance of ISFET devices [10]. This calls for an integrative approach to scalable ISFET sensor development, where the implementation of a platform or framework can be as important as designing the array itself [11].

SPACEMan (**S**imultaneous **P**otentiometric and **A**mperometri**C** m**E**asure**M**ents) is a wirelessly powered analogue front-end IC that is capable of performing potentiometric and amperometric measurements at the same time. The output is modulated via load-shift-keying (LSK), enabling the transmission of data over the wireless power connection [1]. SPACEMan has several potential uses: connecting to an Iridium Oxide (IrOx) pH probe to measure saliva pH levels [12], integrated with an ISFET array to perform readout and etc. The aim of this project is to develop a platform ("the platform") with an intuitive interface to display data collected from the SPACEMan chip. The graphical user interface named SPACEMAN-GUI ("the GUI") will become a open-source repository that anyone can use to develop platforms for other LoCs.

1.2 Report structure

Firstly, Chapter. 2 introduces the previous work done at CBIT and compares their different approaches. Chapter. 3 goes into detail about the expected technical specifications. Chapter .4 focuses on the engineering/design decisions that had to be made during the development. Chapter. 5 then highlights the most important features that was implemented, and how challenging features were implemented. Chapter. 6 covers the tests done to evaluate the performance of said platform. Finally, Chapter. 7 and Chapter. 8 compares the final product with previous work and points out what can be improved in the future.

1.3 Overview of platform

The overview of the platform is shown in Fig. 1.1. The entire platform consists of these elements:

- **SPACEMan**: The aforementioned SPACEMan is the heart of the platform. It performs both amperometry/potentiometry and combines them both into a frequency output over the 433MHa power signal.
- **Raspberry Pi**: The Raspberry Pi connects to the motherboard and extracts frequency directly from the pins of SPACEMan. The full integration with the wireless power module is currently undergoing.
- **Wireless power module (Not available yet)**: The module will operate together with the platform proposed in this paper in the future. It provides power to SPACEMan via a 433MHz RF wave. It also demodulates the LSK encoded signal from SPACEMan.
- **Graphical User Interface (GUI)**: The graphical user interface is an intuitive way for users to monitor the state of SPACEMan. It also has in-built support for data display.
- **Server**: The server connects with the GUI and saves uploaded data into the database. Currently the server also runs on the Raspberry Pi.
- **Motherboard**: The motherboard is currently used to replace the role of the wireless power module, i.e. it powers and bridges SPACEMan with the Raspberry Pi.

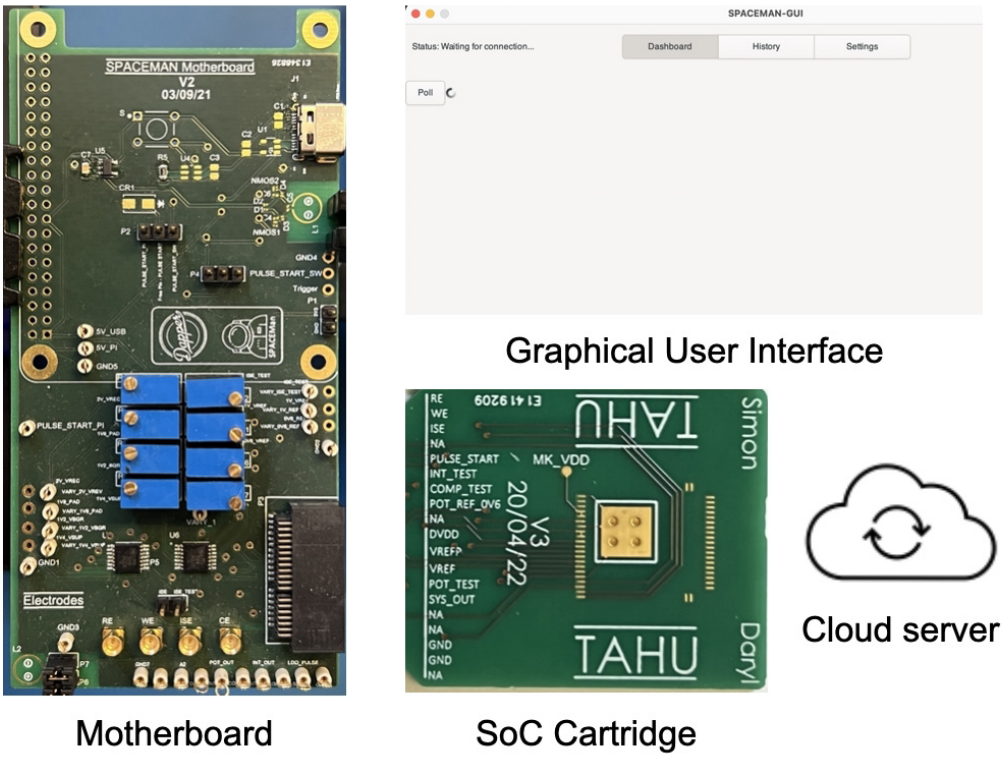


Figure 1.1: Overview of platform

Chapter 2

Background

2.1 Previous work

A number of previous work has been done at CBIT, Imperial College London to extract information from a Lab-On-Chip device by connecting to an external controller, whether that be a Raspberry Pi, a smartphone or a PC. The difference between the approaches are summarised in Table. 2.1.

The team of M. Cacho-Soblechero approached the problem by using a Raspberry Pi running on solar energy, drastically increasing the portability of such a device [3]. The Raspberry Pi is connected to a custom PCB mounted with a STM32 MCU. No details on the specific role of the Pi and the MCU were given. It is speculated that the Pi acts as the controller while the MCU acts as the peripheral. The MCU should be decoding the PWM-encoded signal and transmitting the data to the Pi for visualisation.

The solution of A. Au, et al was also to use an MCU (Microchip PIC24F) to collect and process data. However the processed data is sent to a smartphone via Bluetooth [4]. The MCU interfaces with the digital readout module via SPI. This consumes less power than a Raspberry Pi and allows for the use of batteries to power the device. Also, it is easier to develop a modern-looking GUI by using existing graphical frameworks for smartphones.

N. Miscourides, et al. implemented a trans-impedance-amplifier (TIA) to convert current mode readings to voltages which is then digitised using an analogue-to-digital converter. There is an on-chip SPI controller that interfaces with the motherboard, allowing configuration of output selection, readout cycles and etc. The motherboard is powered by what it would appear to be a Freescale FREEDOM MCU which is then connected to a PC via USB [5]. By leveraging the vast computing power of a PC, real-time data visualisation of DNA amplifications can be achieved.

In an effort to miniaturise the platform presented in [5], S. Karolcik, el al. used a Pi-based solution. A custom shield housing a high-speed ADC was connected to the Raspberry Pi via the GPIO headers. The encapsulated ISFET array shown in [5] was connected to the custom PCB via ribbon cables. Python was used to develop a GUI that controls the ADC and displays real time ion images of speeds up to 10 frames per second. The frame rate of the resulting Pi-based platform was even higher than that of the PC-based platform (7.8 fps) while being much more portable. This showcases the potential of using microprocessor systems like Raspberry Pi.

Lei Kuang and the team presented a novel FPGA-based readout module that can handle input data rates of up to 491.52Mbps. The output from the array is stored into a DDR3 SDRAM before streaming to a PC via PCIe. Most notably, image processing techniques were creatively used on the ISFET array data to reduce noise.

Lei Kuang, et al presented an improved FPGA-based digital readout module that can transmit data up to 762.94 Mbp/s [13]. The on-chip readout module is a 256-to-1 multiplexer (MUX)

that serially scans the 128 ADCs. Output of the MUX is first buffered then fed into the FPGA-based ring buffer which is connected to a PC using a USB 3.0 PHY transceiver. DDR4 SDRAM was used to increase buffering capacity of the ring buffer. The result is a respectable platform that can perform real-time ion-imaging at 6100fps.

Authors	GUI Platform	Data trans.[a]	Data encoding
M. Cacho-Soblechero, et al.[3]	Raspberry Pi	SPI [7]	PWM [7]
A. Au, et al.[4]	Android	SPI [8]	PWM [8]
N. Miscourides, et al.[5]	PC	SPI	PWM
S. Karolcik, et al. [6]	Raspberry Pi	SPI	PWM
Lei, et al.[14]	PC	GPIO	N/A
Lei, et al.[13]	PC	GPIO	N/A
This work	Raspberry Pi	N/A	LSK[1]

[a]Data transmission

Table 2.1: Comparison of different interfacing approaches

Chapter 3

Requirements Capture

SPACEMan poses a unique challenge that previous works did not cover: it is an analogue readout module. The output of SPACEMan is transmitted over the 433MHz power supply signal via Load-Shift Keying (LSK) and will need to be demodulated first to retrieve the original output signal. However, the demodulation is not in the scope of this project. It is assumed that such a demodulator exists and the demodulated signal is already made available to the platform.

Following the research done on prior work, the requirements for the platform can be formulated as follows:

3.1 Software requirements

- Intuitive graphical user interface (GUI).
- Ability to extract frequency from concurrent and single outputs.
- Ability to display measured data.
- Ability to monitor and change state of SPACEMan.
- Some form of IoT capability, e.g. cloud storage.

3.2 Hardware requirements

- A cartridge that SPACEMan can be wire-bonded to.
- A motherboard that can both power SPACEMan and access its pins.

Chapter 4

Analysis and Design

4.1 Hardware platform

The hardware platform is the foundation for the envisioned platform, therefore it must be chosen first. As mentioned above, the platform must be able to interface with the SPACEMan motherboard and read the demodulated LSK signals. It should preferably be able to power the motherboard by providing a 5V or 3.3V power supply.

4.1.1 Microcontroller

As demonstrated in [4], data acquisition can be done using a microcontroller. Microcontrollers often have access to a variety of analogue peripherals like ADCs, opamps. An MCU can easily read its GPIO pins at very high speeds while drawing little power. For example, an STM32L010R8 MCU can sample its GPIO pins with speeds up to 40MHz [33]. Graphical user interfaces can even be hosted on some microcontrollers using software like STM32's TouchGFXDesigner [34] or Qt [35]. The microcontroller option was not considered due to the prototype nature of the platform. Microcontroller-base solutions often require longer development time and is more difficult to debug. However, such a solution will be investigated in the near future.

4.1.2 FPGA

FPGAs can be seen as a more versatile approach than microcontrollers since they can be programmed to suit the specific needs of the application. The FPGA-approach presented in [13] and [14] are designed specifically for high-throughput applications. The output of SPACEMan is nowhere near the speed of these ISFET arrays, hence an FPGA-based solution is an unnecessary excess of performance.

4.1.3 PC

PC undoubtedly has the most powerful general-purpose performance with its desktop-grade CPU/GPUs that often source up to hundreds of watts. However, this is also the downfall of such a solution: using a PC would mean the test platform is completely stationary. This can be beneficial in the use case shown in [5] where the platform is connected to a stationary thermal cycler.

4.1.4 Raspberry Pi

The Raspberry Pi is essentially a Linux PC in a very small form-factor. Its main advantage is its powerful performance and complete IO ports for its size. It is a blend of a PC and a microcontroller, i.e. being moderately powerful and low-power at the same time. Most notably, a Raspberry Pi can be used to directly interface with the current SPACEMan motherboard via its versatile GPIO pins. Also, programming for the Pi is simpler than programming for a microcontroller. Therefore, the Pi was chosen to be the foundation of the platform.

4.2 Choice of GPIO library

The output of SPACEMan has a maximum frequency of 440kHz[1]. To sample a signal of 440kHz, the minimum required sampling frequency can be calculated using the Nyquist–Shannon sampling theorem:

$$f_s > 2f_{max}$$

$$f_s > 880kHz$$

Therefore, the minimum sampling rate required is 880kHz. This requires a fast library to sample the GPIO inputs. Table. 4.1 showcases the write speed of the libraries. It should be noted that the write speed was measured with a simple script toggling a single GPIO pin. The read performance when combined with a GUI will likely be a mere fraction of what was listed in the table.

Library	Language	Write speed
WiringPi	C	4.1MHz
BCM 2835	C	5.4MHz
PIGPIO	C	125MHz[a]
RPi.GPIO	Python	70 kHz
WiringPi	Python bindings	28kHz
`\proc\mem`	Shell	2.8kHz

[a]Using PWM peripheral

Table 4.1: Write speed of GPIO pins on Raspberry Pi 2[16]

As shown in Table. 4.1, only the C libraries are capable of successfully recovering the high-frequency GPIO inputs when used natively. The WiringPi library is no longer being supported by its author while PIGPIO is still being supported. Most notably, PIGPIO has the highest sampling rate of 1MHz, this is achieved by using direct memory access (DMA) [17]. PIGPIO can be run directly using C library or accessed indirectly as daemon (i.e. a process running in the background). The PIGPIO library was used to access the GPIO pins due to its flexibility and sampling rate.

4.3 Language options

Thanks to the added flexibility of the PIGPIO daemon, a number of programming languages can be used. Python and C are two of most popular choices for programming on the Raspberry Pi. A research has been done to determine the language of choice.

4.3.1 Python

Python is an interpreted language, where the code is interpreted and executed by another program: the 'Python interpreter'. This offers some notable advantages over compiled languages like C: dynamic typing, automatic garbage collection, platform independence and etc. Since Python is an open-source language, it is supported by a highly active community. There are a large number of available libraries that can be installed via its built-in package manager 'pip'. Hence Python has great support for rapid application development.

4.3.2 C

C on the other hand, is a compiled language, where the code is first compiled to a binary assembly file then directly executed by the CPU. Combined with the fact that C exposes many 'low-level' features to the programmer, a C program is usually much faster than Python and other programming languages, making C the de facto choice for embedded applications. C was chosen for its powerful performance and the author's familiarity with C/C++ in general.

4.4 Selecting the graphic framework

Since C is a relatively 'low-level' programming language that is rarely used on the front-end, the number of supported GUI libraries is relatively limited. There are three well-established GUI libraries that can be used with C/C++: GTK+, Qt and wxWidgets.

4.4.1 Qt

Qt is one of the most powerful C++ application frameworks. It offers a large number of libraries that cover most application needs. Qt applications can also be easily ported to run on mobile and embedded devices as it is capable of compiling programs to native binaries that use widgets native to any platform. However, setting up a Qt development can be difficult since a cross-compiling toolchain and additional graphics drivers are required. In addition, Qt requires the developer to purchase a commercial license if one chooses to statically link to the Qt libraries unless all the object files are provided.

4.4.2 wxWidgets

Unlike Qt, wxWidgets is a free C++ GUI library, allowing the user to freely use it for proprietary software. wxWidgets uses the native GUI toolkit of the underlying OS. This guarantees a GUI which is indistinguishable from native apps. The main downside of wxWidgets is the complicated API: some functions may work fine on a certain platform but not on another. It should also be noted that wxWidgets also requires a cross-compile tool-chain for development on the Raspberry Pi, further complicating the development process.

4.4.3 GTK

GTK is a native C library to Linux and is included in the most of the Linux distributions, making it easy to install and develop on a Raspberry Pi. It is widely considered as the most comprehensive GUI toolkit for C. GTK offers language bindings for C++, Python, Perl and etc. It is also a completely free software that can be used to develop commercial applications. GTK is backed by the GNOME project and constantly being used in the GNOME desktop development. Therefore, it is expected to be continuously supported by a highly active community of professional developers. However, it should be noted that GTK is rarely used outside of the GNOME desktop, limiting the number of external libraries developed to extend its capabilities.

Framework	Native to Linux	Requires commercial license
Qt	No	Yes
GTK+	Yes	No
wxWidgets	No	No

Table 4.2: Comparison of different GUI libraries

Since it is both difficult and time-consuming to evaluate the performance of each GUI library on Raspberry Pi, GTK+ was chosen to be used for the development of the GUI for its ease of installation. Specifically, GTK4, which is the latest release of GTK+ was used. It has been brought to attention that cross-platform GUI frameworks like Flutter may use C libraries via bindings; however, they were not considered at this stage due to performance concerns.

4.5 Designing the user-interface

The UI is an extremely important part of the app since it along with the functionality defines the user experience. The UI should be easy to navigate, provide useful information and exhibit intuitive interactions. To simplify navigation, the user-interface will ideally have three tabs. In addition, the UI must stay responsive when performing data acquisition or display a visual cue that operation is under way.

The prototype UI is shown in Fig. 4.1, Fig. 4.2 and Fig. 4.3 respectively.

4.5.1 Dashboard

The dashboard will be used to display a snapshot of the current measurement and device state. It should be both informative and concise at the same time. The recent frequency measurement will be converted to the units that the user specifies (e.g. mg/dl for glucose) and displayed in bold text. It will also have buttons for the user to initiate measurement and change the device state. A spinner will be used to indicate to the user that measurement is running in background.

Figure 4.1: Prototype dashboard tab

4.5.2 History

To prevent the dashboard from being cluttered by graphs, another tab name 'History' was added. The history tab will be used to display previous measurements. Another graph will be added to display the signal measured in the most recent polling session.

Figure 4.2: Prototype history tab

4.5.3 Settings

The settings tab is where the user can adjust the settings for the GUI and the device. It will also have buttons dedicated for debugging purposes. Appropriate widgets should be used to capture

user input. A pair of save/reset buttons will also be used to save/reset all of the settings to prevent misclicks.

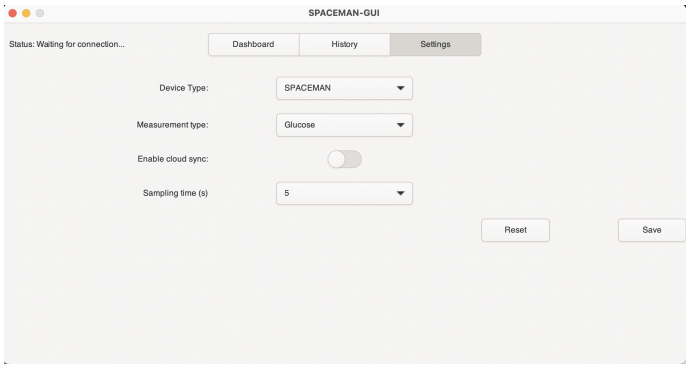


Figure 4.3: Prototype settings tab

4.6 Designing the motherboard

The schematic of the current motherboard is shown in Fig. 9.2 in the Appendix. It serves as the bridge between the SPACEMan (wire-bonded to a cartridge) and the Raspberry Pi. It features a number of potentiometers such that the reference voltages could be tuned to account for any PVT variations. During the development of the motherboard, the GUI was not made available yet. Therefore, an update for the motherboard was urgently needed.

4.6.1 Adding access to SPACEMan pins

Currently, the output pins of SPACEMan are only fanned out but not connected to the Pi. The Pi will need access to the following pins: DATA_OUT, A1, A2. A1 and A2 will be used to determine the current state of SPACEMan by using the state table shown in Table. 4.3. Following a test with TAHU (a chip variant of DAPPER), it is realised that the outputs from the LoC do not have enough driving power to provide a voltage larger than the threshold voltage of 1.2V when directly connected to a GPIO pin set to input. Therefore, the aforementioned pins should be fed into a opamp that boosts the voltages to 3.3V peak-to-peak.

Output	A1	A2
None	0	0
Potentiometry	0	1
Amperometry	1	0
Concurrent	1	1

Table 4.3: State table of SPACEMan. Taken from [1]

4.6.2 Power separation

The previous motherboard had multiple sources of power: antenna receiver, 5V USB and the 3.3V/5V supply from the Raspberry Pi. There is no load balancing between each voltage source. Therefore, in the common case where the voltage sources do not produce the exact voltages, only one source ends up supplying most of the current. The USB connector alongside the 5-3.3V LDO will be removed while the antenna section will be preserved to test functionality of the 433MHz RF antenna circuitry. A header will also be added to isolate the main 3.3V rail from the antenna LDO. Also, in the case that the current draw of SPACEMan and the opamps exceed the current sourcing capabilities of the 3.3V voltage source on the Pi, a header will be added to allow for external supplies.

4.6.3 Final design

Unfortunately, the board is still in production during the write-up of this report. So only the PCB design can be shown in Fig. 4.4. The updated schematic can be found in the Appendix. Multiple improvements were made over the previous design. Most notably, decoupling capacitors of 10nF were added to all three opamp ICs (lmv324lidt) to ensure correct operation [30]. The RF (radio frequency) traces also had their lengths matched. Current limiting resistors of 10kΩ were added to all GPIO pins set to input to limit the maximum current to less than 0.5mA per pin. The potentiometers and test points were rearranged such that the board was easy to route, aesthetically pleasing and easy to debug at the same time.

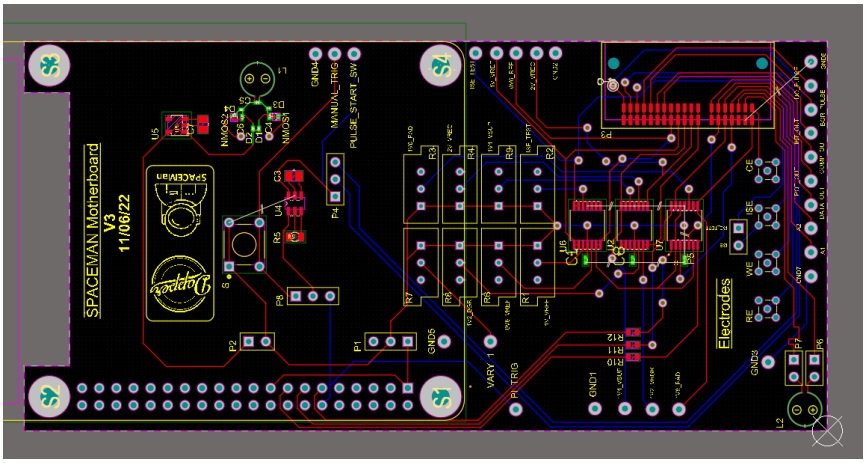


Figure 4.4: Final motherboard PCB layout

4.7 Updating the cartridge

Although the update of the cartridge for DAPPER is technically not in the scope of this project, it has been included since it increases the compatibility of the platform. The current revision of the motherboard is designed to interface with latest SPACEMan cartridges only. However, DAPPER being the predecessor of SPACEMan, shares many of the same pins with its successor.

The previous cartridge designed for DAPPER was shared between two chips. Therefore most of the DAPPER pins were connected to the bottom side of the connector while SPACEMan pins are mainly connected to the top side of the connector. The pinouts were matched to maximise similarity between cartridges. A test point was added in case MK_VDD was a power pin shared by both DAPPER and the other chip. The comparison between the updated cartridge and its predecessor is shown in Fig. 4.5.

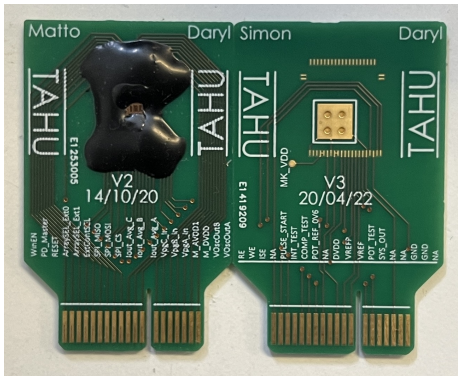


Figure 4.5: Comparison between old (left) and new (right) designs

4.8 Method of calibration

There are two key pathways when converting pH values to frequencies:

- pH to voltage/current
- voltage/current to frequencies

both of which might vary between each individual device due to PVT variations. Hence it is important to for the platform to have a calibration feature that takes these variations into account. There are two options of calibrating:

4.8.1 Direct injection

Both sensitivities can be measured directly by performing a voltage/current sweep. By combining the two sensitivities, we are able to accurately model the pH-frequency relationship. The rated input voltage/current range for SPACEMan can be found in the DAPPER paper to be 0.4V-1V and 250pA-5.6μA[2].

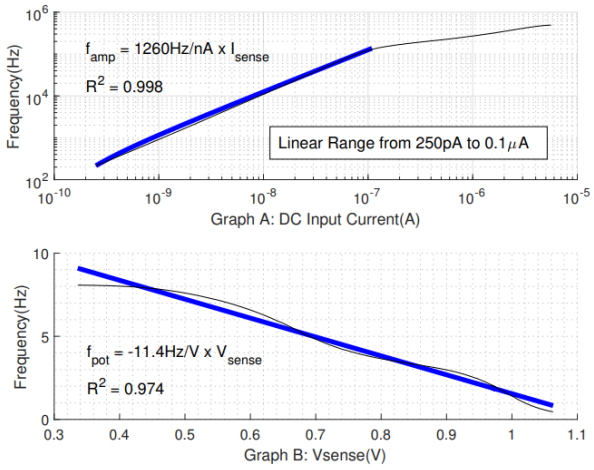


Figure 4.6: Input range of DAPPER/SPACEMan. Taken from [2]

Voltage injection

Voltage injection can be be easily performed by using a proposed circuitry shown in Fig. 4.7. The circuitry will consist of a DAC that can be controlled digitally with the Raspberry Pi through SPI/I2C interface protocols. The output from the DAC is passed into a voltage follower that drives the ISE. A current limit resistor is added before the ISE to prevent large currents from damaging SPACEMan. A suitable DAC would be MCP4921 from Microchip, capable of operating at 3.3V while having 12-bit resolution [36].

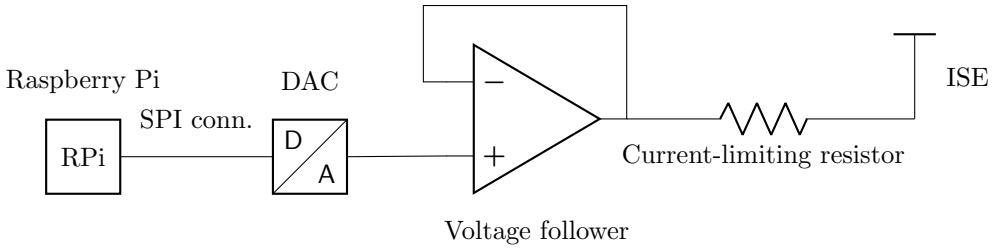


Figure 4.7: Proposed voltage sweep circuitry

Current injection

The current sweep is much harder to implement due to the large current input range. A possible approach would be to use switched-capacitor (SC) resistor to achieve programability of resistance. The V-I relationship of a typical SC resistor shown in Fig. 4.8 can be described by:

$$R = \frac{V}{I} = \frac{1}{C_s f}$$

Where C_s is the capacitance of the capacitor and f is the frequency of switching. The switches are replaced by transmission gate switches in actual circuits.

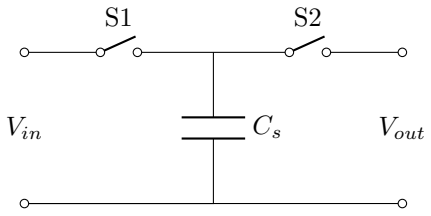


Figure 4.8: Typical switched-capacitor resistor

A constant voltage will be applied across the SC resistor while the resistance is changed by varying the switching frequency. The output current will be first copied by a current mirror then injected into the working electrode of SPACEMan. Assuming the voltage drop across the equivalent resistance is 3.3V and the capacitance used is 10pF, the frequency range required for an output of 250pA-5.6μA can be calculated as follows:

$$250 * 10^{-12} <= 3.3 * 10 * 10^{-12} Nf <= 5.6 * 10^{-6}$$

$$7.6Hz <= Nf <= 0.17MHz$$

where N is the ratio of the current mirror. The frequency range can easily be handled by a microcontroller or even a Raspberry Pi. However, it should be noted that parasitic capacitances of the MOSFETs may affect the final equivalent resistance and thorough testing is required for correct operation.

4.8.2 Using solution with known concentration

Calibration with solutions of known concentrations are widely used in the medical industry. This method allows personnel without electrical/medical background to easily perform calibration. It also has the added benefit of not needing additional hardware blocks. However, the cost of solutions and their storage will need to be taken into account. Many calibrators used by commercial biochemistry analysers need to be stored in a refrigerator or freezer [24].

This method is preferred over direct injection due to its simplicity. pH buffer solutions in the range of 4 to 9 are mostly stable under room temperature and can be obtained at a relatively low cost [25].

4.9 Server design

The platform should have some IoT capabilities in the form of uploading/downloading data from a server such that data could be viewed from different platforms. Server can also be used as a medium between a mobile app and the Raspberry Pi, allowing the user to perform remote measurements.

4.9.1 Choice of transmission protocol

There are two popular transport protocols that can be used: Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Table. 4.4 compares some features of the two protocols.

Protocol	Direction	Packet order	Acknowledgements
TCP	Bidirectional	Guaranteed	Yes
UDP	Unidirectional	No	No

Table 4.4: Feature comparison between TCP and UDP. Taken from [26]

TCP is a robust connection-based protocol offering a reliable connection as it ensures the data to be in the correct byte-order at the recipient. It also tried to resend packets if they were lost. UDP is much more lightweight, therefore, it is often used in performance-oriented scenarios (e.g. gaming and video streaming) where losing a few frames/packets will likely not be perceived by the user.

TCP will be the transport protocol used in the development of SPACEMan-GUI since performance is not as important as data integrity in this particular use case.

4.9.2 Messaging format

A widely used messaging protocol for client-server communication is Hypertext Transfer Protocol (HTTP). It is an application layer protocol that is higher than the previously mentioned transmission protocols (TCP/UDP) in the hierarchical network model. It consists of two types of messages: requests and responses [29]. A HTTP message usually consists of status codes, a header, the body and etc.

JSON (JavaScript Object Notation) is a popular data interchange format. It has several key advantages: human-readable, easy to parse and light-weight [28]. JSON messages can be constructed and serialised in C by using json-c [27]. Instead of packing JSON into the body of an HTTP message, it will be directly sent as a string to the server. Firstly, constructing HTTP messages in C is difficult (without external libraries). Secondly, the platform currently does not benefit from the powerful functionality the HTTP protocol offers.

Chapter 5

Implementation

5.1 Model-based application

Certain functions of the GUI require external data that is out-of-scope. For example, the polling callback attached to a button will need the settings to correctly set the measurement time; the text showing the measured value will need to be modified at the end of the polling callback, etc. A easy fix would be to declare all the widgets as global variables and use them directly in the callbacks. However, this generally does more bad than good. It causes readability issues as it is unclear which function accesses and modifies the global variables. Also, for global variables used across different source files, it becomes difficult keeping track of the original declaration.

Heavily inspired by the Model-View-Controller (MVC) architectural pattern used for front-end development, a similar model that is passed between callbacks was implemented. Fig. 5.1 showcases such model. Definition of the model is shown in Listing. 3 in the Appendix. The model contains pointers to all the widgets that may be accessed during a callback, current state of the connected LoC, current user settings and the GPIO samples collected. It can be said that this exposes too much information to an individual callback/feature, e.g. the login dialog might not need access to the drawing area in a separate tab. However, it saves the effort of creating separate data structures (struct) for every single callback. Since it is a single struct, it is more human-readable than multiple structs stored in every source file.

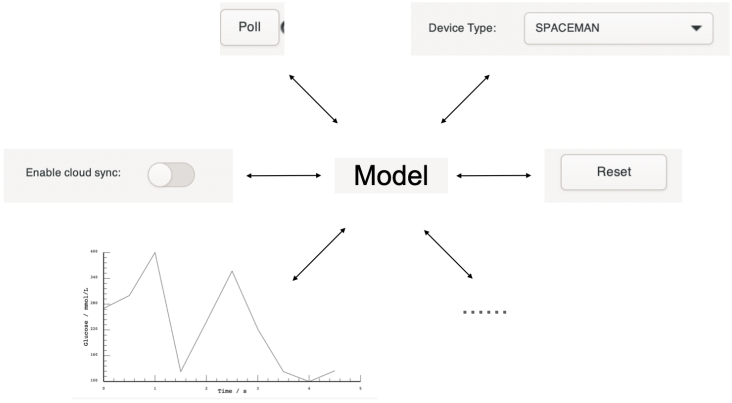


Figure 5.1: Model and widgets

The model ensures all callbacks are operating on the state. However, it should be noted that a centralised model is not thread-safe and neccessary precautions need to be taken to guarantee thread safety. In the current implementation, calls are designed to not access the same variables concurrently.

5.2 Frequency extraction

5.2.1 Data Acquisition

The output of a signal generator was directly connected to the GPIO pins on the Pi. GPIO pins 23 and 24 was used. The data acquisition was handled using the PIGPIO library [17]. GPIO state changes on all pins are captured by attaching a callback that is called every millisecond. The callback function first applies a bit mask to extract the pin of interest, then stores only the positive edges into an array. The array is later used to draw signal plots and to determine the frequency.

```
positiveEdge = ((lastLevel ^ level) & (1<<model->GPIOPinNumber)) & level;
```

Listing 1: Selecting only positive edges

5.2.2 Initial design

The first version of the frequency extraction feature was based on the frequency counter example provided by the PIGPIO library [17]. This is a relatively simple implementation that counts the number of rising edges within a set time window. Table. 5.1 demonstrates its accuracy in POT (1-10Hz) and AMP (400-400kHz) frequency ranges.

Input freq. (Hz)	Measured freq. (Hz)	Measure time (s)	Error
1	0.54	5	46%
1	0.8	10	20%
10	9.64	5	3.6%
350k	334.4k	5	4.5%
400k	365.7k	5	8.6%
440k	389.1k	5	11.6%

Table 5.1: Frequency counter results (averaged over 5 trials)

It can be seen from the table that the accuracy suffers greatly in lower frequencies especially when the measurement time is short. This seems to be a bug with the PIGPIO library. In any given interval, the last two or three positive edges will often not be detected as shown in Fig. 5.2. Another noticeable disadvantage of this method is that it can not be used to differentiate between concurrent outputs.

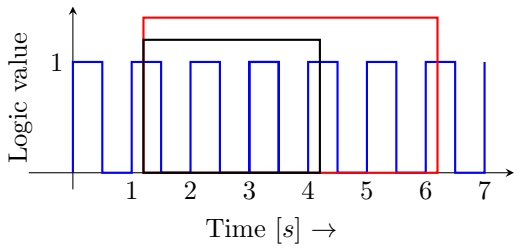


Figure 5.2: Mismatch between actual (red) and measured (black) window

According to the Nyquist-sampling theorem, an 1us sampling period should be able to perfectly reconstruct a signal below 440kHz. However, the measured frequency does not scale linearly with the actual frequency starting from 350kHz. This phenomenon is described by the red scatter points in Fig. 5.3. This is likely due to the fact that there are too many overheads: the GUI application is running on the 64-bit Raspbian GUI OS and the polling function is multi-threaded to keep the GUI responsive. Since Raspbian OS is not a Real-Time OS (RTOS), there is no guarantee that the Raspbian scheduler accurately times the GPIO sampling to be exactly

1us. Fortunately, the value is not capped to a certain value hence a curve could be fitted to estimate the actual frequency through measured frequency. A Python script was used to fit a logarithmic curve described by the equation:

$$y = Alog(x) + B$$

The fitted curve is depicted as the blue line shown in Fig. 5.3. It offers good accuracy while being significantly less complex when compared to higher-order polynomial fitting functions as it can be easily solved using linear regression on logged variables. The estimation function will only be used to recover the actual frequency when the frequency is above 250kHz.

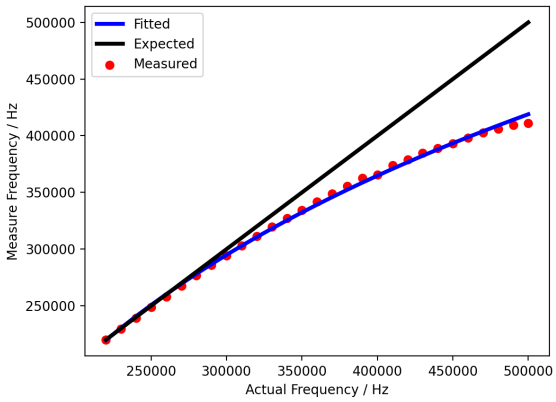


Figure 5.3: Logarithmic estimation (False zero)

5.2.3 Improved design

Due to the shortcomings of the frequency counter, an improved design was implemented to decode concurrent outputs shown in Fig. 5.4. The improved design measures the time between each positive edge and converts it to frequency. Since the frequency ranges are very far apart: AMP frequencies are at least 400 times higher than that of POT, it is possible to directly compare the time difference between two pulses with the maximum and minimum period for AMP and POT to differentiate between them. The average over the measurement time window is used as the final result.

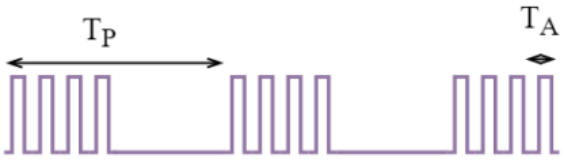


Figure 5.4: Concurrent output waveform

5.2.4 Comparison with existing products

Although the current implementation is accurate, it comes at the expense of computational power. Sampling the GPIO pins with 1us sampling period takes a heavy toll on the CPU, with the CPU nearly reaching 100% utilisation during polling sessions. If the user switches to the another tab (especially the history tab) when polling, the main event loop will compete with the polling function. This not only causes a major reduction in accuracy, but also limits visualisation to be done after polling has finished.

This can be circumvented by placing the polling function in the main event loop, however, unresponsiveness of the UI often results in user dissatisfaction. Another way would be to use a

micro-controller as an SPI slave controlled by the Pi. The Pi would send commands to initiate polling while the Pi.

Table. 5.2 compares the data visualisation technique with other researches done at CBIT. The polling function greatly limits the potential of the entire GUI. However, it can be claimed that real-time data visualisation is only needed for specific fields like ion-imaging.

Authors	Host	Read-out system	Real-time visualisation
Lei, et al.[13]	PC	FPGA	Yes
Lei, et al.[14]	PC	FPGA	Yes
S. Karolcik, et al.[6]	RPi	SPI	Yes
This work	RPi	N/A	No

Table 5.2: Visualisation comparison

5.3 Data Visualisation

5.3.1 Limitations of GTK4

Unlike other GUI frameworks, GTK4 has limited support for graph drawing. Instead of having an encapsulated drawing widget (Qt Chart) or an external drawing library (Matplotlib), GTK4 only offers a bare-bone GtkDrawingArea widget. The GtkDrawingArea widget provides a canvas on which drawings can be made using the Cairo graphics library [18].

5.3.2 The Cairo graphics library

Cairo an powerful open-source cross-device vector graphics library that is often used as the back-end for other plotting libraries. Cairo has a 'low-level' drawing model, where the programmer would need to create drawings by interacting with the three surfaces: source, mask and destination [19]. This drawing model is shown in Fig. 5.5 with the top, middle and bottom layer being source, mask and destination respectively. The source can be selectively mapped to the destination by drawing shapes on the mask layer.

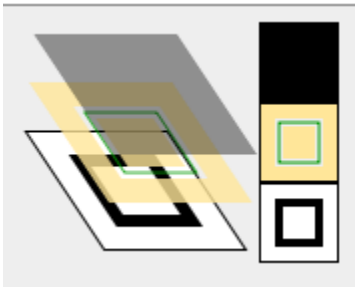


Figure 5.5: Cairo drawing model. Taken from [19]

5.3.3 Adding pan and zoom

Most users expect plots in apps to be interactive by nature, hence it would be counter-intuitive for the graph to be completely static. Due to the limitations of the current drawing technique employed, point selection would be difficult to implement: the 'points' are merely a pen stroke where no data is stored. Clicking would mean backtracking the entire array to find the data point closest to the current mouse position. However, it is expected that this feature would be implemented in the future. Zooming and panning are also important features for any graph to have as they offer users the flexibility to observe different parts of the plot.

Mouse scrolling events and dragging events can be captured using event controllers. The signature of a discrete scroll event is shown in Listing. 2. dx and dy are the distances of scrolling

along the horizontal and vertical axes.

```
gboolean scroll (
  GtkEventControllerScroll* self,
  gdouble dx,
  gdouble dy,
  gpointer user_data
)
```

Listing 2: Scroll signal signature. Taken from [20]

Since typical zooming gestures only use vertical scrolling for zoom, only *dy* was used. Each unit of *dy* would increment/decrement the zoom by 5% (minimum 5%) according to its direction: scrolling upwards zooms in while scrolling downwards zooms out.

Similarly, the panning feature was implemented by capturing drag events. When a drag event fires, it returns the mouse position offsets relative to the starting position. The offset is then used to translate the canvas by the amount (current offset + total offset). When the drag ends, the current offset is added to the total offset. This allows the preservation of all previous panning actions.

However, simply translating and zooming the entire surface will result in unexpected behaviour: axes getting zoomed in/out, label ticks and line plot going beyond the area defined by the axes, etc. Issues like these was circumvented by adding rectangular clipping regions. A clipping region is essentially a mask that only allows traces within it to be displayed. An overview of the locations of such clipping regions are shown in Fig. 5.6. The blue rectangular clipping region applies to both the signal plots and the ticks. The red region applies to tick labels only. The red clipping regions are deliberately set to be wider/taller on the x/y axis to account for the text size.

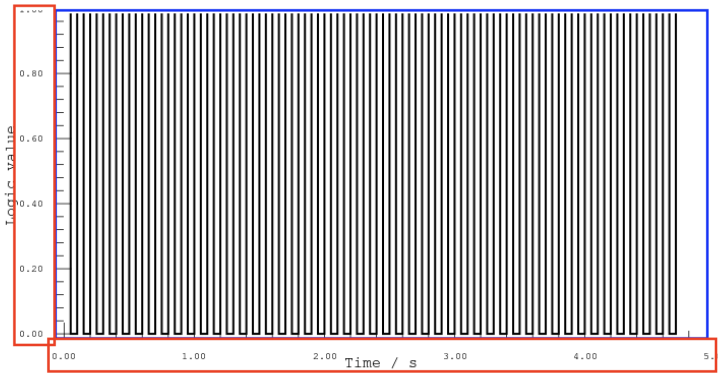


Figure 5.6: Clipping regions

5.3.4 Comparison with state-of-the-art libraries

Table. 5.3 compares the features that this work offers to that by other popular plotting libraries. It should be noted that the features listed are only a mere fraction of what these libraries offer, since most of these libraries are developed and maintained by a large number of developers. Although this work is an imitation of more successful products, it is still novel in the sense that there are no C libraries that can be used with GTK4 to this level of functionality.

Library/Widget	Line plot	Zoom&Pan	Point selection
QtCharts [21]	Yes	Yes	Yes
QCustomPlot [23]	Yes	Yes	Yes
Matplotlib [22]	Yes	Yes	Yes
This work	Yes	Yes	No

Table 5.3: Comparison with other libraries

5.4 Server-client model

As discussed in Chapter. 3, the platform should have IoT capabilities such as cloud storage and remote control. This paper implements the proposed server-client model shown in Fig. 5.7.

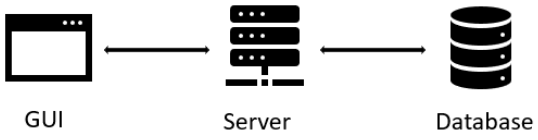


Figure 5.7: Server-client model

When a predefined event occurs, the GUI opens up a socket and attempts to connect to the server which controls the database. When the connection is established, the data will be packed in JSON and sent over the TCP connection. Most messages sent from the client (GUI) are uni-directional (i.e. the server does not respond). However, a few messages like user authentication, pulling previous measurements, etc. will invoke a response from the server. In this case, the client only closes the connection after the server has replied. All the messages and their triggers are listed in Table. 9.1 in the Appendix. The sockets were set to time-out if no connection was made to prevent too much resources being used. Also, any unsent messages are placed in a buffer and will be transmitted in the next successful connection.

5.4.1 Server

The server was written in Python as it would be ran on a remote server where performance is less of an issue. Python also has in-built support for Sqlite3 databases. The server is set to listen on a specific port for new client connections. Once the server receives a message, it parses the message and executes Server Query Language (SQL) commands according to the operation type specified. Currently the server is single-threaded and is only capable of handling one client at a time. When there is an active connection, all other connections are put in a queue and will be processed by the server after the current connection ends.

5.4.2 Database structure

An efficient database structure shown in Fig. 5.8. There are four tables: users, calibration, measurements and settings. The user table is used to store account information for each unique user by using uid (unique identifier) as the primary key. The other three tables are linked to the user table by using uid as the foreign key. This ensures that data entries can only be stored into the database if the associated uid is both valid and exists. Notably, certain fields where the data is stored in an array is stored as a JSON string to simplify storing and loading.

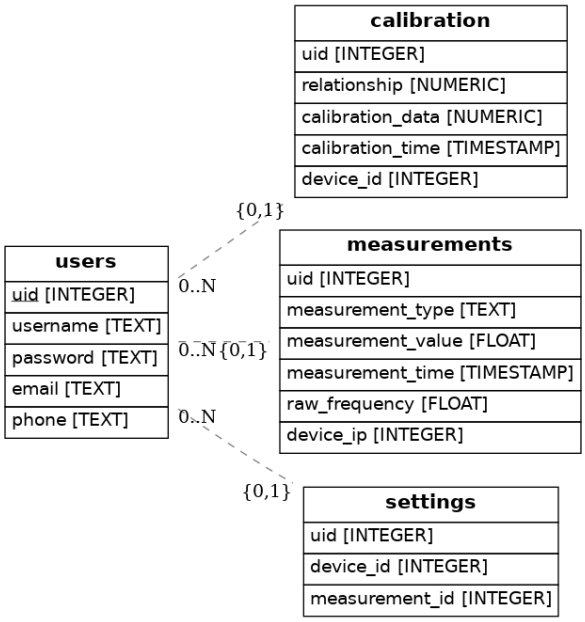


Figure 5.8: Relational diagram of the database

5.4.3 Data protection

SPACEMAN measures vital body signs and collects personal health information, hence it is important that the information of any user is stored securely and anonymously. According to the Data Protection Act, it is required that personal data is managed with appropriate security. Since the database is located on the server, it is assumed that it is less prone to attacks than the communication path between the GUI and the server. Currently, the only form of protection is provided by user authentication via randomly generated tokens. When the user logs in, the user password is first hashed with the known salt and sent to the server. The server then checks if the hashes match, if they do, a token is generated and sent to the client. The server keeps an one-to-one mapping of uid and the token. All messages sent from the client will include this string token. However, it should be noted that the security level of the entire pathway is only as secure as the least secure component which is the channel in this case. If the initial salt was intercepted, the token is no longer secure and can be used to fake the identity of the user via replay attacks.

5.5 Device calibration

On startup, the GUI will look within the directory for a calibration file that contains data from the last calibration session. If it does not exist, the GUI will prompt the user to carry out calibration by clicking the corresponding button. When the button is clicked, four consecutive dialogs (similar to that shown in Fig. 5.9) guide the user to put in technical buffer solutions of different pH levels. After measuring the four outputs, a linear function is fitted to the data via linear regression. The calculated linear relationship is stored in the model mentioned above and is used to convert frequency measurements into pH values.

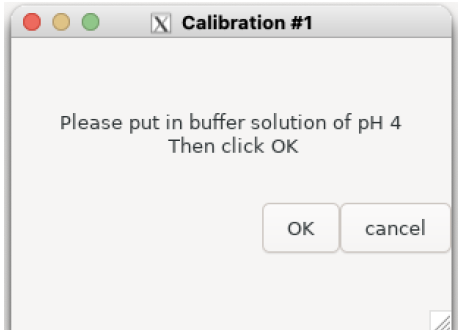


Figure 5.9: Calibration dialog

Since the calibration dialogs are the same except for the title number and the pH value displayed, using separate functions for each individual dialog is both cumbersome and prone to error. Therefore, a generic dialog-spawning function was implemented. The function takes pH values defined in the header according to the current number of dialogs spawned. This allows the behaviour of the calibration function, i.e., number of calibration attempts and pH values for each attempt to be altered by modifying the header files.

However, it should be noted changes to the header file will only be in effect after a full compilation. This is less than ideal as it requires the user to have some programming knowledge, making it difficult to be changed in the field.

Chapter 6

Testing and results

6.1 Testing concurrent output

The most crucial feature of the platform is the decoding the frequency of concurrent outputs, where potentiometry frequencies are modulated with the amperometry frequencies. The function used to to decode the modulated output can be in fact used to decode all three outputs. Due to the fact that the demodulation module for SPACEMAN was not ready in the early stages of testing, the concurrent output was simulated using a configuration similar to that shown in Fig. 6.1.

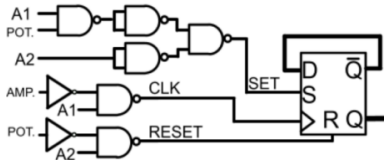


Figure 6.1: Modulator of SPACEMan. Taken from [1]

6.1.1 Experimental setup

A SN54F74 D flip-flop (DFF) was configured as shown in Fig. 6.2 and Fig. 6.3 to generate the output. The DFF was clocked by the higher frequency signal generator while the slower generator was connected to the reset/clear pin. Output pin Q was connected directly to GPIO pin 23 of the Raspberry Pi.

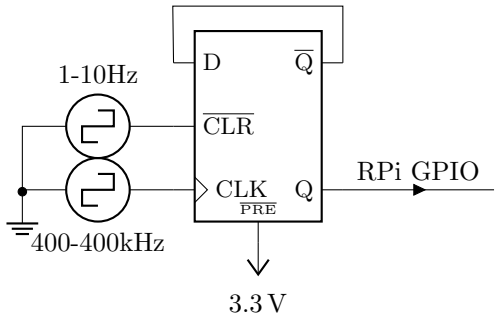


Figure 6.2: Concurrent output generator schematic

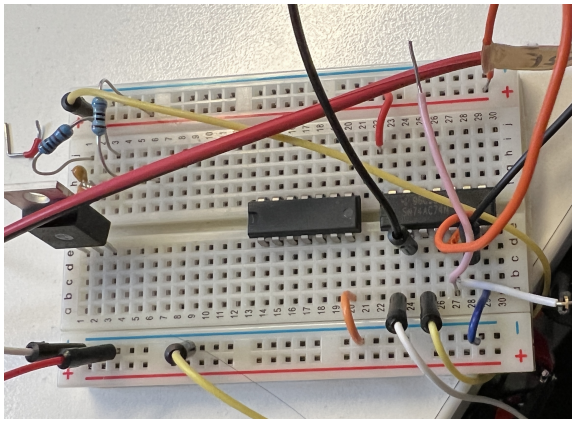


Figure 6.3: Breadboard connections

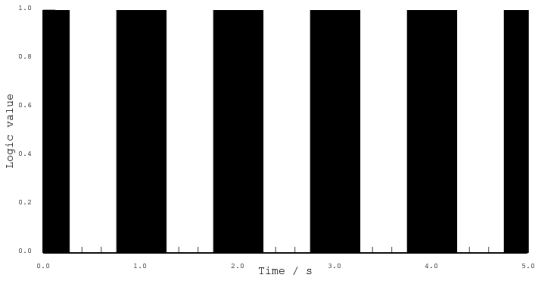
6.1.2 Experiment results

Several input frequency combinations were tested with the results shown in Table. 6.1. It can be seen that the decoder achieves an impressive maximum error of 1.95% across the test frequency range. The full amperometric frequency range of 400-400kHz was not tested due to a bug with one of the signal generators. Any frequency beyond 200kHz will cause it to generate a frequency much smaller than what is set on screen.

Pot/Amp (Hz)	Measured (Hz)	Error
1/25k	1.005/24.512k	0.5%/1.95%
1/50k	1.005/49.578k	0.5%/0.84%
1/100k	1.005/98.181k	0.5%/1.8%
1/200k	1.005/200.364k	0.48%/0.18%

Table 6.1: Concurrent output test results

The program output of 1Hz modulated by 200kHz is shown in Fig. 6.4. Note that a frequency of 200kHz is too high for plot to show, hence the positive half of the 1Hz signal is completely filled black.



(a) Plot output

```
Frequency of pot: 2.009537
Frequency of pot: 2.009561
Frequency of pot: 2.009566
Frequency of pot: 2.009618
Frequency of pot: 2.009687
Number of pot samples: 5
Number of amp samples: 492944
Total number of samples: 492958
Sample function measures average frequency of pot: 1.004797
Sample function measures average frequency of amp: 200364.157251
microsecond per tick: 0.998811
```

(b) Terminal output

Figure 6.4: Output of 200kHz x 1Hz

6.2 Testing client-server communication

6.2.1 Experimental setup

The typical user flow shown in Fig. 6.5 will be simulated. Both the GUI and the server was run on the same Raspberry Pi. The server was set to listen on port 65432. After all the operations are finished, the database entries will be checked to see if they are indeed correct.

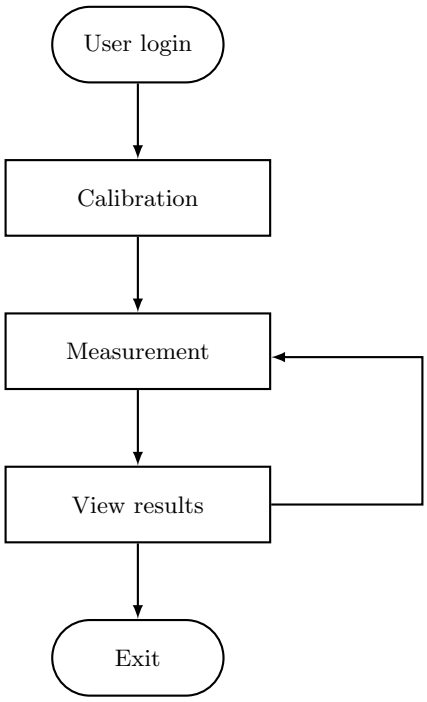


Figure 6.5: Typical user flow

The goal of the experiment would be to test if the client correctly sends the messages and if the server receives and stores them into the database. Also, the GUI will have to exhibit the correct behaviour for each user action, e.g. the measured frequency value must be correct.

6.2.2 Experimental results

The experiment results are shown in Table. 6.2. It can be seen that the server works as intended when use locally. Further testing will be required for remote servers.

Action	Correct GUI behaviour?	Correct server behaviour?
User login	Yes	Yes
Calibration	Yes	Yes
Measurement	Yes	Yes
View graphs	Yes	Yes
Change settings	Yes	Yes

Table 6.2: Test results

6.3 Testing GUI-Arduino communication

This section is dedicated to test the interfacing capabilities of the GUI with the wireless power module developed by Isabella Breslin. The module will be used to wirelessly power SPACEMan and demodulate the LSK output. The module consists of two Arduinos, two phase-locked loops (PLL) and an RF amplifier board. The two PLLs are each controlled by an Arduino via SPI. To trigger the state change of SPACEMan, the wireless power transmission will need to be paused for 100us. This call s for the

6.3.1 Experimental Setup

The experiment was set up as follows: the Raspberry Pi, Arduinos and a laptop was connected to the same router via Ethernet/WiFi. One Arduino is hosting a web server on port 80. After determining the IP address, the GUI will connect to the web server to transmit certain messages in JSON format. The laptop is connected to the Arduinos via USB to monitor the serial output.

6.3.2 Experimental results

A total of four messages was sent to the Arduino. Table. 6.3 showcases the four messages and the results. A screenshot of the serial terminal is shown in Fig. 6.6. It can be seen that all four messages were correctly received and parsed as a JSON object.

```
Ethernet WebServer Example
server is at 192.168.1.8
new client
{ "operation": "change_power_frequency", "frequency": 443000000.0 }
client disconnected
new client
{ "operation": "change_mixer_frequency", "frequency": 100000000.0 }
client disconnected
new client
{ "operation": "start_measurement", "measure_time": 5000000.0 }
client disconnected
new client
{ "operation": "stop_power", "stop_time": 100.0 }
client disconnected
```

Figure 6.6: Serial output from Arduino
Courtesy of Isabella Breslin

Message name	Correct?
change_power_frequency	Yes
change_mixer_frequency	Yes
start_measurement	Yes
stop_power	Yes

Table 6.3: Arduino communication test results

Chapter 7

Evaluation

The final technical specifications can be seen in Table. 7.1. All the requirements set at the beginning of the project were completed. Integration with the wireless module has begun by first consolidating the communication protocol between the platform and the module. The test results are shown in Chapter. 6.

Specification	Met
Frequency measurement	Yes
Calibration	Yes
Data Visualisation	Yes
Compatible hardware design	Yes
IoT capabilities	Yes
Full integration (optional)	No, started

Table 7.1: Final specifications

7.1 Comparison with previous platforms

Table. 7.2 compares this work to state-of-the-art electrochemical sensing platforms that was demonstrated in conferences. It can be seen that this work offers similar or better performance in terms of functionality. However, the IoT capabilities are limited: as of now, the server is running locally on the same device that the GUI is running for testing purposes.

Reference	Data Acquisition	Data visualisation	IoT
[4]	Yes	Yes	Cloud storage
[5]	Yes	Yes	No
[6]	Yes	Yes, real-time	No
This work	Yes	Yes	Local storage

Table 7.2: Comparison with state-of-the-art

Chapter 8

Conclusions and Further Work

This report presented a electrochemical sensing platform consisting of a GUI, a server, a motherboard and compatible cartridges. The platform is capable of interfacing with the wireless LoC SPACEMan and extracting data from it. The most challenging part of the project was the development of the GUI, especially the plotting library. The GUI was developed using GTK4 and a plotting library was developed to display data in an interactive format. However, the UI and the data visualisation is still one of the shortcomings of the project and will need to be improved upon in the future.

Multiple tests were done to ensure correct operation of all the components. Unfortunately, due to the time limit, the platform was not able to be tested with SPACEMan, signal generators were used to simulate output from SPACEMan. Also the new motherboard design was only finalised after all other components were verified. Therefore, its functionality was not able to be tested.

Future work

Due to the time limit, several key features were not implemented. However, it should be noted that the project is still ongoing. The author will continue developing and testing the platform.

The most important feature missing is full integration with the wireless power module developed by Isabella Breslin. Currently, the Pi handles the data acquisition by directly polling the GPIO pins. However, the accuracy was found to be largely limited by the non-RTOS Raspbian scheduler. Polling at 1MHz also takes a heavy toll on the CPU, possibly causing the GUI to lag or even become unresponsive. In the future the measurement will be done by the wireless power module developed by Isabella Breslin. The GUI will command said module to start measurements, stop power transmission, etc. This removes the high performance requirements on the GUI, meaning that using a cross-platform GUI framework such as Flutter becomes possible. A cross-platform app greatly simplifies the development process by cutting down the number of applications to only one. Modern cross-platform frameworks typically have modern-looking widgets and support for data visualisation.

Secondly, the communication protocol for client-server and client-Arduino communication is not secure. The client-server communication is based off the TCP protocol only with JSON formatted data transmitted as cleartext. This is extremely prone to sniffing by eavesdroppers. A solution to this would be to use Secure Sockets Layer (SSL) [32]. Another server-related issue is the current single-threaded implementation of the server, which limits the number of concurrent connections to one only. Parallel socket connections can be achieved by using Python processes while concurrent access to the sqlite database can be done via Write-Ahead Logging (WAL) [37].

An important feature to add would be SPI support. As mentioned in Chapter. 2, a number of chips previously developed at CBIT are integrated with an SPI readout module. With

the additional SPI interface, the platform can also be used with these chips after some minor modifications as well.

The possibility of a MCU-based solution will be considered in the near future when the functionality of the full system has been verified. Although a Pi consumes relatively less power than a PC, it still draws considerably more power than a microcontroller. Considering the envisioned use of SPACEMan as a portable PoC platform, the switch to a microcontroller based solution seems inevitable.

Chapter 9

User Guide

9.1 System requirements

Currently the app can only be run on a Raspberry Pi since it needs access to the PIGPIO library that can only be installed on Raspbian systems. The app was only tested on Raspberry Pi 4 8GB RAM variant and there is no guarantee that the GUI will run smoothly on older versions of RPi.

9.2 Installing required packages

The most important packages are listed as follows:

- **libgtk-4-dev**, you may need to switch on to an unstable or experimental release of Debian to install this package (tutorial of this can be found online).
- **PIGPIO**: the PIGPIO library can be found here: https://abyz.me.uk/rpi/pigpio/.

To install a package on Raspbian (Debian), one would run the following script:

```
sudo apt-get install xxx
```

where xxx is the name of the library you wish to install.

9.3 Installation (client)

9.3.1 Running the executable

There is an executable file named 'gui' under the file directory. You have to run the executable with root privileges as follows:

```
sudo -E ./gui
```

The -E flag keeps the environmental variables. Running without it creates errors for some reason.

9.3.2 Compiling the source code

Each .c source file comes with a header file that has all the function declarations. If you wish to change the function signature, the function must be changed in both the header and the source file.
To compile the source code, run the following command:

```
bash linux_compile.sh
```

To speed up setup, a makefile will be included in the future.

9.4 Installation (server)

Unlike the client, the server can be run on any machine, preferably on the same machine or a remote server with a static public address.

9.4.1 Installing Python

The server is currently a Python script instead of an executable. To run the script, Python will need to be installed. The latest version of Python can be found at: https://www.python.org. Make sure that Python has been added to the PATH of your computer (for convenience).

9.4.2 Installing packages

The server currently only uses one external library, i.e. the sqlite3 library. It can be installed by running the following commmand in a terminal:

```
pip install sqlite3
```

9.4.3 Configuring the database

It will first look for an existing database, if not found, it would create a database with tables shown in Fig. 5.8. It is set to listen on port 65432 by default, this can be changed to the user's liking. Note that the port settings for the client will need to be changed as well.

Bibliography

[1] D. Ma, Y. Chen, S. S. Ghoreishizadeh and P. Georgiou, "SPACEMan: Wireless SoC for Concurrent Potentiometry and Amperometry," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401312.

[2] D. Ma, S. S. Ghoreishizadeh and P. Georgiou, "DAPPER: A Low Power, Dual Amperometric and Potentiometric Single-Channel Front End," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9181213.

[3] M. Cacho-Soblechero, S. Karolcik, D. Haci, C. Cicatiello, C. Maxoutis and P. Georgiou, "Live Demonstration: A Portable ISFET Platform for PoC Diagnosis Powered by Solar Energy," 2019 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2019, pp. 1-1, doi: 10.1109/BIOCAS.2019.8919227.

[4] A. Au, N. Moser, J. Rodriguez-Manzano and P. Georgiou, "Live Demonstration: A Mobile Diagnostic System for Rapid Detection and Tracking of Infectious Diseases," 2018 IEEE International Symposium on Circuits and Systems (ISCAS), 2018, pp. 1-1, doi: 10.1109/ISCAS.2018.8351802.

[5] N. Miscourides, L. Yu, J. Rodriguez-Manzano and P. Georgiou, "A 12.8 k Current-Mode Velocity-Saturation ISFET Array for On-Chip Real-Time DNA Detection," in IEEE Transactions on Biomedical Circuits and Systems, vol. 12, no. 5, pp. 1202-1214, Oct. 2018, doi: 10.1109/TBCAS.2018.2851448.

[6] S. Karolcik, N. Miscourides and P. Georgiou, "Live Demonstration: A Portable High-Speed Ion-Imaging Platform using a Raspberry Pi," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1-1, doi: 10.1109/ISCAS.2019.8702242.

[7] M. Cacho-Soblechero and P. Georgiou, "A Programmable, Highly Linear and PVT-Insensitive ISFET Array for PoC Diagnosis," 2019 IEEE International Symposium on Circuits and Systems (ISCAS), 2019, pp. 1-5, doi: 10.1109/ISCAS.2019.8702532.

[8] Y. Hu, N. Moser and P. Georgiou, "A 32 $\times$ 32 ISFET Chemical Sensing Array With Integrated Trapped Charge and Gain Compensation," in IEEE Sensors Journal, vol. 17, no. 16, pp. 5276-5284, 15 Aug.15, 2017, doi: 10.1109/JSEN.2017.2722043.

[9] Katoba, Juliet et al. "Toward Improving Accessibility of Point-of-Care Diagnostic Services for Maternal and Child Health in Low- and Middle-Income Countries." Point of care vol. 18,1 (2019): 17-25. doi:10.1097/POC.0000000000000180

[10] N. Moser, T. S. Lande, C. Toumazou and P. Georgiou, "ISFETs in CMOS and Emergent Trends in Instrumentation: A Review," in IEEE Sensors Journal, vol. 16, no. 17, pp. 6496-6514, Sept.1, 2016, doi: 10.1109/JSEN.2016.2585920.

[11] N. Moser, L. Keeble, J. Rodriguez-Manzano and P. Georgiou, "ISFET Arrays for Lab-on-Chip Technology: A Review," 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), 2019, pp. 57-60, doi: 10.1109/ICECS46596.2019.8965034.

[12] D. Ma, C. Mason and S. S. Ghoreishizadeh, "A wireless system for continuous in-mouth pH monitoring," 2017 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2017, pp. 1-4, doi: 10.1109/BIOCAS.2017.8325556.

[13] L. Kuang, J. Zeng and P. Georgiou, "High-Throughput Digital Readout System for Real-Time Ion Imaging using CMOS ISFET Arrays," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9181096.

[14] L. Kuang, J. Zeng and P. Georgiou, "A USB 3.0 High Speed Digital Readout System with Dynamic Frame Rate Processing for ISFET Lab-on-Chip Platforms," 2021 IEEE International Symposium on Circuits and Systems (ISCAS), 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401663.

[15] N. Moser, J. Rodriguez-Manzano, T. S. Lande and P. Georgiou, "A Scalable ISFET Sensing and Memory Array With Sensor Auto-Calibration for On-Chip Real-Time DNA Detection," in IEEE Transactions on Biomedical Circuits and Systems, vol. 12, no. 2, pp. 390-401, April 2018, doi: 10.1109/TBCAS.2017.2789161.

[16] Codeandlife.com. 2022. Benchmarking Raspberry Pi GPIO Speed — Code and Life. [online] Available at: `https://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/`[Accessed 22 January 2022].

[17] Abyz.me.uk. 2022. pigpio library. [online] Available at: `https://abyz.me.uk/rpi/pigpio/examples.html` [Accessed 22 January 2022].

[18] Docs.gtk.org. 2022. Gtk.DrawingArea. [online] Available at: `https://docs.gtk.org/gtk4/class.DrawingArea.html` [Accessed 6 June 2022].

[19] Cairographics.org. 2022. cairographics.org. [online] Available at: `https://www.cairographics.org/` [Accessed 6 June 2022].

[20] Docs.gtk.org. 2022. Gtk.EventControllerScroll::scroll. [online] Available at: `https://docs.gtk.org/gtk4/signal.EventControllerScroll.scroll.html` [Accessed 7 June 2022].

[21] Doc.qt.io. 2022. Qt Charts Overview — Qt Charts 6.3.0. [online] Available at: `https://doc.qt.io/qt-6/qtcharts-overview.html`[Accessed 8 June 2022].

[22] "Matplotlib — Visualization with Python", Matplotlib.org, 2022. [Online]. Available: `https://matplotlib.org/`. [Accessed: 08- Jun- 2022].

[23] E. Eichhammer, "Qt Plotting Widget QCustomPlot - Introduction", Qcustomplot.com, 2022. [Online]. Available: `https://www.qcustomplot.com/`. [Accessed: 08- Jun- 2022].

[24] "Chemistry Reference Manual", Beckmancoulter.com, 2022. [Online]. Available: `https://www.beckmancoulter.com/wsrportal/techdocs?docname=A45586.pdf`. [Accessed: 09- Jun- 2022].

[25] "Hydrion Tri-Chek Buffer Set-Vial 4-7-10", Microessentiallab.com, 2022. [Online]. Available: `https://www.microessentiallab.com/ProductInfo/F11-BUFCL-CC471V-BST.aspx`. [Accessed: 09- Jun- 2022].

[26] Fatourou, P., 2022. Introduction to Sockets Programming in C using TCP/IP. [online] Csd.uoc.gr. Available at: `https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf` [Accessed 12 June 2022].

[27] "GitHub - json-c/json-c", GitHub, 2022. [Online]. Available: `https://github.com/json-c/json-c`. [Accessed: 12- Jun- 2022].

[28] "JSON", Json.org, 2022. [Online]. Available: `https://www.json.org/json-en.html`. [Accessed: 12- Jun- 2022].

[29] "An overview of HTTP - HTTP — MDN", Developer.mozilla.org, 2022. [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#http_messages`. [Accessed: 16- Jun- 2022].

[30] STMicroelectronics. 2022. LMV324L - STMicroelectronics. [online] Available at: `https://www.st.com/en/amplifiers-and-comparators/lmv324l.html` [Accessed 16 June 2022].

[31] "Flutter - Build apps for any screen", Flutter.dev, 2022. [Online]. Available: `https://flutter.dev/`. [Accessed: 18- Jun- 2022].

[32] E. Rescorla, "An Introduction to OpenSSL Programming (Part I)", Cs.cmu.edu, 2022. [Online]. Available: `https://www.cs.cmu.edu/~srini/15-441/F02/Projects/lab01/reference/part1.pdf`. [Accessed: 19- Jun- 2022].

[33] "STM32L010R8 - STMicroelectronics", STMicroelectronics, 2022. [Online]. Available: `https://www.st.com/en/microcontrollers-microprocessors/stm32l010r8.html`. [Accessed: 20- Jun- 2022].

[34] "STM32 Graphical User Interface - STMicroelectronics", STMicroelectronics, 2022. [Online]. Available: `https://www.st.com/content/st_com/en/ecosystems/stm32-graphic-user-interface.html`. [Accessed: 20- Jun- 2022].

[35] "Design & Develop for Embedded Microcontrollers — IDE Software — Qt", Qt.io, 2022. [Online]. Available: `https://www.qt.io/product/develop-software-microcontrollers-mcu`. [Accessed: 20- Jun- 2022].

[36] "12-Bit DAC with SPI Interface", Ww1.microchip.com, 2022. [Online]. Available: `http://ww1.microchip.com/downloads/en/devicedoc/21897a.pdf`. [Accessed: 21- Jun- 2022].

[37] "Write-Ahead Logging", Sqlite.org, 2022. [Online]. Available: `https://www.sqlite.org/wal.html`. [Accessed: 22- Jun- 2022].

Appendix

Source code, installation guide and relevant documentation can all be found in the Github repository: https://github.com/YutingXu/SPACEMAN-GUI. The repository also contains the PCB design files for the SPACEMan motherboard and the compatible cartridges. Any comment/-contribution is welcome.

Message name	Origin	Destination	Reply required?
log_in	GUI	Server	Yes
reset_password	GUI	Server	Yes
store_measurement	GUI	Server	No
load_measurement	GUI	Server	Yes
store_calibration	GUI	Server	No
load_calibration	GUI	Server	Yes
start_measurement	GUI	Arduino	No
change_frequency	GUI	Arduino	Yes
stop_power	GUI	Arduino	No

Table 9.1: List of all messages

Listing 3: Definition of model

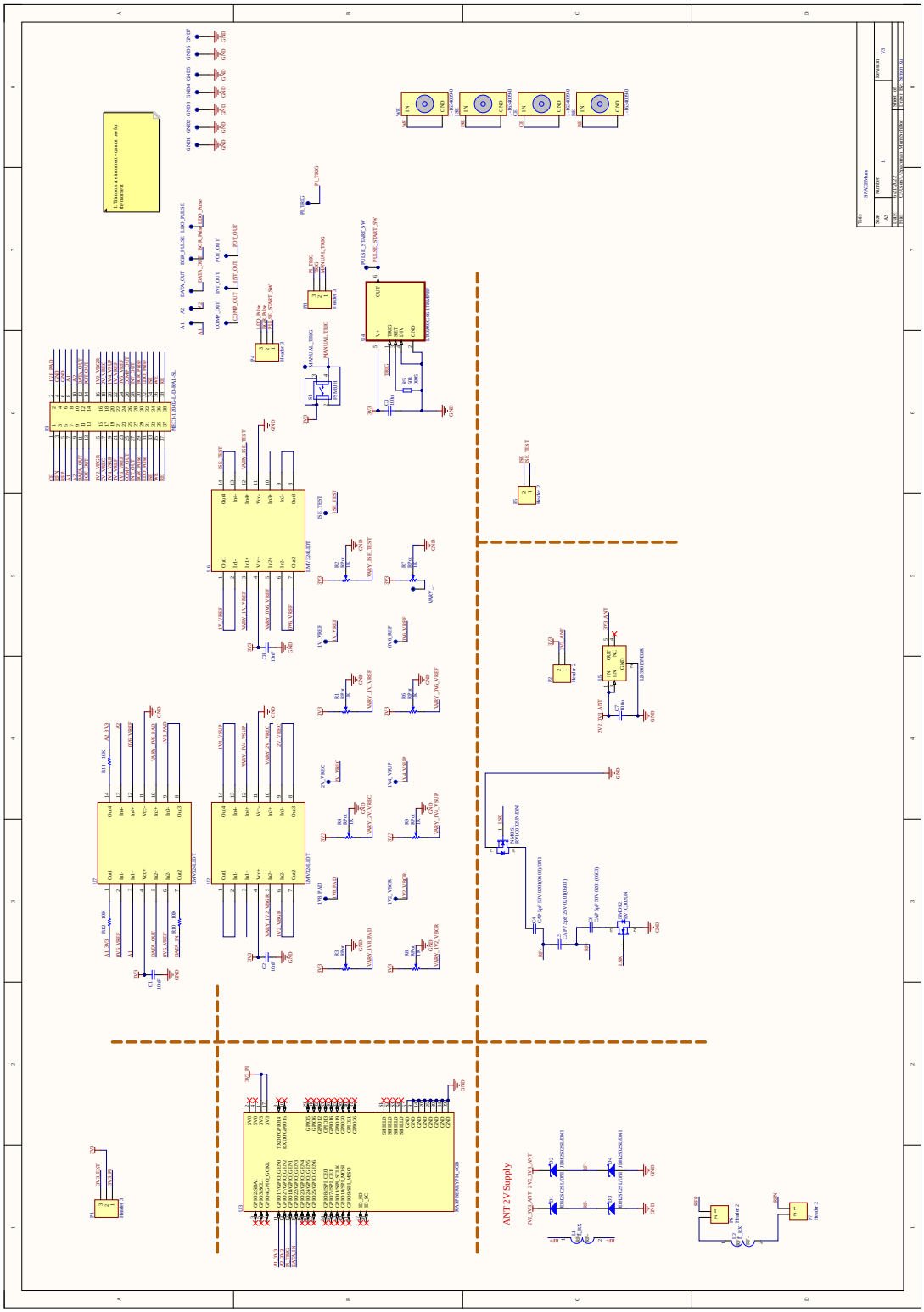


Figure 9.1: Motherboard schematic V3

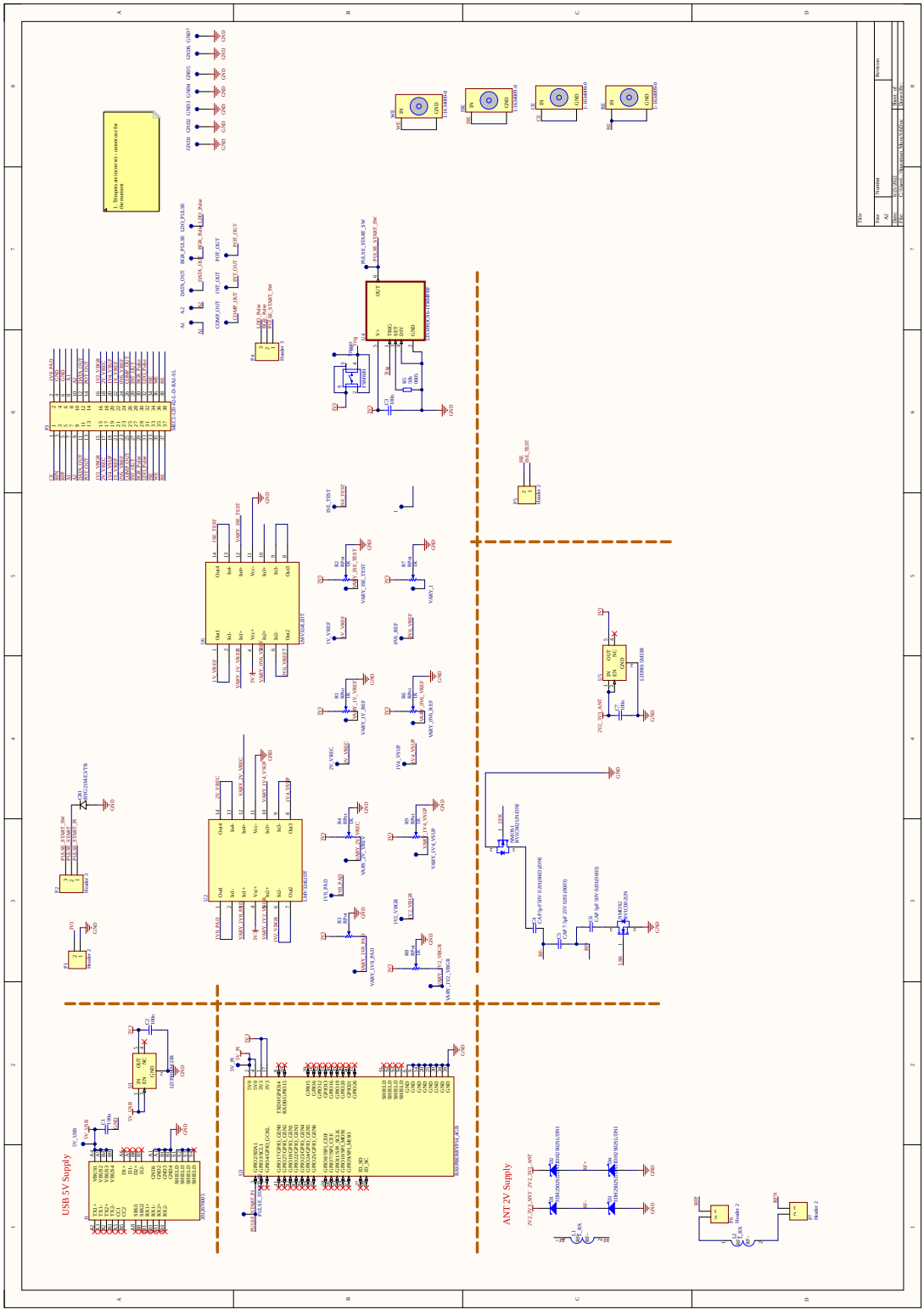


Figure 9.2: Motherboard schematic V2