Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report

May 2020

Project Title: System for real-time evaluation of electrochemical sensors

Student: Yu Sophia Wang

CID: 01342326

Project Supervisor: Daryl Ma

Final Report Plagiarism Statement

I affirm that I have submitted, or will submit, electronic copies of my final year project report to both Blackboard and the EEE coursework submission system.

I affirm that the two copies of the report are identical.

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

Abstract

The quantification of biological and biochemical processes is a paramount part of modern day healthcare; advances in the technologies involved are constantly being sought after. The detection of these processes rely on the techniques of electrochemical sensing.

This project is centered around the use of electrochemical biosensing and involves the design of a test-bench to aid in the better understanding of the processes of biosensing, its potential and its applications. The bulk of the research in this project was done on the AD5940 evaluation board from Analog Devices, which was chosen to perform the electrochemical sensing measurements. The test-bench aims to represent the data from the board in a user-friendly way through a graphical user interface (GUI) implemented using the Raspberry Pi, which was chosen for its size, customizability and versatility.

Amperometric and potentiometric measurements will be available to view through real-time plots on the GUI. The amperometric process will utilise the three electrodes configuration, connected via the provided pins to the Low-Power Transimpedance Amplifier (LPTIA) circuit on the AD5940 board. The potentiometric process will use the ion-selective electrode (ISE) connected directly to an analog input pin, to be outputted through the analog to digital converter (ADC).

Both measurements can be outputted to the Raspberry Pi by configuring the evaluation board to switch between the two measurement blocks at regular

intervals. The Raspberry Pi will interface with the evaluation board through the SPI protocol, configuring the board by writing to the register responsible for the inputs to the ADC multiplexer and receiving the outputs by reading from the ADC raw result registers on the board. The GUI provides a function to calibrate the sensors, required to display the potentiometric results.

The test-bench is tested: the potentiometric aspect of the system is tested using standard alkaline 1.5V batteries or by directly using voltage pins from the board. The amperometric part can be tested using the same batteries connected across a resistor. It will be important to test that the communication between the board and the Raspberry Pi works for the test-bench to be successful.

Acknowledgements

This project exists thanks to the following wonderful people:

Professor Georgiou Pantelis for interesting me in the biomedical aspect of my degree through teaching the module Biomedical Electronics in Autumn term. Thank you for providing me with the opportunity and introducing me to projects proposed by your PhD students.

Daryl Ma for proposing the idea for this project; for guiding me through various unfamiliar territories such as dealing with the Raspberry Pi and bringing the theory of electrochemical sensing into practice. Thank you for being patient with me and my innumerable questions, for considering my interests when structuring the project and for making this an immensely enjoyable journey.

Thank you to the Electrical and Electronics Engineering Department for teaching me what I know.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Project overview

The number of robust technological devices specialising in bioelectrical and electrochemical sensing is continually on the uprise, making these techniques increasingly accessible, contributing to advances in various fields. The techniques of biosensing target a variety of biological events, contributing not only to advances in healthcare such as in neurological disease diagnostics [22] and DNA sequencing [2], but in pathogen detection in agriculture [4], and also aiding in environmental monitoring through detection of heavy metal pollutants in water [1].

Such a robust device is the AD5940 from Analog Devices, offering high precision analysis of electrochemical cells. Although the analog front end (AFE) comes equipped with its own integrated development environment (IDE) and graphical user interface (GUI), this project aims to push the boundaries of the system by exploiting the AFE's existing functions and configuring it for the purpose of an electrochemical sensing platform for analysis in blood and sweat through building a test-bench.

The importance of a test-bench lies in its purpose of providing a controlled process for gauging a particular system and its outputs, hence minimising inconsistencies between different measurements due to errors. A test-bench should be as fully automated as possible, to diminish the likelihood of integrating human involvement and therefore avoidable errors into the system. In this project, the calibration system and the real-time output of data will provide the means of acting as a control for the electrochemical measurements.

1.2 Project deliverables

Due to limited resources, the project deliverables set at the start of the project have since been altered, and is summarised as follows:

System requirements

- Explore the set up of concurrent amperometry and potentiometry measurements using embedded, off-the-shelf components
- Test the individual components of the test-bench system and evaluate the functionalities of the system as a whole
- A GUI for displaying real-time potentiometry and amperometry results
- The test-bench should enable user to calibrate output data by measuring results from input pH

1.3 Report structure

The report will start with an overview of the project background and research in **chapter 2**. The report will then begin on the details of the test-bench

starting from the front-end of the system, the evaluation board, in **chapter 3**. This is followed by **chapter 4**, which details the software aspect of the system, including the implementation of the communication protocol and the GUI. The report then concludes by explaining and evaluating the way the system was tested in **chapter 5**; comparing the end result of the project to the initial deliverables in **section 6.2**, and finally elucidating the work that can be done in the future to improvement the system in **chapter 7**.

Chapter 2

Background

This chapter will touch on the **aims and specifications** of the project, including the motivation behind the project's aims and a rough overview of the roles each component of the test-bench plays in contributing to the overall system. The theories behind **electrochemical sensing** is also explained. The chapter will finish by elucidating on the **high level design** of the project, which covers the results from the literature reviews done on each aspect of the test-bench, elaborating on why each component was chosen and how they are intended to be implemented in the project.

2.1 Project Specification

The overall aim of this project is to create a test bench with a user interface that outputs the results from a given reaction in real time using the techniques of amperometry and potentiometry concurrently.

For this project, an evaluation board was chosen to perform amperometric and potentiometric measurements, namely the AD5940 analog front end system from Analog Devices. This board was chosen as it was more recently updated than other products such as the LMP91000EVM evaluation module from Texas Instruments [14], and at a more affordable price than more capable products such as the Emstat Pico module from PalmSens. More importantly, this evaluation system was chosen for its potential to perform amperometry and potentiometry simultaneously.

The motivation behind achieving concurrent sensing comes from the need to improve accuracy in biosensing measurements. A key challenge in the measurement of biomolecules in environments such as the human body is the presence of interfering molecules. Additionally, factors such as biochemical changes and changes in temperature will further affect results in electrochemical sensing [16]. In environments such as the saliva, pH follows the circadian rhythm of the body and depends on the individual's dietary habits, the presence of gastric acid reflux, symptoms of diabetes and the conditions of their oral health [5]. Concurrent sensing would improve accuracy by calibrating in real-time using electrochemical biomedical readings [25, 23].

The inherent differences in the sensing techniques for potentiometry and amperometry result in advantages and disadvantages in electrochemical measurements. In the case of glucose, amperometry provides a faster speed of

detection, while potentiometric techniques would allow for a lower limit of the concentration of the trace element for detection. Responses for glucose using the two methods also differ, with potentiometry having a logarithmic response while amperometry is linear with respect to the concentration of the detected molecules. To sense concurrently means combining the advantages in each of these individual methods to provide a more accurate analytical platform. Therefore one of the main goals of this project is to configure the AD5940 to perform concurrent sensing. With its potential to do so implicit in its datasheet, albeit seemingly unavailable on the pre-existing graphical user interface (GUI), SensorPal.

The system also required a means of outputting the results, the Raspberry Pi was chosen to fulfill this purpose. The evaluation board interfaces with and outputs onto the Raspberry Pi for a simple display of the results (e.g. pH from potentiometric measurements) on a GUI in real time, communicating through the serial peripheral interface (SPI).

2.2 Electrochemistry background

Chemical sensors are small sized devices comprising a recognition element, a transduction element, and a signal processor capable of continuously reporting a chemical concentration [11]. Multiple electrodes are employed to translate biorecognition responses into electrical signals [20]. The three electrode cell is most often used, consisting of:

- The working electrode (WE) where the redox reaction will take place [11]

- The reference electrode (RE) which applies a constant potential against which the working electrode's potential is measured [11]
- The counter electrode (CE) which completes the electrical circuit and acts as a current supply for the working electrode [11]

The voltage applied between the RE and WE is known as the bias voltage of the cell. The potentiostat, shown in figure 2.1, plays an important role in imposing the bias voltage in potentiometric and amperometric applications of electrochemistry.

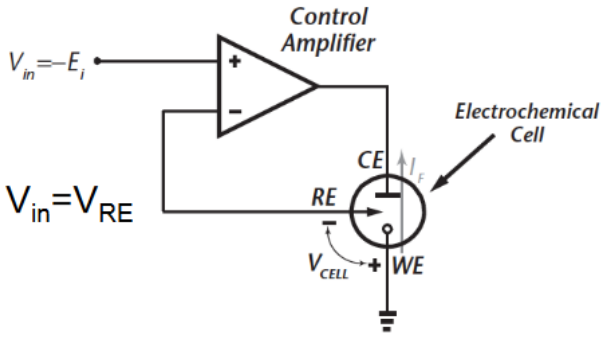

Figure 2.1: Diagram of a typical potentiostat [11]

The principles of electrochemistry is based on the redox phenomenon of ions and elements which involves the gain and loss of electrons. This takes place at the interface between the electrodes and the aqueous solution. It is this behaviour that creates a potential difference between two electrodes and is measured in potentiometry and the resulting current that is measured in amperometry.

2.2.1 Potentiometry

Potentiometry is typically used for detecting the pH which is based on the presence and concentration of H^+ ions. Potentiometry is the measurement of the potential between an ion-selective electrode (ISE) and a reference electrode submerged in the analyte and the reference solution respectively, where the potential difference is generated by the difference in concentrations between the two solutions. The potentiometric circuit requires the input impedance to be high and for there to be no current flow, which is achieved by using an OpAmp [11]. For the purpose of potentiometric or pH measurements, the ISE or AgAgCl electrode are used as the WE.

2.2.2 Amperometry

Amperometry is the measurement of the current resulting from a constant voltage applied to the working electrode. This constant voltage can be found using cyclic voltammetry, where the output current is measured as the reference electrode voltage is ramped up and down linearly. Around the voltage at which the chemical reaction being observed takes place the current will peak, as shown in 2.2.

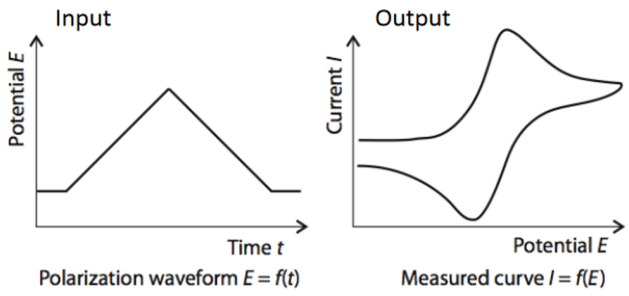

Figure 2.2: Demonstration of cyclic voltammetry [11]

The amplitude of the peak is directly proportional to the concentration of the chemical species and are related by the Randles-Sevcik Equation (2.1).

$$i_p = 2.69 \times 10^5 n^{\frac{3}{2}} A D^{\frac{1}{2}} C v^{\frac{1}{2}} \tag{2.1}$$

Where n is the number of electrons transferred per molecule, $A(cm^2)$ is the electrode surface area, $C(mol/cm^3)$ is the concentration, $D(cm^2/s)$ is the diffusion coefficient, $V(V/s)$ is the scan rate.

Amperometry is typically used for detecting the presence and concentrations of small biomolecules such as lactate, glucose and ATP [25], the glucose sensor being the most common amperometric sensor. Amperometry usually works by coating the WE with a compound that produces a redox current once it encounters the molecule being observed, in the case of the glucose sensor, the enzyme Glucose oxidase is used [11].

The concentration of the molecules found through the detected current using equation 2.1 can be used to find the pH, if the said molecules are hydronium

ions. This is found using equation 2.2.

$$pH = \log_{10}[H_3O^+] \tag{2.2}$$

The transimpedance amplifier (TIA) is a critical part of the measurement of the redox currents produced by the electrochemical reaction. The TIA converts the resulting redox current from the potential difference between the electrodes into an output voltage, as shown in figure 2.3. This is achieved by passing the current, I_F, through a resistor R_F. In the AD5940, it is referred to as R_{TIA}.

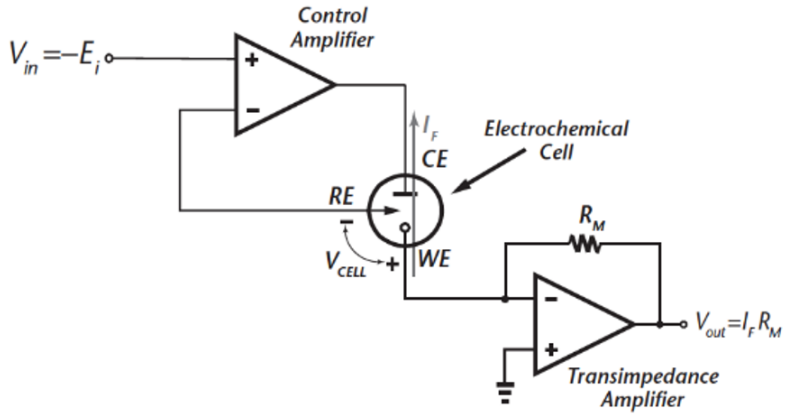

Figure 2.3: Diagram of a typical transimpedance amplifier [11]

The AD5940 uses its low power 6-bit and 12-bit digital to analog converters (DAC) to set the bias voltages on the electrodes, whilst its potentiostat amplifier and LPTIA are used to measure the current. For the purposes of

amperometric measurements, the Platinum electrode is used as the WE.

2.3 High level design

2.3.1 System components

The system required a way to perform amperometry and potentiometry. Building the circuits for these measurements was deemed to be time consuming, difficult and would stray from the project objective. Additionally, many existing products offered sophisticated analog front end systems that suited this project's aims. For these reasons, the use of an evaluation board was looked into.

Most evaluation boards included multiple communication interfaces such as I^2C, SPI and UART, some also had Bluetooth and WIFI modules. The Raspberry Pi 4 Model B was chosen to be used in this project for its abilities to communicate via those protocols. It was also chosen for its capabilities in visually outputting the results from the evaluation board through a GUI. Using the Raspberry Pi seemed more straightforward, its materials more numerous and accessible than other platforms such as Arduino products. For example, the `Tkinter` library seems a lot quicker and more straight forward to use and integrate on the Raspberry Pi when compared to the `ControlP5` library which requires the Processing IDE in addition to the Arduino IDE [31].

2.3.2 Selecting an evaluation board

An initial high level design was constructed from the research conducted prior to the start of the project. The main component of this project is the

evaluation board and several options were looked at, shown in table 2.1.

Board	Producer	Description	Year of last update	Price
LMP91000 [14]	Texas Instruments	Configurable AFE Potentiostat for Low-Power Chemical-Sensing Applications	2014	£101.93
AD5940 [6]	Analog Devices	High Precision, Impedance, and Electrochemical Front End	2020	£277.99
Emstat Pico [28]	PalmSens	Miniaturized potentiostat module for electrochemical measurements	2020	£1381.85

Table 2.1: Comparison of evaluation boards

Three different electrochemical evaluation boards are shown in table 2.1, with varying producers, prices, functions, and some more current than others. The LMP91000 from Texas Instruments is at the most affordable price. It provides communication over the I^2C interface [14] and seems relatively straight forward to program. However, its data sheet was last updated in December of 2014 which can mean that resources on the product is scarce and outdated and would therefore be difficult to work with.

The Emstat Pico from PalmSens seems to be the most suitable for applications requiring concurrent sensing - concurrent sensing is explicitly stated as possible in its datasheet, which weren't the cases for the other two products. Although its functionalities seem the most ideal for this project, the Emstat Pico's price is significantly higher than the other two boards, which isn't desirable.

The AD5940, although ambiguous at first regarding its ability to perform concurrent sensing, has an abundance of up-to-date resources to help develop the board and its applications whilst also priced at a relatively affordable

rate, and therefore chosen for the purposes of this project.

2.3.3 Prior research on the GUI

The AD5940 comes compatible with its existing GUI from Analog Devices, SensorPal, shown in figure 2.4. The functions of which inspired the GUI in this project. SensorPal allows the user to choose from a list of measurement techniques listed on the left hand side of the window. The user can select a measurement technique by dragging the respective label into the "Work Area", the output of which is then viewed through a plot of the data on the bottom right corner. The x-axis data points can be viewed by hovering the cursor over the plot, and the data can be exported into Excel by clicking the Excel icon on the top right of the graph. There is also a frame in the middle of the window which allows the user to customise the range and scan rate of the output data.

SensorPal was undoubtedly going to be more sophisticated a platform when compared to the GUI envisioned in this project. The features of SensorPal not deemed necessary for the test-bench in this project was omitted for the reasons that the GUI in this project focuses on its ability to display both the potentiometric and amperometric results concurrently.

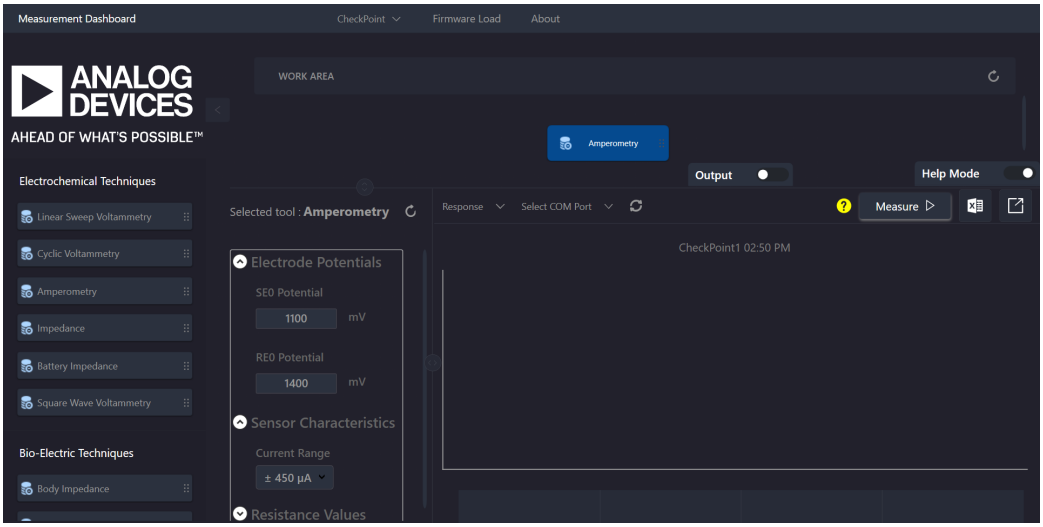

Figure 2.4: AD5940 compatible GUI, SensorPal

There are many libraries available that allow the implementation of a Python-coded GUI on the Raspberry Pi. These include `Tkinter`, `GTK+`, `QT`, `wxWidgets` and many more. When considering the timescale of this project, `Tkinter` was deemed to be the most favourable Python library. This is because `Tkinter` is the most commonly used and therefore arguably has the most resources and help available online. Furthermore, the different libraries don't seem to vary too much in terms of their aesthetics, which made `Tkinter` the foremost option for the implementation of a GUI in this project.

Chapter 3

The AD5940 evaluation board

In this chapter, the **research** conducted on the AD5940 evaluation board are discussed and **the board's functions** introduced. In **section 3.1**, the different aspects of this test-bench and each of their requirements is further detailed. The **configuration of the front end of the system** is presented, including the **electrode connections** to the board and the **configurations of the relevant registers** on the board that need to be dealt with for the purposes of this project. The chapter draws to a close with **section 3.4** which reveals the physical **set up** of the components of the test-bench and the requirements of the protocol with which the AD5940 will use to communicate and **interface to the Raspberry Pi**, the implementation of which will be detailed in the next chapter.

3.1 Prior research on the AD5940

This section reviews the options considered at the initial planning stage concluded from conducting the background research for the various aspects of the test-bench. In the following **sub-section 3.1.1**, an overview of the functions of the AD5940 deemed possibly necessary and relevant to this project are delineated. This is followed by an explanation of the options to interface to the AD5940 and the software requirements.

3.1.1 Overview of functions

The AD5940 has several input channels, including multiple external current and voltage inputs from the sensors. A programmable switch matrix connects the sensors to the measurement blocks and is used to multiplex multiple electronic measurement devices to the same set of electrodes.

The AD5940 measurement board features an analog to digital converter (ADC) with an input multiplexer (MUX) in front of the ADC to allow an input measurement channel to be selected, as shown in the schematic in figure 3.1.

The sequencer provides control of multiple external sensor devices [6]. Preprogrammed sequences can be triggered by general input/ output pins (GPIO) which could be utilised in providing the desired output, potentiometry or amperometry.

The electrodes are attached to the board via a USB to crocodile cable [10]. In amperometry, the current response is measured either through the low power transimpedance amplifier (LPTIA) for or high speed transimpedance amlpifier (HSTIA) depending on the bandwidth of the signal. The tran-

simpedance amplifiers (TIA) convert currents into voltages which are then converted into digital values by the ADC to be outputted by the board [19].

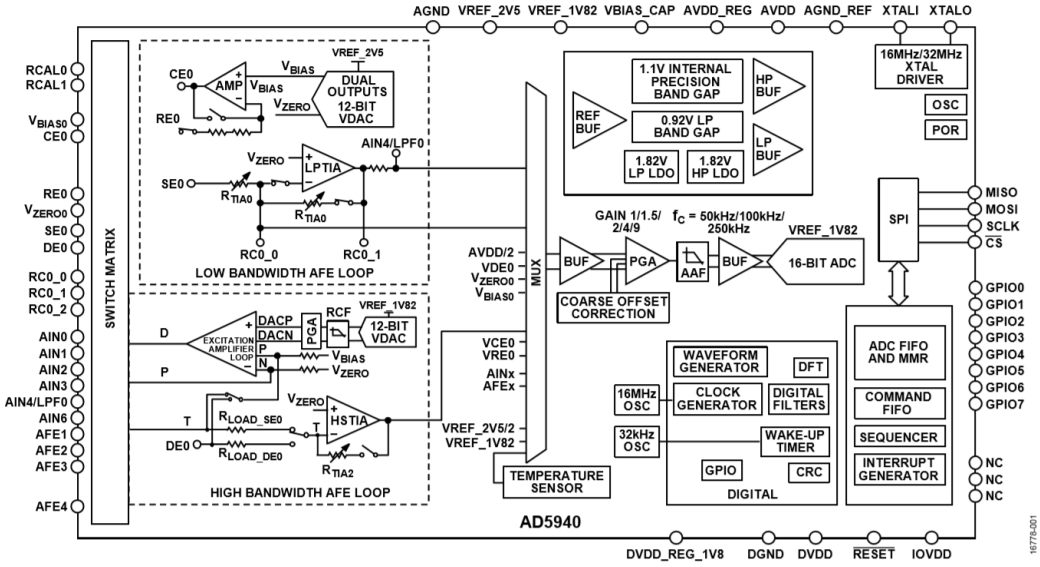

Figure 3.1: AD5940 functional block diagram [6]

The signal from the electrodes to the output can be followed through the flowchart in figure 3.2.

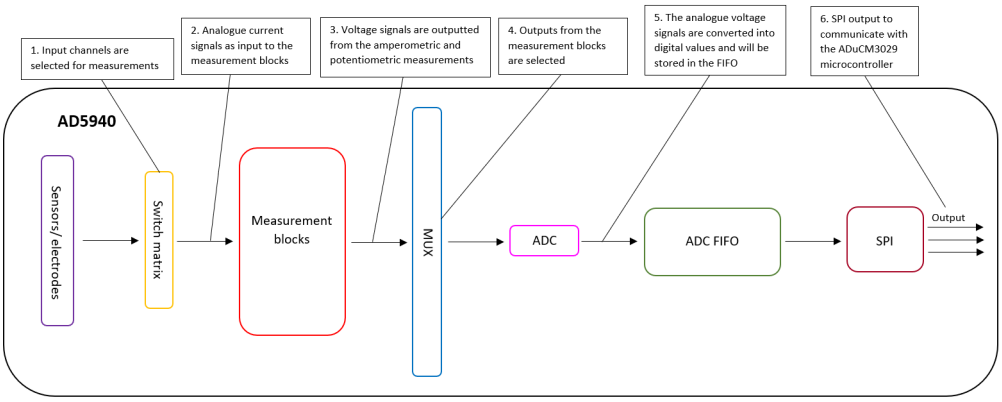

Figure 3.2: The signal path in the AD5940

3.1.2 Communicating with the AD5940

The AD5940 measurement blocks are usually controlled through SPI by the ADuCM3029 microcontroller on the EVAL-ADICUP3029 (the main motherboard) that comes with the evaluation kit [18]. The output of the measurements is meant to be viewed over UART by connecting the board to a computer using a micro USB [19, 9, 21]. Therefore a method of communication between the AD5940 and the Raspberry Pi considered at the start of this project was through the UART. Directly intercepting the output between the AD5940 and the ADuCM3029 microcontroller at the SPI level, bypassing the AduCM3029, to interface to the Raspberry Pi was also put under consideration.

3.1.3 Software

The software components of this project consists of the GUI and the SPI implementation. These are executed using various Python libraries on the

Raspberry Pi.

A calibration algorithm must be incorporated within the GUI in order to correspond the output of the potetionmetric measurements from the AD5940 to readable outputs, such as pH values. This can be built by testing the probes in solutions of a range of pH values. A trend, such as a linear equation [3, 24], should then be found from the values, the characteristics of which would then be used to plot the output potentiometric data in real-time.

3.1.4 GUI

The GUI will be implemented on the Raspberry Pi in Python using the `Tkinter` library. The outputted measurements are displayed through a real time plot of the measured data using the `Matplotlib` library.

3.2 Electrode configuration

As the project progressed, it was clear that the AD5940 is unable to output more than one measurement result per transaction. In spite of this, it was decided that both the measurements can still be outputted if the ADC multiplexer can be configured to switch between its two inputs, amperometry and potentiometry, at a regular time interval to output the results in a staggered manner.

The electrodes for amperometric measurements will be connected to the **RE0**, **CE0**, and **SE0** pins of the AD5940 and the ISE used for measuring potentiometric results will be connected to the analog input pin **DE0**, the voltage of which can be directly measured and converted by the ADC.

3.3 Registers configuration

The AD5940 evaluation board is configured through accessing certain registers on the board.

First, for the AD5940 to function a series of registers must be initialised and set to a certain value, as shown in table 3.1.

Register	Value
0x0908	0x02C9
0x0C08	0x206C
0x21F0	0x0010
0x0410	0x02C9
0x0A28	0x0009
0x238C	0x0104
0x0A04	0x4859
0x0A04	0xF27B
0x0A00	0x8009
0x22F0	0x0000

Table 3.1: AD5940 initialisation registers [6]

Certain bits of the configuration register, AFECON, with the address of 0x2000 [6], must also be set to enable the required functions of the AD5940, such as the ADC. Additionally, the LPTIA measurement block must be configured to perform amperometry measurements through the LPTIASW0 register with address 0x20E4.

The results of the measurements are read from the ADC output from the register named ADCDAT with address 0x2074, which produces a 16-bit un-

signed integer output.

The input of the ADC can be configured using the register 0x21A8 named ADCCON, by writing to it the sources of the positive and negative inputs to its multiplexer. The inputs considered in this project for amperometric and potentiometric measurements are detailed in tables 3.2 and 3.3 respectively.

ADCCON bits	Value	Description
[5:0]	00010	The Low Power TIA low pass output
[8:12]	00010	The Low Power TIA reference voltage

Table 3.2: Inputs to the ADC for amperometric measurements [6]

ADCCON bits	Value	Description
[5:0]	01101	`DE0` pin voltage
[8:12]	01000	`VBIAS_CAP` pin voltage reference, typically 1.11V [6]

Table 3.3: Inputs to the ADC for potentiometric measurements [6]

3.4 Communication with the Raspberry Pi

Since the AD5940 data sheet was very comprehensive about its SPI operations, the SPI protocol is used in the communication between the Raspberry Pi and the AD5940. This meant that the EVAL-ADICUP3029 board needs to be omitted from the system as it would use the SPI bus to communicate with the AD5940 otherwise. Which also meant that the AD5940 source code libraries provided by Analog Devices aren't being used and the necessary parts must be rewritten on the Raspberry Pi. However, using the SPI was deemed to be the best compromise given the timescale of the project.

3.4.1 Set up

The AD5940 board is set up with the Raspberry Pi as shown in figure 3.3.

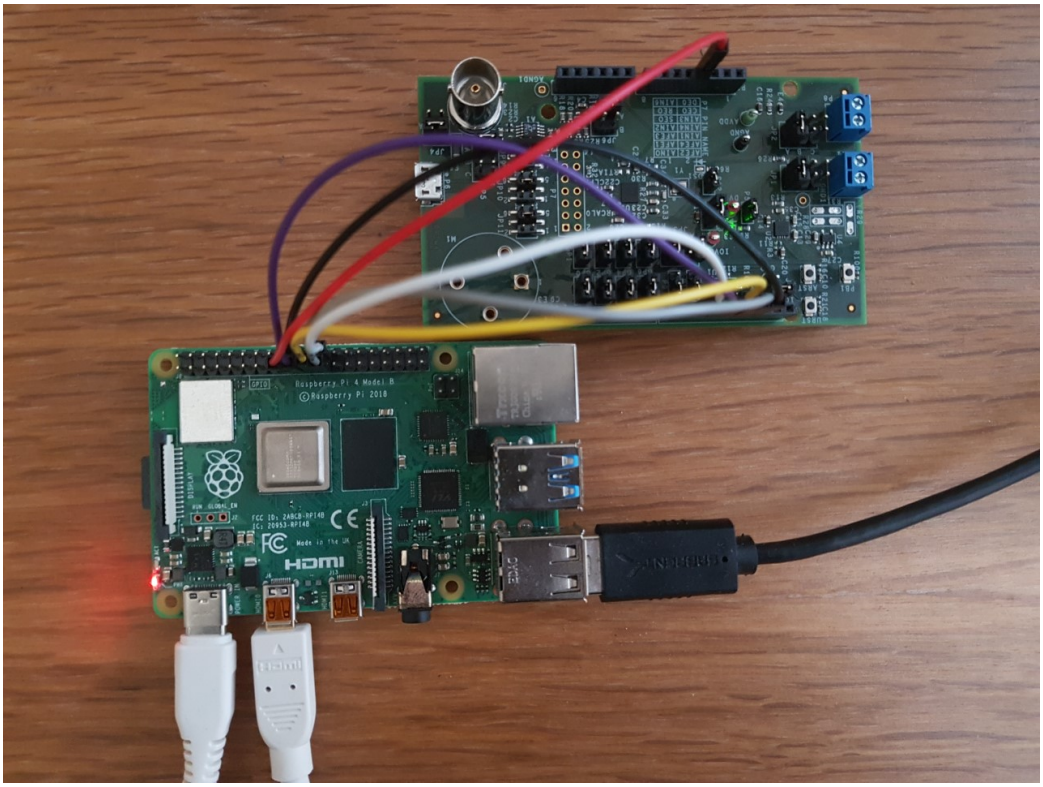

Figure 3.3: AD5940 (top of the picture) interface with the Raspberry Pi (bottom of the picture)

The Raspberry Pi acts as the master device and the AD5940 the slave device. Four connections must be made between the two devices:

- Master Out Slave In (MOSI), data input line driven from the Raspberry Pi to the AD5940.

- Master In Slave out (MISO), data output line from the AD5940 to the Raspberry Pi.
- Chip Select $\overline{CS}$, enable signal indicating the beginning and end of transaction.
- Serial Clock (SCLK), clock signal of maximum frequency 16MHz driven by the Rapsberry Pi to the AD5940.

The AD5940 is powered by the 3.3V supply pin on the Raspberry Pi. The SPI interface is connected by four wires for the MOSI, MISO, CS and SCLK pins on each device.

The Raspberry Pi is connected to a monitor, keyboard and mouse and is powered through the mains supply. The GUI is displayed through the monitor and is navigated using the keyboard and mouse. The GUI will be directly run and displayed on the desktop on the Raspberry Pi interface, shown in figure 3.4.

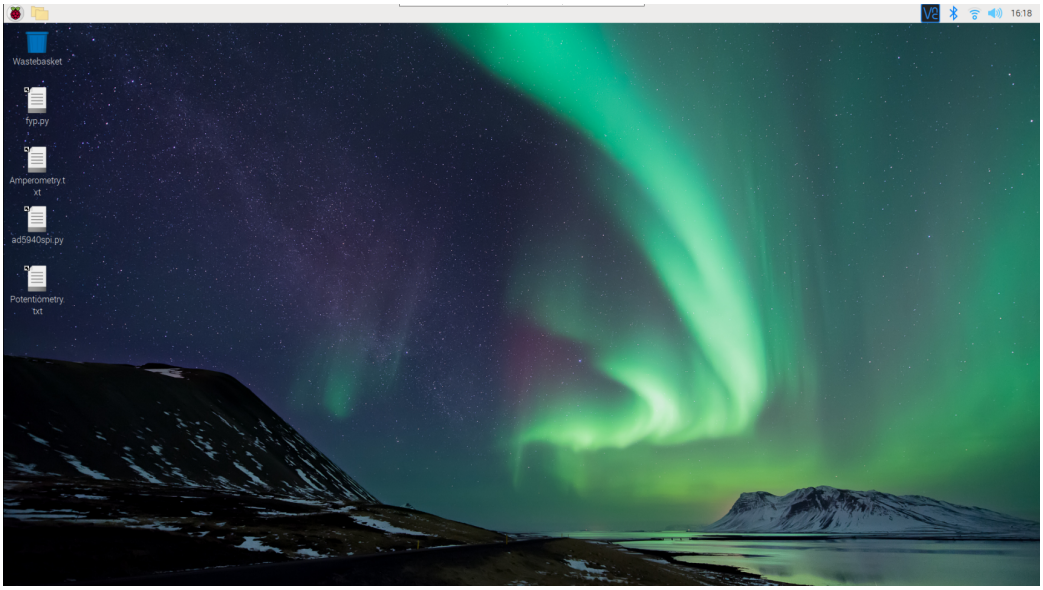

Figure 3.4: The Raspberry Pi Desktop

3.4.2 AD5940 SPI protocol

The SPI protocol used in the communication between the AD5940 and the Raspberry Pi relies on specific command bytes sent to the AD5940 to indicate a certain action taking place, detailed in table 3.4.

Command	Value	Description
`SPICMD_SETADDR`	0x20	Sets register address for SPI transaction
`SPICMD_READREG`	0x6D	Specifies SPI read trasnsaction
`SPICMD_WRITEREG`	0x2D	Specifies SPI write transaction

Table 3.4: AD5940 SPI commands [6]

Every transaction must start with indicating the address with which the master device will be communicating to or receiving from, using the `SPICMD_SETADDR` command byte. This is done by:

1. Driving $\overline{CS}$ low
2. Sending the command byte `SPICMD_SETADDR` followed by the 16-bit address of the register to read or write from.
3. Pulling $\overline{CS}$ high [6].

This is followed by the read or write command, done in a similar fashion.

Read:

1. Drive $\overline{CS}$ low
2. Send the command byte `SPICMD_READREG` followed by a dummy byte to initiate the SPI read.
3. Read returning 16-bit or 32-bit data.
4. Pull $\overline{CS}$ high [6].

Write:

1. Drive $\overline{CS}$ low
2. Send the command byte `SPICMD_WRITEREG` followed by the 16-bit or 32-bit data to write to the register.
3. Pull $\overline{CS}$ high. [6]

When $\overline{CS}$ is driven low, a multiple of eight clock cycles must be generated by the master device before the transaction [6].

Chapter 4

Software

In this chapter, the **implementations** of the requirements mentioned hitherto in previous chapters will be presented and explained. First **the SPI protocol**, how it reads, writes and is used to configure the AD5940 evaluation board into performing the tasks required of it in this project. Then the back-end of **the GUI** is shown, expanding on the two main parts: the **calibration** and **the real-time output** of data; the finalised GUI displayed is presented.

4.1 SPI implementation on the Raspberry Pi

The SPI interface is implemented on the Raspberry Pi using the `spidev` library [8]. The Raspberry Pi pins are shown in figure 4.1, set up as shown in figure 3.3 in **section 3.4.1**. Four main functions are used: *readbytes*,

writebytes, *xfer2* and *cshigh* [33].

Xfer2() *xfer2* performs an SPI transaction, the chip-select signal is held active in between blocks.

Cshigh *cshigh* controls the chip select signal. Since the AD5940 has an active low chip select pin, setting *cshigh* = **True** will drive the pin low.

Readbytes() The *readbytes* function reads from the slave device. It doesn't control the chip select signal and therefore *cshigh* must be set to **True** before the read transaction. The input argument to the *readbytes* function is the number of bytes it should read.

Writebytes() The *writebytes* function writes to the register specified on the slave device. Like the *readbytes* function, the chip select signal must be controlled using *cshigh* for the transaction.

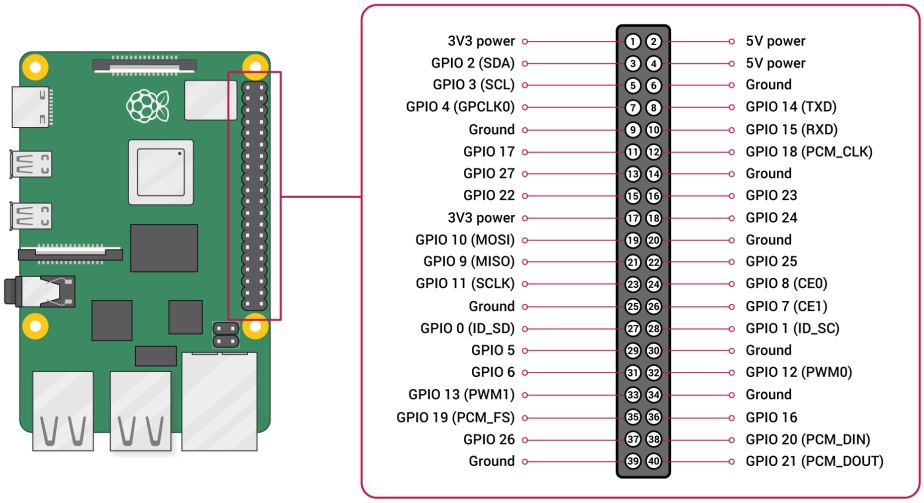

Figure 4.1: Raspberry Pi pins [29]

4.2 SPI code

The SPI protocol on the Raspberry Pi is contained in one Python file, ***ad5940spi.py***. The file contains several functions that perform functions that initialise, write to, and read from particular registers on the AD5940 board. The libraries `spidev` and `time` are imported.

4.2.1 Opening the SPI bus

To initiate and indicate the pins for the SPI connection, the bus on the Raspberry Pi must be opened. The Raspberry Pi 4 has one SPI bus - the SPI bus 0. The Raspberry Pi has two chip select pins, CE0 and CE1, to allow it to connect to more than one slave device. CE0 is used in this instance [15].

The SPI mode dictates the edge of the clock on which data is sampled and shifted out. A total of 4 modes exist when combined with the two different states of the clock polarity [12, 7]. The clock polarity can be either high or low during the idle state, which defines the two segments of time on either end of the transaction during which the $\overline{CS}$ goes from high to low at the beginning of the transaction and back up to high at the end. The AD5940's SPI timing specifications require the MOSI and MISO to be launched on the falling edge of the serial clock (SCLK) and sampled on the rising edge of the SCLK. The clock polarity of the AD5940 seems to be 0 judging from the SPI timing diagram, shown in figure 4.2. Which means that the idle state of the clock signal is low [7] and therefore needs to operate on SPI mode 1.

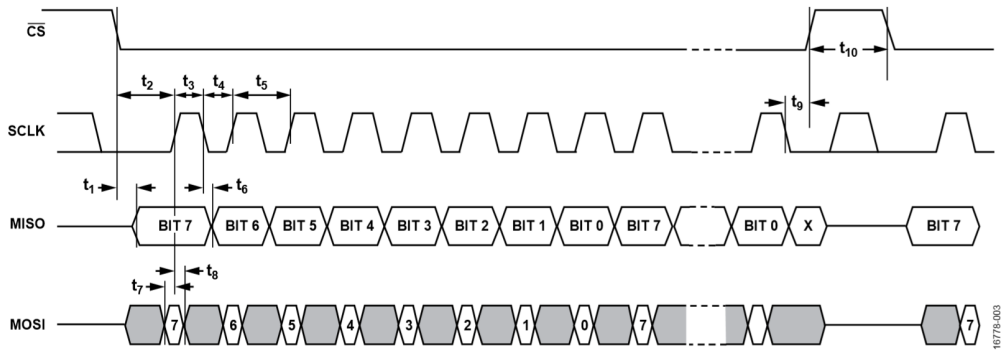

Figure 4.2: The AD5940 SPI timing diagram

Another requirement of the SPI transaction to the AD5940 is that the speed of the serial clock and thus the SPI transaction has to be slower than the system clock on the AD5940, which is 16MHz [6].

```
1         #chip select pin
2         CS = 0
3
```

```
4         #SPI bus 0 on the Pi
5         bus = 0
6
7         #enable spi
8         spi = spidev.SpiDev()
9
10        #open connection to bus and chip select pin
11        spi.open(bus, CS)
12
13        spi.max_speed_hz = 5000
14        spi.mode = 1
```

Listing 4.1: SPI set-up

4.2.2 Command bytes

The command bytes used to indicate a certain type of transaction within the AD5940 detailed in **table 3.4** are defined.

```
1 SPICMD_SETADDR = 0x20
2 SPICMD_READREG = 0x6D
3 SPICMD_WRITEREG = 0x2D
4 SPICMD_READFIFO = 0x5F
```

Listing 4.2: SPI command bytes

4.2.3 Register initialisation

The required initialisation of the registers shown in **table 3.1** is implemented, in the methods described in **sections 3.4.2** and **4.1**.

```
def Ad5940_Init(): #initialising the AD5940 after every reset

    add_1 = [SPICMD_SETADDR, 0x0908]
    spi.xfer2(add_1)
    init_1 = [SPICMD_WRITEREG, 0x02C9]
    spi.xfer2(init_1)
    add_2 = [SPICMD_SETADDR, 0x0C08]
    spi.xfer2(add_2)
    init_2 = [SPICMD_WRITEREG, 0x206C]
    spi.xfer2(init_2)
    add_3 = [SPICMD_SETADDR, 0x21F0]
    spi.xfer2(add_3)
    init_3 = [SPICMD_WRITEREG, 0x0010]
    spi.xfer22(init_3)
    add_4 = [SPICMD_SETADDR, 0x0410]
    spi.xfer(add_4)
    init_4 = [SPICMD_WRITEREG, 0x02C9]
    spi.xfer2(init_4)
    add_5 = [SPICMD_SETADDR, 0x0A28]
    spi.xfer2(add_5)
    init_5 = [SPICMD_WRITEREG, 0x0009]
    spi.xfer2(init_5)
    add_6 = [SPICMD_SETADDR, 0x238C]
    spi.xfer2(add_6)
    init_6 = [SPICMD_WRITEREG, 0x0104]
    spi.xfer2(init_6)
    add_7 = [SPICMD_SETADDR, 0x0A04]
    spi.xfer2(add_7)
    init_8 = [SPICMD_WRITEREG, 0x4859]
    spi.xfer2(init_8)
    add_8 = [SPICMD_SETADDR, 0x0A04]
    spi.xfer2(add_8)
    init_8 = [SPICMD_WRITEREG, 0xF27B]
    spi.xfer2(init_8)
    add_9 = [SPICMD_SETADDR, 0x0A00]
```

```
36      spi.xfer2(add_9)
37      init_9 = [SPICMD_WRITEREG, 0x8009]
38      spi.xfer2(init_9)
39      add_10 = [SPICMD_SETADDR, 0x22F0]
40      spi.xfer2(add_10)
41      init_10 = [SPICMD_WRITEREG, 0x0000]
42      spi.xfer2(init_10)
43
44      #configuring the LPTIA
45      lptiasw0_addr = [SPICMD_SETADDR, 0x20E4]
46      spi.xfer2(lptiasw0_addr)
47      lptiasw0_conf = [SPICMD_WRITEREG, 0x302C]
48      spi.xfer2(lptiasw0_conf)
49
50      #AFECON
51      AFE_addr = [SPICMD_SETADDR, 0x2000]
52      spi.xfer2(AFE_addr)
53      AFE_conf = [SPICMD_WRITEREG, (0b11<<7)] # enable ADC
    power and conversion start enable
54      spi.xfer2(AFE_conf)
```

Listing 4.3: Initialisation registers

4.2.4 Reading data from the ADC

The data from the amperometric and potentiometric measurements are outputted by the ADC through the ADCDAT register and returned as a 16-bit unsigned integer, **val**.

```
1 def SpiRead_ADCDAT():
2     val = None
3
4     ResultsAddr = [SPICMD_SETADDR, 0x2074]
```

```
    spi.cshigh = True
    spi.writebytes(ResultsAddr)
    spi.cshigh = False
    #time.sleep(0.1) #pause

    spi.cshigh = True
    Read_init = [SPICMD_READREG, 0b00] #initialise read
    spi.writebytes(Read_init)
    val = spi.readbytes(2) #reading 2 bytes of data from
   ADCDAT register
    spi.cshigh = False

    return val
```

Listing 4.4: SPI read function

4.2.5 Concurrent measurement configuration

To output both amperometric and potentiomtric measurements onto the Raspberry Pi, the input to the ADC must be switched between the output of the LPTIA for amperometry and the analog **DE0** pin for potentiometry. The following functions configure the ADC for amperometry (`ADCCON_amp`) and potentiometry (`ADCCON_pot`) through the ADCCON register by writing to it the bits that specify the positive and negative inputs mentioned in **section 3.3**.

```
AMP_MUXP = 0b00010 #LPTIA positive input
AMP_MUXN = 0b00010 #LPTIA negative input
POT_MUXP = 0b01101 #DE0
POT_MUXN = 0b01000 #VBIAS_CAP

def ADCCON_amp(): #amperometric inputs
```

```
7
8       ConfAddr = [SPICMD_SETADDR, 0x21A8]
9       spi.cshigh = True
10      spi.writebytes(ConfAddr)
11      spi.cshigh = False
12
13      spi.cshigh = True
14      spi.writebytes((AMP_MUXN<<8)|(AMP_MUXP))
15      spi.cshigh = False
16
17  def ADCCON_pot(): #potentiometric inputs
18
19      ConfAddr = [SPICMD_SETADDR, 0x21A8]
20      spi.cshigh = True
21      spi.writebytes(ConfAddr)
22      spi.cshigh = False
23
24      spi.cshigh = True
25      spi.writebytes((POT_MUXN<<8)|(POT_MUXP))
26      spi.cshigh = False
```

Listing 4.5: SPI measurement configuration

4.3 The GUI

4.3.1 GUI layout

The GUI is built on the Raspberry Pi using the `Tkinter` library in Python [30]. The `Tkinter` library provides three built-in layout managers, `pack`, `grid` and `place`. In this project, the `grid` manager is chosen to specify the position of elements in the GUI without having to use the absolute positioning specified by pixels as is the case with the `place` manager, or the inability

to customise at all in the case of the `pack` manager.

An initial design of the GUI layout is shown in figure 4.3. The `grid` manager is used for the GUI window and the `frame` widget, the latter of which is used to contain the calibration features and the GUI title. The real-time plot is displayed on the `Tkinter canvas` widget. The plot comes with a navigation toolbar feature which allows the user to zoom into the plot, save the plot and view values on the plot.

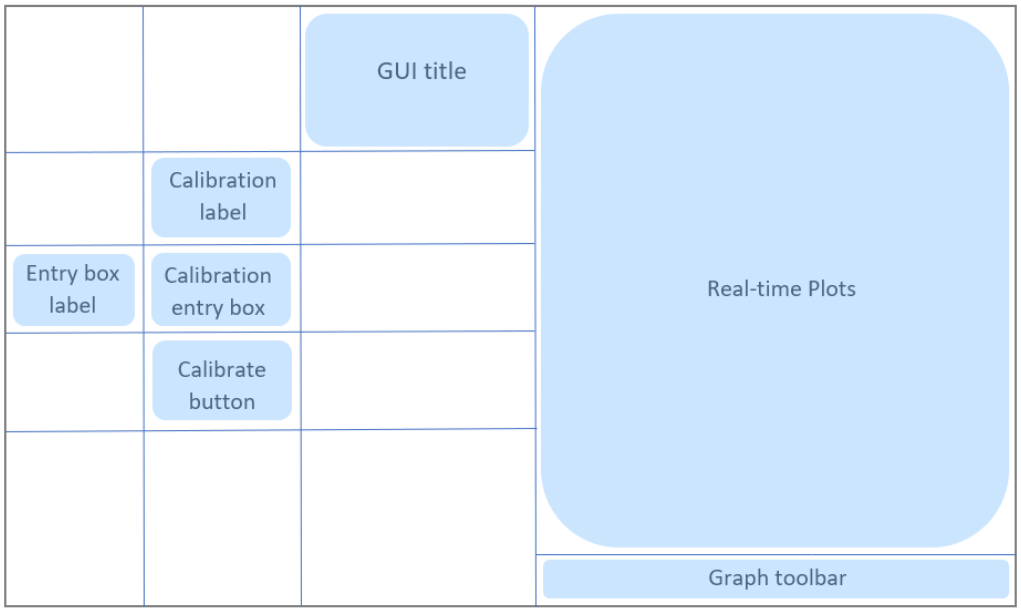

Figure 4.3: Initial design of the GUI layout

The final GUI implementation is shown in figure 4.4.

Figure 4.4: Final GUI design

4.3.2 Calibration

The calibration is done using the *polyfit* function of the `NumPy` library [17]. An input of the pH value from the user and the resulting output voltage corresponding to that inputted pH will be appended to their respective lists. These lists will correspond to the **x** and **y** values used to calculate attributes of the line of best fit. The attributes, the gradient and the y-intercept, of that straight line is found using the *polyfit* function. Using these attributes, the voltage outputted by the measurement blocks can then be mapped to their approximate corresponding pH, and subsequently plotted in real-time.

The function `get_mb` obtains the global variables **m** and **b** which represent the gradient and y-intercept respectively, later used in the real-time plot of the potentiometric results.

```
m = None
b = None

def get_mb():

    global m #gradient
    global b #y intercept

    x = [3, 5, 7, 9] #pH values
    y = [3.3211, 2.0242, 1.8564, 1.8615]
    #voltage values corresponding to the pH values

    m, b = np.polyfit(x, y, 1)
    #potentionmetric results = m*pH + b
```

Listing 4.6: Function to find line of best fit attributes

With the default values of **x** and **y** in the function, the calculated **m** and **b** are: -0.227 and 3.630 respectively.

The **x** list correspond to the pH values and the **y** lists, their respective potentiometric voltages. The GUI, as shown in figure 4.4, features an entry box for the user input of pH values which, when combined with the SPI read of the ADCDAT register in the AD5940 would ideally provide the values in the **x** and **y** lists instead. In this case however, the values are in the function by default. The potentiometric voltage values are obtained from the first value of each pH from pre-obtained data. The `get_mb` function is called when the "Calibrate" button is clicked. In this case, the button needs to only be clicked once before the **m** and **b** values are obtained.

```
button_calibrate = ttk.Button(frame, text="Calibrate",
```

```
2     command=get_mb)
```

Listing 4.7: Calibration button code

4.3.3 Real-time plot

The output data is plotted using the `Matplotlib` library [26]. The plot in real-time is achieved using its *animation* function [32, 13], which samples the input from the SPI at a regular interval and plots it against time.

With no real data input from the SPI bus, previously simulated data for a constant pH of 9 was read from a text file, `Potentiometry.txt` and outputted at time intervals of one second for each value in the file. The resulting values of **m** and **b** from the calibration function, `get_mb`, is used to convert the outputted potentiometric voltage into pH values on line 21.

The potentiometric plot function is shown below, the amperometric function plots the amperometric current in a similar way except for the conversion into pH.

```
1  yp = []
2  t = []
3  fig_p = plt.Figure(figsize = (9, 4))
4  p = fig_p.add_subplot(111)
5  pot_voltage = []
6  indexp = count()
7
8  def animate_p(i, t, yp):
9      global pot_voltage
10
11     if m != None:
```

```
12          data_pot = open('Potentiometry.txt', "r")
13          lines = data_pot.readlines()
14          data_pot.close()
15
16          for num in lines:
17              value = float(num)
18              pot_voltage.append(value)
19
20          xp = pot_voltage[next(indexp)]
21          pH_pot = (xp - b)/m
22          yp.append(pH_pot)
23          t.append(dt.datetime.now().strftime('%H:%M:%S'))
24          t = t[-50:]
25          yp = yp[-50:] #x and y limits t-50
26
27          p.clear()
28          p.plot(t, yp, color="g")
29          p.set_title("Potentiometry pH")
30          p.set_xticklabels(t, rotation=45, ha='right')
31          p.set_ylabel("pH")
32          fig_p.subplots_adjust(left=0.15, bottom=0.15)
```

Listing 4.8: Real-time plotting function

Since the values of the variables **m** and **b** are initialised to the type **None** the potentiometric data can only get plotted once the values of **m** and **b** have been calculated through calibration. The logic of this part of the function is shown in a flow diagram in figure 4.5 where the user actions for each stage of the process is situated in the orange boxes on the right.

The `for` loop converts the strings into integers, `int`, form as the data read from the text file is stored as an array of strings.

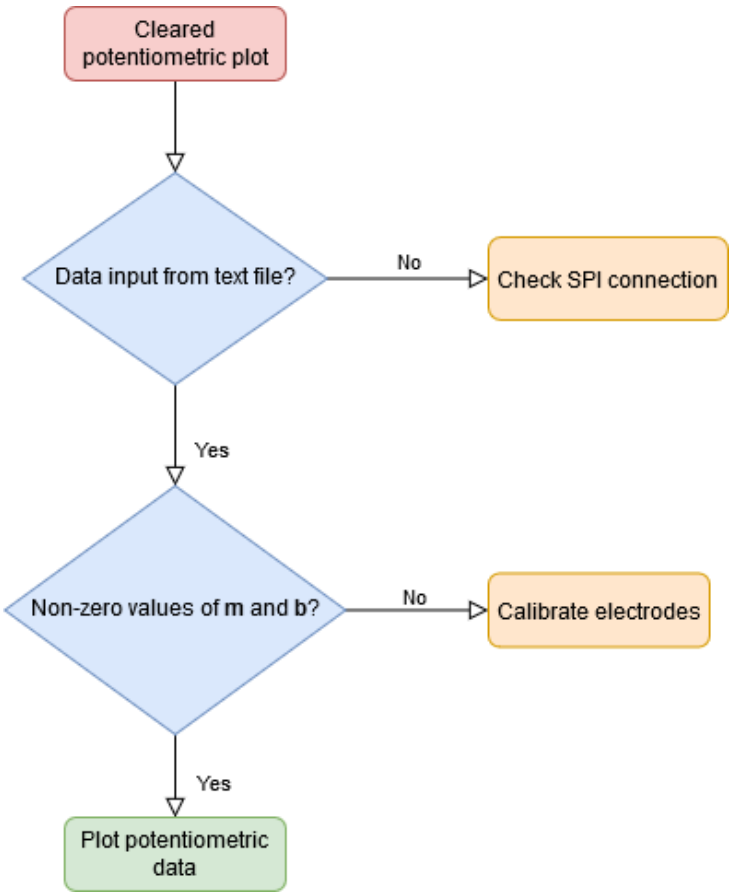

Figure 4.5: Logic flowchart for function `animate_p`

The `animate` function is called by the `FuncAnimation` class [27] in the `Animation` object of `Matplotlib`:

```
ani_pot = animation.FuncAnimation(fig_p, animate_p,
    fargs=(t, yp), interval=1000)
```

Listing 4.9: Real-time plotting potentiometric results code

The input argument, **interval**, indicates the time interval between plotting

each point in milliseconds. Here one point is plotted every second to allow the user to see the data being plotted and catch any anomalies.

Chapter 5

Components testing

This chapter explains the way different parts of the **system is tested**, the **results** from those tests and the conclusions and evaluation drawn from them.

5.1 The SPI connection

The SPI functions on the Raspberry Pi was tested using the `init_check` and `CS_low` function. The *readbytes* function is used to read from the identification registers of the AD5940 which should return constant values of 0x4144. The `CS_low` function is used to test the *cshigh* structure of `spidev`, using a multi-meter. This made sure of the kinf of change in the state of the CS pin on the AD5940 as a result of *cshigh* being set to **True** or **False**.

```
1
2 def init_check():
3
```

```
    check_addr = [SPICMD_SETADDR, 0x0400] #ADIID test
   register
    spi.cshigh = True #cs low
    #CS_status = GPIO.input(channel)
    #print(CS_status)
    spi.writebytes(check_addr)
    spi.cshigh = False

    spi.cshigh = True
    Read_check = [SPICMD_READREG, 0x00]
    spi.writebytes(Read_check)
    var = spi.readbytes(1)
    spi.cshigh = False

    print(hex(var[0]))

def CS_low():
    spi.cshigh = True
```

Listing 5.1: Testing the SPI transaction

However, the SPI read bus returned with 0x0. This would normally be debugged by connecting the pins to the oscilloscope for the SPI transaction signals to be looked at in detail. In the current circumstances this was not possible.

The problem could be that the AD5940 needs to receive a multiple of eight clock cycles after the $\overline{CS}$ signal is brought low, which hasn't been implemented in the software part of the system yet, for the reason that the SPI protocol implemented by the `spidev` library has transaction and switching delays which need to be inspected before determining the right amount of clock cycles to implement following the $\overline{CS}$ signal. The connection between the Raspberry Pi and the AD5940 has therefore not been successfully estab-

lished from theory.

5.2 Potentiometric tests

Should the SPI connection work, the potentiometric measurements would be tested with the ISE submerged in a solution such as Hydrogen Peroxide. This can also be tested with a battery connected to the analog input pin, DE0 and ground without access to laboratory equipment and solutions.

5.3 Amperometric tests

Similarly, the amperometric measurements would ideally be tested with the platinum electrodes submerged in the same solution as the ISE in potentiometry, such as Hydrogen Peroxide. This can also be tested with a battery connected in parallel with a resistor to generate a current from ground to the CE0 counter electrode pin. This current can then be measured by the transimpedance amplifier.

Chapter 6

Conclusion

6.1 Revisiting project aims and deliverables

The project deliverables stated the following:

System requirements

- ☑ To explore the set up of a concurrent amperometry and potentiometry measurements using embedded, off-the-shelf components
- ☐ To test the individual components of the test-bench system and evaluate the functionalities of the system as a whole
- ☑ A GUI for displaying real-time potentiometry and amperometry results
- ☑ The test-bench should enable user to calibrate output data by measuring results from input pH

6.2 Evaluation

The requirements for the GUI have been met, although the user-friendliness and aesthetic of the interface could be improved upon.

The system requirements were more difficult to meet and assembling the different components of the test-bench was challenging. The SPI connection between the AD5940 and the Raspberry Pi could not be properly tested with the lack of an oscilloscope and lab equipment. Fortunately, some parts of the protocol were debugged using a multimeter. The code relating to the SPI interface has been written to work theoretically, shown subsequently in **section 7**.

Unfortunately, the different components of the test-bench could not be tested together and ultimately the test-bench could not be tested as a whole on pH solutions due to the lack of lab access.

The practical aspects of the project became more orientated towards the research of the AD5940 module, its ability to perform concurrent sensing with the Raspberry Pi, and the theoretical usage of the two devices together in providing a GUI for use in the test-bench.

Chapter 7

Future work

7.1 SPI Incorporated functions

The functions described previously in **sections 4.3.2** and **4.3.3** will need to be altered when communicating to the AD5940 via the SPI bus using functions for the SPI protocol detailed in **section 4.2**.

Firstly, the `get_mb` function which obtains the gradient and y-intercept of the line of best fit for potentiometric data must correspond the user-input to the data read by the SPI bus. This is done by appending the user-input of the pH to an array named `pHs`, and appending the data read by the SPI bus to an array named `op`.

```
1 m = None
2 b = None
3 pHs = []
4 op = []
5
6 def get_mb(pH_val):
7
```

```
8       global pHs
9       global op
10      global m #gradient
11      global b #y intercept
12
13      pHs.append(pH_val)
14      val = SpiRead_ADCDAT() #read data from ADC
15      op.append(val)
16
17      m, b = np.polyfit(pHs, op, 1)
18      #ADCDAT = m*pH + b
```

Listing 7.1: Line of best fit function with SPI input

The `get_mb` function is called when the "Calibrate" button is clicked on, as before. The user-inputted pH value is taken as a variable, `pH_val`, from the entry box, `entry_pH`.

```
1 entry_pH = tk.Entry(frame, textvariable=pH_val)
2 button_calibrate = ttk.Button(frame, text="Calibrate",
3     command=get_mb(pH_val))
```

Listing 7.2: SPI set-up

The animation function must also be altered such that the input data to the real-time plot isn't obtained from a text file but rather from the SPI input directly. Every time the `animate_p` function is called it calls the `SpiRead_ADCDAT` function from ***ad5940spi.py*** file which reads from the `ADCDAT` register returning a 16-bit unsigned number.

```
1 yp = []
2 t = []
3 fig_p = plt.Figure(figsize = (9, 4))
```

```
p = fig_p.add_subplot(111)
pot_voltage = None

def animate_p(i, t, yp):
    global pot_voltage


    if m != None:
        pot_voltage = ad5940spi.SpiRead_ADCDAT()
        pH_pot = (pot_voltage - bp)/mp
        yp.append(pH_pot)

        t.append(dt.datetime.now().strftime('%H:%M:%S'))
        t = t[-50:]
        yp = yp[-50:] #x and y limits t-20
        p.clear()
        p.plot(t, yp, color="g")

        p.set_title("Potentiometry pH")
        p.set_xticklabels(t, rotation=45, ha='right')
        p.set_ylabel("pH")
        fig_p.subplots_adjust(left=0.15, bottom=0.15)
```

Listing 7.3: Real-time plotting function with SPI

7.2 Further automation

The system could benefit from being more automated, for optimising the purposes of the test-bench serving as a control for gauging electrochemical measurements as explained in the introduction (**section 1.1**) of this report. One way to further automise this test-bench is to include a flow control valve to precisely inject the solvents.

7.3 Reference voltages and electrode configuration

For the purposes of concurrent sensing, there need to be tests to determine whether the amperometric circuit and the potentiometric circuit can utilise the same reference electrode as this may affect the results chemically.

7.4 Using the sequencer

The sequencer on the AD5940 has the capability to have up to 4 preprogrammed sequences which can be triggered by GPIO pins to implement a predestined function such as performing potentiometric or amperometric measurements.

The sequences' most significant bit (MSB) indiicates the function it is to perform, such as the write or the timer function. This is followed by a 16-bit address between the values 0x0000 and 0x21FC. The address field in the sequence is only 7-bits wide, namely bits [8:2] of the address used by the external controller [6]. This is then followed by the 24-bit wide data to be written into the address.

7.5 I^2C protocol

I^2C is possible on the board albeit the seemingly lack of information in the datasheets and source code regarding how it can be used to access the registers on the AD5940 to configure the board and read the results. Though if it were possible, it would make communicating with the Raspberry Pi easier as the information for the library for I^2C protocol, `SMBus`, seems to be more

accessible and easier to implement.

Using the I^2C protocol would also allow simultaneous usage of the microcontroller unit (MCU), which means the source code from the AD5940 library can be included and therefore wouldn't have to be re-written on the Raspberry Pi.

Bibliography

[1] K A M Akbar, R V Manurung, and B Mulyanti. "Development of K500 Signal Processing System using Raspberry PI in the 3-electrodes Electrochemical Sensor". In: *IOP Conf. Series: Materials Science and Engineering* 384 (2018), p. 1. DOI: doi:10.1088/1757-899X/384/1/012030.

[2] Stefano Cagninn et al. "Overview of Electrochemical DNA Biosensors: New Approaches to Detect the Expression of Life". In: *Sensors* (Apr. 2009). DOI: doi:10.3390/s90403122.

[3] *Calibrating an Analog pH Sensor*. URL: https://www.youtube.com/watch?v=9EIbTbh80gA&t=613s. (accessed: 28.04.2020).

[4] Adetunji Charles Oluwaseun, Paomipem Phazang, and Neera Bhalla Sarin. "Biosensors: A Fast-Growing Technology for Pathogen Detection in Agriculture and Food Sector". In: *Sensors* (Apr. 2009). DOI: doi:10.3390/s90403122.

[5] J E Choi et al. "Intraoral pH and Temperature During Sleep With and Without Mouth Breathing". In: *Oral Rehabilitation* 43 (2015), pp. 356–363. DOI: 10.1111/joor.12372.

[6] Analog Devices. *AD5940/AD5941 Datasheet - High Precision, Impedance, and Electrochemical Front End*.

[7] Piyu Dhaker. *Introduction to SPI Interface*. URL: https://kite.com/python/answers/how-to-plot-a-line-of-best-fit-in-python. (accessed: 01.06.2020).

[8] doceme. *py-SpiDev*. URL: https://github.com/doceme/py-spidev. (accessed: 01.06.2020).

[9] Analog Electronics. *AD5940 ChronoAmperometric*. URL: https://wiki.analog.com/resources/eval/user-guides/eval-ad5940/software_examples/ad5940_chronoamperometric. (accessed: 28.04.2020).

[10] Analog Electronics. *AD5940 Electrochemical Shield User Guide*. URL: https://wiki.analog.com/resources/eval/user-guides/eval-ad5940/hardware/eval-ad5940elcz. (accessed: 28.04.2020).

[11] Pantelis Georgiou. *Biomedical Electronics*. Department of Electrical and Electronic Engineering Imperial College London, 2020.

[12] Mark Hughes. *Back to Basics: SPI (Serial Peripheral Interface)*. URL: https://www.allaboutcircuits.com/technical-articles/spi-serial-peripheral-interface/. (accessed: 01.06.2020).

[13] Shawn Hymel. *Graph Sensor Data with Python and Matplotlib*. URL: https://learn.sparkfun.com/tutorials/graph-sensor-data-with-python-and-matplotlib/update-a-graph-in-real-time. (accessed: 01.06.2020).

[14] Texas Instruments. *LMP91000EVM Evaluation Module*. URL: http://www.ti.com/tool/LMP91000EVM. (accessed: 28.04.2020).

[15] Byron J and Shawn Hymel. *Raspberry Pi SPI and I2C Tutorial*. URL: https://learn.sparkfun.com/tutorials/raspberry-pi-spi-and-i2c-tutorial/all. (accessed: 01.06.2020).

[16] Allen J. Bard and Larry R. Faulkner. *ELECTROCHEMICAL METHODS, Fundamentals and Applications*. Vol. 2. John Wiley Sons, Inc, 1980. ISBN: 0-471-04372-9.

[17] Kite. *How to plot a line of best fit in Python*. URL: `https://www.analog.com/en/analog-dialogue/articles/introduction-to-spi-interface.html`. (accessed: 01.06.2020).

[18] Micheál Lambe. *Implementing the AD5940 and AD8233 in a Full Bioelectric System*.

[19] Micheál Lambe. *Optimizing the AD5940 for Electrochemical Measurements*.

[20] Haitao Li et al. "CMOS Electrochemical Instrumentation for Biosensor Microsystems: A Review". In: *Sensors* (2016), p. 4. DOI: `doi:10.3390/s17010074`.

[21] Raspberry Pi Trading Ltd. *The Raspberry Pi UARTs*. URL: `https://www.raspberrypi.org/documentation/configuration/uart.md`. (accessed: 28.04.2020).

[22] Catarina M. Abreu et al. "Emerging Biosensing Technologies for Neuroinflammatory and Neurodegenerative Disease Diagnostics". In: *Frontiers in Molecular Neuroscience* (May 2018). DOI: `https://doi.org/10.3389/fnmol.2018.00164`.

[23] Daryl Ma. *Motivation of Concurrent Sensing*.

[24] Daryl Ma. "Smart Orthodontic Bracket for In-mouth Metabolite Measurements". 2017, pp. 36–38.

[25] Daryl Ma, Sara S. Ghoreishizadeh, and Pantelis Georgiou. *DAPPER: A Low Power, Dual Amperometric and Potentiometric Single-Channel Front End*.

[26] Matplotlib. *Matplotlib version 3.2.1*. URL: https://matplotlib.org/. (accessed: 01.06.2020).

[27] Matplotlib. *matplotlib.animation.FuncAnimation*. URL: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.animation.FuncAnimation.html. (accessed: 01.06.2020).

[28] PalmSens. *Emstat Pico*.

[29] Raspberry Pi. *GPIO*. URL: https://www.raspberrypi.org/documentation/usage/gpio/. (accessed: 01.06.2020).

[30] Python. *24.1. Tkinter — Python interface to Tcl/Tk*. URL: https://docs.python.org/2/library/tkinter.html. (accessed: 01.06.2020).

[31] Hardik Rathod. *Control Arduino Using GUI (Arduino + Processing)*. URL: https://www.hackster.io/hardikrathod/control-arduino-using-gui-arduino-processing-2c9c6c. (accessed: 01.06.2020).

[32] Corey Schafer. *Matplotlib Tutorial (Part 9): Plotting Live Data in Real-Time*. URL: https://www.youtube.com/watch?v=Ercd-Ip5PfQ. (accessed: 01.06.2020).

[33] Radiostud.io Staff. *Understanding SPI Communication using Raspberry Pi*. URL: https://radiostud.io/understanding-spi-in-raspberry-pi/. (accessed: 01.06.2020).