DISSERTATION

CREXENS^{TM}: AN EXPANDABLE GENERAL-PURPOSE ELECTROCHEMICAL ANALYZER

Submitted by

Lang Yang

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2019

Doctoral Committee:

Advisor: Tom Chen

George J Collins
Jesse Wilson
Stuart Tobet

ABSTRACT

CREXENS™: AN EXPANDABLE GENERAL-PURPOSE ELECTROCHEMICAL ANALYZER

Electrochemical analysis has gained a great deal of attention of late due to its low-cost, easy-to-perform, and easy-to-miniaturize, especially in personal health care where accuracy and mobility are key factors to bring diagnostics to patients. According to data from Centers for Medicare & Medicaid Services (CMS) in the US [1], the share of health expenditure in the US has been kept growing in the past 3 decades and reached 17.9% of its overall Gross Domestic Product till 2016, which is equivalent to $10,348 for every person in the US per year. On the other hand, health care resources are often limited not only in rural area but also appeared in well-developed countries. The urgent need and the lack of health resource brings to front the research interest of Point-of-Care (PoC) diagnosis devices. Electrochemical methods have been largely adopted by chemist and biologist for their research purposes. However, several issues exist within current commercial benchtop instruments for electrochemical measurement. First of all, the current commercial instruments are usually bulky and do not have handheld feature for point-of-care applications and the cost are easily near $5,000 each or above. Secondly, most of the instruments do not have good integration level that can perform different types of electrochemical measurements for different applications. The last but not the least, the existing generic benchtops instruments for electrochemical measurements have complex operational procedures that require users to have a sufficient biochemistry and electrochemistry background to operate them correctly. The proposed Crexens™ analyzer platform is aimed to present an affordable electrochemical analyzerwhile achieving comparable performance to the existing commercial instruments, thus, making general electrochemical measurement applications accessible to general public.

In this dissertation, the overall Crexens^{TM} electrochemical analyzer architecture and its evolution are presented. The foundation of the Crexens^{TM} architecture was derived from two separate but related research in electrochemical sensing. One of them is a microelectrode sensor array using CMOS for neurotransmitter sensing; the other one is a DNA affinity-based capacitive sensor for infectious disease, such as ZIKA. The CMOS microelectrode sensor array achieved a 320uM sensitivity for norepinephrine, whereas the capacitive sensor achieved a dynamic range of detection from 1 /uL to 105 /uL target molecules (20 to 2 million targets), which makes it be within the detection range in a typical clinical application environment. This dissertation also covers the design details of the CMOS microelectrode array sensor and the capacitive sensor design as a prelude to the development of the Crexens^{TM} analyzer architecture.

Finally, an expandable integrated electrochemical analyzer architecture(Crexens^{TM}) has been designed for mobile point-of-care (POC) applications. Electrochemical methods have been explored in detecting various bio-molecules such as glucose, lactate, protein, DNA, neurotransmitter, steroid hormone, which resulted in good sensitivity and selectivity. The proposed system is capable of running electrochemical experiments including cyclic voltammetry (CV), electrochemical impedance spectroscopy (EIS), electrochemical capacitive spectroscopy (ECS), amperometry, potentiometry, and other derived electrochemical based tests. This system consist of a front-end interface to sensor electrodes, a back-end user interface on smart phone and PC, a base unit as master module, a low-noise add-on module, a high-speed add-on module, and a multi-channel add-on module. The architecture allows LEGO^{TM}-like capability to stack add-on modules on to the base-unit for performance enhancements in noise, speed or parallelism. The analyzer is capable of performing up to 1900 V/s CV with 10 mV step, up to 12 kHz EIS scan range and a limit of detection at 637 pA for amperometric applications with the base module. With high performance module, the EIS scan range can be extended upto 5 MHz. The limit of detection can be further improved to be at 333 fA using the low-noise module. The form factor of the electrochemical analyzer is designed for its mobile/point-of-care applications, integrating its entire functionality on to a 70 cm^2 area of surface space. A glutamine enzymatic sensor was used to valid the capability of

the proposed electrochemical analyzer and turned out to give good linearity and reached a limit of detection at 50 uM.

ACKNOWLEDGEMENTS

I would like to acknowledge and thank the following people and groups for their support in this work, without you none of this would have been possible. I would like to thank Dr. Tom Chen for his persistent support during my graduate school. His patience and perspective helped out in most of the bumps during research. He is also a good mentor in guiding me throughout my graduate career and advising for my long-term future. The past 6 years with him has become one of my most precious and enlightening experience in my life. John Wydallis and Rachel Feeny were the first couple of people I've been worked with when I just joined the research group. Their strict scholarship and talented inspiration introduced me into the career for a doctoral degree. The impact from them has been last over all these years on most of my decisions and plans for my researches. I appreciate the good example they've been presenting. I'd like to thank Jessie Filer for her persistence and patience during our collaboration. The fact that Jessie and me come from totally different background surprisingly worked well in our joint research. It's mostly because she always keeps an open mind to listen to different perspectives and knowledge, and eager to understand. I appreciate her patience to help me understand the story from bio side regardless how silly the question was. Additionally, she is also a diligent researcher that accelerate the progress with her persistence and all the late nights spent in lab. I'd also like to thank my group colleagues, William Tedjo, Abhiram Reddy Gundla and Yusra Obeidat for all the supporting and encouragement over the years as well as every valuable discussion that inspires me during research. I'd like to acknowledge August DeMann for his voluntary help in manufacturing our sensor for the research, some of my researches could be stopped without his help. Many thanks to my wife, Shumei. Thank you for keeping being my soul mate and being supportive. None of any of these could happen if you didn't decide to quit your job in China and come with me to CSU 6 years ago. The journey at CSU together between you and me not merely just witnessed the process in order to earn a doctoral degree, but also witnessed how you made me the luckiest person for having you in the rest of my life. Special thanks to my upcoming kid. Your arrival caught a good timing during

your parents' temporary separation. I appreciate the distraction you made to keep your mom busy in taking good care of your growth and health, and made your dad a secondary person so your mom can feel better from this separation. You've done great job to accompany your mom and cheer her up, which means a lot to this family. At last but not the least, I'd like to thank Stacy Willett, Milena Veselinovic and Lei Wang for being helpful during our collaboration and thanks to everyone who contributed, advised or helped in any way during my program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Health Care in the US and Beyond

Health care is a critical topic as it is related to every individual. In the United States, the health care expenditure is the highest among developed countries. According to the data from the National Health Expenditure Accounts (NHEA) as shown in Figure 1.1, US reached $ 3.3 trillion total health care expenditure, or $10,348 per person in 2016 [2]. This is about 17.9% of the gross domestic product (GDP). In other developed countries, such as Britain, Germany,and France, the spending on health care is around 10% of their GDP and the percentage is keeping growing over the years.

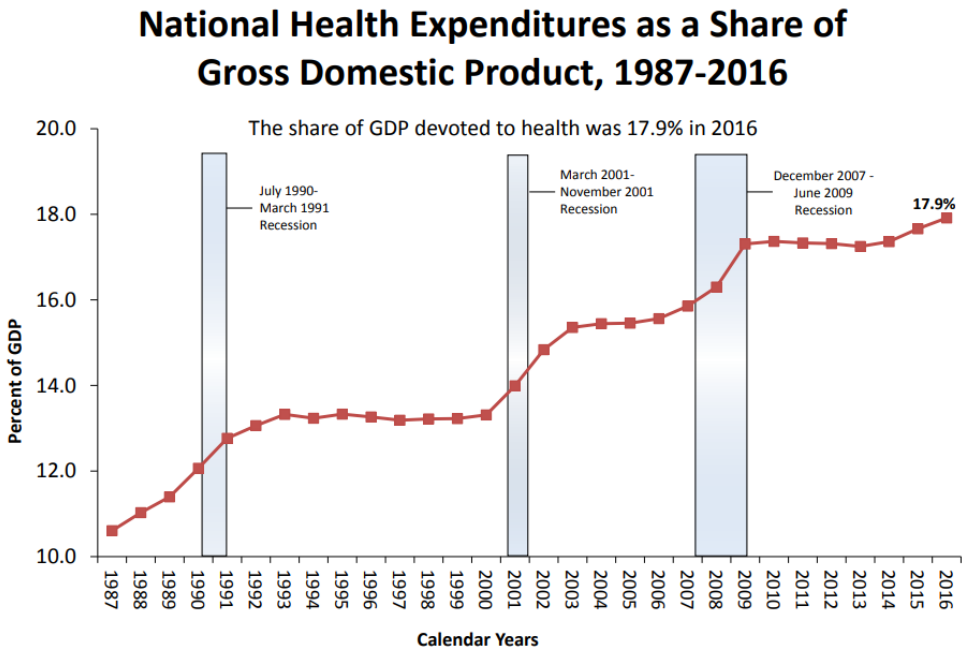


Figure 1.1: The health care expenditure as share of GDP [2].

While the market in health care is promising and prosperous, from a patient perspective, limited health care resource and the resulting frustration in waiting time can prevent needed diagnosis in a

timely fashion. Data from the Commonwealth Fund in 2013 indicated 76% of patients can manage to make an appointment for specialist within 4 weeks, and 6% need to wait for two month or more in the US [3], as shown in Figure 1.2. In other countries, such as Canada, the scenario could be even worse with only 39% in four weeks and 29% in more than two months. Diseases like cancer or any viral infections tend not to be discovered until they reach a late stage, which could be avoided and make the treatment easier if can be diagnosed at an early stage.

Figure 1.2: Waiting times for specialist at different countries [3].

Personal health care devices in this case draw people's attention to relieve the burden on health care diagnosis. Companies own such techniques can provide domestic personal health care service and allow users to do daily or weekly routing tracking on their health status. Health issue, therefore, has a better chance to be found in an early stage without adding load to health care centers.

1.2 Development of Personal Health Care Devices

Moore's law was a prediction in development of semiconductor electronics, which generalized the number of transistors would be doubled in every two years per integrated circuit. Although the slope has been flattened in recent years, the exponential growth has been successfully observed for past decades due to scaling down of transistor size. As a fact of the global industry revenue involved with consumer electronics, such like Wifi-enabled cellphones, tablet, laptop and other personal assistant device (PDA) was increased in recent years, the success of technology scaling down could be largely attributed to market stimulus from the needs of being entertained or being facilitated. Personal health care electronics, therefore, have become an emerging topic by taking the advantage of very large scale integrated circuit (VLSI) and to fulfill a need of timely diagnosis from individuals [4] [5]. Bio-electronic circuit designs for monitoring important signals and molecules,such as ECG [6], neurotransmitter [7], human metabolism [8], were proposed to serve the various needs. One of the issues needs to be addressed is that the geometry of bio-electronics is required to be scaled down from benchtop as much as possible to become more portable. On one hand, scaling down is a trend to catch up with the technology scaling of the semiconductor industry. On the other hand, bioelectronic circuits are typically used in field where a portable feature provides better accessibility to the users. The portable bioelectronics have different forms. Works in [9] and [10] proposed CMOS based integrated chip designs for applications uses impedance spectroscopy technique to achieve the goal. Consequently, wearable device, handhold device and implantable device are three major solutions to reach the goal, such as works in [11], [12] and [13] for sensing glucose, Bovine serum albumin (BSA) proteins and neurotransmitters respectively.

1.3 Challenges in the Development of Personal Health Care Devices

Challenges in developing bio-electronics lie in cost, mobility, simplicity and accuracy, other challenges in bio-interface involves bio-compatibility, sensitivity, specificity, reproducibility and

reliability [14] [15] [16] [17]. Multi-disciplinary background of knowledge is required to resolve these problems as a whole from a system level's perspective. Among various challenges, cost and simplicity seem to be the most urgent features from the viewpoint of usability and mobility. Cost and size, are typically determined by the type of detection principle and implementation details. Based on the type of signal being sensed from bio-targets, different solutions could be proposed and a decision needs to be made to have an optimum viable approach. The corresponding simplest circuit/chip design should be made for realization. As for simplicity, the whole system operation protocol should be established to have enough prework done by biosensor providers to reduce the amount of steps that require users to execute. A common example is the choice of labeled techniques, such as florescence labeling. Since this class of techniques usually requires expertise to handle samples, and therefore, slows down the diagnosis cycle and adds costs, other alternatives should be explored instead if achievable. On the other hand, it is essential to realize that personal health care devices differ from research benchtop in its application specificity. This suggests the specification for personal health care device can be narrowed down compared to general purpose benchtop. Keep this in mind, reconfigurability comes out as new solution for customization. A re-configurable personal health care device allows the designers to reduce redundant design and unnecessary competing effort for specifications by providing the adequate functionality to users that fits almost exactly the requirements. With introducing reconfigurability, personal health care devices will be more affordable and flexible in the market. In reality, however, there is no electrochemical analyzer benchtop with reconfigurability features since circuit designers tend to design general purpose electronics to fit for the needs of every electrochemical researcher. Consequently, electrochemical analyzer design process was a research-oriented rather than application-oriented , while the later one is one step further to the public and the mass market. Nevertheless, circuit designers for personal health care device will be required to have biosensor application experiences in order to gain enough background to complete multidisciplinary design loop and know how to develop a reconfigurable bio-electronic.

Chapter 2

Existing approaches

2.1 Bio-sensor Basics

Biosensor device is a class of sensor that converts parameters with bio-interest, such as concentration, to electrical signals for storage and further processing. As an analytical device, a biosensor is able to interact with bioelements such as nuclieu acids, antibodies, tissues, microorganisms, enzymes, etc., electrochemically (direct and indirect) or through binding. A common biosensor consists of bioreceptor, biotransducer, amplifier, processor and display, as shown in Figure 2.1.

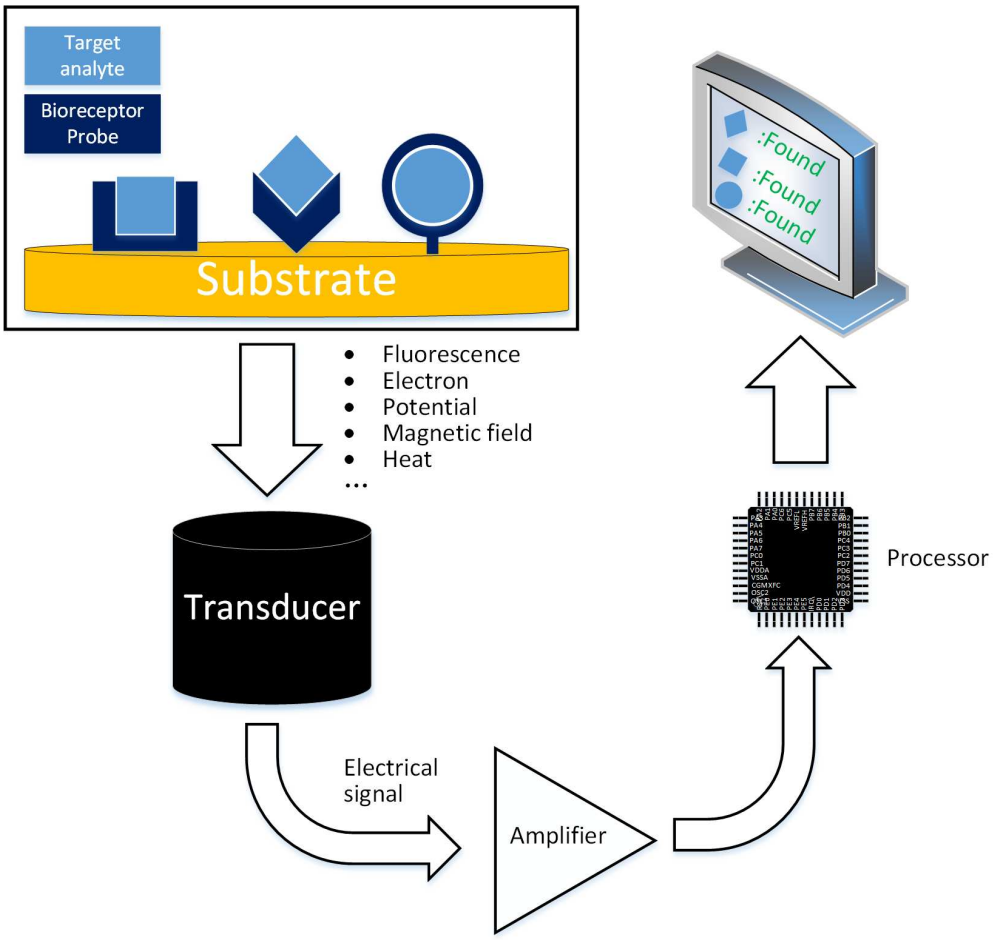


Figure 2.1: Illustration of a biosensor system.

A bioreceptor is a bio-molecule that can recognize target analyte through their reaction, such as binding between antigen/antibody due to hydrophobic force, ionic force and other forces among molecules, complementary DNA hybridization due to the form of hydrogen binding in base pairing. Other bioreactions include enzymatic interactions such as glucose oxidase that consumes oxygen when glucose exists.

As a critical component which determines the actual implementation of biosensor, transducer is the mechanism that makes a bioreceptor event observable or detectable. Upon binding or other bioreaction events, transducers take advantage of side effects, such as heat, potentiometric, amperametric, magnetical or optical phenomenon, and translate them into electrical signals. The following discussion will focus on the principle and details on different transducer designs.

2.2 Different Bio-sensing Techniques

2.2.1 Labeled Biosensors

Fluorescent Imaging Sensors

One of the techniques used by the majority of biologists is fluorescence labeling. In this case, the target analyte is reprocessed to be attached with a fluorescent label. During the experiment, the analyte can bind to the modified substrate which provides specificity to the analyte under study. The system setup is shown as in Figure 2.2. An excitation source with a tuned filter is adopted to generate narrow bandwidth frequency light that can trigger the emission. Then the emitted photon can be sensed by the filtered detector and therefore to detect the existence of target analytes and visualize their location. The excitation source is implemented using a homogeneous light source [18]. Filters are needed to reduce unwanted excitation light and emitted photons. Detector in this system is the transducer that convert photon into electrical signal through imaging electronics such as charge-coupled device (CCD) cameras [19]. The CCD active area is typically formed by a 2D array of metal-oxide-semiconductors (MOS) capacitor which can generate current flow proportional to photon intensity. The resolution of image is determined by the actual pixel pitch and density.

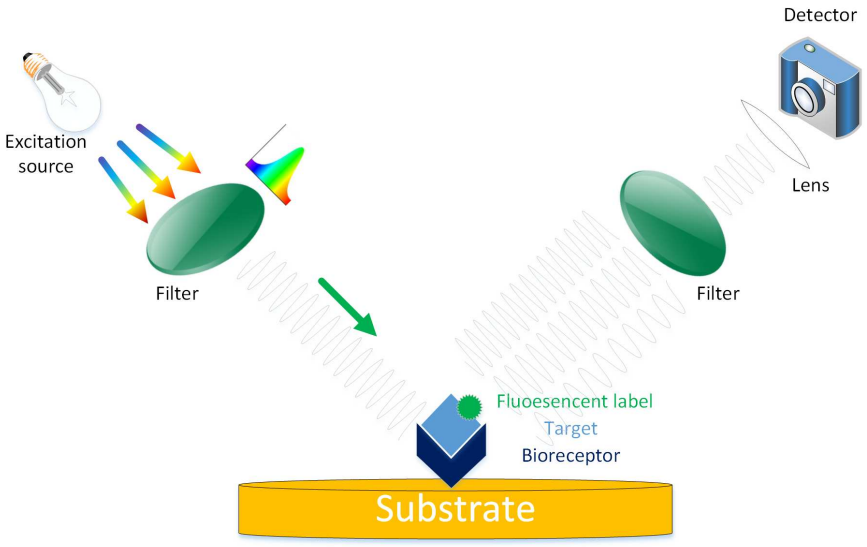


Figure 2.2: System setup of a fluorescent imaging sensor.

Magnetic-particle Labeled Sensors

Another labeled technique is by attaching a magnetic nano-particle to the target analyte. The system is illustrated in Figure 2.3. In this technique, the magnetic particle serves as an extra component to store magnetic field. A LC resonator is designed and tuned to oscillate at frequency determined by the designed capacitance and inductance. Upon the reaction between bioreceptor and target analyte, the magnetic particle will sit around the electrode surface. Because of the oscillation through the electrode, an alternating magnetic field is generated. The polarized magnetic particles in the vicinity of the electrode, therefore, provide additional magnetic flux and result in inductance change.Consequently, the oscillation frequency will be changed due to inductance change [20]. After the transducer have converted the binding/hybridization to frequency shift, the change can be further detected by introducing a counter circuit that counts number of cycle within a given time interval. The quantitative difference in number of cycles prior and post to target analyte injection indicates not only its existence, but the amount or concentration based on proper calibration.

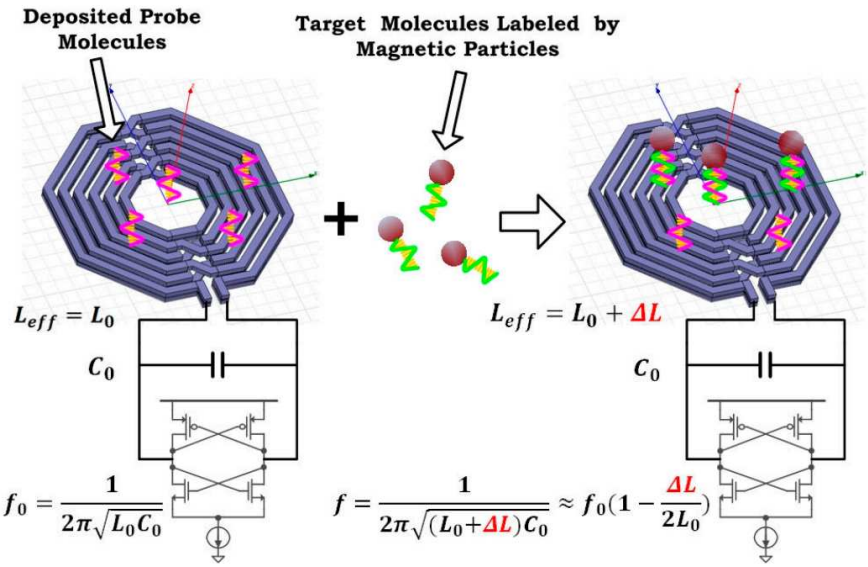


Figure 2.3: Principle of magnetic particle labeled biosensor [20].

2.2.2 Label-free Biosensors

Ion-sensitive Field Effect Transistor(ISFET) Based Sensors

While the labeled techniques uses different transducers to detect analytes by recognizing labels, another class of biosensors are label-free biosensors. Ion-sensitive Field Effect Transistor (ISFET) based sensor is a type of label-free biosensor, as shown in Figure 2.4. Field effect transistor in general is a switch whose conductivity is determined by the electrical potential to the gate. With the interference of electrical potential due to ions in electrolyte and analyte, the charge transfer response can be distinguished, and therefore, used as transducer for biosensing purposes. For instance, such technique is applied to detect DNA in [21]. Since DNA naturally carries backbone negative charges that opposed the applied electrical field, the capture of its complementary DNA will cause further change of chargers accumulated beneath the sensing area. Taking into account the pH effect in the solution, the actual potential applied to the gate area is described by Eq. (2.1), where V_{ref} is the bias voltage applied to the electrolyte solution, α_{H+} is the hydrogen ion concentration, R is gas constant, T is temperature, F is Faraday constant, V_{DNA} and $V_{ComplementaryDNA}$ are equivalent potential change due to opposing electrical field from backbone negative charges. With φ at the gate of ISFET, the charge Q accumulated to the floating diffusion (FD) capacitor

varies accordingly. The sensor signal is eventually represented as the voltage signal at the output of the source follower buffer according to Eq. (2.2), where C_{FD} is the capacitance at the drain terminal of ISFET, Q is the total charges collected by FD capacitor, A_{SF} is the gain of the source follower. The ISFET based technique is commonly used for pH sensor [22] as well.

$$\varphi = V_{ref} + \frac{RT}{F} ln\alpha_{H+} - V_{DNA} - V_{ComplementaryDNA}, \tag{2.1}$$

$$Vout = \frac{Q}{C_{FD}} \times A_{SF}, \tag{2.2}$$

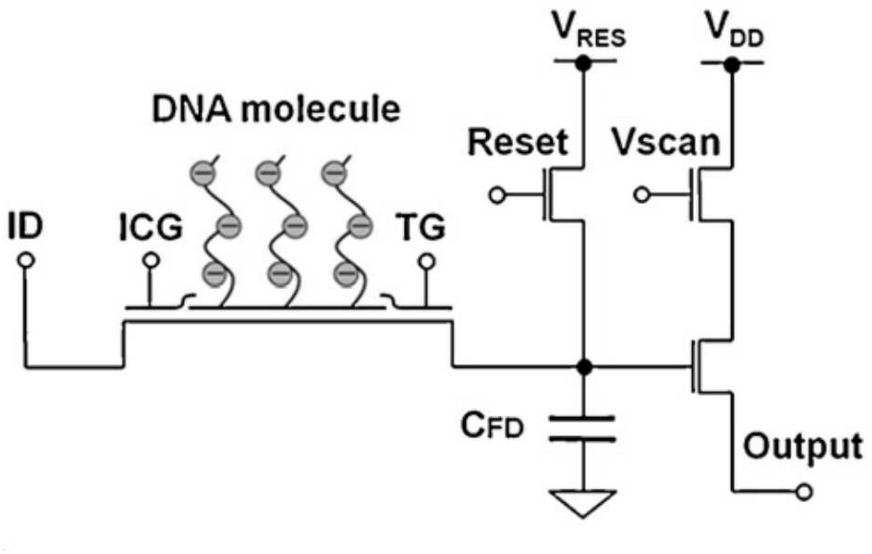


Figure 2.4: An ISFET based biosensor [21].

Capacitive Sensors

When a pair of electrodes is immersed in a solution and form a complete electrical loop, electrodes can repel ion with same polarity and attract counterions due to the electrical field. The charges on electrode and counterions in the electrolyte solution in vicinity, therefore, form two layers of conductive plates. Since the molecules in between the plates act as dielectric material, a double layer capacitor is formed at the electrode/electrolyte interface. As illustrated in Figure 2.5, the bare electrode/electrolyte interface can be modeled as a resistor R_p in parallel with a capacitor

C_p and then in series with another resistor R_p. R_p is to model the leakage current through electrode, R_s represents solution resistance and C_p is the double layer capacitor. When target analytes bind near the electrode, the biomolecules displace the counterions and increase the dielectric layer thickness [23]. Eq. (2.3) defines the capacitance involved in sensing, where ϵ is dielectric constant, A is effective plate area, d is distance between plates, as d increases the overall capacitor will decrease.

There are several ways to measure the capacitance change. For instance, [23] uses an integrator technique that correlates the capacitance with charge/discharge time delay. In [24], a RC oscillator was proposed to convert the capacitance change into oscillation frequency. Nevertheless, since capacitor is the reactant component, the alternative to measure capacitance could be directly derived from impedance detection [25].

$$C = \epsilon \frac{A}{d} \tag{2.3}$$

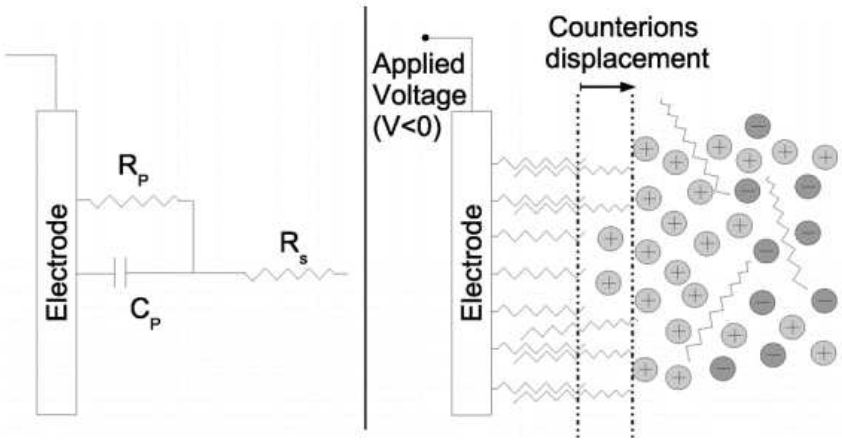


Figure 2.5: Principle and model of capacitive biosensor [23].

Electrochemical Biosensor and Electrochemical Impedance Spectroscopy

Another type of biosensors is based on electrochemistry. A reduction or oxidization reaction typically happens at electrode/electrolyte interface as shown in Figure 2.6. When the reaction happens, the reactant will either absorb or release electron, and therefore, cause electron transfer through electrodes. The transducer, hence, view the detection of target analyte as change in current due to binding or reaction [26]. The mechanism that causes changes in electron transfer can be either generated by the target analyte through enzymatic response [27] or from the interference of target blocking [28]. The enzymatic approach can be applied when detecting molecules such as lactate and glucose, that each type enzyme bind on electrode is dedicated to catalyze a specific target to generate electron transfer. In electrochemical affinity sensing that relies on the blocking effect of carrier movement. A redox couple is commonly introduced in the solution to enhance the electron transfer current. When binding occurs, the target molecule which is not redox active reduces the effective contact area for the electron movement from the redox couple, causing the impedance at the electrode/electrolyte interface to increase. This type of transducer directly converts biosignal into current. Among other electrochemical techniques for biosensor,

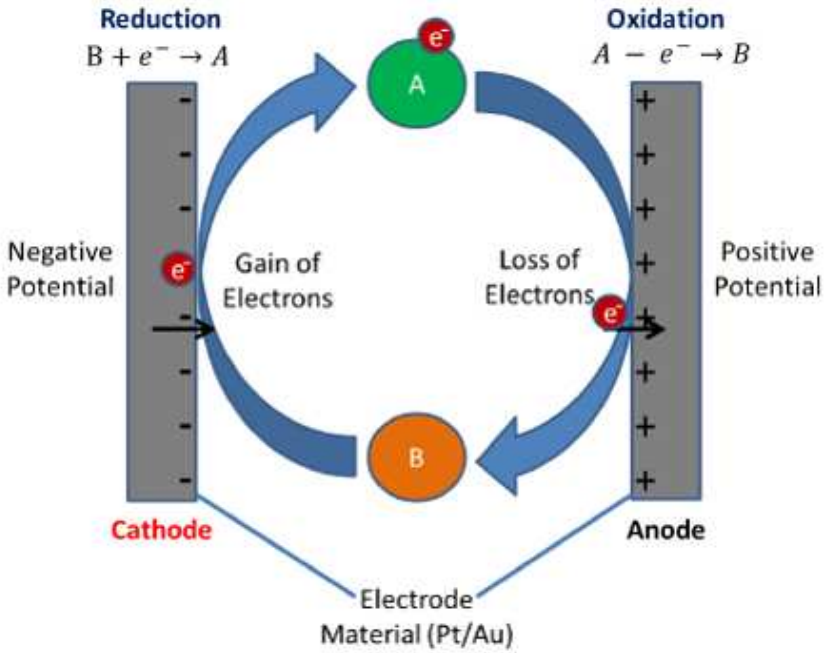


Figure 2.6: Electrode/electrolyte interface response in electrochemical Biosensors.

electrodechemical impedance spectroscopy (EIS) stands out due to its informative nature. EIS is achieved by sweeping the applied AC signal at the electrode-electrolyte interface In this case more data can be collected as a response of impedance at different frequency, the dimensionality of data will be increased and results in more detailed analysis at the bio-interface. Figure 2.7 illustrates an example in Nyquist plot, which can resolve the equivalent electron transfer resistors and solution resistor in a more complicated circuit model by finding the best fitting curve to the actual data. EIS has been largely used by researchers for detecting various proteins such as folate receptor [29] and C reactive protein [30]. The results achieved so far indicated a good detection limit as low as picomolar range.

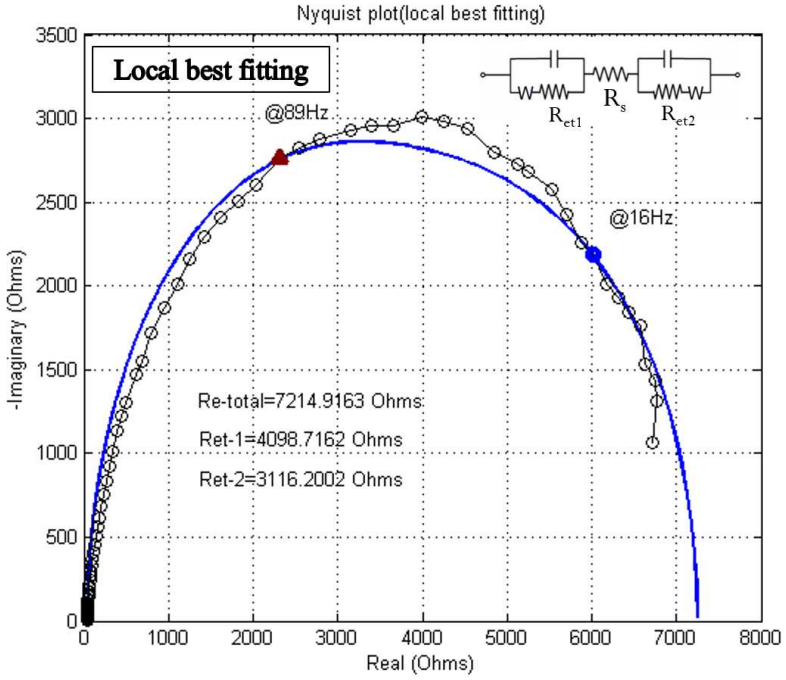


Figure 2.7: Example for EIS fitting results.

2.2.3 Discussion for Different Biosensor Techniques

Among different types of techniques, the label-free techniques are simpler to use and, therefore, more appropriate for devices used for POC applications. From the electronics' perspective,

electrochemical methods act as a natural bio-transducer that generates current for analysis. This fits perfectly for electronic design and reduces design complexity. Additionally, the EIS approach provides the current or impedance information in a complex form which allows the user to do applications that care about the phase shift in addition to the magnitude. Because EIS can provide both real and imaginary parts, it fits for capacitive biosensor applications as well. Therefore, the implementation and validation of a low cost EIS system act as a perfect entrance point to develop the reconfigurable electrochemical analyzer as a main base module.

2.3 Electrical Impedance and Different Measurement Methods

The term electrical impedance is to describe a characteristic defined as the total opposition a device or circuit offers to the flow of an alternating current (AC) at a given frequency. It's often referred to complex impedance since it has a complex quantity and can be graphically shown in complex plane.

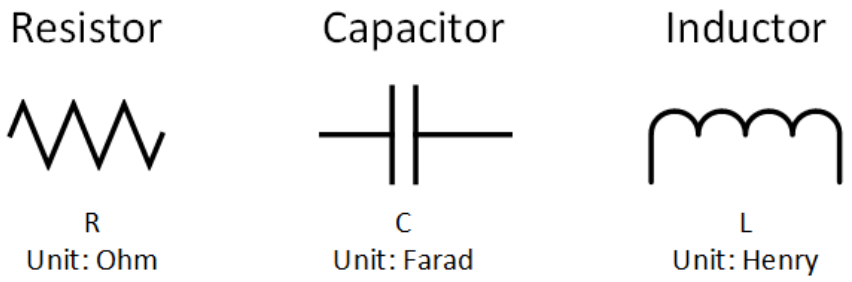


Figure 2.8: Symbolic representation for RLC.

In basic electrical theory, there are three fundamental electrical components, which are resistors, capacitors and inductors, Figure 2.8 shows their symbolic representations. A resistor is a component that impedes the movement of carrier in flow path. The resistance is proportional by resistivity of material and the length along with carrier flow, is inversely proportional to the cross sectional area. The relationship is simply the ratio of voltage and current without any phase shift, $Z_R = R = \frac{|V|}{|I|}$. A capacitor is a storage component, which allows charge to accumulate on both

capacitor plates. The size of capacitor indicates the capability to store the amount of charge per unit voltage. No current can go through a capacitor at steady state when direct current (DC) voltage is applied. While an AC signal is applied across a capacitor, an altering current is created with a -90 degree phase shift, its mathematical form in complex plane is $Z_C = -\frac{j}{\omega C} = -\frac{1}{j\omega C}$, j is Imaginary unit, ω is Angular frequency. An inductor can store magnetic field as energy when AC signal is applied. The inductive impedance expression is $Z_L = j\omega L$, L is the inductance. In complex domain, an inductor can cause +90 degree phase shift.

Theoretically, the impedance of any device under test (DUT) is affected by its resistive, capacitive and inductive properties. Therefore an impedance usually has both a resistive part and a reactive part. The undesirable electrical component is call parasitic. For instance, a resistor could have unwanted inductor if it has wire-wound shape, a capacitor can potentially be modeled with a big resistor in parallel with it for leakage current, and an inductor could have resistor in series with it since any material has its resistivity. However, in reality, the unwanted parasitic resistance/capacitance/inductance can be ignored in most of case, if we know what's the test setup. Such as the inductive impedance can be ignored at low frequency, the big resistor in parallel with capacitor can be ignored at high frequency. Therefore it's reasonable to neglect some parasitic when analyzing an impedance response.

2.4 Bridge Method to Detect Impedance

The history to detect impedance can be traced back to early twentieth-century [31]. Some of the early techniques are derived based on the bridge method. In Figure 2.9, it shows the schematic of a Transformer Ratio Arm Bridge. In this technique, an oscillated AC signal is carried through the transform and the magnitude of the signal can be controlled by its turns ratio. Once the bridge is balanced with tuning the standard resistor and capacitor as well the ration of transform, the detector will give a null signal, Eq. (2.4) shows the relationship among the parameters when it's balanced. This approach can provide wide impedance detection range, high accuracy and is insensitive to

parasitic capacitance [32]. On the other hand, since it relies on the transformer, the low frequency, such as at 200 Hz or less, the detection response is poor due to signal loss.

$$\frac{r_1}{Z_x} = \frac{r_2}{R_s} + j\omega C_s r_3 \tag{2.4}$$

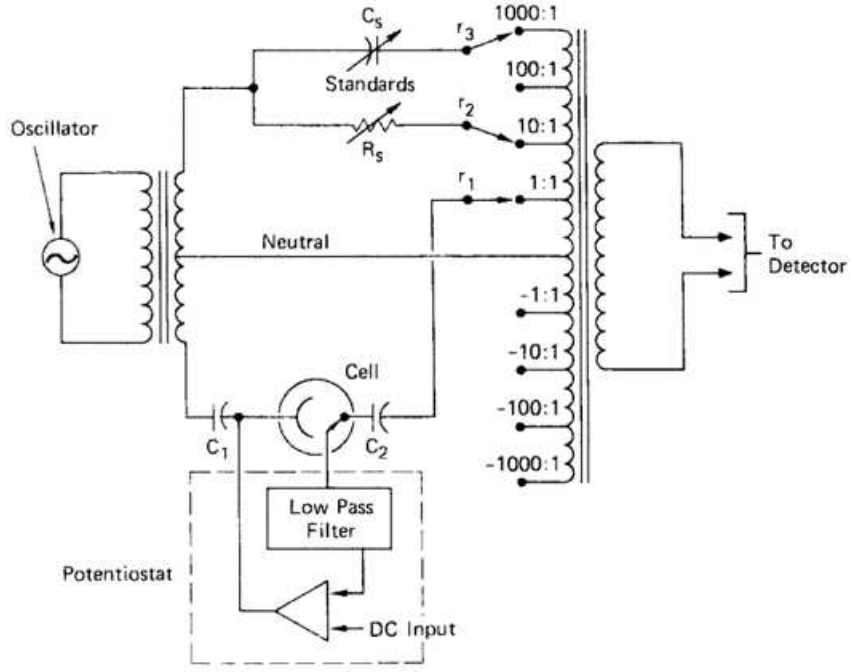


Figure 2.9: Schematic of transformer ratio arm bridge [31].

Another technique is based on audio frequency bridges, as illustrated in Figure 2.10. This measurement is achieved by applying an audio frequency (20 Hz-20 kHz) and monitor the output null status to find a balance point. The DUT impedance Z_x equals calculated by Eq. (2.5), Z_s is the standard impedance consists of resistor and capacitor. Such technique, however, is limited at high frequency as a result of the inductor effect of resistor and parasitic capacitance shunt.

$$Z_x = \frac{R_1}{R_2} Z_s \tag{2.5}$$

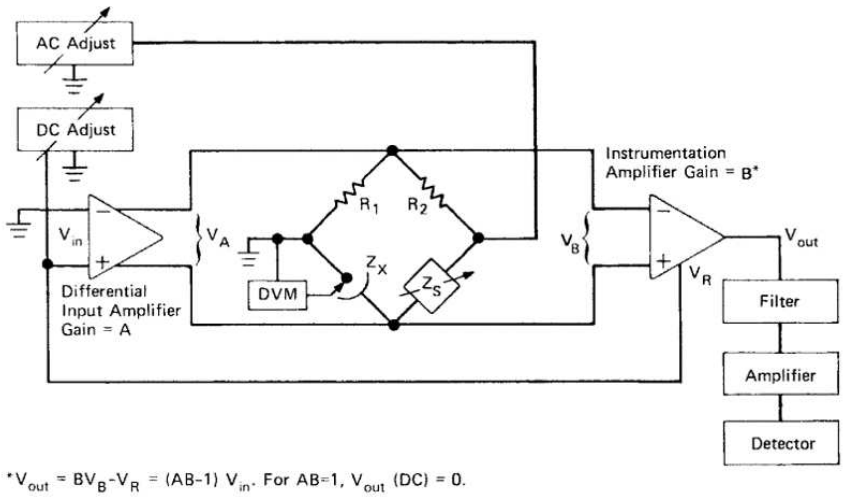


Figure 2.10: Schematic of audio frequency bridges method [31].

The Berberian-Cole Bridge is a technique that overcomes the drawbacks from the previous bridge measurement methods. It has a wide detection frequency range and can operate as low as DC, Figure 2.11 shows the schematic. This technique uses a potentiostat to stabilize the stimulus signal. At the output of the potentiostat, a resistor R' convert the voltage into current, and the current will go directly to the ground since the input impedance at the input of amplifiers are usually high. Two amplifiers are adopted to measure the voltage across the reference resistor R' as well as DUT. Then, the signals at output of amplifiers are further converted into current through resistor and capacitor and merged at same node. According to the Kirchhoff's Current Law (KCL), $i_1 + i_2 + i_3 = 0$ when detector has high input impedance. Eq. (2.6) (2.7) (2.8) and (2.9) represent the impedance when the bridge is balanced, in other word when V_s equals to 0 V. If the operating frequency go beyond the bandwidth, the measurement will suffer from gain and phase error.

$$i_1 = A\frac{V_A}{R_1}, \quad (V_A = IZ) \tag{2.6}$$

$$i_2 = A\frac{V_A}{j\omega C}, \quad (V_A = IZ) \tag{2.7}$$

$$i_3 = B\frac{V_B}{R''}, \quad (V_B = IZ) \tag{2.8}$$

$$Z = \frac{BR'R_1}{AR''} \frac{1 - j\omega R_1 C}{1 + \omega^2 r_1^2 C^2} \tag{2.9}$$

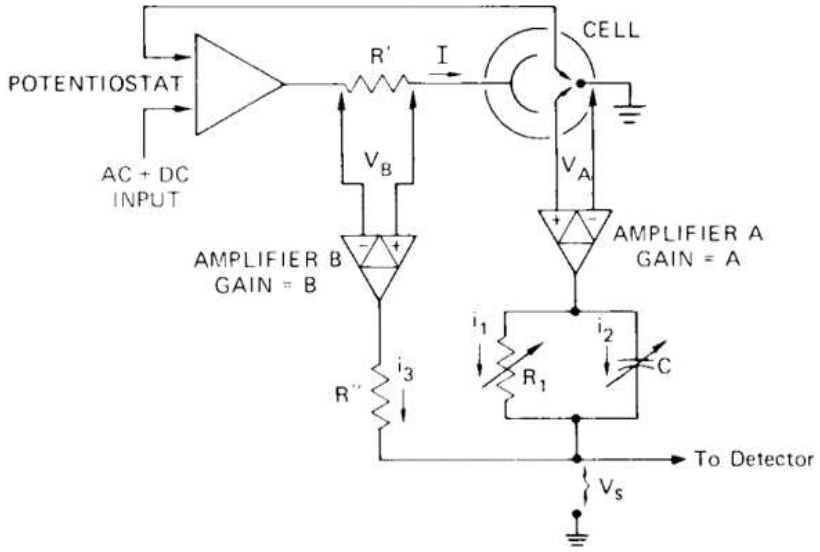


Figure 2.11: Schematic of Berberian-Cole bridge method [31].

The bridges methods are mainly used in early and middle twentieth century when transistor technique was not advanced and processors were not prevailing. These techniques have a common drawback that requires tuning to reach a null status in order to measure. This increases cost by adding more standard resistors and capacitors for better resolution. At the same time, the manual tuning process makes the measurement inconvenient.

2.5 CMOS Based Impedance Measurement Circuits

2.5.1 CMOS Technology

Complementary Metal Oxide silicon (CMOS) is a modern technology for integrated circuit. The transistor used in this technique is metal-oxide-semiconductor field-effect transistor (MOSFET) and term complementary is referring to the complementary and symmetrical nature between p-type and n-type doped MOSFET [33]. Figure 2.12 shows the MOSFET structure for both n-type and p-type. The n or p notation indicates the type of carriers in the device are either electrons or

holes. MOSFET is a four terminal device, which has body, source ,drain and gate. Gate is where the electrical potential can be applied to create a conductive channel. Source and drain are named after the active region which acts the source/drain of carrier in the device, but not the source/drain of current direction. Body is the lightly doped substrate by complementary carrier. MOSFET has three operating regions as the cutoff region, the triode region, and the saturation region. Eq. (2.10) (2.11) and (2.12) show the condition for each region, where V_{th} is the threshold voltage that creates an inversion layer in MOSFET substrate to form the conductive channel.

$$|V_{gs}| < |V_{th}|,\ (Cut-off) \tag{2.10}$$

$$|V_{gs}| > |V_{th}|\ and\ |V_{gd}| > |V_{th}|,\ (Triode) \tag{2.11}$$

$$|V_{gs}| > |V_{th}|\ and\ |V_{gd}| < |V_{th}|,\ (Saturation) \tag{2.12}$$

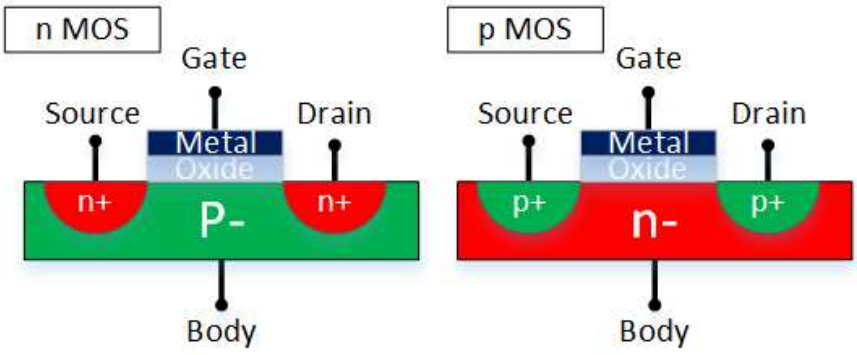


Figure 2.12: CMOS structure for both n-type and p-type.

MOSFET has several advantages over other types of transistors. First of all, the on-off operation is straightforward as a result of applied gate voltage, this makes it ideal to perform as a switch in any analog or digital circuit. The oxide insulating layer at the gate can almost get rid of input bias current, and therefore reduces the power consumption. On the other hand, the oxide layer provides a high impedance configuration, which reduce the effect as a load to previous stage circuitry. From manufacturing perspective, MOSFET is also area efficient, the transistor length

and width are controllable. With academic and industry researcher's effort, MOSFET technology is scaled down to as small as 5nm [34] which enabled the integration of billion of transistor on a single chip.

2.5.2 Synchronous Voltage-to-frequency Convertert Technique

In [10], a synchronous voltage-to-frequency converter (SVFC) technique was proposed to measure the impedance, as shown in Figure 2.13. A sinusoidal voltage signal is applied as stimulus through the DUT and the response signal in current coming out to charge/discharge the integrator circuit. The push/pull current source switched by the output of the D-flipflop can help accelerate the charge and discharge processes and provides the counting function. When $V_{int} > V_{ref}$, $I_{ref1} + I_{in}$ charges C_f, and causes $Vint$ to drop. Once $V_{int} < V_{ref}$, the Q output flips and enables I_{ref2}. $-I_{ref2} + I_{in}$ in this case starts to discharge C_f and causes $Vint$ to raise. The duty cycle of Q indicates whether I_{in} is positive or negative. These happen periodically during an input cycle and a counter circuit is used to record the flipping event. Eq. (2.13) is an approximation of transferred charge q_{in}, the real part expression of the current in this approach is based on Eq. (2.14), where T_{clk} is the time period of clock and D is the counts within one cycle period of input signal. The imaginary part of the current is acquired by introducing a quadrature phase shifted signal that controls the integrator sample period. Eq. (2.15) represents the calculation for the imaginary part of the input current. The impedance in complex number can be presented as in Eq. (2.16).

$$q_{in} \approx I_{ref} T_{clk} D, \tag{2.13}$$

$$Re\{I_{in}\} = I_{ref} T_{clk} D \frac{\pi}{T_{in}}, \tag{2.14}$$

$$Im\{I_{in}\} = -I_{ref} T_{clk} D \frac{\pi}{T_{in}}, \tag{2.15}$$

$$Z = \frac{V_{in}}{Re\{I_{in}\} + Im\{I_{in}\}}, \tag{2.16}$$

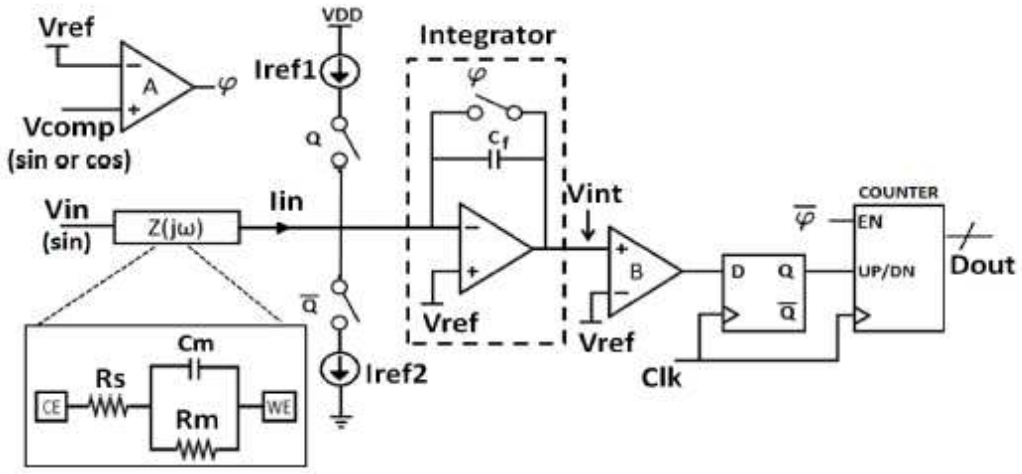


Figure 2.13: SVFC technique schematics [10].

The dynamic detection range in this technique is largely determined by the choice of I_{ref}, current source with different magnitude needs implemented for practical measurement. The measured frequency in this technique is largely limited by the clk frequency in order to have a counter output. Nevertheless, compared to the bridge methods, such approach has better integration and converts the output into digital signal which is more accessible to process. There is no need for tuning resistor/capacitor in this technique, since the impedance can be calculated by the digital counts.

2.5.3 Lock-in Amplifier Based Technique

Lock-in amplifier is a system that extracts a specific frequency of interest to analyze. As shown in Figure 2.14, it usually consists of a pre-amplifier, such as transimpedance amplifier (TIA), a signal mixer and a low pass filter. The pre-amplifier is to convert the current signal from DUT to voltage. Signal mixer is a unit that multiplies the signal from DUT with a reference signal from the local oscillator (LO) and results in a superposed signal. Eq. (2.18) expresses the math representation for signal at the mixer output. When the frequency of the input and reference signals are equal, the expression can be simplified as Eq. (2.19) shows which consists of a DC and high frequency component. The low pass filter (LPF) is used to eliminate the high frequency part and the signal ends up with a pure DC, eventually $V_{out} = \frac{1}{2}V_{sig}V_{LO}cos(\Delta\theta)$.

$$V_{mix} = V_{sig}sin(\omega_{sig}t + \theta_{sig}) \times V_{LO}sin(\omega_{LO}t + \theta_{LO}) \tag{2.17}$$

$$= \frac{1}{2}V_{sig}V_{LO}cos([\omega_{sig} - \omega_{LO}]t + \theta_{sig} - \theta_{LO}) + cos([\omega_{sig} + \omega_{LO}]t + \theta_{sig} + \theta_{LO})) \tag{2.18}$$

$$V_{mix} = \frac{1}{2}V_{sig}V_{LO}(cos(\Delta\theta) + cos(2\omega_{sig}t + \theta_{sig}) + \theta_{LO})) \tag{2.19}$$

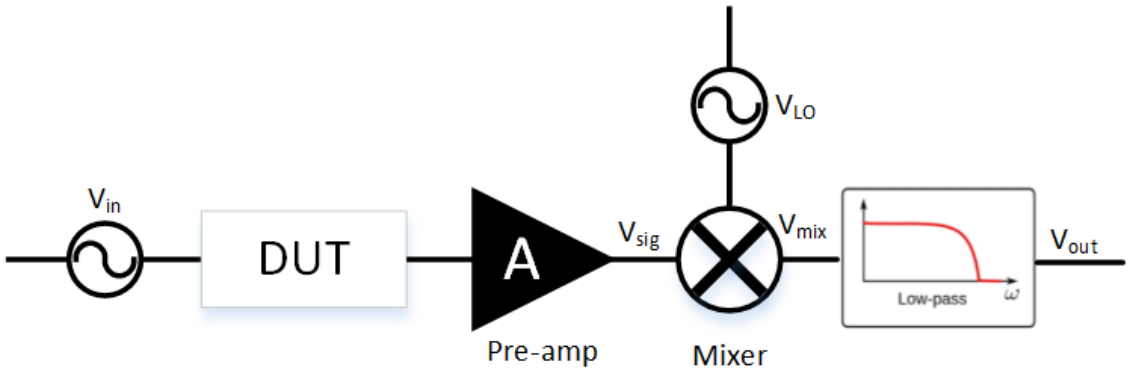


Figure 2.14: Block diagram of lock-in amplifier

In [9], a CMOS based lock-in amplifier technique for impedance measurement was proposed, as shown in Figure 2.15. An in-phase and 90 degree out of phase signal are applied to lock-in the signal, and to acquire the real and imaginary part respectively. The frequency detection range can go as high as 50 MHz till it reaches the TIA bandwidth limit. The impedance detection dynamic range depends on the gain range of TIA. Once the in-phase and quadrature phase voltage response are acquired, the impedance magnitude and phase can be calculate from Eq. (2.20) and (2.21), where A is the total gain from TIA and mixer.

$$Z(\omega) = \frac{A|V_{x(\omega)}|}{\sqrt{V_I^2 + V_Q^2}} \tag{2.20}$$

$$\angle Z(\omega) = 90 - arctan(\frac{V_Q}{V_I}) \tag{2.21}$$

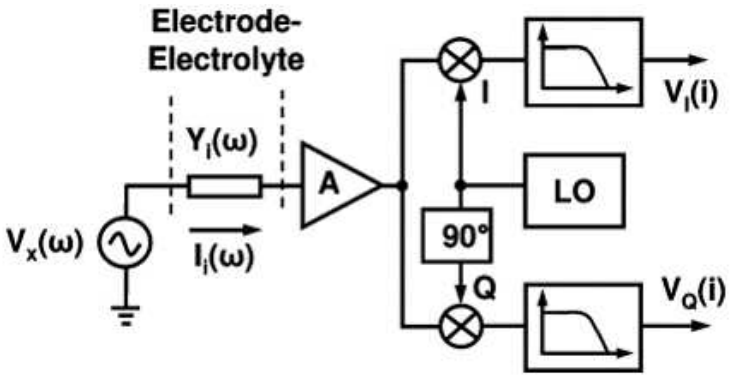


Figure 2.15: Lock-in amplifier based impedance measurement diagram [9].

2.6 Some Existing Potentiostat/Impedance Analyzer System

So far, various approaches in terms of bio-sensing as well as current/impedance measurement have been discussed. It bring more practicality when it comes to integrated system that physically allows any user for their specific applications. In this subsection, several existing designs which consist of complete front-end user interface and back-end sensor interface will be discussed.

2.6.1 CheapStat

The concept in CheapStat, proposed in [35], is to develop a home-made potentiostat at low cost to help users to explore their sensor application. In this design, costs have been lowered down to less than $80. Figure 2.16 shows the PCB layout for this design.

The proposed device supported cyclic, square wave, linear sweep and stripping voltammetry over the potential range -990 to +990 mV and over frequencies from 1 to 1000 Hz. The device was also validated with 1mM/2mM Ferricyanide and observed peak current difference in uA range. Additionally, the device was applied in a DNA hybridization experiment to measure the event of binding from complementary DNA. The results, in Figure 2.17, showed a decreasing peak current when apply linear scan voltammetry.

While this integrated system had been proven to be suitable for educational use and some analytical applications. The unavailability in applying impedance measurement set limits to applications that require impedance spectroscopy.

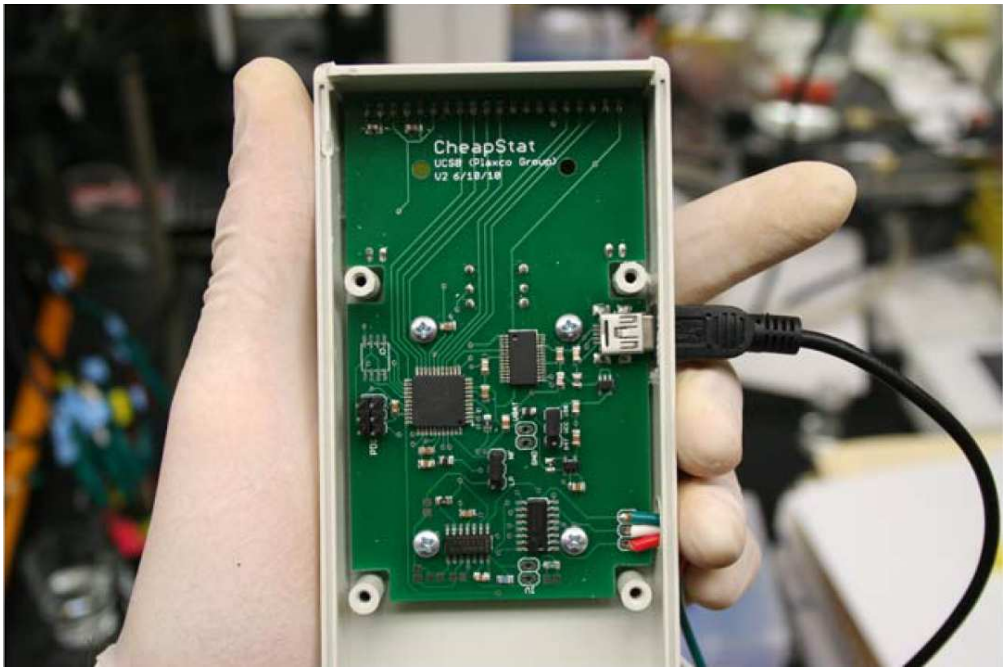


Figure 2.16: Physical PCB layout for CheapStat [35].

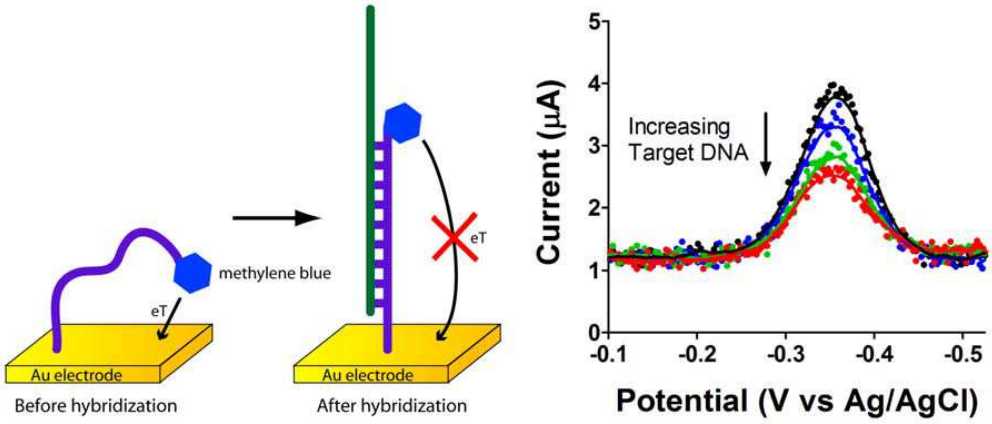


Figure 2.17: DNA hybridization experiment measured by CheapStat [35].

2.6.2 A Smartphone-controlled Electrochemical Impedance Spectroscopy Analyzer

In [12], the researchers proposed a smartphone-controlled electrochemical impedance spectroscopy analyzer that can scan from 10 Hz to 10 kHz. The design architecture is shown in Figure 2.18, and Figure 2.19 shows the physical PCB boards and device. The device had been validated by BSA absorption test through EIS, and a limit of detection at 1.78 ug/ml with 3δ slope calculation had been achieved. Different non-specific target proteins had also been study and shown lower signal in impedance change to prove the specificity of the proposed protein sensor.

By taking advantage of AD5933, which is a commercial impedance analyzer chip that have integrated signal generator, data acquisition, signal processor as well as I2C protocol converter, this overall design efforts had been relieved to simplified the board design. This design was battery operated, and able to communicate wirelessly through Bluetooth protocol with an android app. All these features allow the device to be suitable for point-of-care application which requires portability and simplicity. However, the designers didn't take further step to integrate the system and reduce the redundancy to cut down the cost, such as getting rid of commercial Arduino board or integrate the Bluetooth module on board. Further, the dedicated impedance analyzer chip can only perform EIS experiments and can't be applied to other electrochemical methods, such as cyclic voltammetry (CV), amperometry, differential pulse voltammetry (DPV). Therefore, it's not applicable to general electrochemical applications.

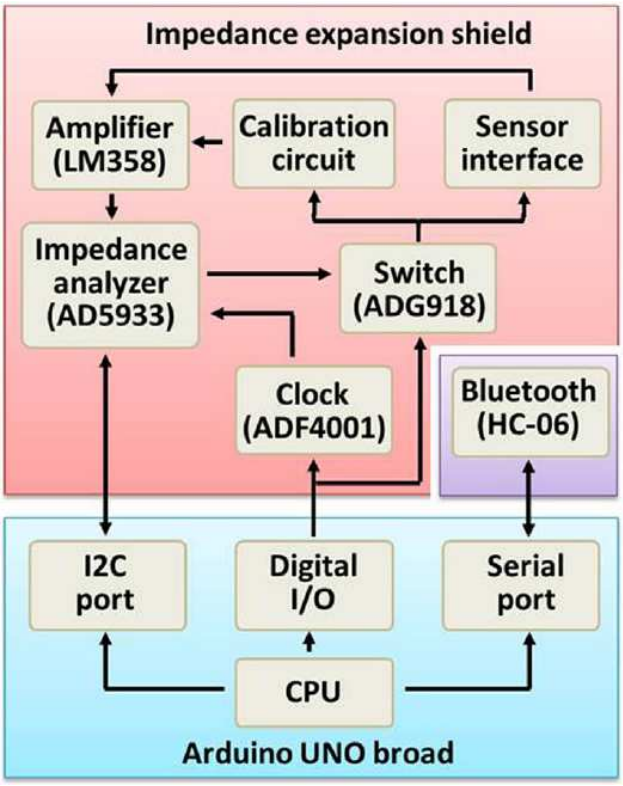


Figure 2.18: The smartphone-controlled EIS analyzer architecture [12].

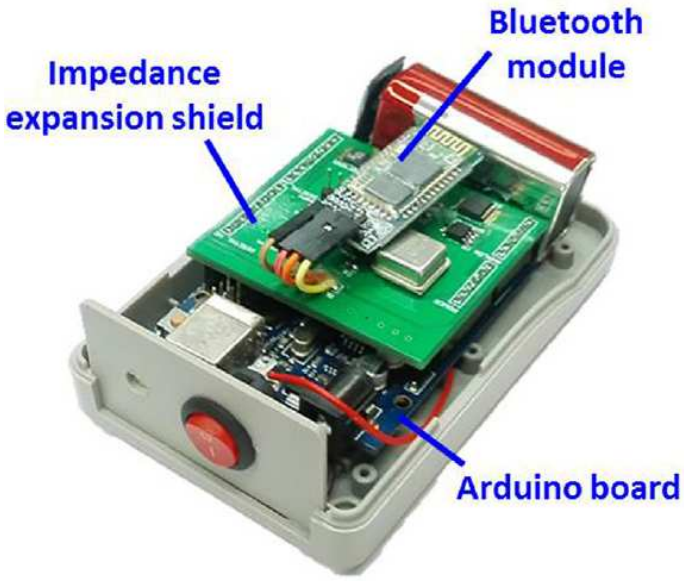


Figure 2.19: The smartphone-controlled EIS analyzer device [12].

Chapter 3

The Components Used in Crexens^{TM} Architecture

3.1 ADC Techniques

Signals, such as voltage, audio, light, temperature, are classified as analog signals. This type of signal is naturally continuous and time varying [36]. However, an analog signal can't be directly processed by digital signal processor (DSP) to do any calculation or storage. Therefore, an analog-to-digital conversion (ADC) method is needed as an interface to turn the analog signal into binary numbers.

3.1.1 ADC Resolution and Errors

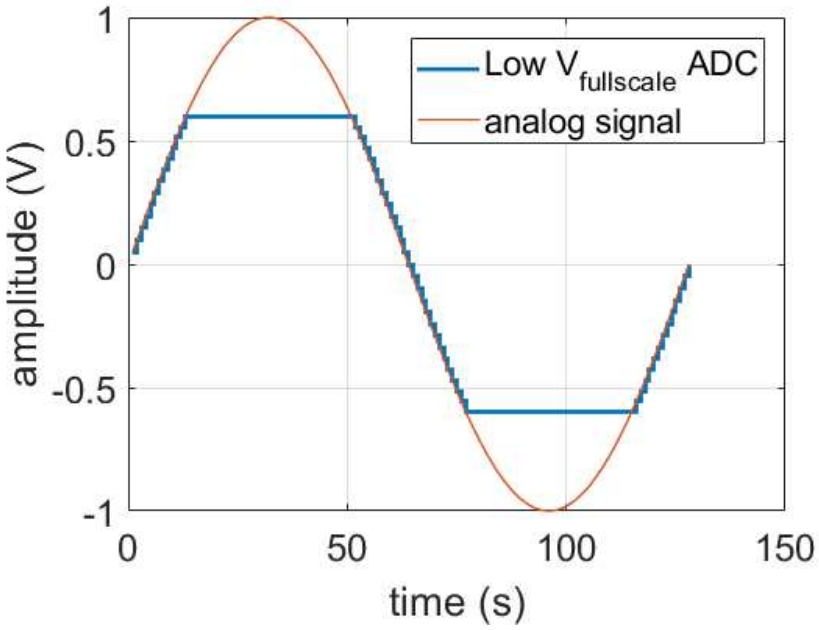


Figure 3.1: Distortion error due to low full-scale voltage .

There are many different implementations in ADC family such as, successive-approximation (SAR), direct conversion, sigma-delta ($\Sigma - \Delta$), and pipelined ADCs. Different techniques can

make different trade-off and be applied to various applications. ADCs have common design specifications and need to improve to avoid certain problems. One of important aspects is the resolution.

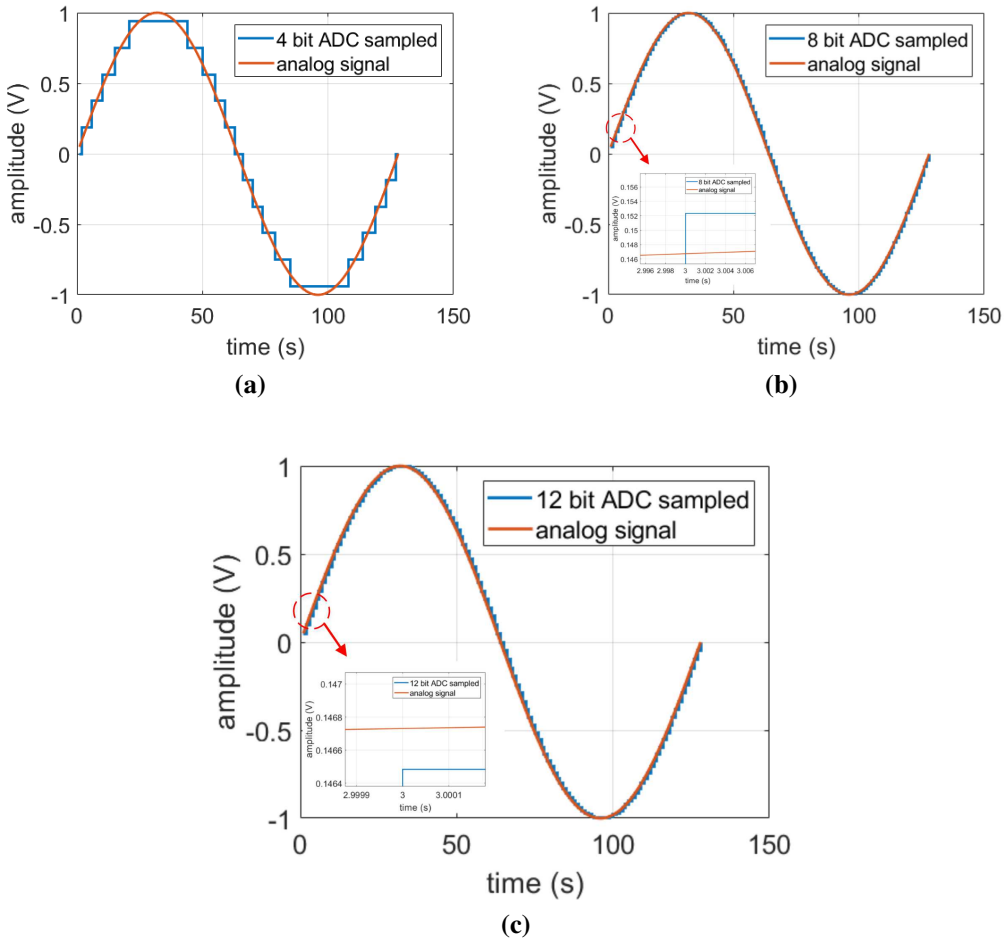


Figure 3.2: Quantization error for different LSB

Resolution of an ADC indicates the minimum difference in an analog signal that can cause one bit change in digital output. Therefore, it's also referred to least significant bit (LSB). When an ADC is sampling voltage signals, the resolution is determined by the full-scale voltage and the number of bit. Since each bit is storing a '1' and '0', for N-bit ADC it can represent 2^N decimal values. And the LSB voltage is defined as $\frac{V_{fullscale}}{2^N}$.

The choice of the full-scale voltage and LSB are both essential. If the full scale voltage is smaller than peak to peak amplitude of the sampled signal, the digitized signal will be distorted. Figure 3.1 shows an example in which the signal is distorted from the original sinusoidal due to insufficient full scale voltage.

Quantization error is another error due to large LSB. It happens on each sampled point. Just like the sample result from ADC is discrete in time, the resolution in amplitude can't be infinitely small. Therefore the quantization error is the difference between the original value and the sampled value at the same time. Figure 3.2 shows the quantization error under different ADC bit which indicates the error is reduced as resolution is increased. Since LSB is affected by both the full-scale voltage and number of bits, low full-scale voltage and more ADC output bits are preferred to reduce such error.

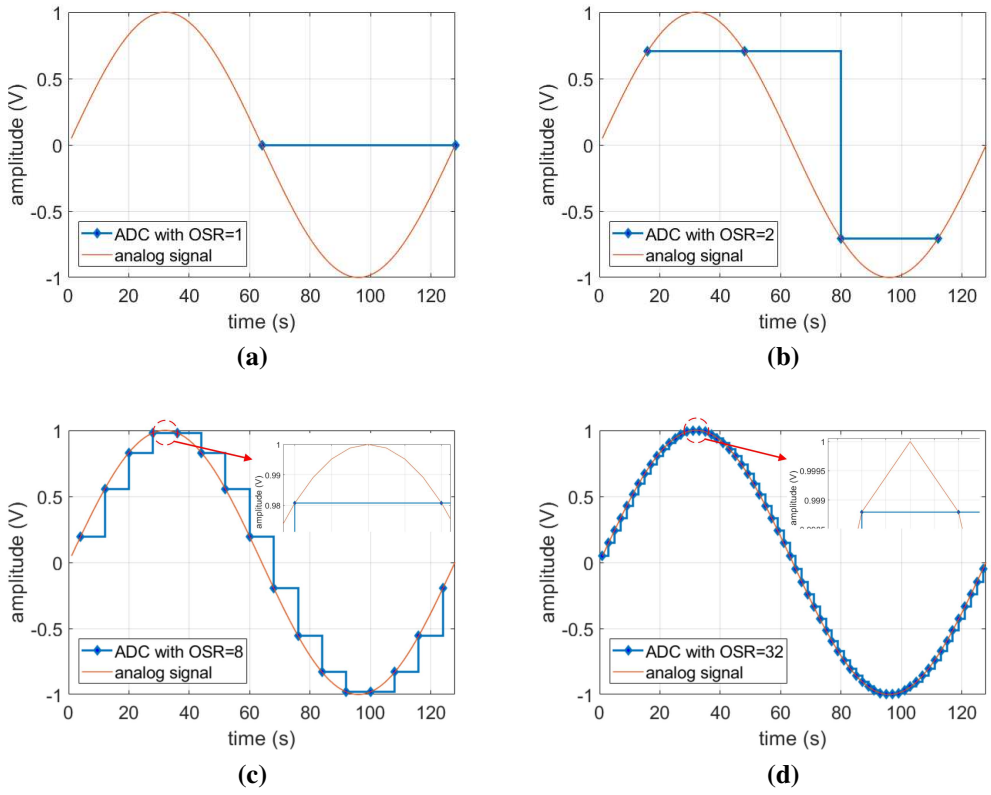


Figure 3.3: Sample error due to low OSR in worst case scenario

3.1.2 ADC Sample Rate

Sample rate determines the time interval between two data acquisition attempt for an analog signal. According to the Nyquist theorem, the sample rate should be at least twice faster than the analog signal needs to be sampled, which is also referred as Nyquist rate. Oversampling ratio (OSR) is parameter indicates how much faster the actual sample rate is than Nyquist rate. Even though samples at Nyquist rate is plausible in principle, there are issues brought by such scenario. One of the critical problems is the sample error due to phase shift. For instance, when ADC samples at the Nyquist rate for single sinusoidal signal, the reconstructed digital signal amplitude can be any value in between the original signal amplitude to 0. Figure 3.3 shows the reconstructed sampled signal with phase shift sample error in the worst case, which implies its preferable to choose higher sample rate to reduce sample error.

3.1.3 Choice of ADC

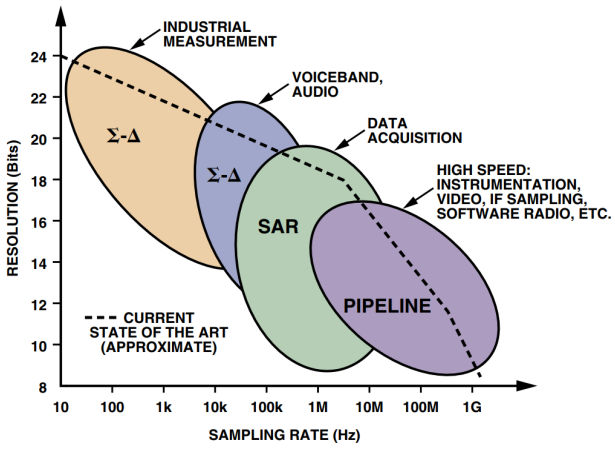


Figure 3.4: Design trade-off chart for ADC [37].

The conclusion drawn from the previous discussion can be generalized as it's always preferable to have more ADC output bits and high sample rate. However, this is hard to achieve since the design of ADC among different technique are typically making trade-off between sample resolution and sample rate [37]. Figure 3.4 generalizes the design trade off in ADC, which shows

a clear trend that lower resolution can achieve higher sample rate and vice versa. Consequently, a high resolution and high sample rate ADCs can be costly, since these ADCs are usually taking advantage of advanced technology scale. Therefore, it is important to know the tolerance of error from the non-ideal ADC when comes to make a decision on specification of ADC.

3.2 Signal Processing Technique

3.2.1 Fast Fourier Transform Algorithms

In order to measure impedance, the key is to find out its complex expression. The CMOS techniques discussed in the previous chapter all impulsed a quadrature phase shift signal for the imaginary part and a in-phase signal for the real part with analog approaches. With ADC technique, once signal gets sampled and have binary representation, algorithms can be introduced to process the signal and acquire the information needed. One of the many algorithms for this purpose is Fast Fourier Transform (FFT).

FFT is one of the most widely used algorithms for signal processing, which can convert signals from the time domain to the frequency domain, Figure 3.5 shows an example of such conversion from the time domain to the frequency domain by FFT.In this example, the analyzed signal is superposed by two pure sinusoidal signal with different magnitude and frequency, while it's hard to tell from the time domain, the individual signals are clear once projected into frequency domain through FFT. Mathematically, FFT can be expressed by Eq. (3.1) and Eq. (3.2), where $\mathbf{W_N}$ is the twiddle factor, $\mathbf{X_k}$ is signal in the frequency domain and $\mathbf{x}_n$ is signal in the time domain.

By performing FFT, signals with different frequencies and magnitudes can be extracted from their time domain counterpart, which allows subsequent analyzers to determine the signal components that form a given input signal.

$$\mathbf{X_k} = \frac{1}{N}\sum_{n=0}^{N-1}\mathbf{x}_n \exp(\frac{-j2\pi nk}{N}) = \frac{1}{N}\sum_{n=0}^{N-1}\mathbf{x}_n {\mathbf{W_N}}^{nk} \tag{3.1}$$

$$\mathbf{W_N} = \exp(\frac{-j2\pi}{N}) = \cos(\frac{2\pi}{N}) - j\sin(\frac{2\pi}{N}),\ where\ 0 \leq k \leq N-1 \tag{3.2}$$

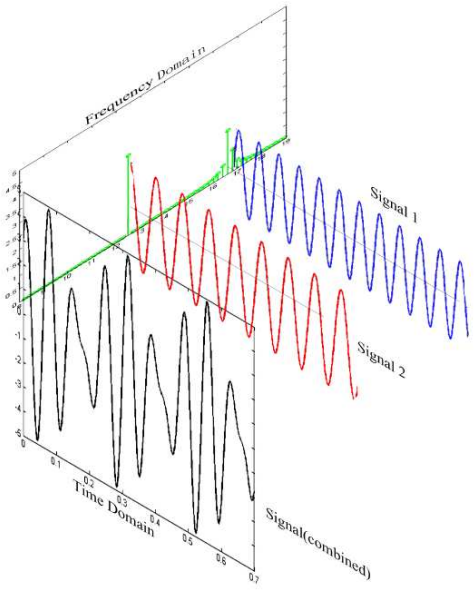


Figure 3.5: Signal after fast Fourier transform.

3.2.2 FFT and Radix-2 Structure

Hardware implementation of FFT can be realized by using the Radix-2 butterfly structure [38], as shown in Figure 3.6. The Radix-2 butterfly structure essentially consists of two multiplications and three additions/subtractions.

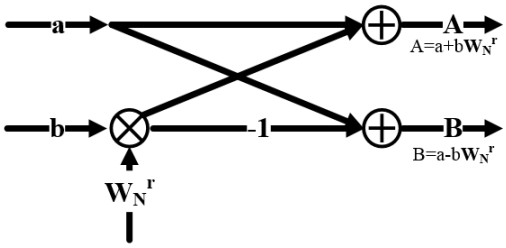


Figure 3.6: Radix-2 butterfly structure unit

For FFT to be practical for signal processing applications in biomedical devices, the number of data points an FFT can handle is an important parameter. With a fixed sample frequency f_s

from ADC, the frequency resolution is determined by $\frac{f_s}{N}$, where N is the FFT points.The more number of data points an FFT can handle, the more accurate the spectrum output will be. However, increasing the number of data points will significantly increase the hardware complexity of FFT, and consequently, its power consumption. For example, a simple 2-point FFT needs 4 multipliers and 6 adders/subtractors as datapath for both real and imaginary part, a 64-point will need $32 \times 6 = 192$ times bigger area than 2-point one, a 1024-point then will be $512X10 = 5120$ times greater compared to 2-points.This is because the complexity of FFT increases not only as a function of the number of input points but also that of the number of stages. Specifically, for 2^N points FFT, it will end up with N stages and 2N entrances using the Radix-2 architecture, Figure 3.7 shows an example for 64-point FFT based on radix-2 butterfly.

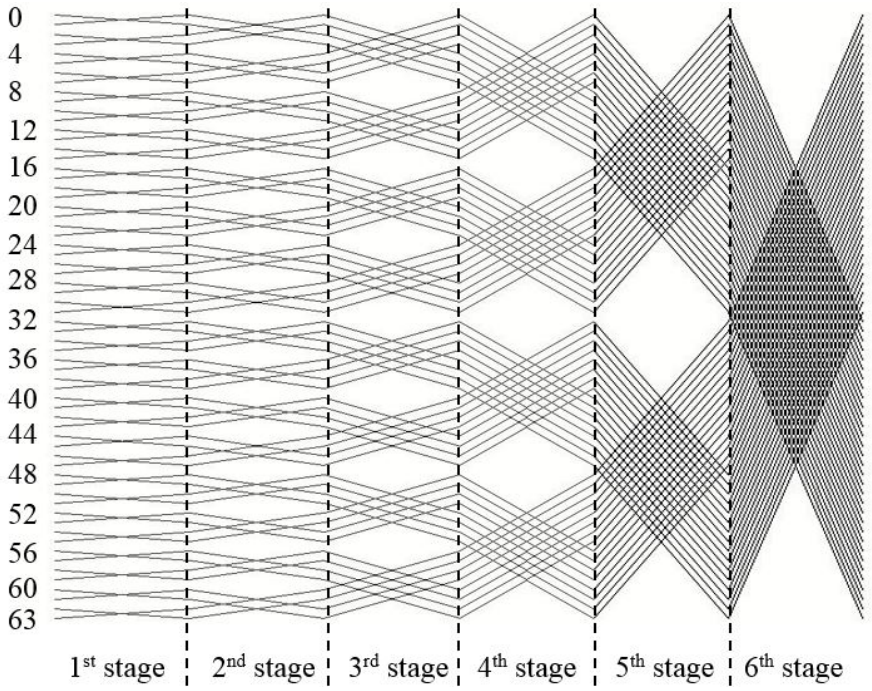


Figure 3.7: 64-point FFT butterfly structure

3.2.3 Smoothing Algorithms

Noise is a practical concern when deal with analog signals. When electrical signals propagate through electrical circuits, the final output will add up the accumulated noise upon the ideal signal.

Noise source can come from the components such as resistor and transistor which contribute to thermal noise and flicker noise. Thermal noise is caused by the thermal agitation to carriers in any circuit [39]. For resistive component, the noise power spectrum is proportional to $4k_BTR$, where k_B is Boltzmann constant which approximately equals to $1.38 \times 10^{-23} J/K$, T is temperature and R is resistance. The flicker noise is caused by the trapping and release of carrier when it flows through [40]. In MOSFET, the flicker noise occurs when carrier is trapped by the oxide layer beneath the gate, and therefore fluctuate the surface mobility. Because the flickers noise power is inversely proportional to frequency, it's also referred as $\frac{1}{f}$ noise. Additionally, the signal from elctronics also suffer from electrical magnetic interference (EMI) from on-board and off-board sources.

Smoothing algorithms can be applied to relieve the burden on hardware filtering. The idea behind the smoothing is generally moving average. Such approach smooths a data point by taking reference points at its vicinity, as in Eq. (3.3). The number of reference points for each calculation determines the average window. A proper window size should be selected to avoid distortion as well as to maximize the filtering effect. Figure 3.8 shows the effect after applying the smoothing algorithm including an distortion example when the smoothing window size is too large .

$$Y_{i,smoothed} = \frac{Y_{i-j} + Y_{i-j+1} + ... + Y_i + Y_{i+1} + ... + Y_{i+k}}{n},\ n = j + k + 1 \tag{3.3}$$

3.3 A Low Power 64-point Bit-Serial FFT Engine

As FFT acts as the core algorithm to acquire the complex impedance data, we proposed a customized FFT CMOS processor chip to minimize the area and cost.

3.3.1 Existing FFT Implementations

FFT algorithm essentially consists of multiplications and additions/subtractions, therefore it can be implemented with using existing co-processor such as FPGA with programming [41]. However, FPGA board is normally designed for general purpose application rather than FFT, which

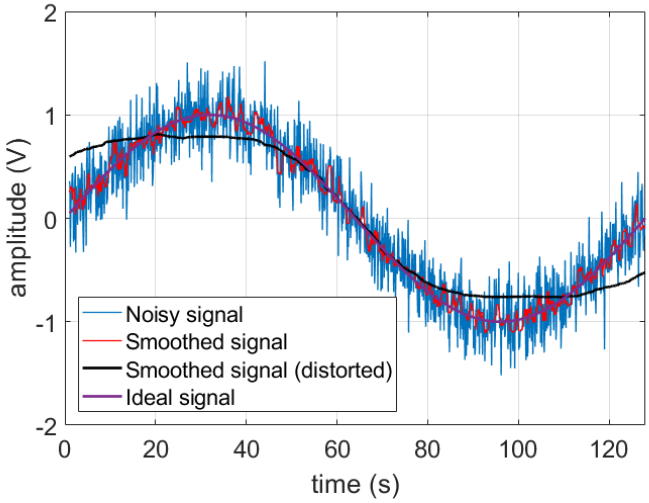


Figure 3.8: Results With smoothing algorithms

introduce a lot of redundancy in terms of chip size and power consumption. For applications that don't need digital signal processing functions other than FFT, customization would be a better choice.

Existing works such as [42] [43] are customized FFT implementation. The design complexity of such FFT grows as number of points and number of bits increase, the size of chip and power consumption could be too high that they are not suitable for battery based implantable devices. The proposed implementation in [44] attempted to make a better tradeoff among performance, complexity and power consumption by developing a two-dimensional structure of 8-point FFTs, which reduced the complexity of multiplication and managed to save storage unit. In [45], Radix-4 butterfly is used to implement the FFT for latency purpose. As the arithmetic components are getting reduced, it takes more time to wait for data from previous stage of Radix-2 butterfly structure and proceed operation, which is the cause of its latency growth. With the Radix-4 structure, more incoming data are grouped and proceeded together which reduce the number of stages equivalently compared to the Radix-2 counterpart. The amount of time for the FFT to hold and wait for data is therefore reduced, which improve the latency.

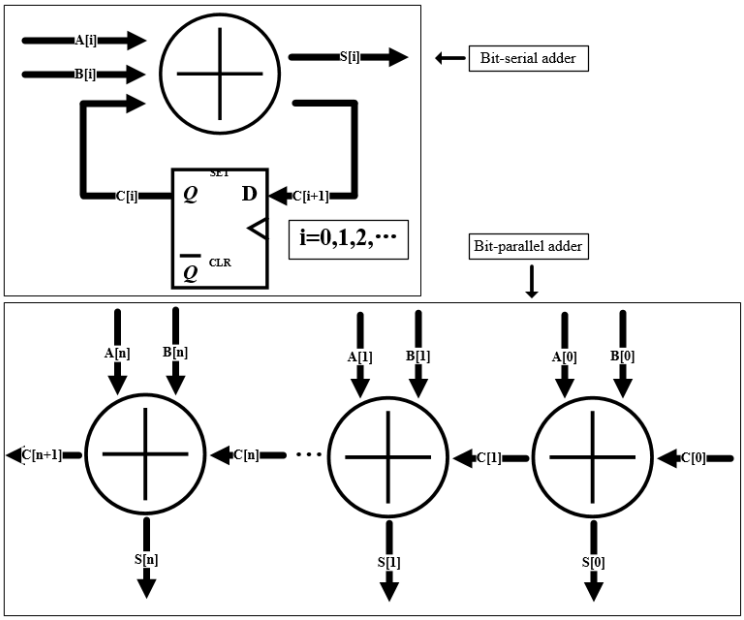


Figure 3.9: Bit-serial vs bit-parallel adder

3.3.2 Bit-serial vs Bit-parallel Logic

The hardware implementation complexity of FFT grows as the number of point increases. In general, the complexity of N-point FFT requires $\frac{N}{2} \times (log_2 N)$ Radix-2 butterfly structures. In addition, the complexity of the basic arithmetic components as multipliers and adders are getting more complicated as the number of bits grows. Even though the problem in the adder is not significantly severe since its complexity goes up linearly, complexity of multiplier is a quadratic function of the number of bits it handles which can be huge in terms of area when dealing with 10 more bit and results in more power consumption.

Bit-serial logic is different from traditional bit-parallel logic in a way that the data flow is coming out in a serial manner. Figure 3.9 compares a bit-serial adder with a bit-parallel adder. Bit-serial arithmetic trades performance with complexity and scalability.

In traditional bit parallel logic, the number of full adder in a multiplier is a quadratic function of number of bits. In bit-serial logic, as shown in Figure 3.10, the complexity of full adder is a linear function of the number of input bits [46] [47]. Although extra components such as D-flip flops

(D-FF) are needed, the amount of D-FF is still in a linear relationship with number of bits. Such complexity would become more prominent as the number of bits increases for high resolutions.

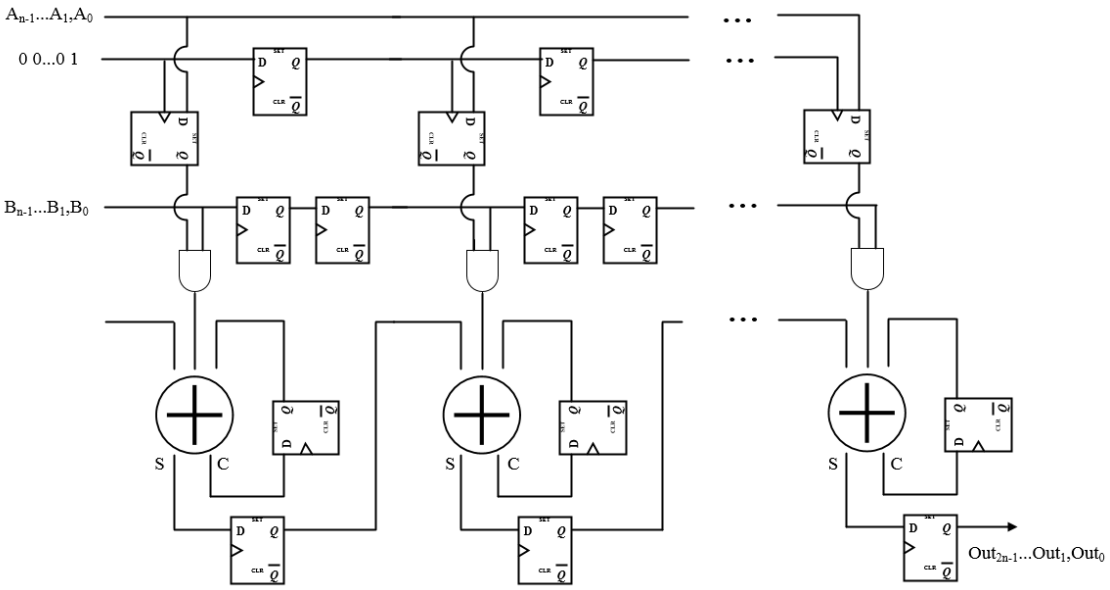


Figure 3.10: Bit-serial multiplier

At the logic level, the design complexity can be represented by the number of transistors. Using the standard CMOS gate structures, Table 3.1 shows the typical transistor cost for basic building blocks for digital FFTs.

Table 3.1: Transistor counts for different gates.

Logic gate component	Transistor counts
Inverter	2
NAND-2	4
NOR-2	4
D-FF	12

Figure 3.11 shows the complexity growth of bit-serial and bit-parallel implementations of adders and multipliers as a function of data word length. However, bit-serial logic has longer latency. For an N-bits adder, the latency is N cycles, and for an N-bits multiplier, the latency is

3N cycles. Nevertheless, in applications not sensitive to prompt data processing, bit-serial technique can be a worthy tradeoff since silicon area and power consumption are more stringent than performance and therefore can be safely applied for FFT implementation.

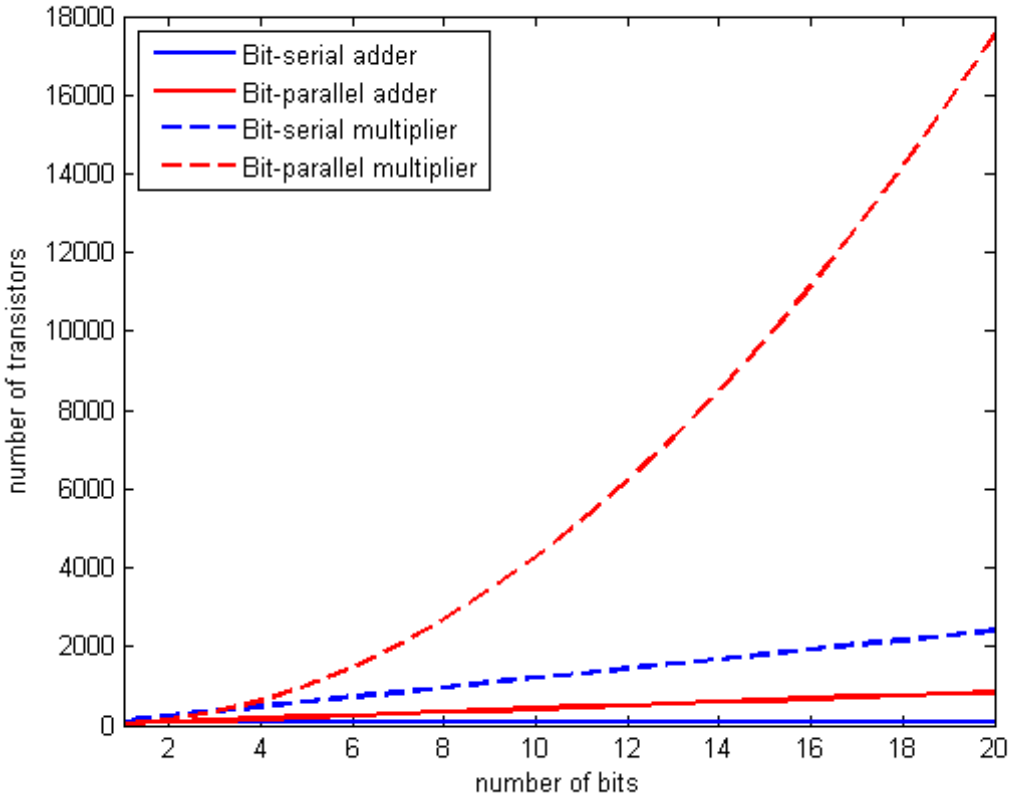


Figure 3.11: Bit-serial vs bit-parallel in number of transistors.

3.3.3 Top Level Architecture

The top level diagram for the 64 point bit-serial FFT is illustrated in Figure 3.12. In this design, the bit inputs go into logic component in serial and pass through a multiplier, two adders/subtractors and a first in first out shift register (FIFO) at each stage. For 64-point design, it has 6 stages in total, but only with one row of Radix-2 unit in a serial manner rather than 32 of them. In other words, all the Radix-2 butterfly operations at each stage are sharing the same arithmetic unit. Processing gets started from the first stage, the digital data bits that represent time domain information

for given signal would be aligned in serial and go into the Radix-2 butterfly in pairs. Equivalently speaking, such structure would finish the operation row by row in each stage and then proceed to the next one. On the other hand, since each stage does not need to wait for the previous stage to finish all of calculations for initiating itself, the advantage of pipelining will result in improvement in throughput. Once the result from the Radix-2 butterfly has been stored in the FIFO, a logic controller will gate the clock signal in the FIFO to make the data bit stays in each D flip flop (DFF) as well as save power. Meanwhile, it would allow the next set of inputs to proceed starting a new period. In addition, the control logic for each stage is also responsible for the next stage to have it fetch data stored in the previous stage's FIFO and initiate the next stage's operation when it's ready, which makes such design highly dependent on control logic.

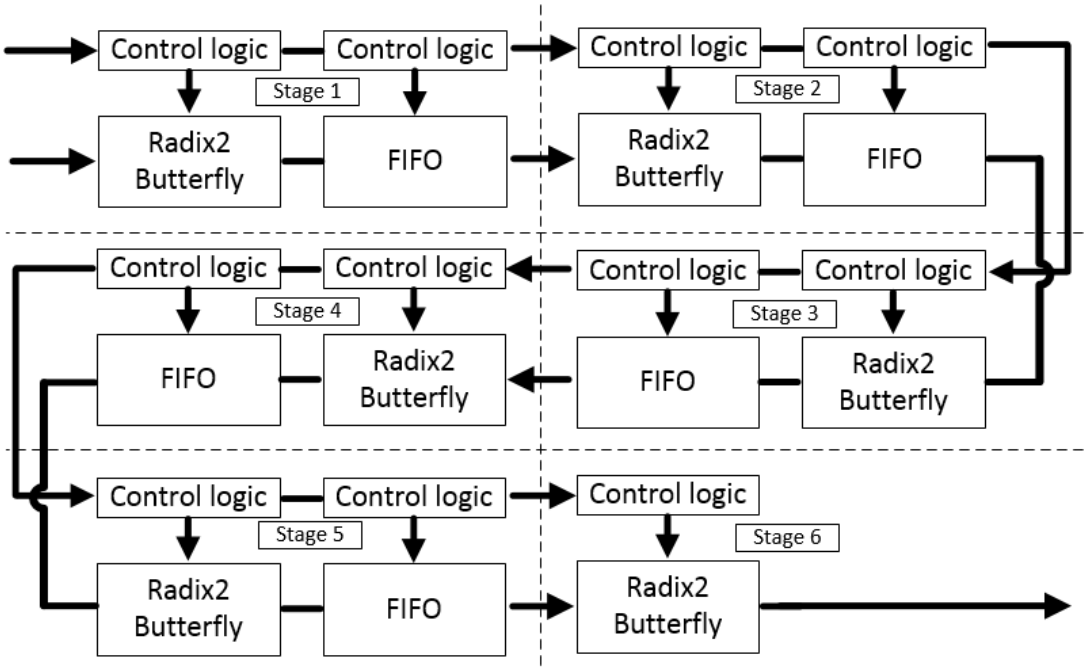


Figure 3.12: Proposed bit-serial FFT topology

3.3.4 Radix-2 Unit Design

Inputs a , b as well as twiddle factor $\mathbf{W_N}$ in FFT algorithm are complex number which consist of real part and imaginary part, therefore $a = a_R + a_I i$, $b = b_R + b_I i$ and $\mathbf{W_N} = \mathbf{W_{N}}_R - \mathbf{W_{N}}_I i$.

The equations for calculating A and B are shown in Eq. (3.4) and (3.5), respectively. And if the real part and imaginary part are separated out, it ends up with Eq. (3.6) (3.7) (3.8)and (3.9). These equations indicate that the Radix-2 butterfly can be implemented by using 2 multipliers and 3 adders/subtractors for either the real part or the imaginary part. In order to calculate a complex number, a Radix-2 for A_R and B_R and another for A_I and B_I are needed.

$$A = a_R + b_R\mathbf{W_{N}}_R + b_I\mathbf{W_{N}}_I + (a_I + b_I\mathbf{W_{N}}_R - b_R\mathbf{W_{N}}_I)i \tag{3.4}$$

$$B = a_R - (b_R\mathbf{W_{N}}_R + b_I\mathbf{W_{N}}_I) + (a_I - (b_I\mathbf{W_{N}}_R - b_R\mathbf{W_{N}}_I))i \tag{3.5}$$

$$A_R = a_R + b_R\mathbf{W_{N}}_R + b_I\mathbf{W_{N}}_I \tag{3.6}$$

$$B_R = a_R - (b_R\mathbf{W_{N}}_R + b_I\mathbf{W_{N}}_I) \tag{3.7}$$

$$A_I = a_I + b_I\mathbf{W_{N}}_R - b_R\mathbf{W_{N}}_I \tag{3.8}$$

$$B_I = a_I - (b_I\mathbf{W_{N}}_R - b_R\mathbf{W_{N}}_I) \tag{3.9}$$

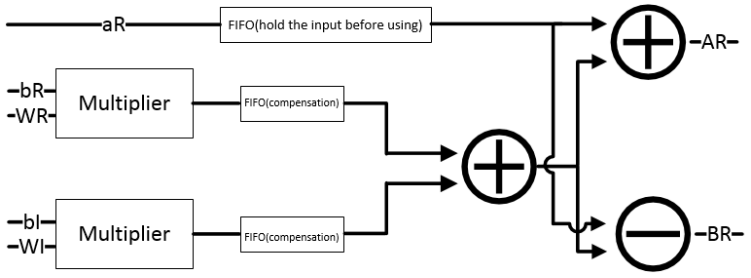


Figure 3.13: Radix-2 implementation for real part

Figure 3.13 shows the diagram for real part Radix-2 hardware realization. Since the number of bits will grow after each stage, the latency for each stage Radix-2 will be different. To simplify the control logic design, data words are padded to make the execution period uniform at 90 cycles for each input pair in each Radix-2 calculation throughout the pipeline. Shift register string(D-FF string) is introduced in Radix-2 butterfly as a compensation delay line to allow data padding. In

addition, another D-FF set is needed to hold the input data bits, because even the input pair is latched into Radix-2 component, one of them needs to go through multiplier and compensation D-FF string while the other does not.

3.3.5 Multiplier Design

The implementation of the bit-serial multiplier is shown in Figure 3.10.In addition to standard bit-serial multiplier design using bi-serial adders, we would like to highlight two design details. First of all, since there is a pulse signal that will be needed as a trigger to latch in the data coming from D-FF of the previous stage, it may cause logic error if the incoming data bit does not catch the trigger edge. Therefore, buffers were inserted in this design to delay the clock signal slightly to eliminate this hazard. Secondly, for N-bit two's complementary multiplication, there is very often overflow condition that results in incorrect outcome. For instance, for 20-bit binary number 1111 1111 1010.0000 0111 $\times$ 0000 0000 0000.1011 0101, rather than result like 1111 1111 1011.1100 0110, it ends up with 1011 0100 1011.1100 0110 instead. A solution in [48] was to pad the input. If there is an input as 16-bit, it will be extended to be 32 and then proceed processing. However, in this design, 8 MSB bits are assigned for integer and 8 LSB bits are for decimal, it would end up with 64-bit output which contains the highest 48-bit for integer and the lowest 16-bit for decimal after multiplication. Nevertheless, since the twiddle factor is in -1 to +1 range, the highest 40-bit and lowest 8-bit can be discarded which makes the output bit remains at 16. In other words, the number of bits after multiplication does not grow.

3.3.6 Adder Design

Although the result after multiplication does not increase, the result grows after addition/-subtraction, and therefore overflow detection and bit extension techniques are required for bit-serial adder implementation. The overflow detection is designed by comparing $Carryout_{N+1}$ and $Carryout_N$ with XOR gate, which is a well-known technique. For bit-serial logic, extra D-FF and a control logic are need to store the $Carryout_N$ and give signal at certain clock cycle to detect

overflow event receptively. And control logic is also responsible for extending the MSB of digital output to grow the number of bits.

3.3.7 FIFO Design

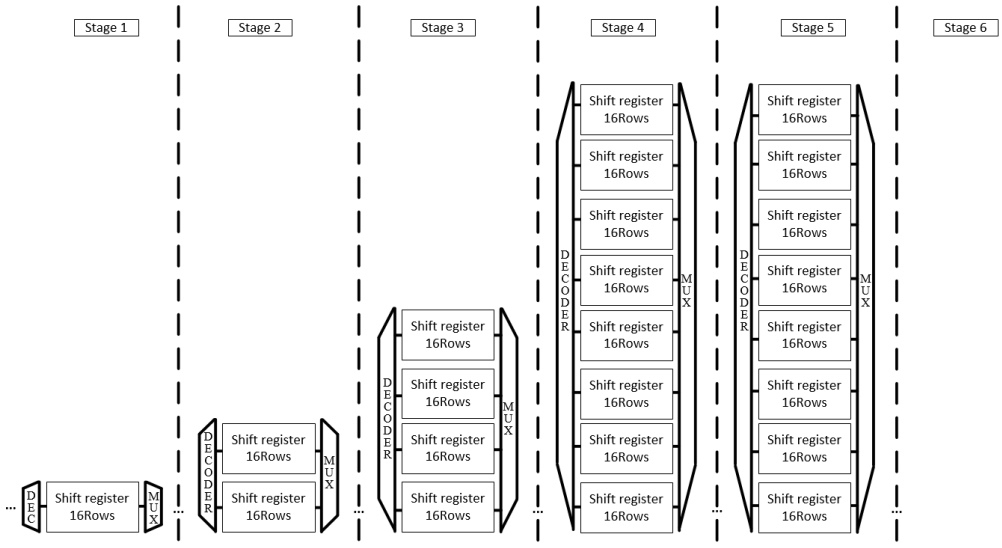


Figure 3.14: FIFO structure for each stage.

As the number of Radix-2 units is reduced in bit-serial architecture, the availability of intermediate value for FFT operation is an issue. Since the FFT implementation is based on the Radix-2 butterfly,the later stage will need results coming from previous one. For instance, in stage two, in order to calculate the result from Row0 and Row3, it needs to hold for at least two operation periods (180 clock cycles) to have both of them available. As a consequence, storage components are needed to hold the values that can't yet be operated. FIFO structure is chosen in this design to store intermediate results, Figure 3.14 shows the FIFO structure in the proposed design. Since it needs more time to hold the intermediate variable as processing progresses, FIFO structure is lesser reusable and therefore results in more register components as stage increments. Nevertheless, since FIFO is normally built based on D flip flop which can be as simple as 12 transistors, it can still be area efficient. Each FIFO has a corresponding set of mux and decoder to handle FIFO inputs and outputs.

Chapter 4

The Crexens^{TM} Architecture

4.1 Crexens^{TM} Electrochemical Analyzer: Generation 1

In this section, an initial electrochemical analyzer design is proposed. The design consists of signal stimulus generation and signal acquisition for EIS. Figure 4.1 shows the overall architecture. The circuit is capable of generating a stimulus signal consisting of 32 frequencies with a resolution of 2 Hz at low frequency band and 62Hz at medium frequency band, the signal is designed as shown in Figure 4.2. The frequency range for the stimulus signal is from 2 Hz to 2 kHz which is divided into two bands. The circuit generates a composite signal for one band at a time. The composite signal provides a compact representation of the desired signal spectrum with 0.3% error in amplitude. The response signal acquisition is accomplished with signal amplification using a transimpedance amplifier, followed by an analog-to-digital converter (ADC). An average error lesser than 2.14% in measured impedance was achieved. The limit of detection 10 Ohms was obtained. The design was implemented on a 2-layer PCB board with a total footprint of 75.26 cm^2. The design was intended as a low-cost and high accuracy point-of-care (POC) platform for pathogen detection.

4.1.1 Stimulus Signal Generator

The stimulus signal generator combines 32 sinusoidal signals into an aggregate analog signal. Figure 4.2 illustrates the processing of combining sinusoidal signals at different frequencies into one composite signal (black line). To reduce the peak-to-peak value of the combined signal, a low crest factor signal was designed by inserting phase offset to each individual sinusoidal signal frequency similar to that in [49]. Table 4.1 shows the phase pattern used to generate the aggregate signal. Each frequency component was designed to have a 10 mV amplitude. A programmable microcontroller unit (MCU) is used to initialize the SRAM with digital samples of the

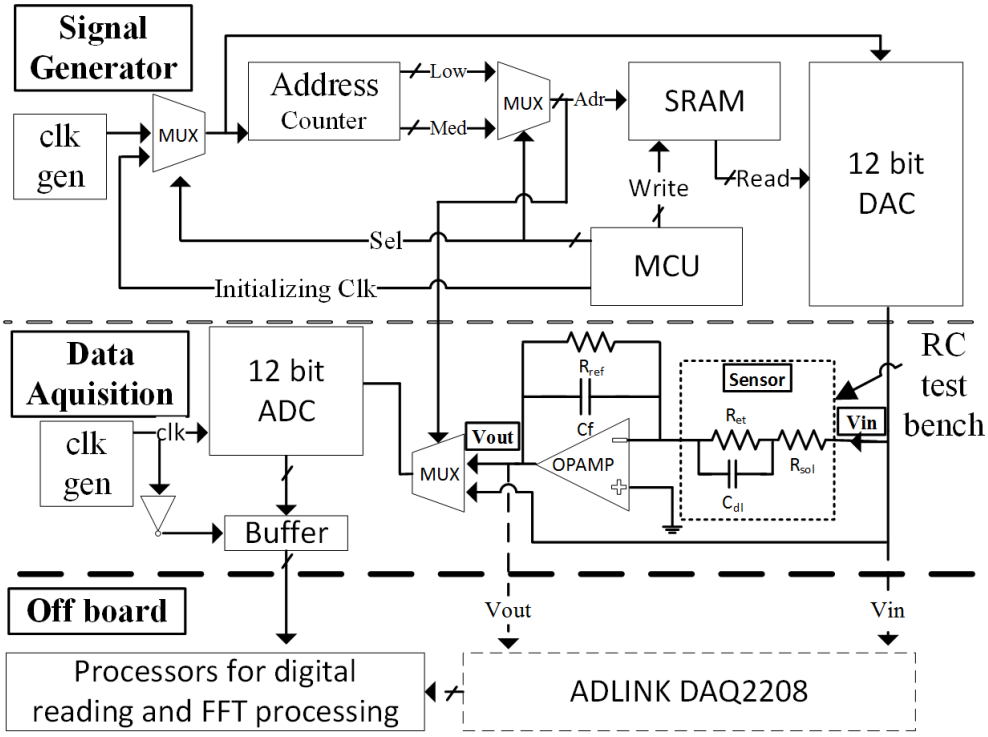


Figure 4.1: System level architecture for generation 1 design.

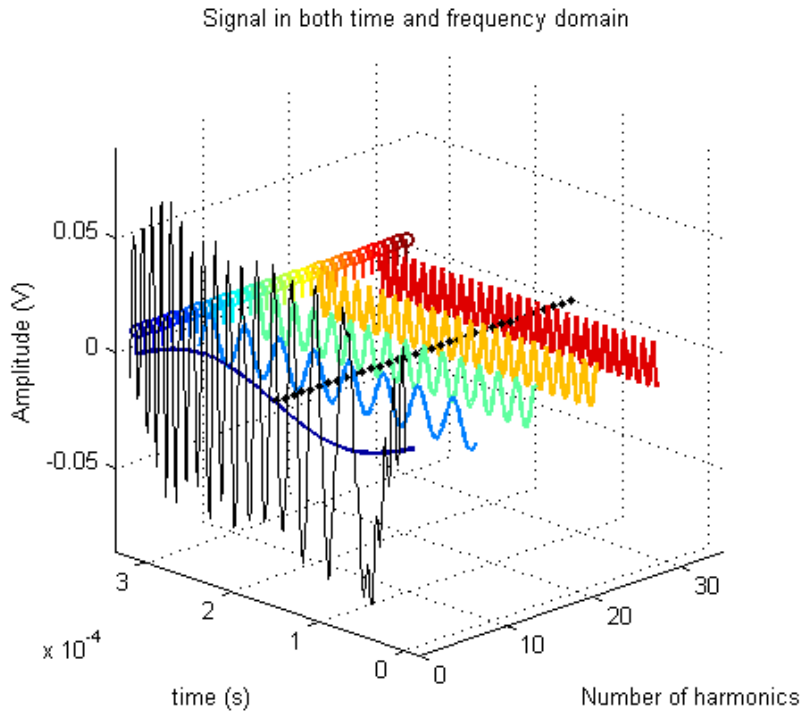


Figure 4.2: Illustration of multi-sine signal in time and frequency domain.

Table 4.1: Phase shift for each frequency component.

Sine i	ϕ_i (rad)	Sine i	ϕ_i (rad)	Sine i	ϕ_i (rad)
1	0	12	2.7873	23	-1.2014
2	3.1067	13	-2.0961	24	0.8386
3	0.0835	14	1.3176	25	0.3342
4	-2.2936	15	2.0630	26	-0.3860
5	2.0544	16	2.6231	27	2.0323
6	-1.5791	17	-0.9922	28	-2.7892
7	-3.0282	18	0.7026	29	0.9178
8	1.6772	19	-0.6575	30	-2.0236
9	-2.2803	20	-2.0723	31	-0.9686
10	2.4914	21	-2.4123	32	2.6719
11	1.7254	22	-2.0217		

composite signal. Each composite signal has 256 data points. The SRAM is driven by an address counter to select composite signal values in an incremental fashion. Multiplexers (MUX) controls the selection of two different frequency bands. The digital values of the composite signal are converted to their analog counterparts by a 12-bit DAC before being applied to sensor for impedance measurement. A clock generator synchronizes the composite signal generation process.

4.1.2 Impedance Measurement Unit

The impedance measurement unit consists of a transimpedance amplier (TIA) for converting the response current to an output voltage. The transimpedance amplifier uses an operational amplier (opamp) with a reference resistor in the feedback. A capacitor Cf in parallel with the reference resistor is used for reducing TIAs output noise at the cost of reducing TIA bandwidth. For a given Rref and Vin , the sensor impedance Zsensor is determined by Vout as shown in Eq. (4.1). The choice of Rref and Cf is to prevent output railing within the estimated range of Zsensor and TIAs bandwidth and noise tradeoff requirement. In this design, a 1.91K ohms resistor and 470 pF feedback capacitor were chosen.

$$Z_{DUT} = -\frac{R_{ref} \times V_{in}}{V_{out}} \tag{4.1}$$

Two steps are taken to measure sensor impedance: 1) .Select and sample Vin and Vout channel respectively to acquire digital data, 2) Use PC processor to run FFT and calculate impedance for both magnitude and phase.

4.1.3 Response Signal Preparation and Data Processing

The sensor response signal from the TIA is digitized by a 12-bit analog-to-digital converter (ADC). The digitized data can then be sent to host computer for further processing. The data processing consists of calculating magnitude and phase information about the sensor impedance using Eq. (4.2) and (4.3).

$$Z_{DUT} = \frac{R_{ref} \times \sqrt{V_{in,Real}^2 + V_{in,Imag}^2}}{\sqrt{V_{out,Real}^2 + V_{out,Imag}^2}} \tag{4.2}$$

$$\theta_{DUT} = \arctan \frac{V_{out,Imag}}{V_{out,Real}} - \arctan \frac{V_{in,Imag}}{V_{in,Real}} - 180 \tag{4.3}$$

4.2 Crexens^{TM} Electrochemical Analyzer: Generation 2

Figure 4.3 shows the system level view for the proposed platform and the goal is to increase the system's functionality for applications. The platform is powered by a USB based rechargeable battery 3.3 V. Signal generation, digital data acquisition/processing are managed by an on-board microcontroller. Automatic gain control and noise reduction filter are implemented to meet the signal-to-noise ratio (SNR) requirements of the applications. The final data are transmitted through Bluetooth 4.0 protocal to user's smart-phone or tablet.

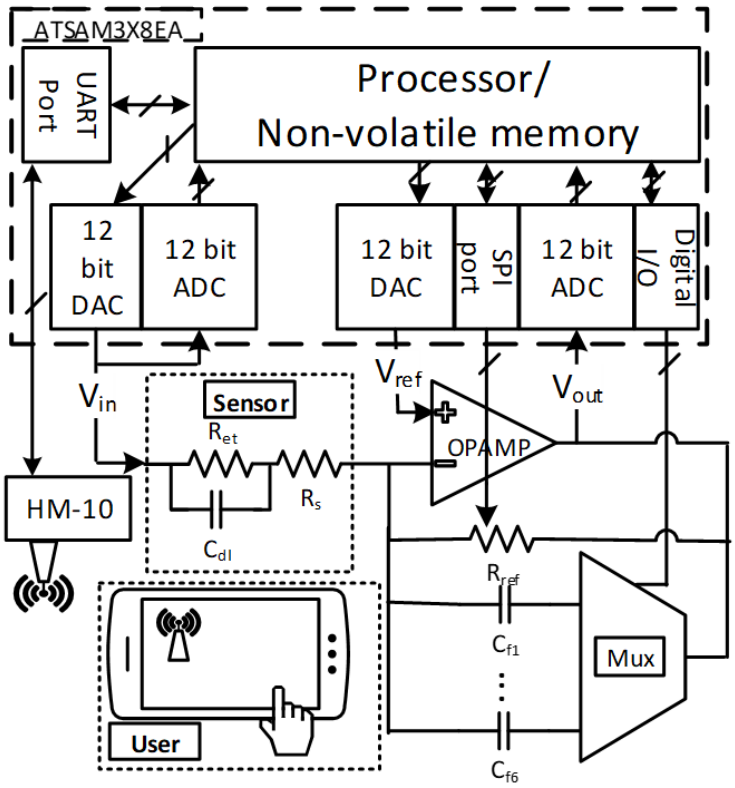


Figure 4.3: System level architecture for generation 2 design.

4.2.1 Signal Generation

This approach has two modes to generate stimulus, a signle-tone mode and a multi-tone mode. In the single-tone mode, one sinusoidal signal at specific frequency will be applied to the DUT and gathered for processing at each time. In the multi-tone mode, a 32 sinsoidal at different frequencies are superposed to become one stimulus, this composite analog signal will then go through DUT, transimpedance amplifier (TIA) and sampled by ADC. The signal processing step will analyze the 32 tone together at once. Therefore, the multi-tone approach can reduce the sweeping effort and save time. One of the issue in multi-tone mode is overshooting as a result of supperposition. To reduce the peak-to-peak magnitude of the combined signal, a low crest factor signal was designed by inserting phase offset to each individual sinusoidal signal frequency similar to that in [49].

The signal for both the single-tone and the multi-tone modes are pre-sampled in Matlab by 256 data points. These data points were then loaded into the non-volatile memory in the microcontroller

unit (MCU) for the 12-bit digital-to-analog converter (DAC). To reduce the impact of stimulus on electrodes, the amplitude of stimulus was controlled to be no greater than 30mV rms.

4.2.2 Read Channel and Data Acquisition

The analog front-end of the read channel is a transimpedance amplifier (TIA). The close-loop gain of the TIA is controlled by a potentiometer or a multiplex resistor array. An array of filtering capacitors in parallel with resistive feedback are multiplexed to reduce noise. The data are sampled at the output of the DAC and TIA through two 12-bit analog-digital converter (ADC) in MCU. Based on ADC data feedback, the potentiometer and mux for the capacitors can be automatically adjusted to maintain the maximum gain without railing issue as well as the best filtering without affecting of signal. This auto gain control and auto filter control feedback can help SNR.

4.2.3 Data Processing and Transmission/Receiving

Once the response signal was sampled, they were fed to the micro-controller to obtain the spectrum information using FFT. The impedance and phase can be further deduced based on Eq. (4.4) and (4.5).

$$Z(\omega) = \frac{R_{ref} \times \sqrt{V_{in_Re}^2 + V_{in_Im}^2}}{\sqrt{V_{out_Re}^2 + V_{out_Im}^2}} \tag{4.4}$$

$$\angle Z(\omega) = arctan(\frac{V_{out,Im}}{V_{out,Re}}) - arctan(\frac{V_{in,Im}}{V_{in,Re}}) - 180 \tag{4.5}$$

The impedance and phase data were then sent out to the Bluetooth chip and further transmitted wirelessly to the user's cellphone. The Bluetooth chip is CC2540 from Texas Instruments. It supports the Bluetooth 4.0 protocol which makes it more compatible with the latest versions of smart-phone devices or tablets.

4.2.4 The Graphic User Interface (GUI) on Android Smart-phone

The handheld device is controlled by an Android smartphone. The phone application is programmed in Java using Android studio. The application can be freely installed on any android system has an android 4.0 or higher version. Bluetooth 4.0 is needed on the smart phone for data transceiver. At the beginning when the GUI is started, a scanning sequence will be launched to find all available Bluetooth device around as well as signal strength, as shown in Figure 4.4. Users need to choose the correct device name to pair with the one in use.

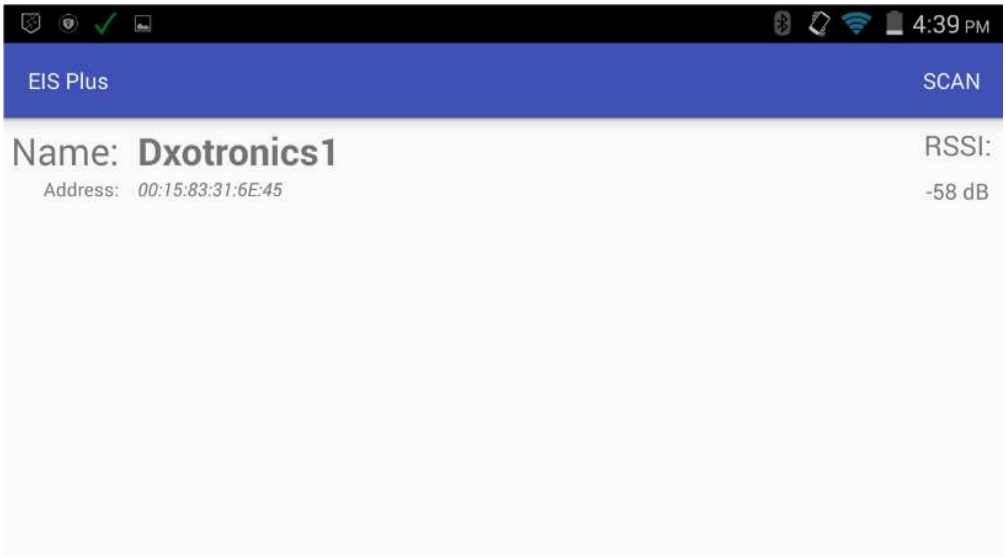


Figure 4.4: GUI interface for device pairing.

The GUI allows user to start EIS test under +200 mV,+100 mV,0 V,-100 mV or -200 mV bias, and automatically sweep from 0.16 Hz to 15 kHz with 96 data point.

Additionally, the system allows user to launch cyclic voltametry (CV) between -1.1 V to +1.1 V in 100 mV/s scan rate, and fast scan cyclic voltametry (FSCV) between -1.1 V to +1.1 V in 100 V/s. The monitor can track the data when it's collecting from the handheld unit, and generate plot once finished, as shown in Figure 4.5. The data will be saved in smartphone's internal memory for the record.

The final product is enclosed in a plastic package. The case has a USB port to connect device to electrode, as in Figure 4.6. It has an on-off rocker button and micro USB-port for charging the

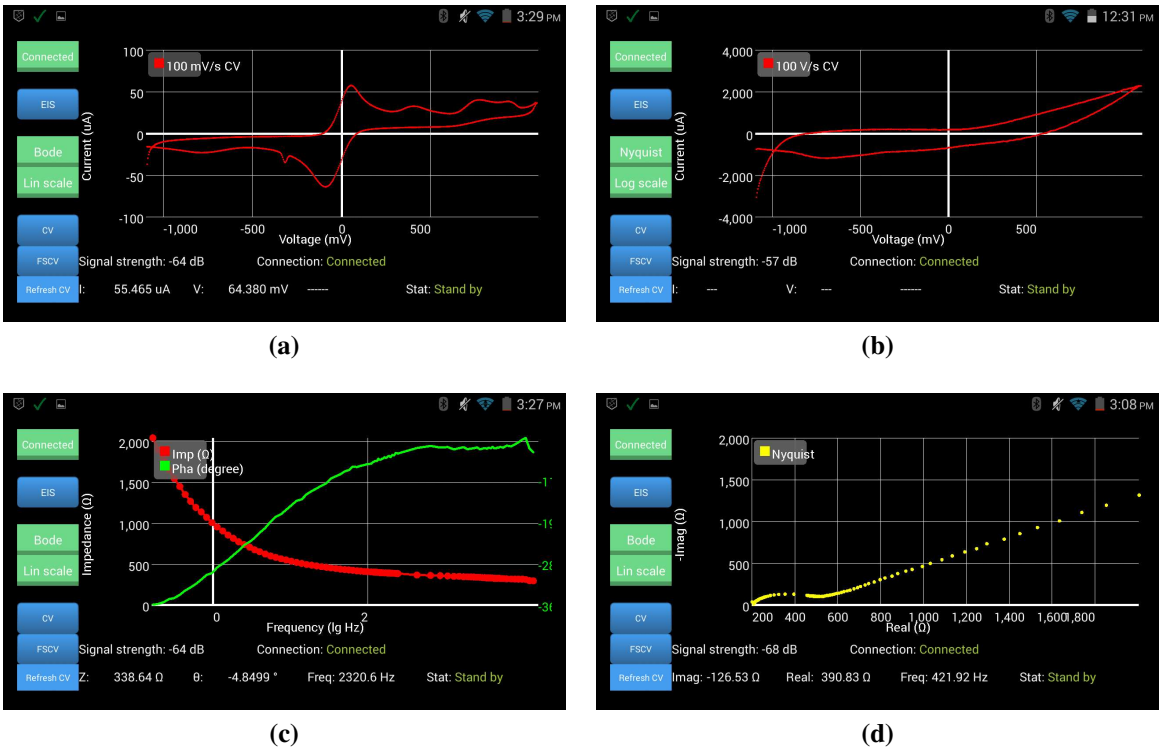


Figure 4.5: (a). CV plot. (b) FSCV plot. (c) Bode Plot for EIS. d) Nyquist plot for EIS.

rechargeable battery. The package is designed with Fusion 360 3D model software and printed through 3D printer at CSU idea to product (I2P) prototyping lab.

4.3 Crexens^{TM} Electrochemical Analyzer: Generation 3

The 3rd generation of electrochemical analyzer is aimed to explore its reconfigurability that allows user to define the capability of the electrochemical analyzer without having too much redundancy that adds up the cost.

4.3.1 The Reconfigurable Electrochemical Analyzer Platform

In order to achieve the desired degree of reconfigurability, both digital and analog parts are made reconfigurable. The general design philosophy for the proposed platform is to have separate analog or mixed signal component on add-on modules, and have a digital controller on the base module to be reprogrammed depending on which type of module is used. The Crexens^{TM}

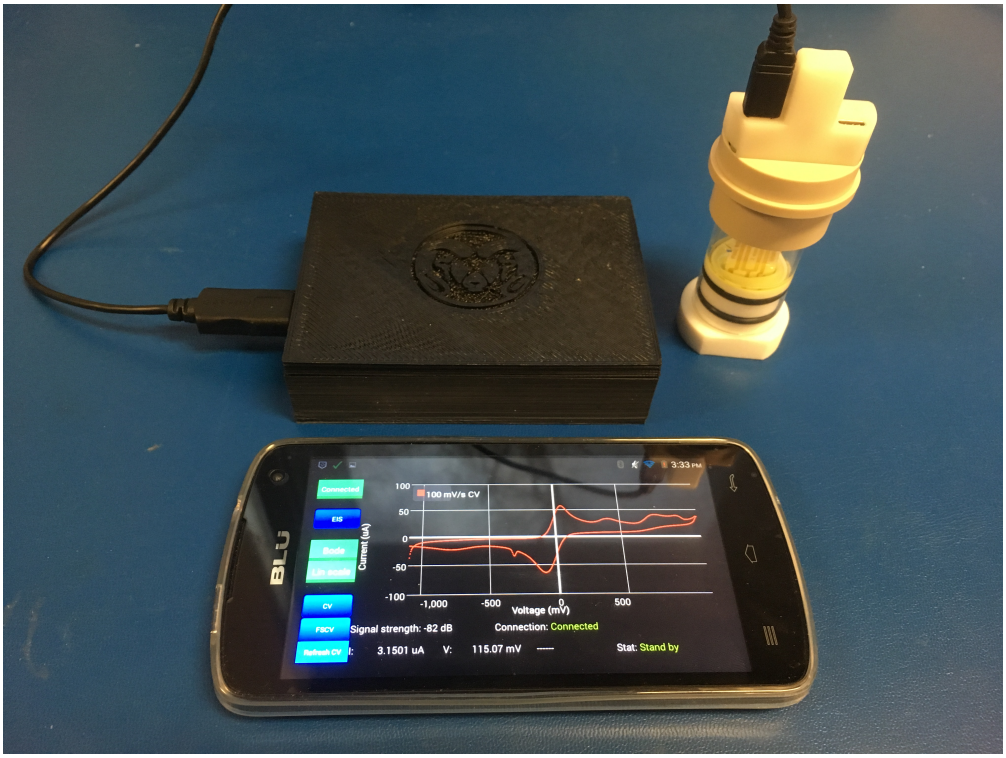


Figure 4.6: Final Product Package (Top View).

Figure 4.7: Final Product Package (Side View).

architecture is shown in Figure 4.8. Spartan-6 series field programmable gate array (FPGA) unit was chosen to act as the central digital controller on the base module due to its high speed and its parallelism capability for handling multiple tasks/modules.

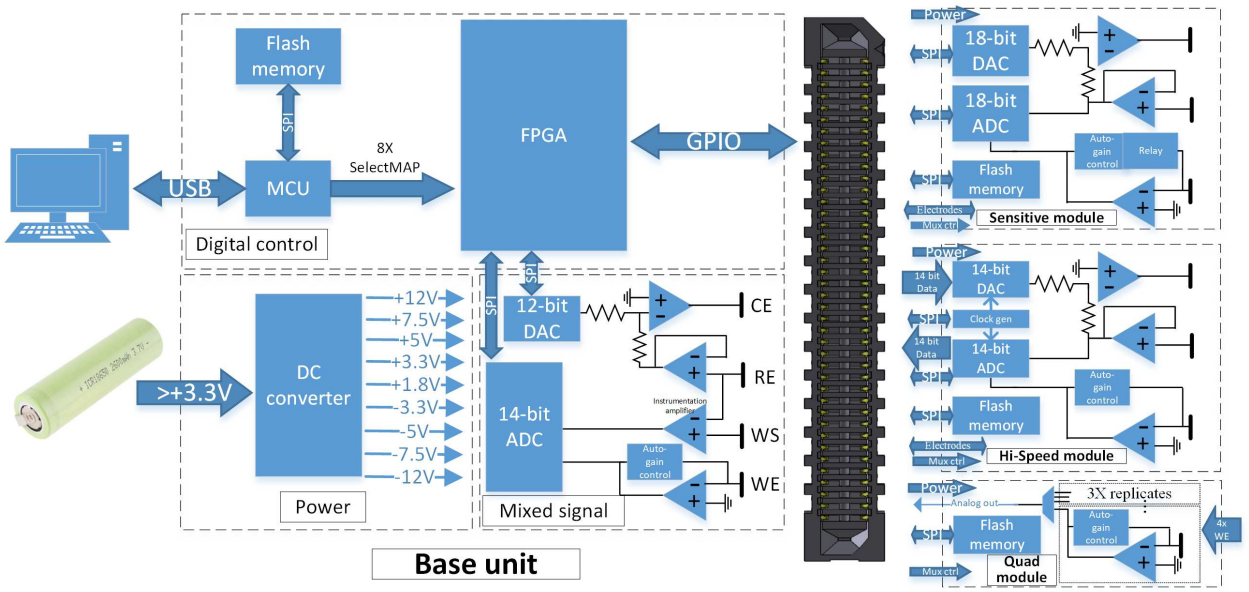


Figure 4.8: System level architecture for generation 3 design.

4.3.2 The Base Unit

The base unit acts as the motherboard for all other add-on modules and it is by itself a fully functional electrochemical analyzer. It comprises three parts: power supply, digital control, and mixed signal circuit. The power supply part is designed to have a number of boost-buck converters and voltage regulators to provide a wide range of supply voltages for not only the base unit itself, but also for the potential add-on modules. The base unit can be battery driven and can operate at a potential >+3.3V. A microcontroller unit (MCU) (ATMEGA32U4 from Atmel) was used as interface between the FPGA unit and the PC-based graphical user interface (GUI). It also acts as pro- gram loader to configure the FPGA routing for specific functionalities. Since the FPGA does not have the internal dedicated memory to store configuration data, an external SPI-based flash memory was used to save the configuration file for the FPGA unit. The configuration file in the

flash memory is read automatically by MCU when the device is powered on and then MCU will load the current configuration data into the FPGA unit. For the reconfiguration purpose, all add-on modules have their own flash memory for storing the preloaded configuration file. The mixed signal part in the base unit is for signal genera- tion and data acquisition, it consists of a 12-bit digital-to-analog converter (DAC), a 14-bit analog-to-digital converter (ADC), a potentiostat, and a read channel. The potentiostat was designed using two closed-loop opamps in the unity gain configuration. The read-channel provides automatic gain control in the range from 10 Ohms to 100 MOhms. The output voltage scale of potentiostat was designed in the range of +-3V with resolution at 1.46 mV. The compliance voltage for the potentiostat is 10V.

4.3.3 The Low-noise Module

The low-noise module provides high sensitivity measurement for applications that require detection of current in the range of sub-nanoamp scale. The low-noise module can be in- serted into the base unit and be automatically recognized by the based unit and allow it to be reconfigured for high-sensitivity measurements. One of the biggest issues in detecting small scale current is the possible leakage on the critical signal path, such as the leakage through transistor gate in the opamp, analog multiplexer channel in off-state, or even poor PCB routing/guarding scheme. To address these issues, opamps that have input bias current at fA range were used for both potentiostat and read channel. Additionally, relay switches were used in- stead of analog multiplexers since they have nearly infinity off- state impedance similar to mechanical switch. These efforts can reduce the distortion of original electrochemical signal before it gets amplified. Meanwhile, the gain control stage was designed to have 4 different gains from 40 kOhms to 5 GOhms to reduce the relay area. An 18-bit ADC and an 18-bit DAC were chosen for the low-noise module with the compromise of operating the module at a lower sampling rate. The potentiostat was de- signed to have +-5 V full scale range with 38 uV resolution. The compliance voltage was in this module becomes 24 V.

4.3.4 The Quad-channel Module

This module was designed for applications that measures four sensor outputs simultaneously. This module consists of four replicated read channel and an auto-gain control circuit. The signal generation and data acquisition were carried out by the DAC, the ADC and the potentiostat on the base unit. The four sensor signals are multiplexed from the on-board multiplexer controlled by the based unit in a user preferred fashion.

4.3.5 The High Performance Module

When it comes to EIS measurements that require operating at a higher frequency range (up to tens of MHz), the high-speed module can be used as a plug-in to extend the performance of the base unit. For high speed applications, the use of ADC/DAC with SPI interface is not appropriate since the serial data communication will push the digital processor to operate at an even higher clock rate, making it cost prohibitive. Therefore, a 14-bit parallel ADC and a 14-bit DAC were used in the high-speed module. They communicate with the FPGA unit on the base unit through the parallel input/output (I/O) ports. By taking advantage of the parallel I/O pins, the FPGA unit does not have to be operating at an extremely high frequency, thus, reducing the overall cost for the Crexens^{TM} analyzer. The ADC and DAC in the high-performance module were synchronized by a dedicated clock generator and operate at 150 MHz and 75 MHz, respectively. The FPGA unit was also synchronized through an output clock from the ADC whenever samples are ready in order to avoid any control failure due to the asynchronous nature of the design in this part. The potentiostat in the high performance module has output swing from -3 V to 3 V with a resolution of 366 uV, and compliance voltage of 10 V.

4.3.6 GUI Design

The GUI provides user control functions such as connection/disconnection from the device, experiment mode options (Amperometry, CV, EIS, etc.), experiment run/stop the measurement, reset the data buffer and export data into excel automatically in real time. It also allows user to set

run-time parameters depend on the user's need. The GUI was implemented in Python. Figure 4.9 shows the snapshot of the GUI.

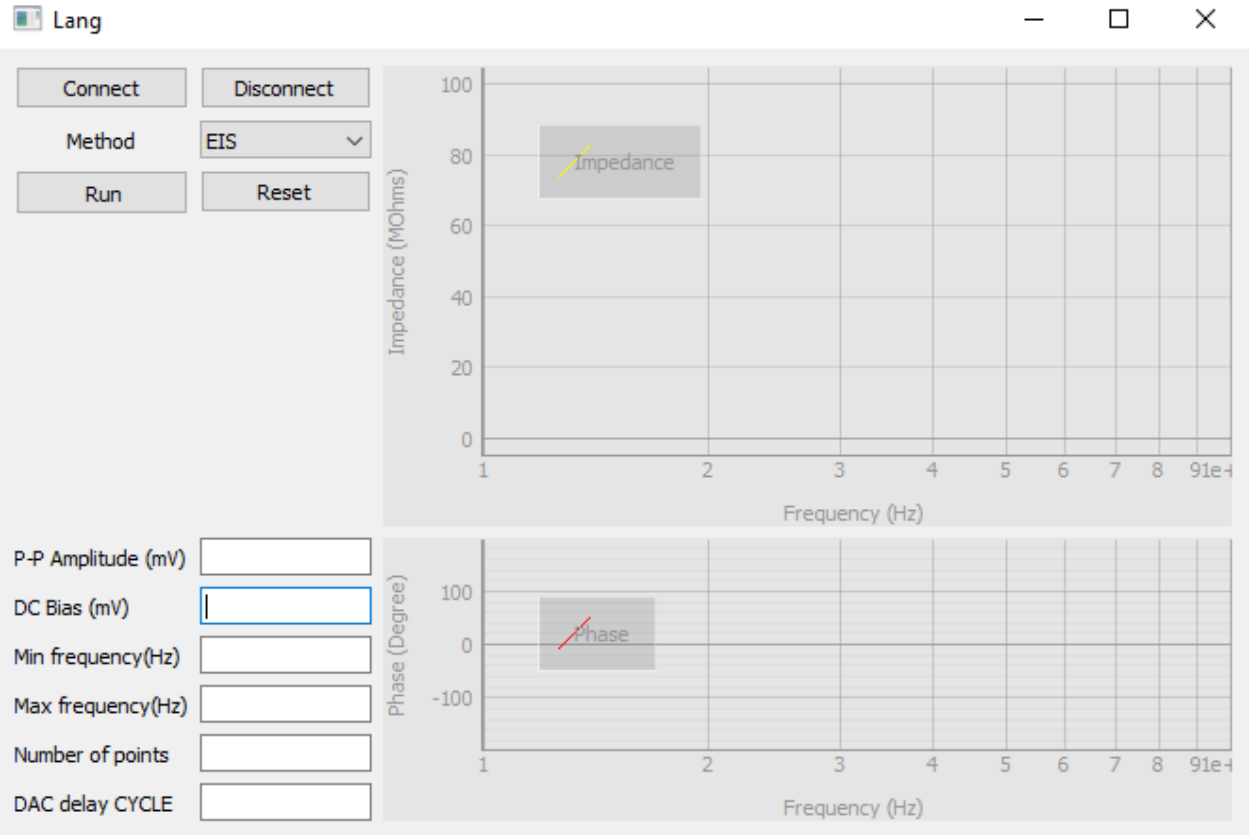


Figure 4.9: Crexens^{TM} GUI on windows PC.

Chapter 5

Design Implementation and Validation

In these chapter, design implementation and validation results from all proposed electronic designs will be shown and discussed.

5.1 Result for the Proposed 64-point Bit-serial FFT Processor

5.1.1 Simulation Setup

To validate the proposed FFT design, a time domain signal is given for testing. The data input to FFT is a superposition of two signals as shown in Eq. (5.1). The sine signal has a 2 V amplitude with 16.67 Hz frequency and cosine signal has a 3V amplitude with 12.5 Hz frequency respectively. A Matlab testbench was created as a baseline comparison to the results from the proposed hardware design. With 100Hz sample rate and 64 data points, the results for both time domain and frequency domain output from Matlab testbench are shown in Figure 5.1 (a), (b).

$$x_n = 2 \times \sin(2 \times \pi \times \frac{100}{6} \times t) + 3 \times \cos(2 \times \pi \times \frac{25}{2} \times t) \tag{5.1}$$

The entire 64-point FFT was implemented in a commercial $0.18\mu m$ CMOS process. The test for proposed FFT processor is subject to the same input signal as shown in Eq. (5.1) with the same sampling rate. Figure 5.1 (b) shows the FFT outputs from both the proposed hardware implementation and from the Matlab testbench. Figure 5.1 (c) (d) further illustrates the error between two implementations in percentage, which shows that the error is a function of amplitude. As spectrum amplitude goes up, error goes down. This is because 8 bits are assigned for decimal value in the proposed design which provides a resolution of 3.90625×10^{-3}. Therefore for spectrum amplitude close or even lesser than that, error will increase a lot. That's why the errors for the very last two frequency are significantly higher, since their amplitude are 3.6×10^{-3}(with 12.6% error)

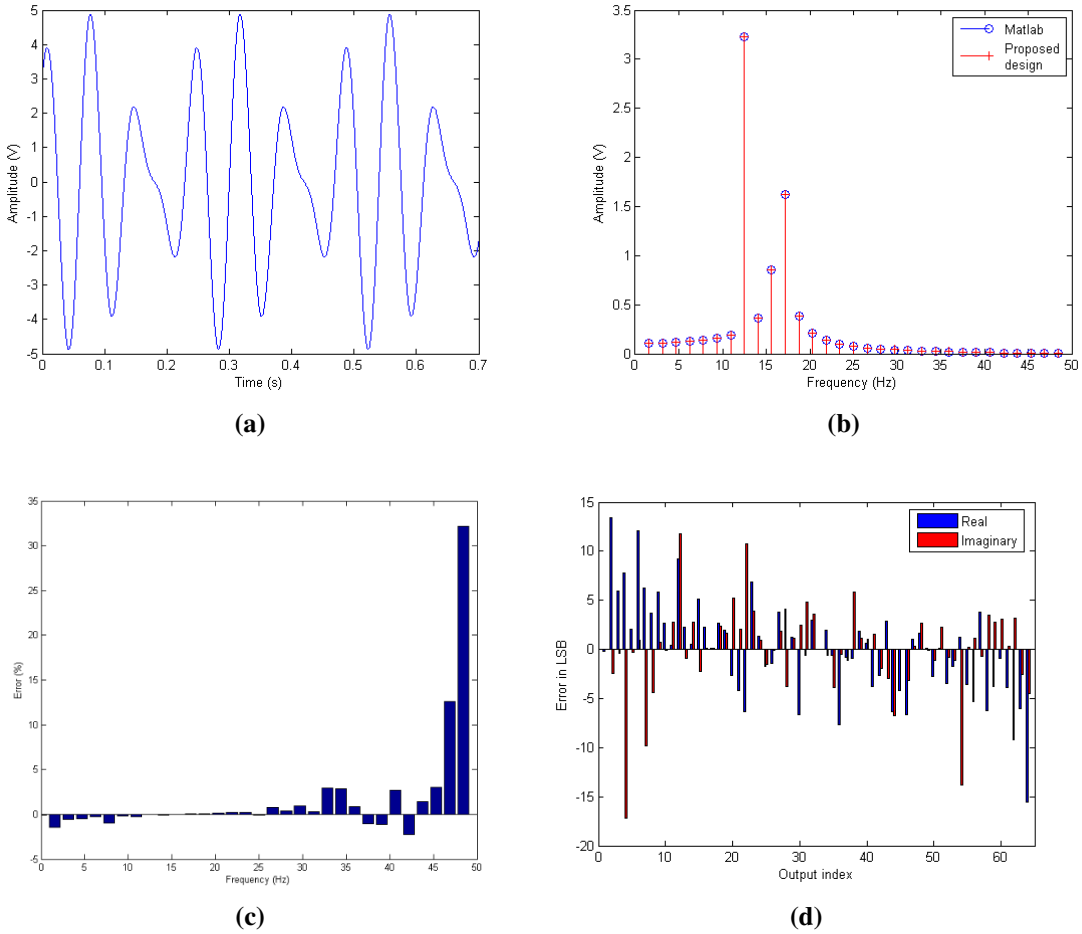


Figure 5.1: (a) Test Signal in Time Domain. (b) Frequency Domain Signal. (c) Error Compared to Matlab Implementation. (d) Error in Real Part and Imaginary Part.

and 1.8×10^{-3}(with 32.2% error) respectively. Therefore, the errors in these frequency bias are meaningless. The errors near the input frequency are extremely small.

5.1.2 Performance

In proposed design, it needs 8861 clock cycles to complete one set of 64-point operation. This is 177.22 μs latency at 50MHz clock rate. By taking advantage of pipelining, the execution cycle can be reduced to $2 \times 32 \times 90 = 5760$ for continuous input, which results in 115.2μs for each set of 64-point FFT. Therefore, the throughput is 0.55 Mpoints/s. Since this design is aimed at minimizing complexity, the FIFO size of the penultimate stage is designed as half as it supposes

to be, which consequently reduces the capability to store intermediate data and results in longer execution time. By doubling the FIFO size of second last stage, the throughput can potentially be improved to be $32 \times 90 = 2880$ clock cycles, which is 57.6μs or 1.11 Mpoints/s. Although it looks like a good tradeoff by increasing some area and power to get throughput twice faster, the execution time within hundred of microsecond is sufficient for slow throughput application.

5.1.3 Comparison Among Different Designs

$$Normalized\ Area = \frac{Area}{(Technology/0.5\mu m)^2} \tag{5.2}$$

Table 5.1 compares the proposed design with the existing FFT implementations in the past. To make the comparison more objectively, the silicon areas of all the reported design have been normalized to an equivalent of 0.5μm process as shown by Eq. (5.2). The power consumption has been normalized by measuring power per FFT point to make different data point implementation comparable. Performance is measured by the processing time per FFT point. Table Table 5.1 shows the proposed design can reduce the power consumption and size. However, this is largely at the expense of performance as intended.

Table 5.1: Comparison among different FFT approaches

Work	CMOS Tech (μm)	Power Supply (V)	FFT Point	Clock Freq (MHz)	Power Per Point (mW/point)	Norm Area (mm^2)	Perf per point (μs/point)
This	0.18	1.8	64	50	0.22	11.1	1.8
[44]	0.25	1.8	64	20	0.64	27.2	0.050
[50]	0.13	1.2	64	20	0.35	24.5	0.050
[51]	0.6	3.3	64	36	15.625	43.3	0.056
[52]	0.35	3.3	64	65	8.51	13.8	0.05
[53]	0.5	3.3	1024	66	5.86	167	0.59×10^{-3}
[43]	0.75	/	1024	40	7.52	1104	9.3×10^{-3}

5.1.4 Layout Implementation

Since the overall design costs around half million transistors, which largely increase the routing effort, it was converted into Verilog files and then imported into Cadence Encounter for auto-routing. In order to minimize the overall design area, the utilization effort was set to be 95 %. The design was implemented based on a commercial standard cell library that's built upon 180 nm CMOS technology. The overall layout is shown in Figure 5.2 , which is measured at $1.44mm^2$ and is adequate for manufacturing.

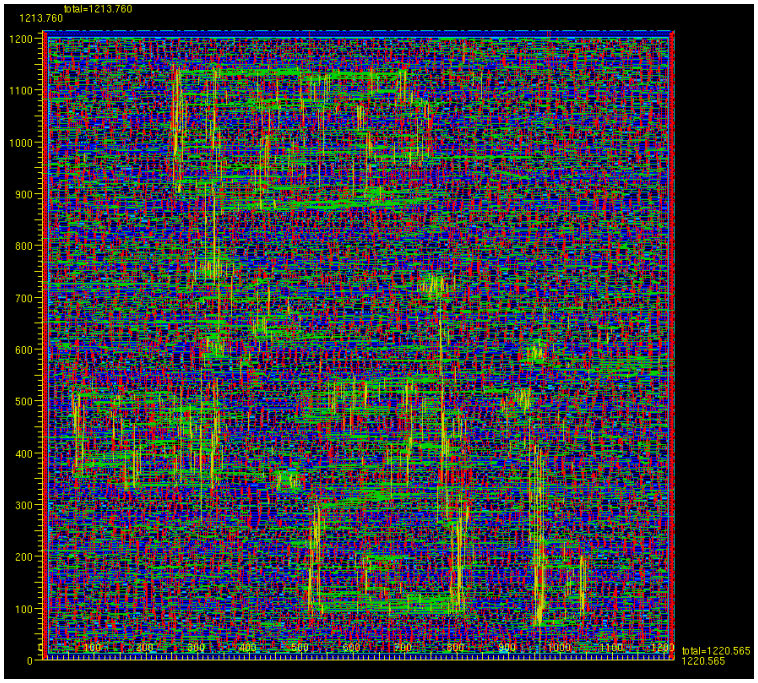


Figure 5.2: Layout of Proposed FFT Design.

5.2 System Validation: Electrochemical Analyzer, Generation 1

5.2.1 Final PCB Implementations and Measurement Results

The input stimulus generation and output response acquisition circuits were implemented on a PCB using off-the-shelf components. Figure 5.3 shows the PCB realization of the proposed design.An equivalent RC test bench was built to mimic the electrode-electrolyte interface with a

Cdl=15.6 uF, Ret=100 Ohms and Rsol=100 Ohms as shown in Figure 4.1, where Cdl and Ret are the double layer capacitance and electron transfer resistance respectively at electrode-electrolyte interface, Rsol is the solution resistance.

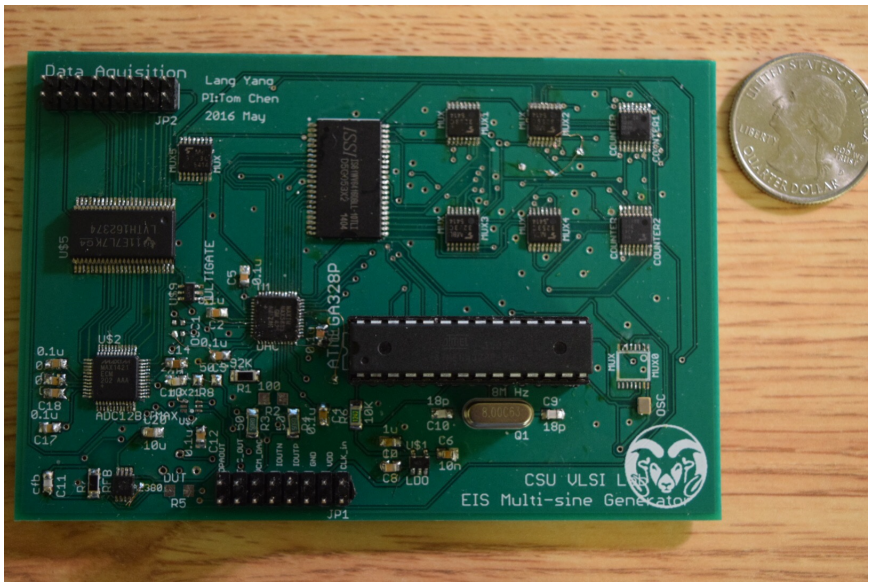


Figure 5.3: PCB layout for proposed design

Figure 5.4 show the output from signal generator at output of DAC. The average amplitude for each frequency component is about 10.02 mV with a 3 standard deviation of 85.9 uV. The signal spectrum is obtained using an ADLINK DAQ2208 data acquisition unit with an off-line FFT on a host computer. The output signal shows an excellent spectrum purity for the desired frequency range.

The response from the RC test bench under the input stimulus from 2Hz to 2KHz was acquired with the two measurement steps. Figure 5.5 shows the bode plot of the response signal from the RC test bench using the on-board acquisition circuit (blue line) and using HP4192A impedance analyzer. Figure 5.6 shows impedance measurement error for each frequency in the desired range. The averaged impedance magnitude error is in 0.8% while the phase error is 0.95 degree. The corresponding Nyquist result from this setup is presented in Figure 5.7 for both the results generated by the on-board circuits and that generated by HP4192A impedance analyzer.

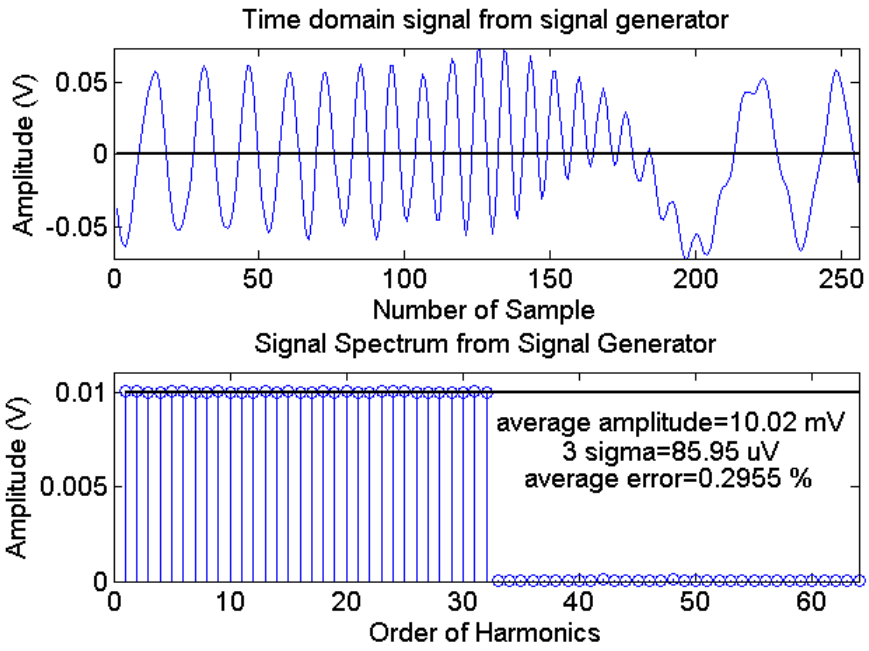


Figure 5.4: Time and frequency domain composite signal from the stimulus generator

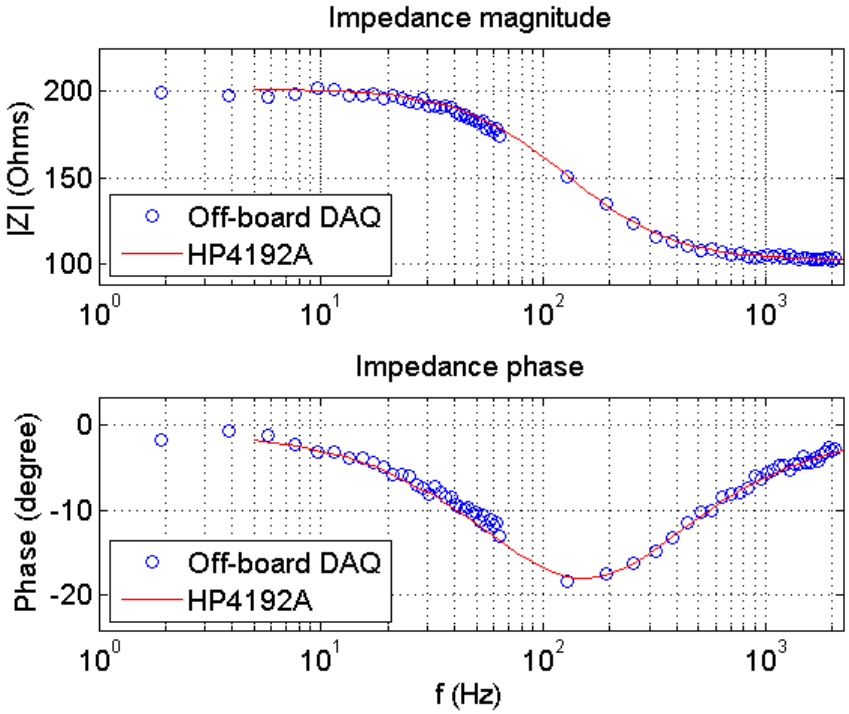


Figure 5.5: Measured Bode plot with off board DAQ

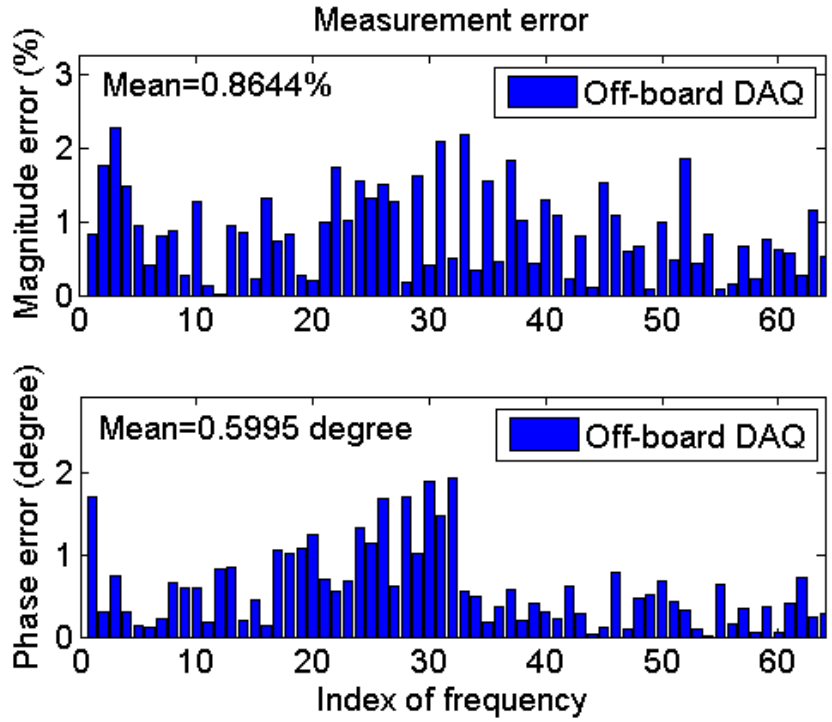


Figure 5.6: Measurement error with off board DAQ

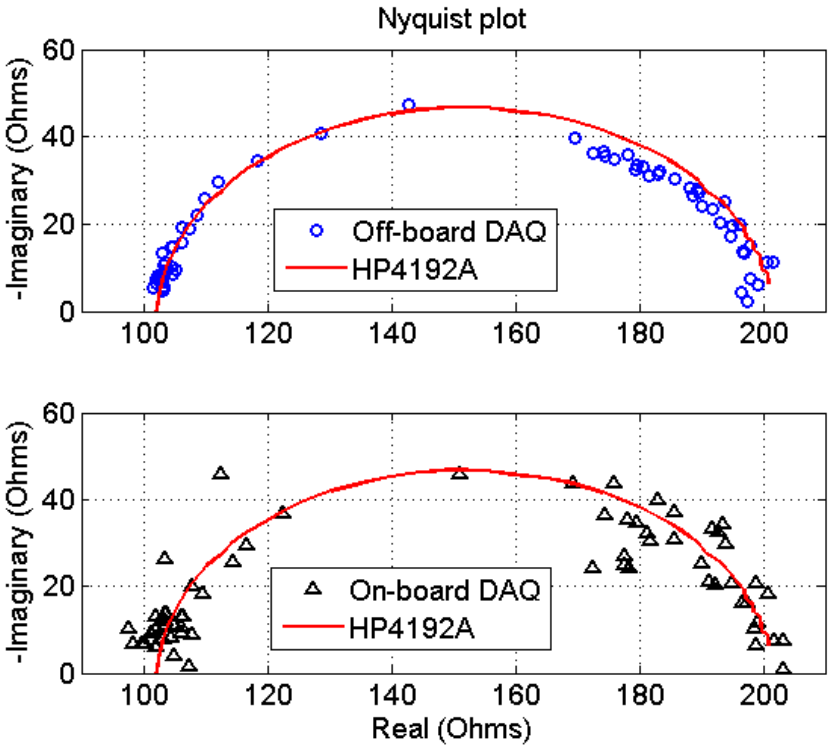


Figure 5.7: Measured Nyquist plot with off board DAQ

Figure 5.8 illustrates the noise spectrum density at output of TIA measured using the ADLINK data acquisition board. The integrated input inferred noise of TIA is 1.58 uA , the limit of detection is when the input signal is equal to the noise. Eq. (5.3) and (5.4) shows the relationships between the overall sensor impedance and the components in the equivalent circuit in Figure 4.1. If Vrms = 7.14 mV with 10mV amplitude, I = 1.58 uA, Ret = 100 Ohms, Rsol = 100 Ohms, Cdl = 15.6 uF, the detection limit is 9.2 Ohms.

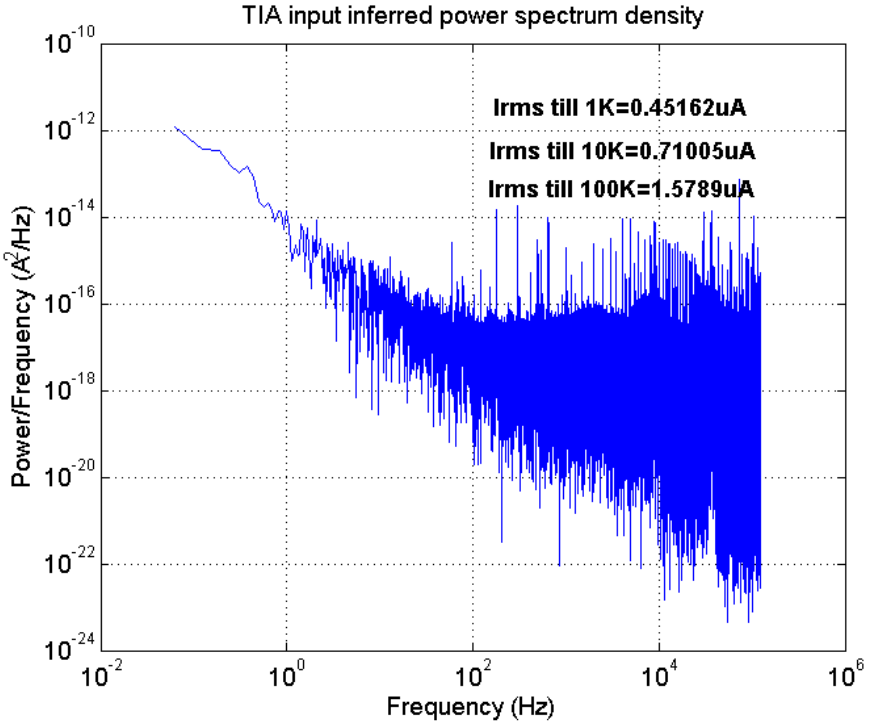


Figure 5.8: Input inferred noise spectrum density from TIA

$$\delta Z_{DUT} = \frac{\delta I \times |Z(\omega)|^2}{V_{RMS} - \delta I \times |Z(\omega)|} \tag{5.3}$$

$$Z(\omega) = R_s + \frac{R_{et}}{1 + \omega^2 R_{et}^2 C_{dl}^2} - \frac{j\omega R_{et}^2 C_{dl}}{1 + \omega^2 R_{et}^2 C_{dl}^2} \tag{5.4}$$

Test with on-board analog digital converter is also included in this session to be compared with the performance from commercial data acquisition board. As results shown in Figure 5.9 and

Figure 5.10 the averaged error for impedance magnitude is 2.14 % and is 1.58 degree for phase. This is because on board ADC is a much simpler and cheaper implementation than a commercial data acquisition board. In order to save a ADC from design, signal from input and output of TIA was multiplexed which introduced more variation from sampling error. Figure 5.11 shows the resulting Nyquist plot, which looks more zigzagging. Nevertheless, this approach reaches a lower cost and higher integration level as a point of care device.

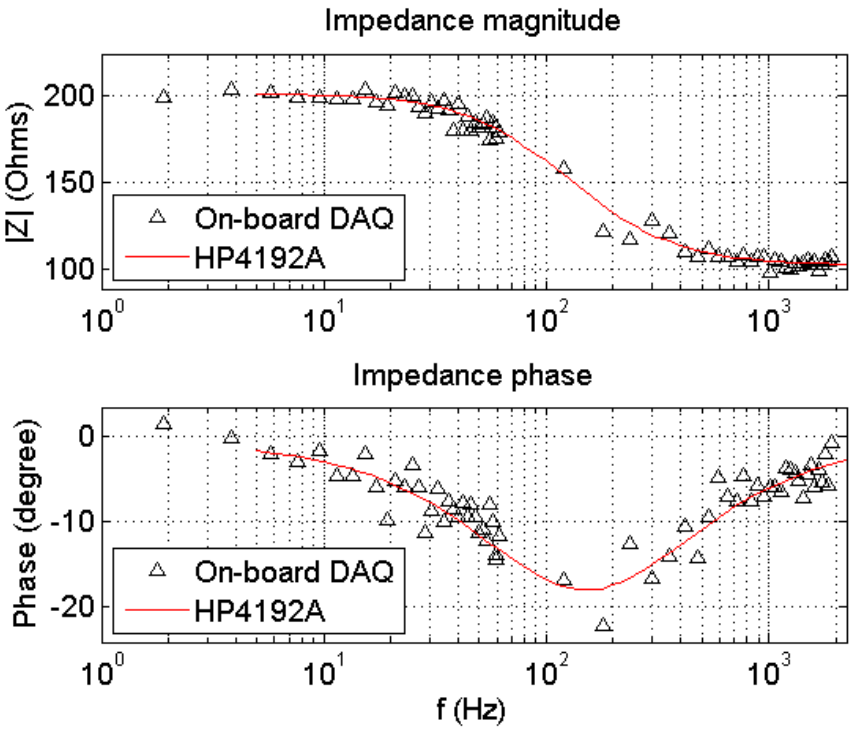


Figure 5.9: Measured Bode plot with on board DAQ

5.2.2 Improvement with Double Sampling Technique

Correlated double sampling (CDS) is a technique frequently used in switched-capacitor circuit to reduce the common mode noise, by sampling the signal twice at a known and unknown condition respectively. Borrowing the concept from such technique, the multi-tone signal was modified as shown in Figure 5.13. In this case, the sampling rate will be doubled in order to sample both the

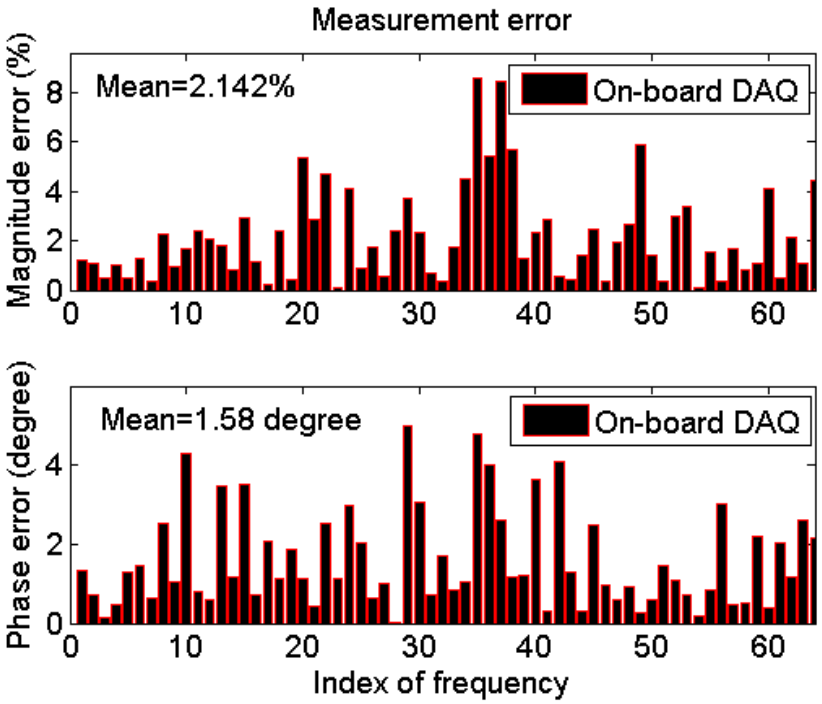


Figure 5.10: Measurement error with on board DAQ

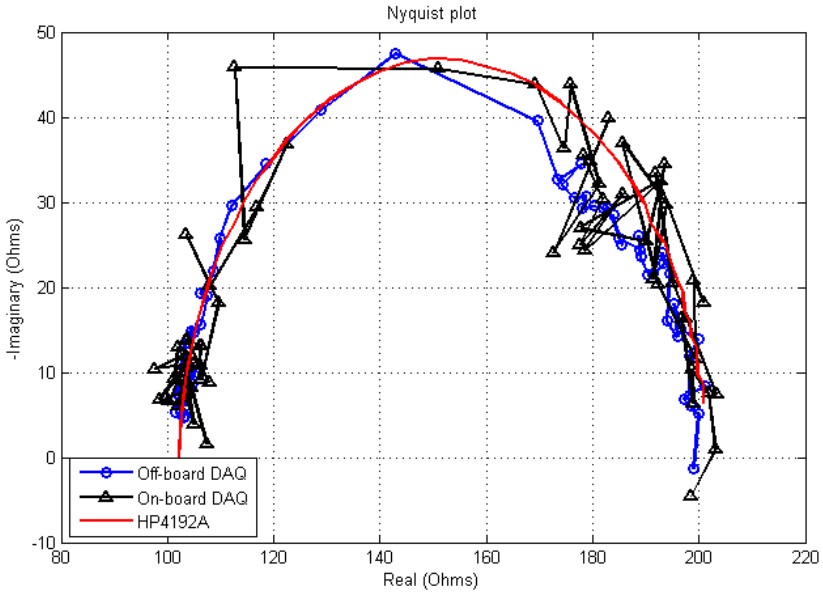


Figure 5.11: Measured Nyquist plot with on board DAQ

unknown signal and the common mode. The common mode error then gets subtracted from it's own signal period to calibrate the error cause by common mode noise or shift.

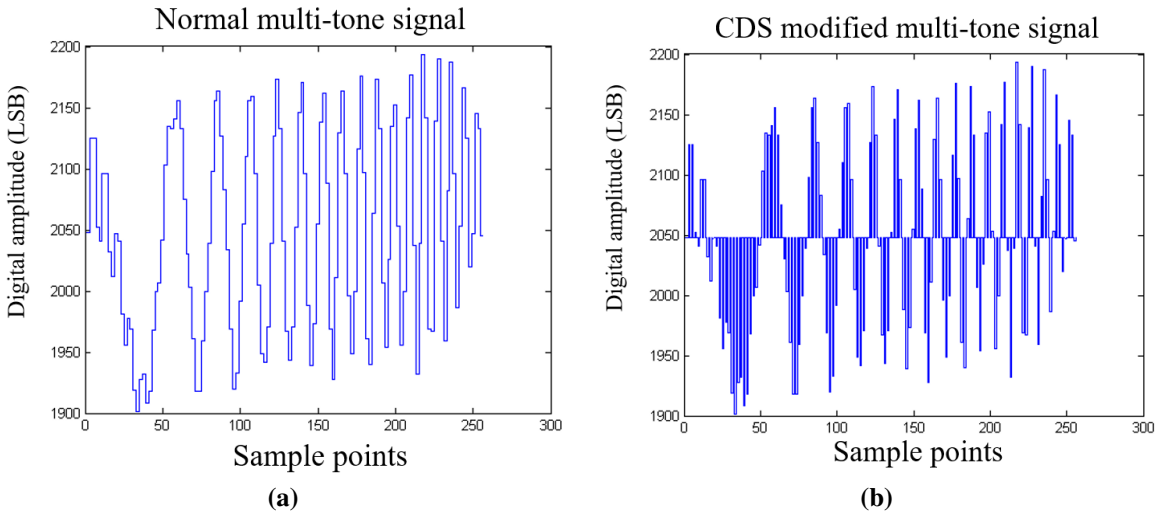


Figure 5.12: (a) Original multi-tone signal. (b) CDS modified multi-tone signal

Without applying smoothing algorithms, the averaged error with and without CDS technique are very similar, and the improvements were nearly negligible. As shown in Figure 5.13, the measurements with CDS modified signal reached an average error of 2.628 % in impedance and 2.1 degree in phase, while the error without CDS technique were 2.363 % in impedance and 1.957 degree in phase. However, when the smoothing algorithms were applied, measurements were improved in general, while signal modified by CDS technique had a much significant improvement. The CDS modified signal with smoothing algorithms (with a smoothing window size at 70 points) had a 1.269% impedance error and 0.4587 degree phase error. The error without CDS technique after smoothing were 1.91% and 1.084 degree in phase. These data have shown that by smoothing and canceling common mode noise can improve and reduce the measurement error.

5.3 System Validation: Electrochemical Analyzer, Generation 2

Figure 5.14 shows the major components of the proposed platform and the physical implementations. A $R_{et}|C_{dl} - R_s$ model was built with standard resistor and capacitor to mimic electrode-

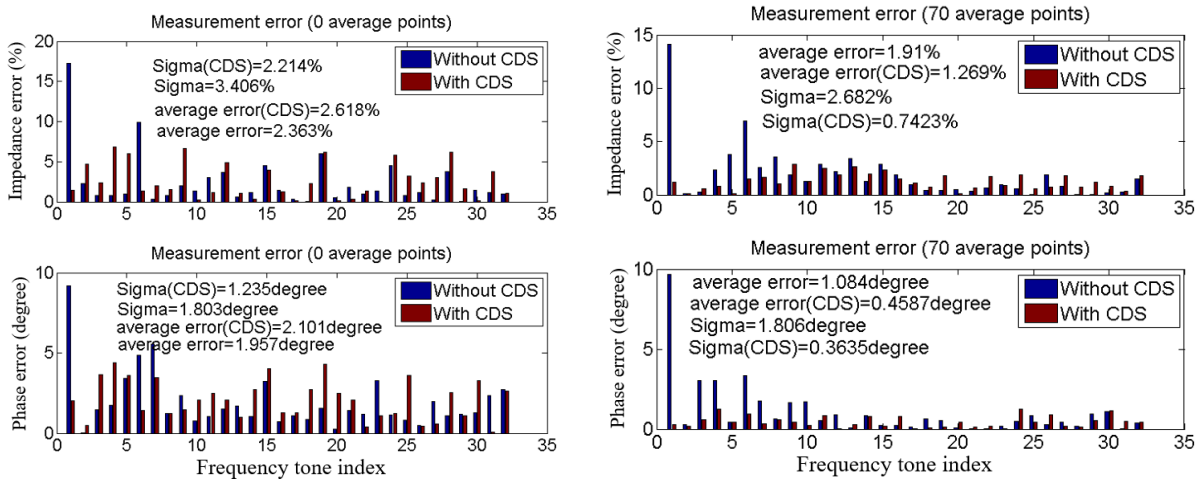


Figure 5.13: Measurement error with/without CDS (a)without smoothing algorithms. (b)with smoothing algorithms

electrolyte interface. The C_{dl}=2 uF, R_{et}=2088 Ohms and R_s=50.7 Ohms. The Cdl, Ret and Rs represent double layer capacitor, electron transfer resistor and solution resistor respectively.

5.3.1 Single-tone Response

The system was first tested under the single-tone mode. In this mode, signals were sent with one frequency component at a time. The frequency sweeping range was from 0.16 Hz to 15.1 kHz with 96 frequency points in between. The results were compared with the measurements from Zive SP1 EIS benchtop (Seoul, Korea) as well as the ideal value derived from the circuit transfer function shown in Eq. (5.5) and (5.6).

$$Real = R_s + \frac{R_{et}}{1 + (2\pi f)^2 R_{et}^2 C_{dl}^2} \tag{5.5}$$

$$Imag = -\frac{2\pi f C_{dl} R_{et}^2}{1 + (2\pi f)^2 R_{et}^2 C_{dl}^2} \tag{5.6}$$

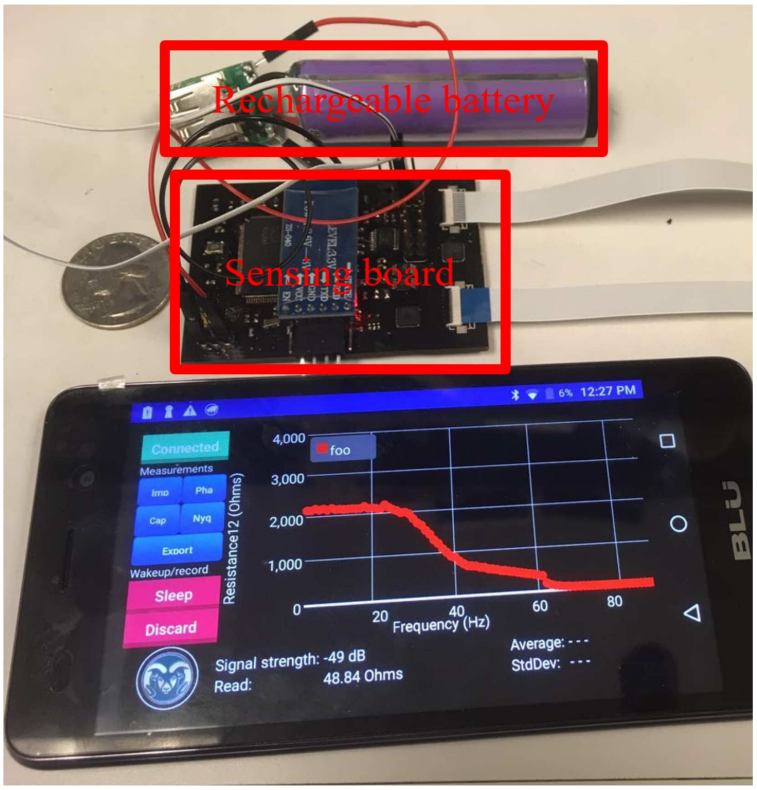


Figure 5.14: Proposed EIS Platform Components.

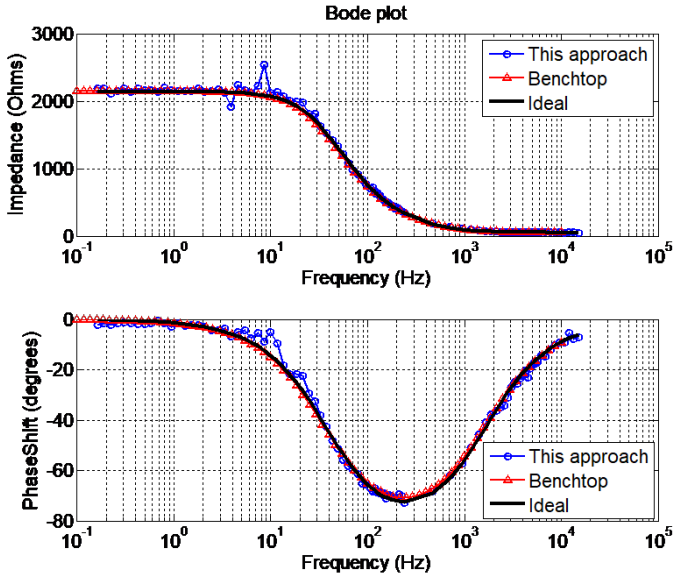


Figure 5.15: Bode Plot for the Single-tone Measurement.

Figure 5.15 shows the measured results in Bode plot. The average error of the proposed system compared to the ideal expected data is 2.6% for impedance and 1.4 degree for phase, while the benchtop counterpart is at 1.6% average impedance error and 0.67 degree average phase error.

5.3.2 Multi-tone Response

A 32-tone combined signal was designed and generated from DAC. The signal was fed to the same RC model, and ran for 3 times in low, medium and high band to form 96 frequency points. The frequency range in this case is from 0.16 Hz to 8 kHz. Figure 5.16 shows the measured results in Bode plot for the multi-tone operation. The average error at the multi-tone mode was 9.2% in impedance and 6.4 degree in phase. The increased errors in the multi-tone mode can be attributed to the additional errors generated by the multi-tone stimulus signal in hardware.

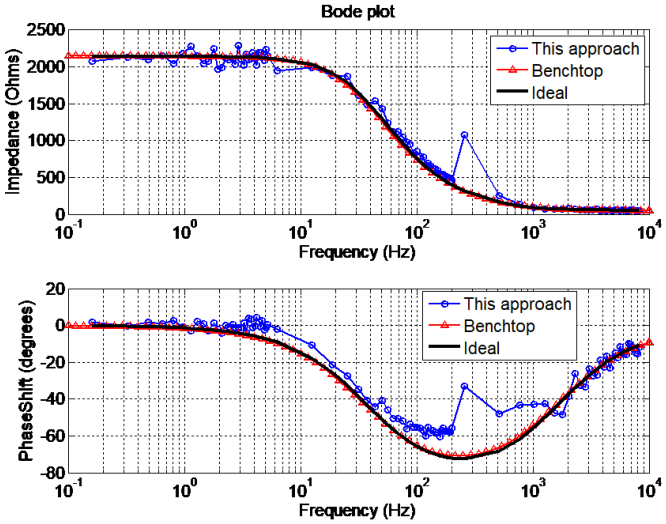


Figure 5.16: Bode Plot for the Multi-tone Measurement.

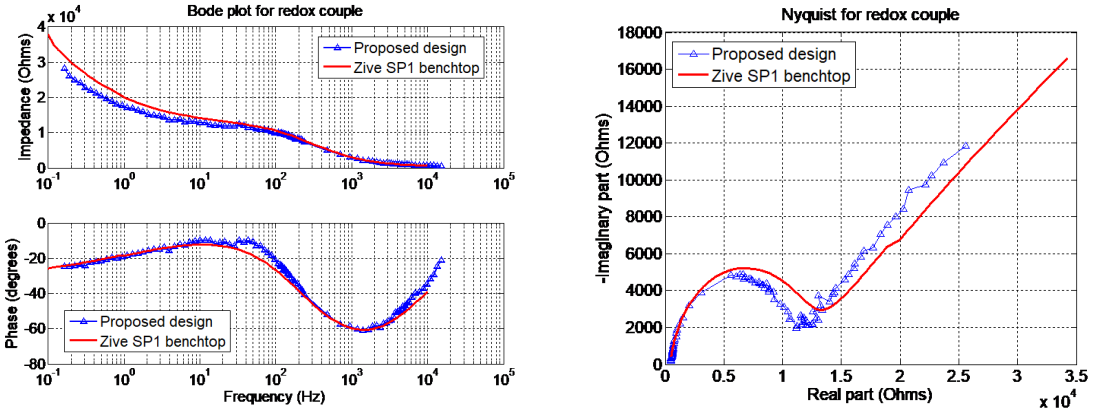
5.3.3 Simulation Result with Redox Couple

In electrochemistry, oxidation and reduction reaction happen at electrode/electrolyte interface. Potassium ferricyanide ($K_3[Fe(CN)_6]$) and ferrocyanide ($K_4[Fe(CN)_6]$)are commonly used as redox couple to provide electron transfer. The chemical reaction is as in Eq. (5.7) and (5.8). The

redox measurement was performed under the single-tone mode. 5 mM potassium ferricyanide and 5 mM potassium ferrocyanide were mixed with 100 mM KCl solution for testing. Figure 5.17 shows the redox measurement results in Bode plot and Nyquist plot. The proposed system performs similarly compared to Zive SP1 benchtop. Because the Redox couple are responsive to the bias voltage across the electrode, the offset voltage difference between instrument could introduce some variation.

$$Reduction : [Fe(CN)_6]^{3-} + e \rightarrow [Fe(CN)_6]^{4-} \tag{5.7}$$

$$Oxidation : [Fe(CN)_6]^{4-} - e \rightarrow [Fe(CN)_6]^{3-} \tag{5.8}$$



(a) (b)

Figure 5.17: (a). Bode plot for the Redox Couple Measurement. (b) Nyquist Plot for the Redox Measurement.

5.4 System Validation: Electrochemical Analyzer, Generation 3

5.4.1 Performance Analysis

The noise performance of the 3rd gen analyzer was analyzed by integrating the power spectrum density among different modules as shown in Figure 5.18.

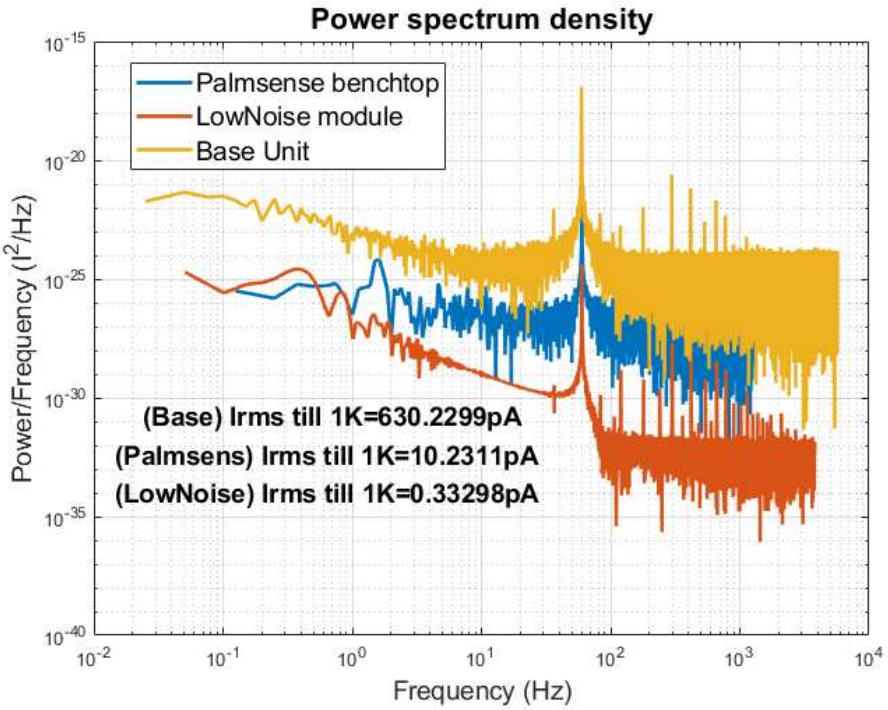


Figure 5.18: Power spectrum density in current for different modules.

The noise spectrum was integrated up to 1 kHz to obtain the root-mean-square (rms) input-inferred current noise. The low- noise module reached input inferred noise in 333 fA. Detection as low as picoamp current can , therefore, be achieved. The base unit has the input-inferred current noise of 630 pA. This noise performance for the base unit is sufficient for applications with input current above the nanoampere range. The results in Figure 6 also compare the noise performance of the 3rd gen Crexens™ analyzer with that of Palmsens 4 analyzer (Houten, Netherland) which had a 10 pA noise.

The measurement accuracy of the Crexens™ analyzer was examined using the EIS mode as it is more indicative of the overall accuracy than that from the other modes due to its high performance nature. RC calibration circuits were built and used for validation.

Figure 5.19 shows the RC model and the measured result compared to its theoretical Bode plot for the base unit. The results were also compared to these obtained using the Palmsens 4 unit (Houten, Netherlands). The average impedance magnitude error from the 3rd gen Crexens™ base unit was 0.55% while the aver- age magnitude error from the Palmsens 4 unit was 1.28%. The

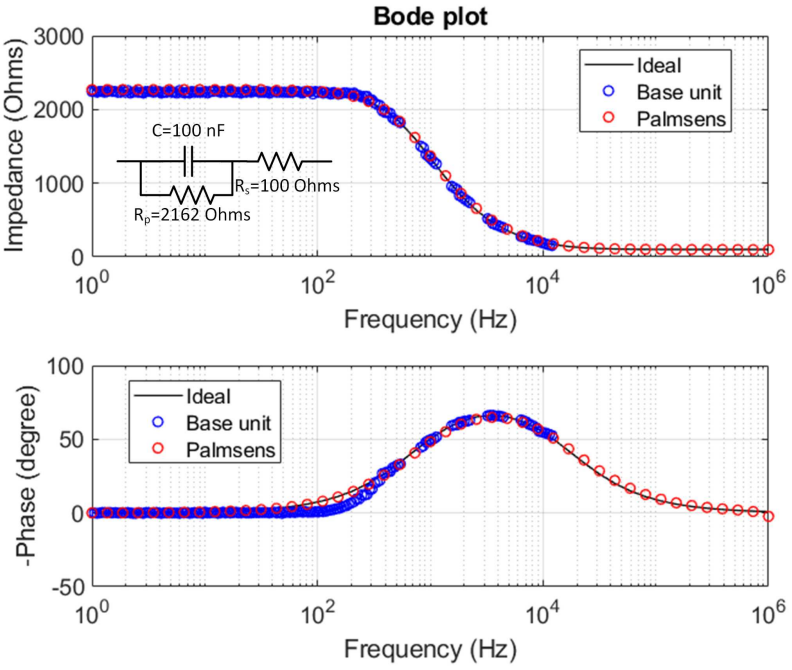


Figure 5.19: Base unit EIS measurement result in Bode plot.

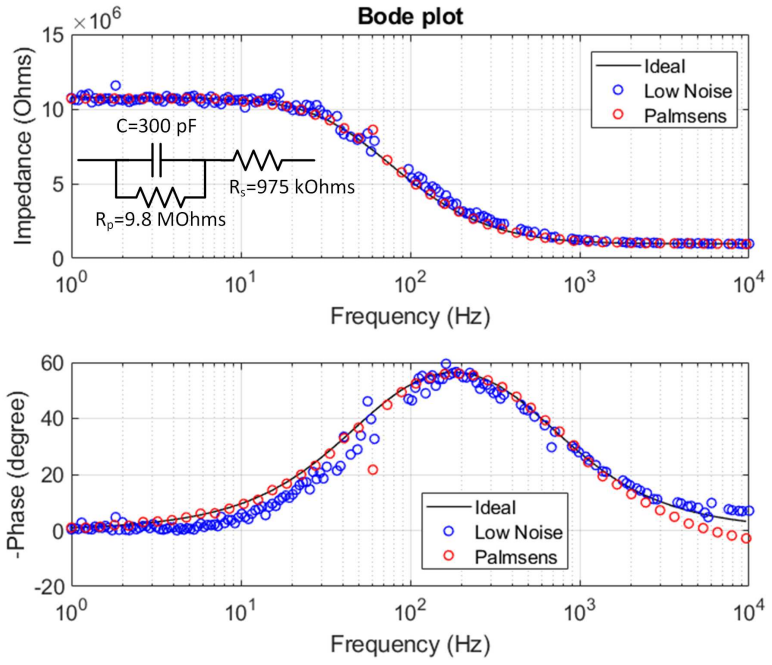


Figure 5.20: Low noise module EIS measurement result in Bode plot.

average phase error from the 3rd gen Crexens™ base unit was 1.89 degrees while the average phase error from the Palmsens 4 unit was 0.1 degree. For the low-noise module, a different RC calibration circuit was chosen to provide a relative high impedance to suitable for testing the scenario of low input current. Figure 5.20 shows the RC model and the measurement results for the low-noise module. The average impedance magnitude error from 3rd gen Crexens™ was 4.4% while average magnitude error from the Palmsens 4 unit was 1.2%. The average phase error from 3rd gen Crexens™ for the low- noise module was 2.16 degrees while the average phase error from the Palmsens 4 unit was 1.47 degrees. The measurement errors in impedance and phase were expected to be slightly higher for the Crexens™ analyzer due to its target of significantly low cost. However, the overall accuracy is comparable to that of Palmsens analyzer. Nonetheless, the use of a better adaptive filtering scheme that shift cut-off frequency for filtering according to EIS frequency can improve 3rd gen Crexens™ accuracy further.

Figure 5.21 shows the implementations of all the modules used in Crexens™. The add-on modules are inserted face to face to the base unit by the 80 pin connector strip. In Table 5.2, the full performance specification for the base unit and low-noise module are listed. The performance specification for the quad- module is not listed in the table as it has the same specs as base unit. The base unit has sufficient performance that can cover a wide range of electrochemical applications, and the add-on modules are able to further extend the base unit's capabilities in sensitivity and speed to cover applications where the performance of the base unit is unable to reach. The power consumption for the proposed platform is 0.62 W in the worst case which allows it to operate for more than 10 hours when the device is driven by 2000 mAh Lithium-ion battery. The goal of the implementation of the base unit and the modules was to pack the design into a PCB area less than 70 cm^2.Adding the plug-in modules shifts the area constrain to the thickness of the device instead of further increasing the overall area. With low power, small area and good performance, the proposed Crexens™ electrochemical analyzer is capable of providing general-purpose electrochemical analysis with comparable performance as the existing commercially available units, but with desired flexibility and lower cost. It is, therefore, more suitable for point-of-care use.

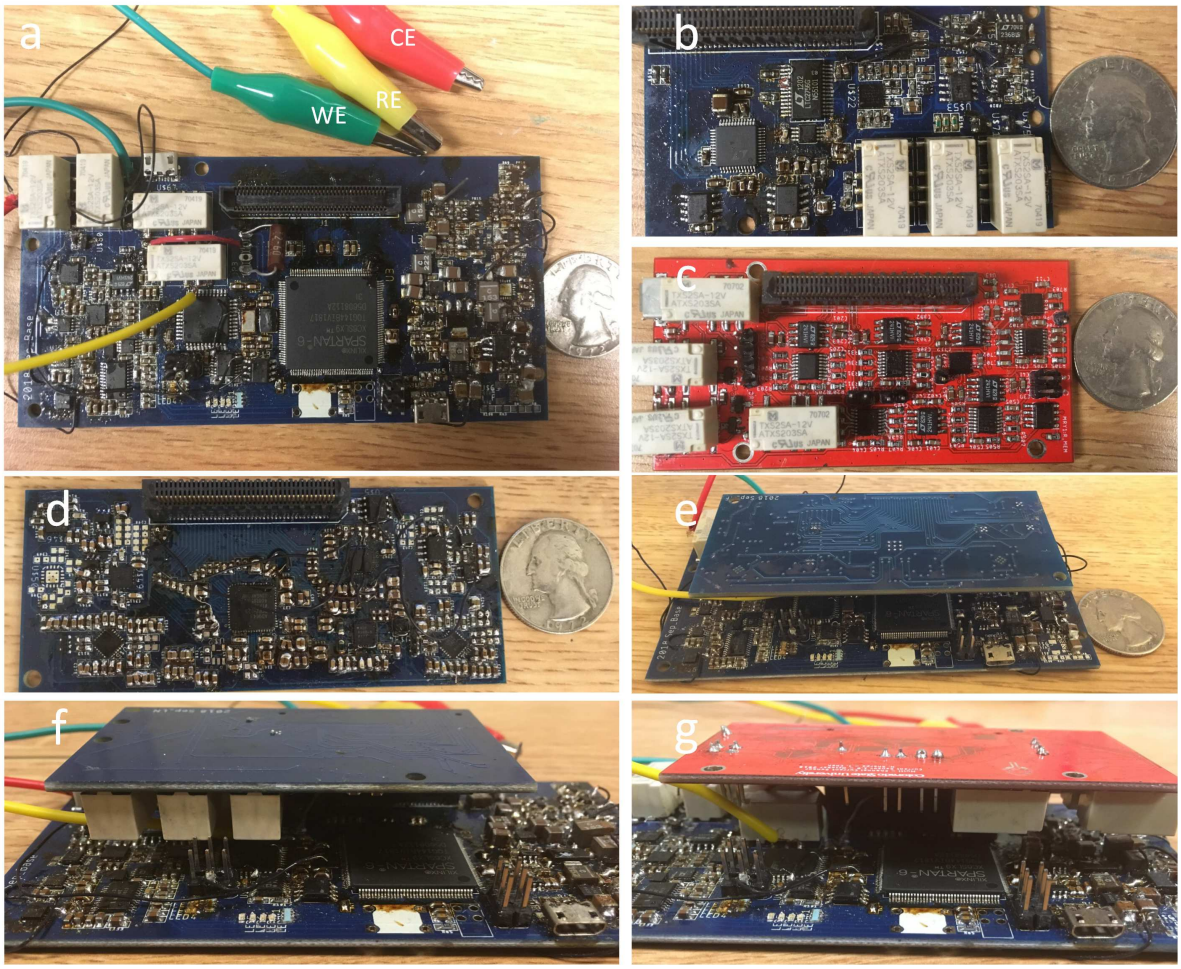


Figure 5.21: PCB implementation for proposed platform. a) based unit, b) low noise add-on module, c) quad-channel add-on module,d) high performance add-on module, e) base module with high performance add-on module, f) base module with low-noise add-on, g) base module with quad-channel add-on.

Table 5.2: SPEC for the proposed reconfigurable electrochemical analyzer in details.

Module	V_{out}	V_{LSB}	V_{comp}	I_{limit}	f_{max} (Hz)	CV rate	Power	Area
Base	±3 V	1.46 mV	10 V	630 pA	0.2-12 k	1900 V/s	0.62 W	67 cm^2
Low noise	±5 V	38 uV	24 V	333 fA	0.1-10 k	1700 V/s	1.25 W	29 cm^2
High perf	±3 V	366 uV	10 V	/	0.1-9.375 M	>10 kV/s	2.3 W	37 cm^2

5.4.2 Discussion for High Performance Module

The high performance add-on module turned out to be problematic for measurements due to unexpected dominant noise while being operated. The unexpected noise occurred when the ADC was sampling at 150 MHz and it caused significant switching noise on all associated power supply rails in the system. The analog read channels were, therefore, affected and caused sampling error that resulted in inaccurate measurement. After detailed diagnosis, it was determined that this could be caused by the poor routing scheme on a 4 layer PCB, as shown in Figure 5.22, plates for different power supplies were largely splitted by routes and increases the ground loop due to limited space [54]. Furthermore, by failing to do common mode shielding, the high speed traces could cause coupling among them (highlighted in Figure 5.22) resulting in digital glitches [55].

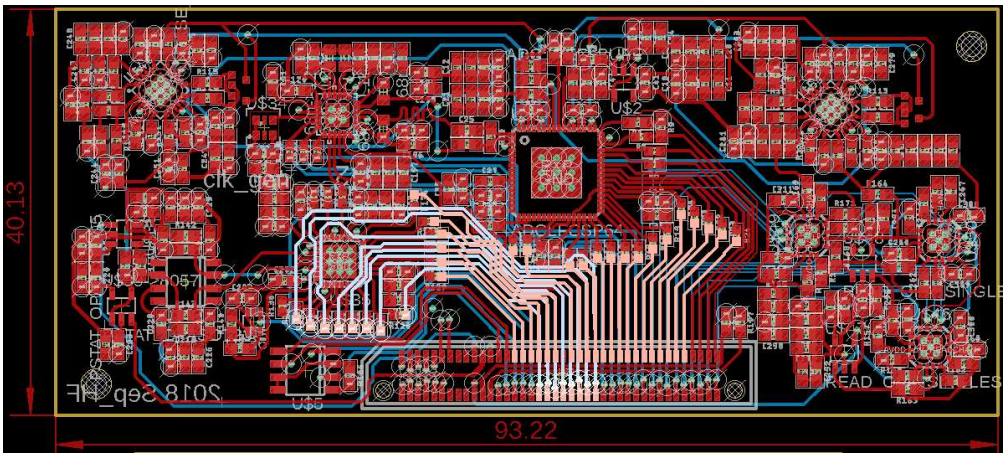


Figure 5.22: 3rd gen Crexens™ high performance module 4 layer PCB layout. The highlight traces are the high speed traces running up to 150 MHz, they were not well shielded from one to another

The PCB layout was redesigned on a 6-layer PCB, as shown in Figure 5.23, with shielding for every high speed digital trace. The high speed traces are also buried between two power plate as much as we can to reduce Electromagnetic Interference (EMI). By taking advantage of more routing space, the power and ground plate can be more complete with bigger area to reduce the return current path.

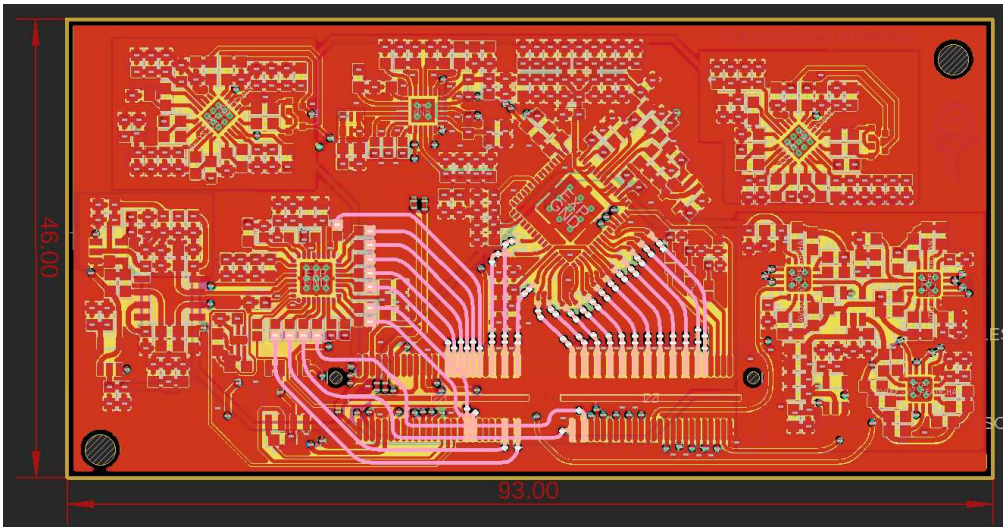


Figure 5.23: 3rd gen Crexens™ high performance module 6 layer PCB layout. The highlighted high speed traces were shielded by large ground plate and buried as much as they can be on the middle layer.

Chapter 6

Experimental Results

In this chapter, several electrochemical experiments will be introduced and discussed as proof of concepts. First of all, a micro-electrode array based neurotransmitter sensor will be presented. Electron transfer events were measured by home-made transimpedance amplifiers (TIA) at different gain to observe the existence of electrochemical active analyte, which in this case is norepinephrine. A capacitive DNA sensor will be discovered to monitor complementary hybridization event. And another ZIKV sensor measurement by electrochemical impedance spectroscopy (EIS) technique will be talked. At the end, a glutamine enzymatic sensor will be introduced which was validated by the proposed electrochemical analyzer.

6.1 A Neurotransmitter Sensor with Micro-array Electrodes

6.1.1 Motivation

As biologists have grown interests in monitoring biologic events, different approaches have been developed in order to provide a real time feedback. Among various techniques, optical and fluorescence microscopes have been widely adopted to observe biologic behavior in molecular level [56]. For some target molecules in a living tissue that's interested, further modifications won't be needed since they have endogenous fluorophores, such as nicotinamide adenine dinucleotide (NADH) and hemoglobin [57] [58]. This can maintain the integrity and sustainability of tissue cells. On the other hand, some other molecules don't have such fluorophores by nature can be conjugated with fluorescent proteins to keep track of those. However, the extra probe required typically add undesired activities to the molecule behavior [59] [60] [61]. While microscopy methods have been accessible and popular in monitoring biological events, electrochemical based sensing techniques stands out as alternatives to track certain molecules due to its label-free and non-optical features. These inherent features allow to reduce the cost and simplify the monitoring process

without introducing expensive optical objectives and well controlled environment. In this work, norepinephrine was chosen to be studied as a model neurotransmitter to mimic the neurotransmitter released by $exvivo$ tissue slice for imaging purpose when the tissue are stimulated.

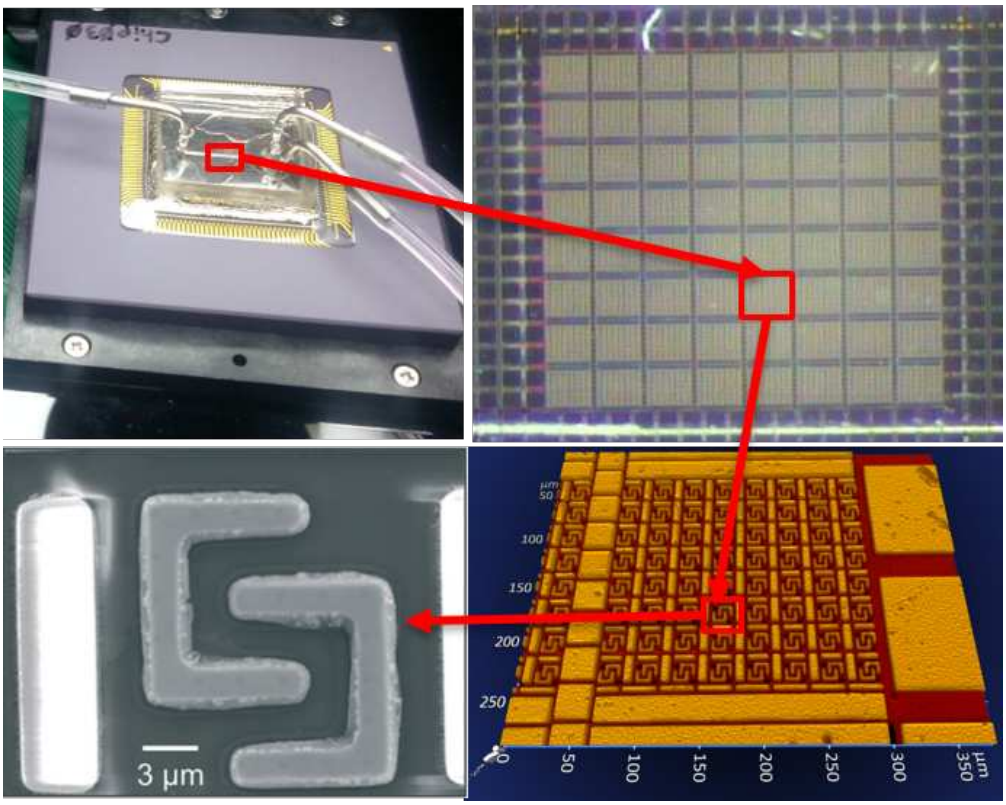


Figure 6.1: Electrode array on CMOS chip [56].

6.1.2 Experiment Setup

The sensing electrodes were fabricated on 500 nm 4-layer metal CMOS chip by Avago (Fort Collins, Colorado). The electrodes were topped by Pt metal, and each chip has 8 by 8 electrode subarrays. Each individual electrode subarray has 64 pairs of working electrode in interdigitated shape [62], and a global Pt counter electrode and Pt pseudo-reference electrode, as shown in Figure 6.1. The CMOS chip has an internal on-chip potentiostat to maintain the potential. External trans-impedance amplifiers (TIA) were designed on PCB board and a data acquisition card (ADLINK DAQe-2200) was used to sample the analog signals for further processing. A two layer

polydimethylsiloxane (PDMS) well was made and attached onto the CMOS chip with electrode area exposed. The top layer was made with 10-parts base elastomer and 1-part curing agent, while to bottom layer that's attached to the CMOS chip was made with 30-parts base elastomer and 1-part curing agent in order to take advantage of the stickiness. The peripheral of the well were then sealed by resin to avoid possible leaking.

6.1.3 Norepinephrine Calibration Curve

Different concentrations of norepinephrine were diluted into neurobasal media for the measurement. The reference potential for the on-chip potentiostat was set to be at +0.6V (vs. Pt). Figure 6.2 shows the amperometric redox reaction for norepinephrine. Two different TIA gains have been applied in these experiment to confirm the results and linearity. The measurement of norepinephrine went from 100uM, 200uM, 400uM, 700uM to 1mM. In this case, 8 random channels among 128 working electrodes were picked to observe the signal trend. Four of selected channels are measured through TIA at 10 MOhms gain and the other four were measured at 22 MOhms. Each channel has 4 data points as replicates. As it illustrated in Figure 6.3, the channels with different gain can be separable after 320 mM concentration in the worst case. The slope among different channels with the same TIA gain are nearly the same expect some baseline drift. With the linear fitting, the calibration curve around 500 nA/M for Norepinephrine.

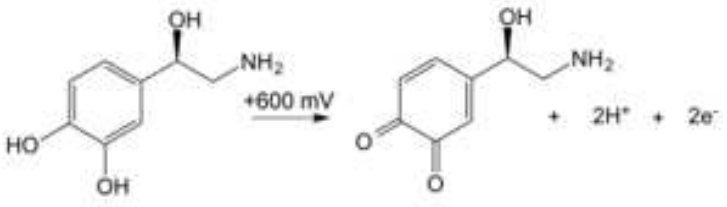


Figure 6.2: Norepinephrine redox reaction.

These experiments was preliminary for tissue imaging and set good foundations in terms of experimental protocol and signal processing for future works [63].

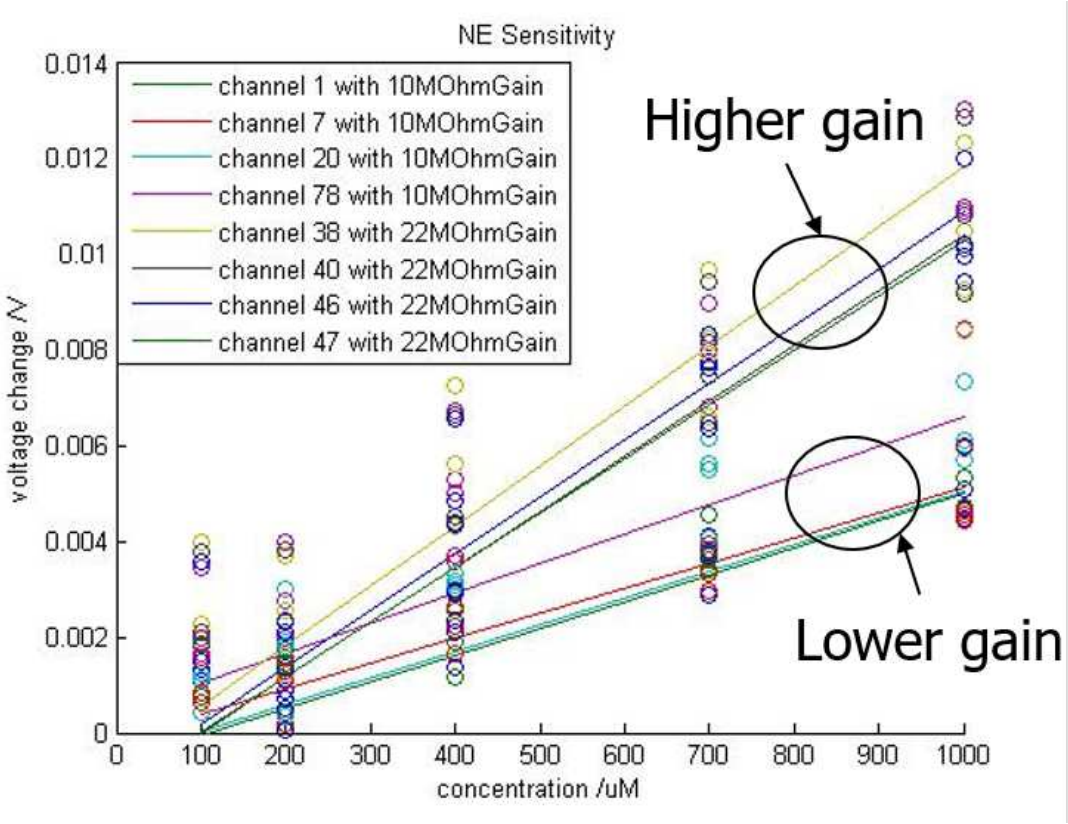


Figure 6.3: Norepinephrine calibration curve.

6.2 A DNA Capacitive Sensor Measured by Commercial Impedance Analyzer

In this section, a DNA capacitive sensor will be discussed. This work was done in collaboration with Lei Wang and Milena Veselinovic.

6.2.1 DNA Structure

As universal biological information storage entities, nucleic acids (DNA and RNA) are unique biorecognition molecules, and the detection of pathogen genomic DNA or RNA provides one of the most reliable methods for viral infectious disease diagnostics [64]. DNA also has relatively simple structure results in less complex to study. The basic unit is nucleotide which is made of a phosphate group, a Five-carbon sugar and Nitrogenous base, as shown in Figure 6.4. The phosphate group at back bone caries a negative charge. The nitrogenous base determines the sequence of a DNA strain as every nitrogenous base can be bonded with it's complementary nitrogenous base through hydrogen bond, such as Adenine (A)-Thymine (T),Cytosine (C)-Guanine (G). Therefore the specificity to detect a target single strain DNA can be achieved by designing its complementary DNA as bioreceptor probe.

6.2.2 Sensing DNA Protocol and Measurement Model

As capacitive sensor, the detection is based the change at electrode/electrolyte interface, which is known as double layer capacitor [65]. Upon binding to the electrode area, the biomolecule act as dielectric material that repels the free ions in the solution, and therefore causes a drop in capacitance.

In this proposed DNA sensor, the bare gold electrode surfaces are cleaned before use. To prepare the surfaces, the chips with the gold microelectrodes were immersed in a solution of 50 mM KOH and 25% H_2O_2 for 10 min [66], then followed by a through rinse with Milli-Q water. The surface is oxygen plasma treated afterward by Plasma Etch PE-25 (Plasma Etch, Carson City, NV, USA) for 5 mins. The 5' thiol-modified oligomers are used as single stranded DNA probe and cre-

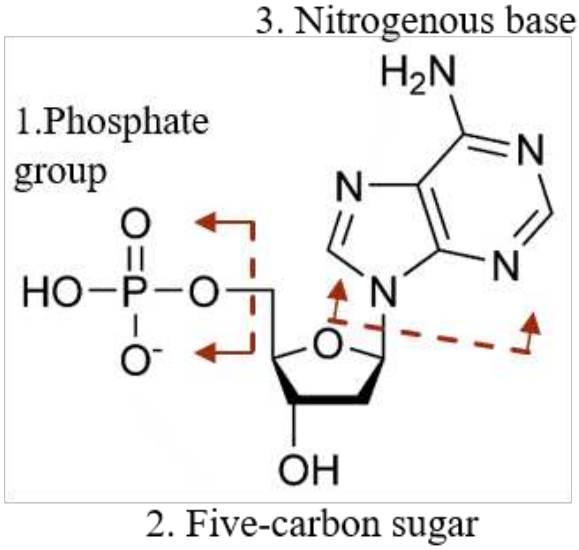


Figure 6.4: Nucleotide structure.

ate covalent bond on the gold-surface [67], the structure is shown in Eq. (6.1). The disulfide bonds on oligomers need to be reduced for a strong Au-S bond. The 10 uM ssDNA probe solution was prepared in 100 uM 1×TE-MgSO4 buffer, and the gold microelectrodes were immersed overnight in 30 uL of the solution.

$$5' - ThioMC6 - D - TAGTATGCACTGGTGTCTATCCCT - 3' \tag{6.1}$$

The 11-Mercapto-1-undecanol (MCU) is used as self-assembled monolayer to block the empty space to reduce any non-specificity for target detection. In this protocol, a 100% complementary ssDNA and a non-complementary ssDNA are used for specific detection and non-specific detection, the structures are shown in Eq. (6.2) and (6.3).

$$5' - AGGGATAGACACCAGTGCATACTA - 3',\ (100\%complementary) \tag{6.2}$$

$$5' - GCAATATAGATAACGCCAGATGGC - 3',\ (Noncomplementary) \tag{6.3}$$

6.2.3 Equivalent Circuit Model and Estimation

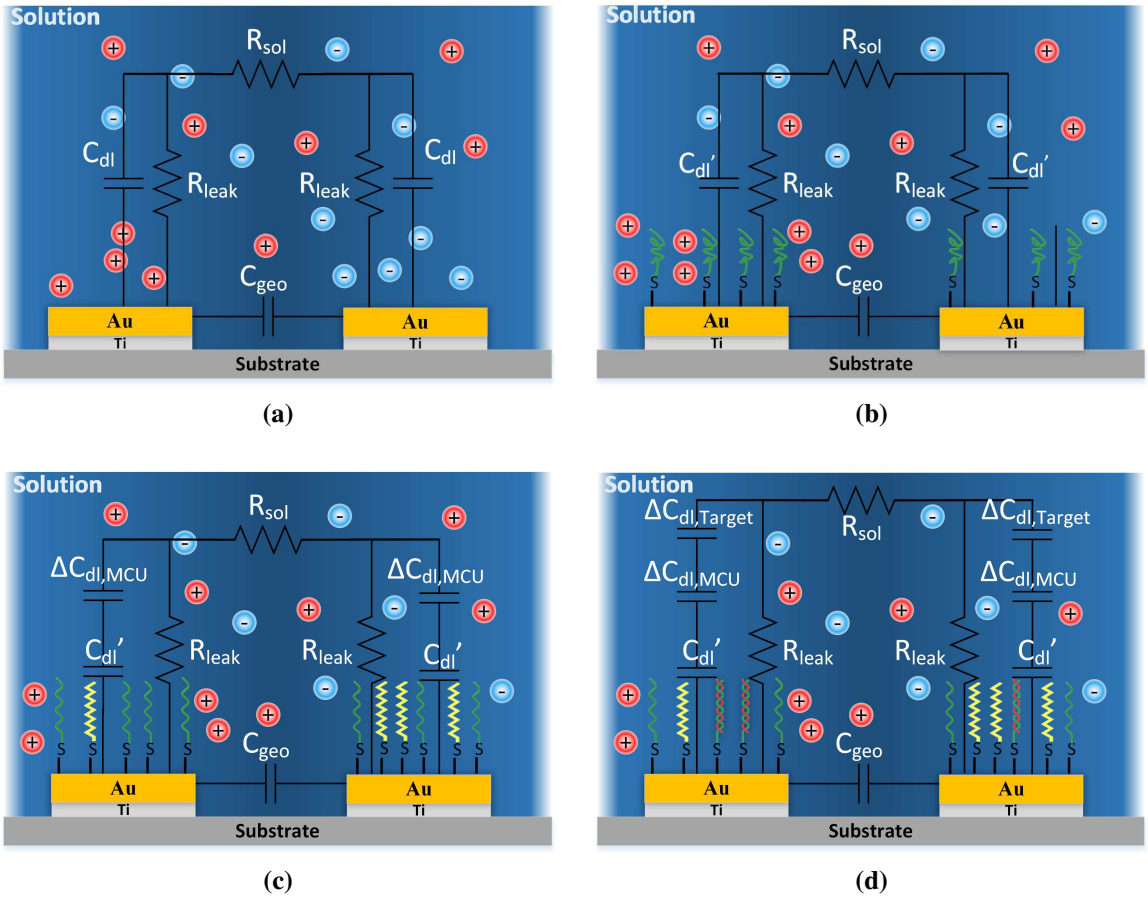


Figure 6.5: Equivalent Circuit Model (a). for Bare Electrode. (b) After Probe Immobilization. (c) After MCU Incubation . d) After Complementary Target Hybridization.

The Debye length in solution is calculated by Eq. (6.4) [68], where k is Boltzmann's constant, T is the absolute temperature, e is the proton charge, ϵ is the permittivity of the solvent, Z_i and n_i^B are the charge and bulk concentration of ion species i, respectively, and the sum extends over all ion species in solution. By assuming symmetric monovalent electrolyte and solvent is water, at room temperature a 100 uM $MgSO_4$ solution can have about 15 nm, the Debye length is $\propto \frac{1}{\sqrt{C}}$. Consider the area within the Debye length is pure dielectric layer with water, the double layer capacitance can be estimated from $C/A = \epsilon_o \epsilon_r \frac{1}{d}$, which is about $5uF/cm^2$. Given the electrode are is about $8.45mm^2$ and two double layer capacitor are in series, the theoretical bare electrode

capacitance should be around 200 nF. For each step of incubation, the equivalent circuit model are build as in Figure 6.5, which indicates an extra capacitor can be modeled in series for MCU and target DNA hybridization .

$$k_D^{-1} = (\frac{\epsilon kT}{e^2 \sum_i Z_i^2 n_j^B})^{\frac{1}{2}} \tag{6.4}$$

6.2.4 DNA Capacitive Sensor Measurement Setup

Capacitance measurement data were collected using the Instek LCR-821 benchtop LCR meter (New Taipei City, Taiwan), which interfaces with a PC for data acquisition. A graphical user interface (GUI) on the PC was used for sending command signals to the LCR meter. Since the measurement is obtained from non-faradaic current, a 0 V DC bias voltage was applied across the IDE sensor. A 20 mV root mean square (RMS) AC voltage with 20 Hz frequency was applied to the IDE sensors. All capacitance readouts were recorded under 20 uL of 100 uM 1×TE-MgSO4 buffer on the electrodes and 50 data points were collected per reading.

From this impedance analyzer, the original data is collected in real and imaginary as part of the complex form for impedance, as in Eq. (6.6). The equipment has two build-in model to analyze the data, which are series mode and parallel mode. In series mode, the sensor equivalent circuit is modeled as a resistor (R_s) in series with capacitor(C_s), Cs and R_s are calculated from Eq. (6.6) and (6.7). In parallel mode, the model is a resistor(R_p) in parallel with capacitor(C_p), the values derive from Eq. (6.8) and (6.9). Since the parallel mode is more resemble to the proposed model, the analysis of capacitance are based on this.

$$Z = Re + jIm \tag{6.5}$$

$$C_s = -\frac{1}{\omega Im} \tag{6.6}$$

$$R_s = Re \tag{6.7}$$

$$C_p = \frac{C_s}{1 + \frac{Re}{Im}} \tag{6.8}$$

$$R_p = Rs(1 + \frac{Im}{Re}) \tag{6.9}$$

6.2.5 DNA Capacitive Sensor Measurement Result

Specific and non-specific target DNA are tested from 20 to 2 Million molecule number for sensitivity. By taking the difference after and before target DNA incubation, the measurement result is shown in Figure 6.6. The complimentary targets showed a linear response in capacitance change with increasing target concentration, indicating an excellent correlation between low-range target concentrations and capacitance responses. While the non-specific target response is negligible compared to complementary counterpart. The fitting curve shows about -12 nF/log(number of molecule).

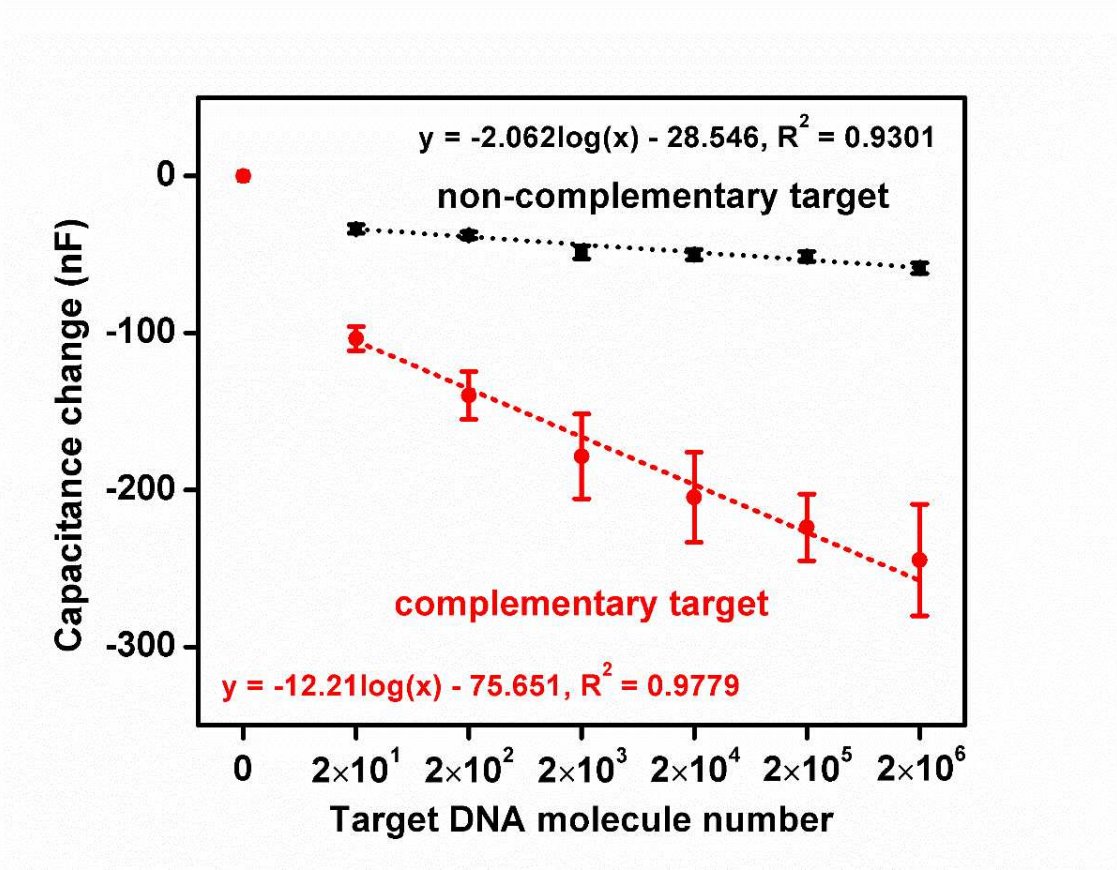


Figure 6.6: Capacitance Response from Complementary (red line) and Non-complementary (black line) DNA Targets [64].

6.3 An EIS Affinity Sensor for ZIKV Detection

6.3.1 Motivation

Although, the methods already exist for direct flavivirus detection, such as reverse transcription and polymerase chain reaction (RT-PCR) [69] [70], enzyme-linked immunosorbent assay (ELISA) [71], Reflective Phantom Interface technology [72], these techniques typically require complex process, well trained personals and specialized laboratory. As for point-of-care (POC) application which people need instant response and simplicity, elelctrochemical methods become an alternative to fit such need.

Researchers in the field have been studying affinity sensors for proteins and virus for a long period of time. Protein detections have been explored in folate receptor (FR) [29], Human prostatic acid phosphatase (PAP) [30], C-reactive protein (CRP) [28] [73], hepatitis B virus surface antigen(HBsAg) [74], Prostate-Specific Antigen (PSA) [75], etc, and good sensitivities had been reported. Although these researches have laid foundation for the proof of concepts towards protein sensing based on EIS method, a direct measurement for viral pathogen would be more interesting as a step further for point-of-care diagnosis.

In this study, done in collaboration with Jessie Filer, an EIS based affinity sensor was introduced for ZIKV detection. By taking advantage of multi-array electrode, 24 electrode pairs on a single chip, this project can achieve high throughput for replicates data. Similar researches in detection of flavivirus with EIS approaches had been done in [76] [77] [78] [79] [80], limit of detection from 0.12 Plaque-forming unit (PFU) to 167 PFU were reported. A limit of detection 22 focus forming units (FFU) in this work have been achieved.

6.3.2 Electrode Fabrication

The sensor electrode array were fabricated by photolithography process [81]. Figure 6.7 illustrates the process step by step. First of all, a 1 inch by 3 inches glass slide is cleaned by acetone then followed by isopropyl alcohol (IPA) and deionized (DI) water rinsing. Then, the glass was dried with nitrogen gas and left on hot plate at 135 Celsius to evaporate any residual moisture for

5 minutes. The cleaned glass was loaded onto spin-coater, and S-1813 photo-resist was applied onto the glass surface to fill and entire area. Spin-coater was set to be accelerated to 3000 rpm in 5 s, and stay 3000 rpm for 30 s. After the photo-resist coating was done, the coated glass was placed onto hotplate for soft-baking for 1 min at 135 Celsius. And the sample was transferred into a UV light chamber, and a photo-mask, designed by AutoCAD and printed by CAD/Art Services,Inc.(Bandon, OR), was aligned onto the coated glass slide. The glass slide was exposed under UV light for 6 seconds. Remove and store the photomask from the sample for next time use and place the sample into S-1813 developer for 1 minute. This was when a clear electrode pattern with photo-resist on should reveal, and the glass slide was then rinse thoroughly with DI water to stop the reaction. Inspection was needed to ensure there was no photo-resist residue on the path where the electrode will be, a disconnection or chip failure would be expected if the inspection was not careful. After all the photo-lithography processes were well finished, the sample was sent to a evaporator machine for metal deposition. In this process, 10 nm Chromium (Cr) first deposit onto the glass as adhesion layer between gold and glass, and then 150 nm gold (Au) was deposit on top of Cr. The metal coated chip was thoroughly rinsed by acetone to remove the photo-resist layer as well as the unwanted metal area and followed by IPA and DI water rinsing for cleaning. After dried by nitrogen gas, the chip was stored and ready for future use.

6.3.3 PDMS Fabrication

A PDMS well that's adaptive to the 24 electrodes array was made. The PDMS base and curing reagent were mixed in 10:1 ratio on a clean silicon wafer [82], and stacked to a height of 3 mm. Then the mixed PMDS in fluid form was cured in oven at 70 Celsius for 30 mins. The PDMS was peeled off from the silicon wafer and cut into the size that fits on the sensor. A 3.5 mm diameter biopsy punch (Technical Innovations, FL, Inc. USA) was used to punch through 24 holes to expose the electrode area and form the well. Finally, both electrode array and PDMS were plasma cleaned in an O2 Plasma Etch PE-25 (Plasma Etch, Carson City, NV, USA) at 200 mTorr pressure and application of 150 W to the RF coil for 5 minutes. PDMS can form covalent bond with glass after

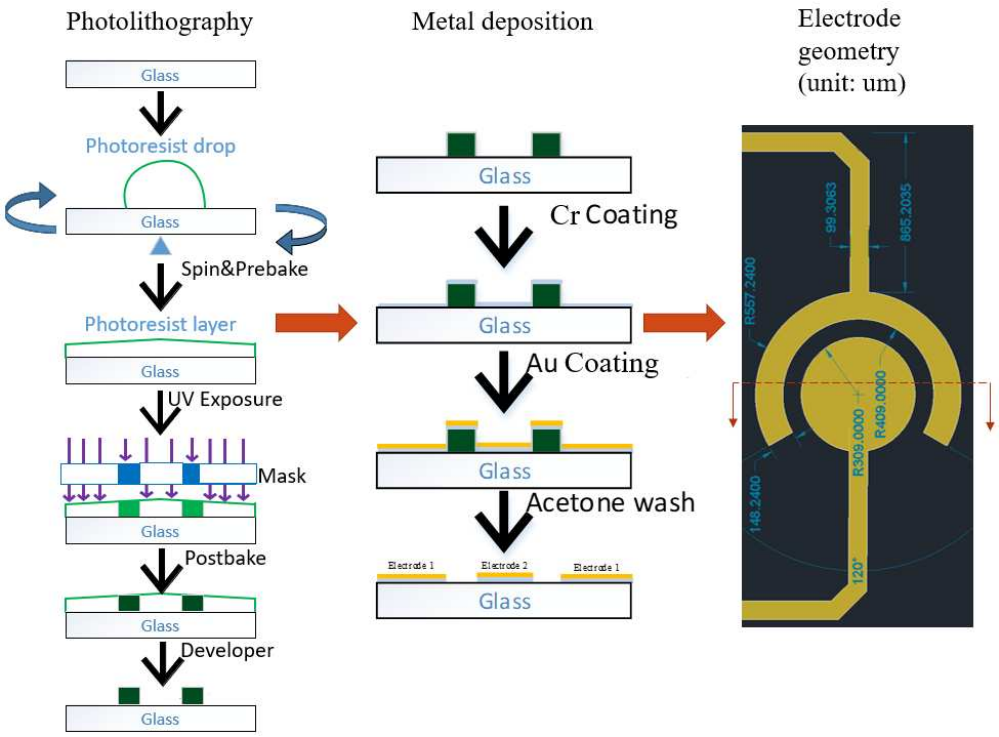


Figure 6.7: Gold electrode sensor fabrication process.

plasma treatment, the side attached to silicon wafer should be faced down to the glass substrate for better adhesion.

6.3.4 Surface Modification of Electrodes

The sensor needs to be cleaned at the beginning of any surface modification. The cleaning was done by apply 50mM NaOH/25% H_2O_2 mixed solution for around 45 mins [66]. This time was determined by keeping track of impedance change of the bare electrode, and stop till the impedance become steady, as shown in Figure 6.8. The array was then rinsed in nanopure water and dried with N_2 gas. The array was plasma cleaned for 5 minutes in an O_2 Plasma Etch PE-25 (Plasma Etch, Carson City, NV, USA) at 200 mTorr pressure and application of 150 W to the RF coil. Right after the O_2 plasma cleaning, the whole chip was dipped into 20 mL mixed solution with 9 mM MUA and 9 mM MPOH in reagent alcohol. This can create self-assembled monolayer (SAM), which on one side has thiol group that binds to gold, and on the other side has carboxylic acid to bind with protein [84].The array was then rinsed by reagent alcohol and immersed in 100 mM NHS/100

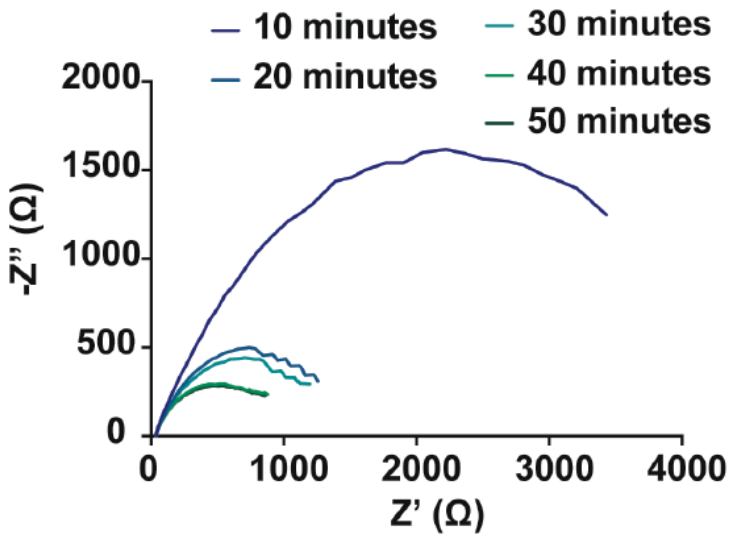


Figure 6.8: EIS measurement for bare electrode after as a function cleaning time [83].

mM EDC in 0.1 M MES (PH=5.0) for 30 minutes to activate the SAM surface and followed by rinsing with 20 mL 0.1 M MES (PH=5.0). 10 uL of 2.55 mg/mL 4G2 antibody was added to each electrode and incubated for 2 hours for bioconjugation. Each electrode was then rinsed twice with 10 uL 1 M ethanolamine in PBS and then incubated with 10 uL 1 M ethanolamine for 30 minutes, the ethanolamine was acting as blocking reagent to reduce non-specificity. The electrode were rinsed by PBS buffer, and incubated with 10 uL 2.5 mg/mL BSA in PBS, the BSA is acting as additional blocking reagent. Finally, the electrode were thoroughly rinsed five times with 10 uL PBS and incubated with 10 uL clarified ZIKV for 30 minutes [83]. The electrodes were rinsed five times again with 10 uL PBS and ready for measurement.

6.3.5 EIS Measurement and Result Discussion

EIS measurement was performed by ZIVE SP1 potentiostat (WonATech Co,Ltd. Seoul, South Korea). 10 uL 5mM $K_3Fe(CN)_6$/5 mM $K_4Fe(CN)_6$ in PBS was added onto each electrode pair. 0 V bias and stimulus at 10mV V_{rms} was applied for measuring. Frequency was swept from 800 kHz to 1 Hz. Figure 6.9 shows the actual sensor, and its corresponding circuit model for EIS and the final surface structure after immobilization steps. Since there are two asymmetric electrode/electrolyte interface, 2 different double layer capacitance and electron transfer resistance should be modeled.

Warburg impedance were ignored to simplify the analysis. The transfer function for impedance can be calculated by Eq. (6.11), and double layer capacitance as well as charge transfer resistance can be back calculated with known impedance and frequency.

$$Z(\omega) = Z' + jZ'' \tag{6.10}$$

$$Z(\omega) = R_s + \frac{R_{ct1}}{1+\omega^2 R_{ct1}^2 C_{dl1}^2} - \frac{j\omega R_{ct1}^2 C_{dl1}^2}{1+\omega^2 R_{ct1}^2 C_{dl1}^2} + \frac{R_{ct2}}{1+\omega^2 R_{ct2}^2 C_{dl2}^2} - \frac{j\omega R_{ct2}^2 C_{dl2}^2}{1+\omega^2 R_{ct2}^2 C_{dl2}^2} \tag{6.11}$$

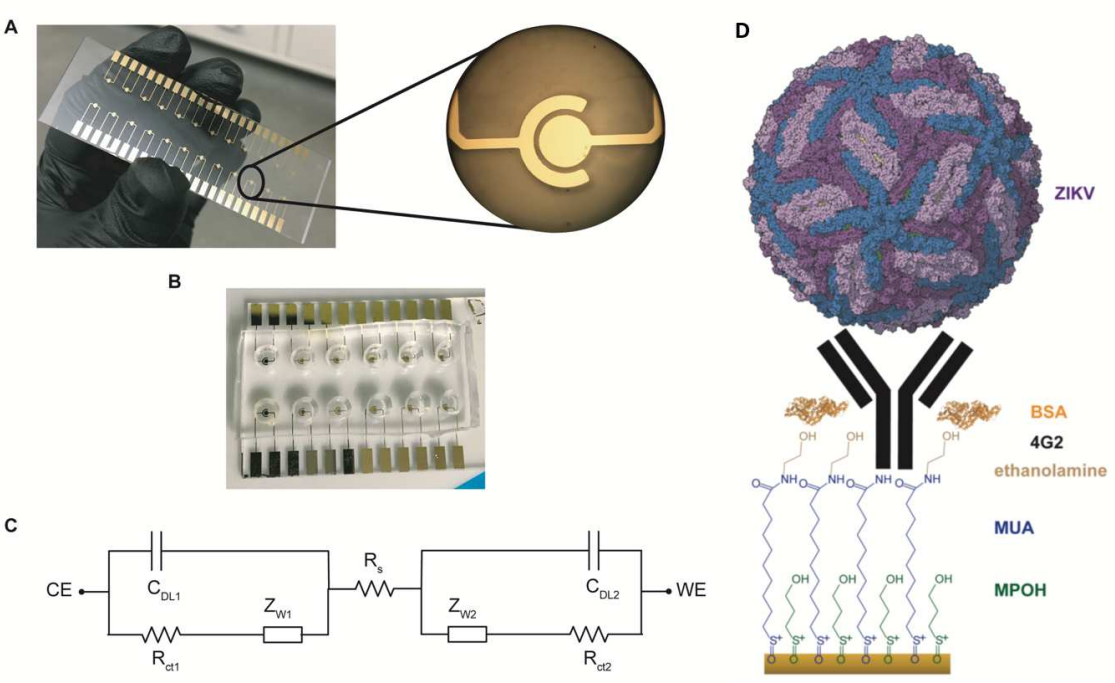


Figure 6.9: The gold electrode array on glass. A)2 by 12 electrode array and individual gold electrode pair. B) Electrode array with PDMS well bound. C) The equivalent circuit model for EIS, R_{ct} stands for equivalent charge transfer resistance, C_{dl} stands for double layer capacitance, Z_w stands for Warburg impedance and R_s stands for solution impedance. D) Final surface modification for ZIKV detection [83].

The measured EIS data are shown in Figure 6.10. In this case, PBS was added and measurement for each step as blank control to confirm the detection of ZIKV. ZIKV concentration from 10 focus forming units (FFU) to 11110 FFU were investigated. The final calibration results were presented in $\%\delta R_{ct}$ to reduce variation introduced by different active area for electron transfer, while each concentration has 4 replicates. The linearity for ZIKV was reasonably good (with R^2=0.9843).

Although we can observe an unwanted signal change from PBS background, the ZIKV signal has a sharper slope to make it distinguishable to its blank control. By taking signal-noise-ratio (SNR) equals to 3 as the limit of detection (LOD), the LOD was calculated to be 22.4 FFU, which was comparable to other reported viral loads of 80 PFU/mL of ZIKV in saliva [76].

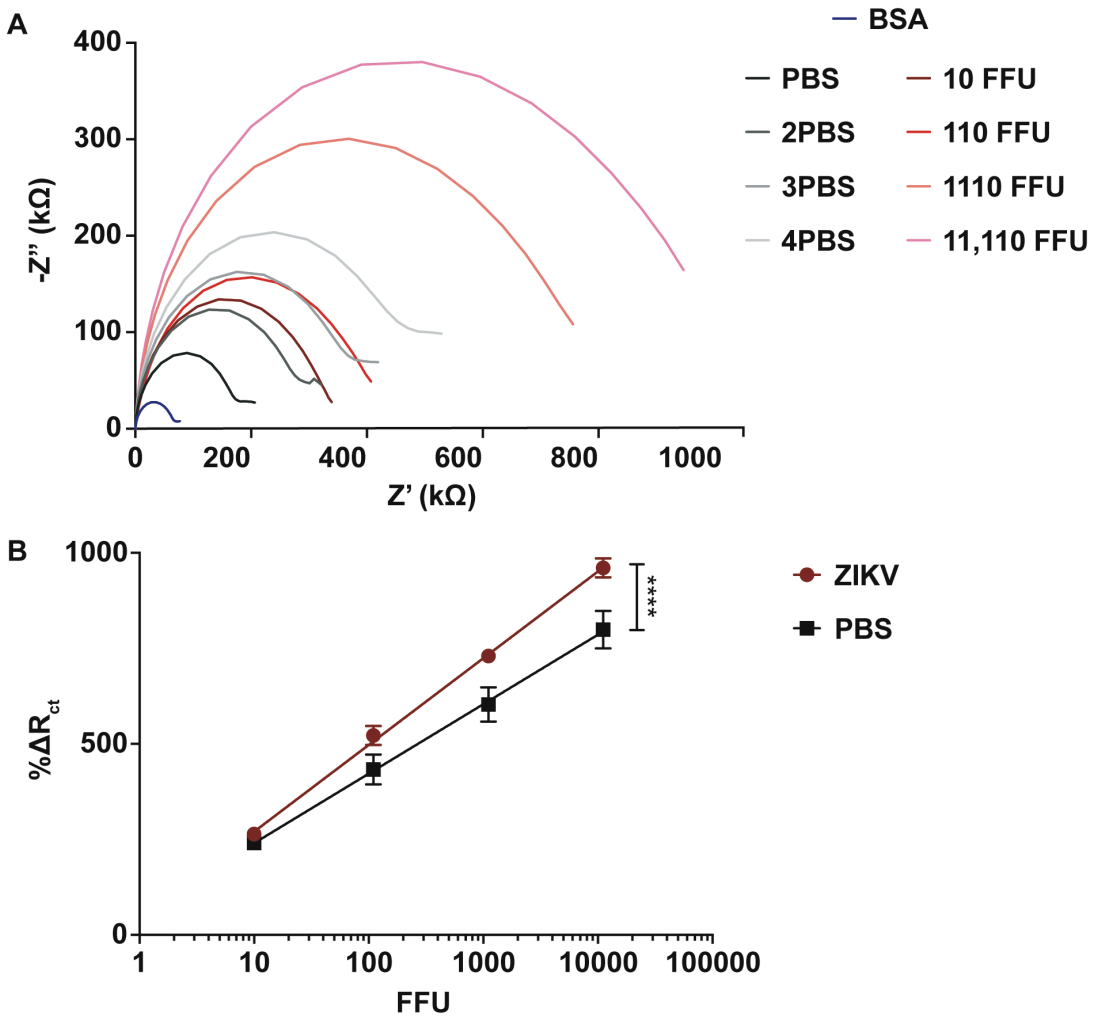


Figure 6.10: EIS measurement for ZIKV sensor. A)Nyquist plot result for ZIKV concentration at 10 FFU,110 FFU,1110 FFU and 11110 FFU. B) Calibration for ZIKV versus PBS control [83].

6.4 An Glutamine Sensor Measured by Porposed Electrochemical Analyzer

In additional to electrical testing resulting to show the proposed electrochemical analyzer's performance, we present its validation results using a custom-designed glutamine sensor. The glutamine sensor has gained its growing importance in cancer metabolism [85] [86]. The sensor was built as three electrode system, where we had platinum as counter electrode, gold as working electrode and a silver wire as pseudo-reference electrode. The glutamine sensor using the proposed analyzer achieved linearity in the range of 500 uM and a limit of detection at 50uM.

6.4.1 Glutamine Sensor Preparation and Enzyme Immobilization

The gold and platinum electrode were attached face to face with a 4 mm thick treated PDMS in between to set the gap. The PDMS was treated on both sides by plasma for 3 minutes in order to create covalent bond when it binds to glass substrate on both glad and platinum electrode. A 1 mm diameter silver was then pierce through the PMDS in the middle to form Au-Ag-Pt three electrode sensor for the next use. The sensor was first rinse by acetone, then followed by isopropyl alcohol and deionized water as initial cleaning step. The sensor was then cleaned with 50 mM NaOH+25% H_2O_2 [66] for 15 minutes before replacing the cleaning solution, and the process was repeated for 3 times. Both L-glutamate oxidase and glutaminase were dissolved in 0.1M PBS, and the aqueous was then mixed by BSA (10% wt%), glutaraldehyde (2% vol%) [87] as well as tween-20 (2% vol%). BSA was used to help adhere to the sensor surface. Glutaraldehyde was used as corss-linker for the proteins. Tween-20 was used to help dissolve BSA in PBS. The L-glutamate oxidase membrane was first immobilized onto the Au electrode surface by adding 2 uL (0.1 Unit) mixed solution, and waited for drying out. Then 10 uL (0.1 Unit) glutaminase solution was added on top of the L-glutamate oxidase membrane to create a layer with glutaminase enzyme. The immobilized sensor is illustrated in Figure 6.12, the two steps immobilization was meant to have a nature gradient to have H_2O_2 at the end from glutamine. The prepared sensor was then store in

refrigerator at 4 Celsius and ready for the next day's use for the measurement. Figure 6.11 shows the prepared sensor unit.

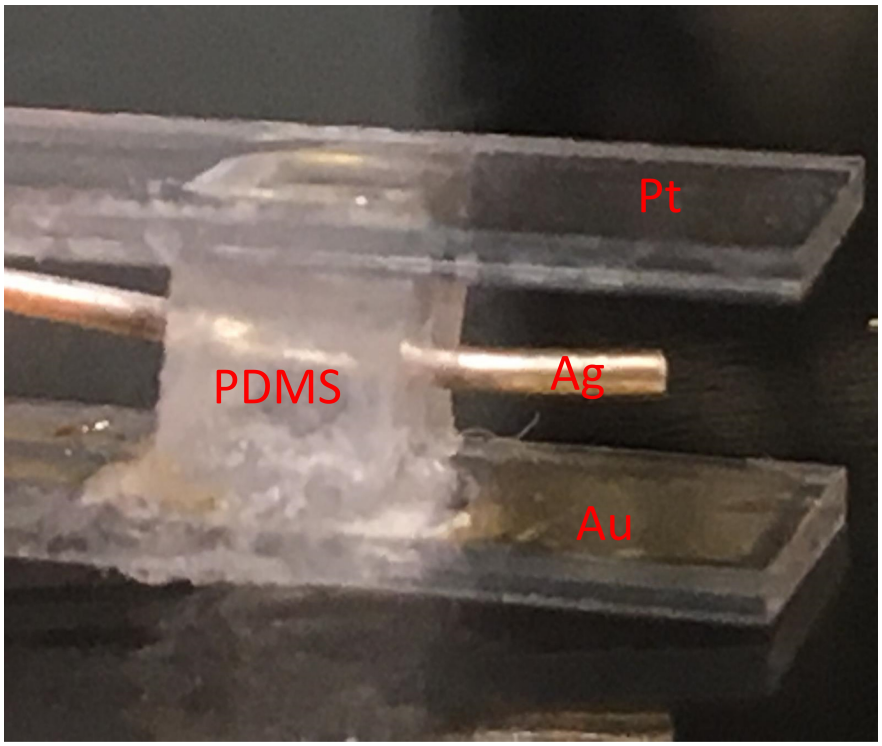


Figure 6.11: The 3 electrode system for glutamine sensor.

6.4.2 Sensor Characterization

Before the experiment starts, it is important to characterize the electrochemical active area on the sensor for later analysis. 5mM Ferri/Ferrocyanide in 0.1 M KCl were used as standard redox couple for testing. Cyclic voltametry running at 100 mV/s was applied to the electrodes and scanned from -0.8 V to +1 V. As shown in Figure 6.13, the oxidization peak is around +680 uA and reduction peak is about -550 uA. By applying Randles-Sevcik equation as in Eq.(6.12), the area can be estimated. For $K_3Fe(CN)_6$, $n = 1, D = 7.6 \times 10^{-6} cm^2 s^{-1} (0.1 MKCl)$ [88], C is the concentration of $K_3Fe(CN)_6$ which was 5 mM, v is the scan rate. The electrode active area was calculated to be at $0.47 cm^2$.

$$I_p = 2.69 \times 10^5 AC\sqrt{n^3 Dv} \tag{6.12}$$

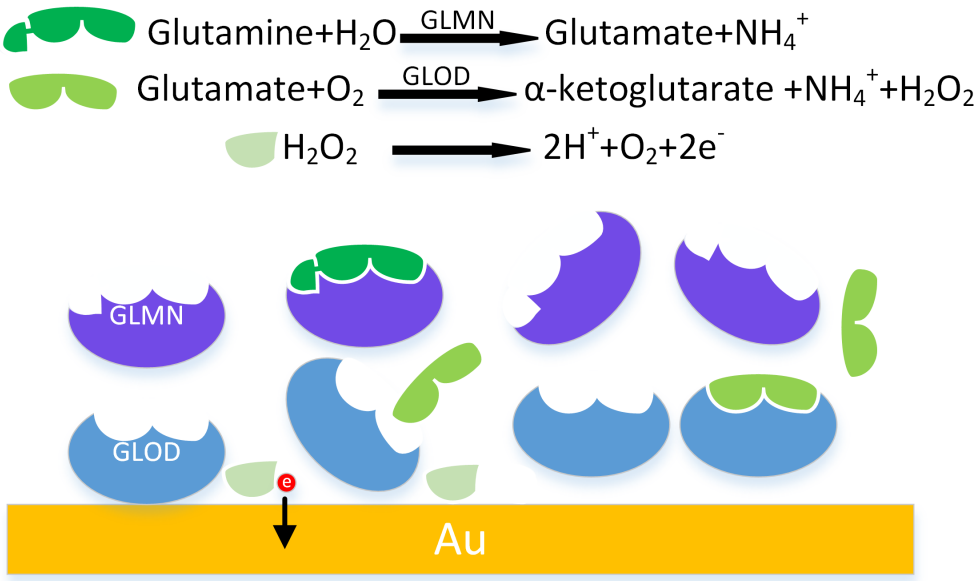


Figure 6.12: Surface of immobilized enzymatic sensor.

6.4.3 Glutamine Sensor Calibration Curve

Amperometry was conducted for glutamine enzymatic sensor with +600 mV potential [87] applied to the working electrode vs Ag reference electrode. For each measurement, 2 ml glutamine solution in 0.1X PBS buffer was replaced at concentration from 100 uM to 2mM within 5 steps. The calibration curve is shown in Figure 6.14 with linear fitting in its linear region. The glutamine sensor reached a sensitivity at 36.67 nA/mM, or 78 nA/(mM*cm^2) if taking into account the active area calculated from CV in ferri/ferrocyanide. The limit of detection is determined when the signal is equal to standard deviation error, which turns to be around 50 uM. The fitted curve also indicated a 0.99556 R^2 value which implies a good linearity. However, when the concentration was increased close to 1 mM range, the signal flatten out and reached a saturation region. This can be a result of electron transfer kinetic limit at the electrode surface, in other words the electron transfer event can not deplete H_2O_2 generate from glutamine, therefore the current will not keep increasing linearly as the concentration raises. Nevertheless, this sensor could be applied to applications, such as cancer cell monitoring, when glutamine concentration lies above the limit of detection.

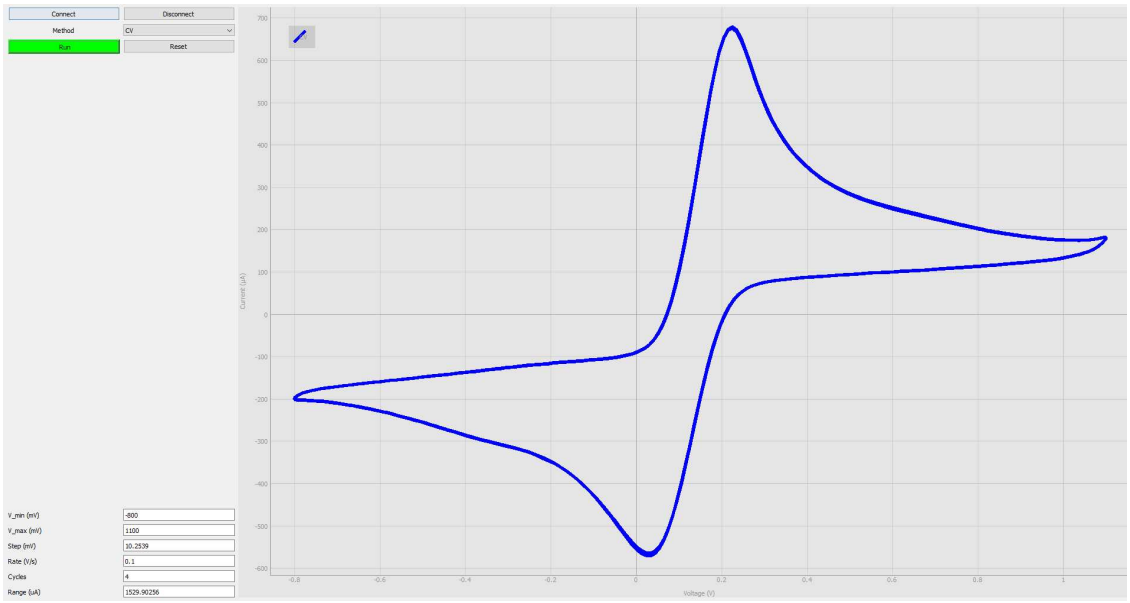


Figure 6.13: CV for glutamine sensor characterization.

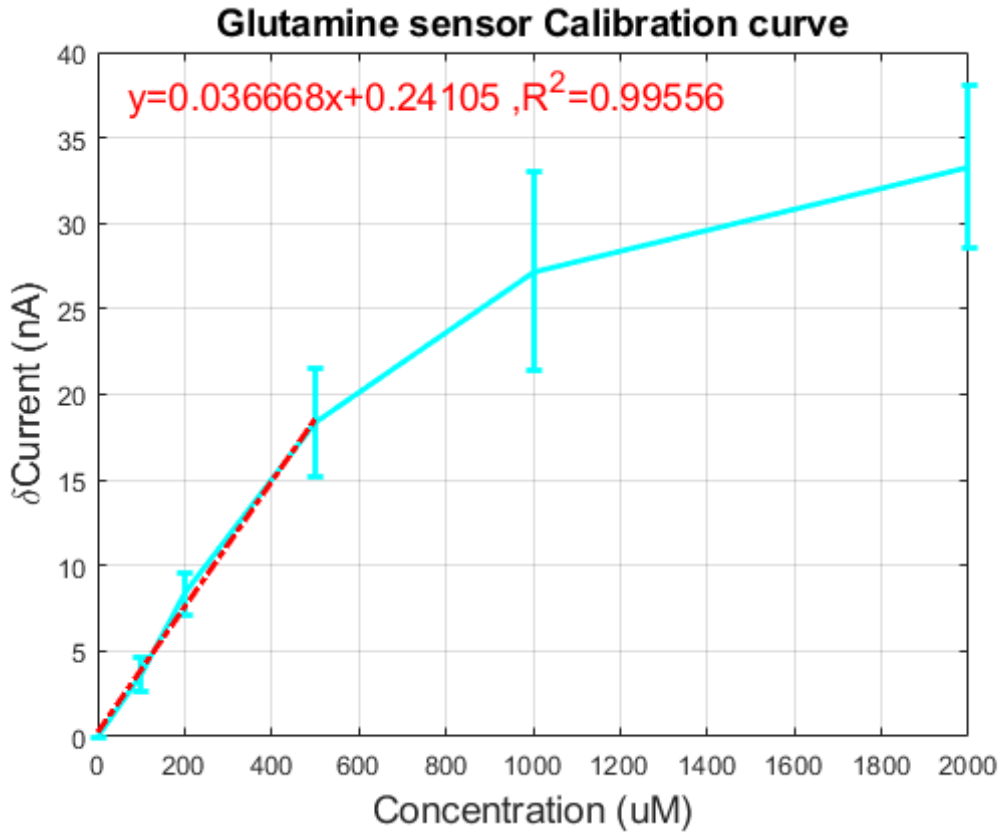


Figure 6.14: Glutamine sensor calibration curve with linear fitting.

Chapter 7

Conclusion

In this work, various electrochemical sensors have been studied and proven to have practical meaning for point-of-care (POT) applications. The neurotransmitter sensor worked as good foundation for larger scale tissue imaging technique [63], as it confirmed the electron transfer events in amperometric experiment. The DNA capacitive sensor was carefully design, and achieved a dynamic range for detection as low as 1 molecule/uL for target DNA. This experiment proved the possibility of detection from non-faradic electrochemical event. Additionally, a ZIKV sensor was developed and tested by customized gold electrode array. A clinically relevant LOD of 22.4 FFU was calculated. All these experiments have emphasized opportunities for rapid measurement and low-cost, low-power, ease of use features provided by electrochemical methods.

A low cost and reconfigurable electrochemical analyzer platform (Crexens^{TM}) was designed and tested. First of all, a CMOS FFT processor was customized. The proposed architecture attempts to minimized the overall FFT silicon area and power consumption while maximizing performance. The proposed bit-serial implementation may not be suitable for applications that require high performance, it certainly provides a desired tradeoff for implantable biomedical applications where size and power consumption are more important. Three generations of electrochemical analyzer were designed. In the first generation, a multi-frequency EIS device is implemented to capture impedance spectroscopy without a sweeping effort. Low input amplitude and the test with 64 frequency components between 2Hz to 2KHz were accomplished. However, its lack of auto-gain control and adaptive filtering made its detection dynamic range inferior compared to existing designs. In addition, the unavailability of user interface made such design unfriendly to untrained personnel. In the second generation, the platform was greatly improved by introducing gain/filter control mechanisms, as well as an Android app. The multi-tone and single-tone modes were also available in the second generation design and compared. The single-tone mode was in general a better approach for measurements that don't have high demand on EIS timing. The platform's

capability and performance were validated using an electrolyte-electrode interface model circuit with an average error of 2.6% in impedance and 1.4 degree in phase in the single-tone mode. To further broaden the capability without adding too much cost to the system, the third generation Crexens™ was designed. The idea of 3rd generation Crexens™ was to make it an expandable general-purpose electro- chemical analyzer. The modular architecture allows the base unit to run most of the electrochemical experiments without going for expensive benchtop instruments, while the add-on modules are available as options if users need performance beyond the base unit. By partitioning the baseline and low noise specifications, the proposed analyzer can act as a general-purpose analyzer for electrochemistry without having users to pay for functions they do not use. Such a customization feature for functionality can significantly reduce the cost for average users, thus, increasing the accessibility of electrochemical instrumentation for a wider user community. It is estimated that the cost for the based unit will be in the range of $200-300. The proposed electrochemical analyzer obtains low cost, low power, small area, and battery-driven features that are essential for Point-of-Care (PoC) devices. Additionally, a glutamine enzymatic sensor was developed and verified by the proposed analyzer, which have shown good linearity in the linear region from its calibration curve.

Bibliography

[1] Andrea M Sisko, Sean P Keehan, John A Poisal, Gigi A Cuckler, Sheila D Smith, Andrew J Madison, Kathryn E Rennie, and James C Hardesty. National health expenditure projections, 2018–27: Economic and demographic trends drive spending and enrollment growth. *Health Affairs*, 38(3):491–501, 2019.

[2] Office of the Actuary National Health Statistics Group. National health care spending in 2016. https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/NationalHealthExpendData/NationalHealthAccountsHistorical.html.

[3] The Commonwealth Fund. Wait times for specialist appointment. http://www.commonwealthfund.org/interactives-and-data/chart-cart/survey/2013-international-health-policy-survey/wait-times-for-specialist-appointment.

[4] Joon Myong Song, Mustafa Culha, Paul M Kasili, Guy D Griffin, and Tuan Vo-Dinh. A compact cmos biochip immunosensor towards the detection of a single bacteria. *Biosensors and Bioelectronics*, 20(11):2203–2209, 2005.

[5] David C Ng, Takashi Tokuda, Akio Yamamoto, Masamichi Matsuo, Masahiro Nunoshita, Hideki Tamura, Yasuyuki Ishikawa, Sadao Shiosaka, and Jun Ohta. On-chip biofluorescence imaging inside a brain tissue phantom using a cmos image sensor for in vivo brain imaging verification. *Sensors and Actuators B: Chemical*, 119(1):262–274, 2006.

[6] Hyejung Kim, Sunyoung Kim, Nick Van Helleputte, Antonio Artes, Mario Konijnenburg, Jos Huisken, Chris Van Hoof, and Refet Firat Yazicioglu. A configurable and low-power mixed signal soc for portable ecg monitoring applications. *IEEE transactions on biomedical circuits and systems*, 8(2):257–267, 2014.

[7] William Tedjo, Rachel Feeny, Chad Eitel, Luke Schwerdtfeger, Stacy Willett, Charles Henry, Stuart Tobet, and Tom Chen. Design of an integrated microelectrode array system for high

spatiotemporal resolution chemical imaging. In *Biomedical Circuits and Systems Conference (BioCAS), 2016 IEEE*, pages 46–49. IEEE, 2016.

[8] Daniel L Bellin, Hassan Sakhtah, Jacob K Rosenstein, Peter M Levine, Jordan Thimot, Kevin Emmett, Lars EP Dietrich, and Kenneth L Shepard. Cmos electrochemical sensing platform for spatially resolved detection of redox-active metabolites released by multicellular films. *Biophysical Journal*, 106(2):812a, 2014.

[9] Arun Manickam, Aaron Chevalier, Mark McDermott, Andrew D Ellington, and Arjang Hassibi. A cmos electrochemical impedance spectroscopy (eis) biosensor array. *IEEE Transactions on Biomedical Circuits and Systems*, 4(6):379–390, 2010.

[10] Areeb Ali, Neelanjana Pal, and Peter M Levine. Cmos impedance spectroscopy sensor array with synchronous voltage-to-frequency converters. In *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pages 1–4. IEEE, 2015.

[11] A Alharbi, B Nasri, T Wu, and D Shahrjerdi. Advanced integrated sensor and layer transfer technologies for wearable bioelectronics. In *Electron Devices Meeting (IEDM), 2016 IEEE International*, pages 6–5. IEEE, 2016.

[12] Diming Zhang, Yanli Lu, Qian Zhang, Lei Liu, Shuang Li, Yao Yao, Jing Jiang, Gang Logan Liu, and Qingjun Liu. Protein detecting with smartphone-controlled electrochemical impedance spectroscopy for point-of-care applications. *Sensors and Actuators, B: Chemical*, 222:994–1002, 1 2016.

[13] P Heiduschka and S Thanos. Implantable bioelectronic interfaces for lost nerve functions. *Progress in neurobiology*, 55(5):433–461, 1998.

[14] Oscar Mayora, Bert Arnrich, Jakob Bardram, Carsten Dräger, Andrea Finke, Mads Frost, Silvia Giordano, Franz Gravenhorst, Agnes Grunerbl, Christian Haring, et al. Personal health systems for bipolar disorder: anecdotes, challenges and lessons learnt from monarca project.

In *Proceedings of the 7th International Conference on Pervasive Computing Technologies for Healthcare*, pages 424–429. ICST, 2013.

[15] Roland Thewes, Franz Hofmann, Alexander Frey, Birgit Holzapfl, Meinrad Schienle, Christian Paulus, Petra Schindler, Gerald Eckstein, Christian Kassel, Manfred Stanzel, et al. Sensor arrays for fully-electronic dna detection on cmos. In *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 02CH37315)*, volume 1, pages 350–473. IEEE, 2002.

[16] Anantha P Chandrakasan, Naveen Verma, and Denis C Daly. Ultralow-power electronics for biomedical applications. *Annu. Rev. Biomed. Eng.*, 10:247–274, 2008.

[17] Loren Schwiebert, Sandeep KS Gupta, and Jennifer Weinmann. Research challenges in wireless networks of biomedical sensors. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165. ACM, 2001.

[18] Ali Agah, Arjang Hassibi, James D Plummer, and Peter B Griffin. Design requirements for integrated biosensor arrays. In *Imaging, Manipulation, and Analysis of Biomolecules and Cells: Fundamentals and Applications III*, volume 5699, pages 403–414. International Society for Optics and Photonics, 2005.

[19] Philip E Stanley. Commercially available fluorometers, luminometers and imaging devices for low-light level measurements and allied kits and reagents: survey update 6. *Luminescence*, 14(4):201–213, 1999.

[20] Hua Wang, Yan Chen, Arjang Hassibi, Axel Scherer, and Ali Hajimiri. A frequency-shift cmos magnetic biosensor array with single-bead sensitivity and no external magnet. In *Solid-State Circuits Conference-Digest of Technical Papers, 2009. ISSCC 2009. IEEE International*, pages 438–439. IEEE, 2009.

[21] Y Maruyama, S Terao, and K Sawada. Label free cmos dna image sensor based on the charge transfer technique. *Biosensors and Bioelectronics*, 24(10):3108–3112, 2009.

[22] Takeshi Hizawa, Kazuaki Sawada, Hidekuni Takao, and Makoto Ishida. Fabrication of a two-dimensional ph image sensor using a charge transfer technique. *Sensors and Actuators B: Chemical*, 117(2):509–515, 2006.

[23] Claudio Stagni, Carlotta Guiducci, Luca Benini, Bruno Riccò, Sandro Carrara, Christian Paulus, Meinrad Schienle, and Roland Thewes. A fully electronic label-free dna sensor chip. *IEEE Sensors Journal*, 7(4):577–585, 2007.

[24] Sam Wright and Tom Chen. A pvt-compensated capacitive sensor with sub-1ff sensitivity. In *Circuits and Systems (MWSCAS), 2015 IEEE 58th International Midwest Symposium on*, pages 1–4. IEEE, 2015.

[25] A Zeng, E Liu, IF Annergren, SN Tan, S Zhang, P Hing, and J Gao. Eis capacitance diagnosis of nanoporosity effect on the corrosion protection of dlc films. *Diamond and Related Materials*, 11(2):160–168, 2002.

[26] Meinrad Schienle, Christian Paulus, Alexander Frey, Franz Hofmann, Birgit Holzapfl, Petra Schindler-Bauer, and Roland Thewes. A fully electronic dna sensor with 128 positions and in-pixel a/d conversion. *IEEE Journal of Solid-state circuits*, 39(12):2438–2445, 2004.

[27] Yi Xiao, Linru Xu, and Lian-Wen Qi. Electrochemiluminescence bipolar electrode array for the multiplexed detection of glucose, lactate and choline based on a versatile enzymatic approach. *Talanta*, 165:577–583, 2017.

[28] Flávio CB Fernandes, Adriano Santos, Denise C Martins, Márcio S Góes, and Paulo R Bueno. Comparing label free electrochemical impedimetric and capacitive biosensing architectures. *Biosensors and Bioelectronics*, 57:96–102, 2014.

[29] Hai-Bo Wang, Hong-Ding Zhang, Shu-Ping Xu, Tian Gan, Ke-Jing Huang, and Yan-Ming Liu. A sensitive and label-free electrochemical impedance biosensor for protein detection based on terminal protection of small molecule-linked dna. *Sensors and Actuators B: Chemical*, 194:478–483, 2014.

[30] Joshua Lehr, FlaÌĄvio C Bedatty Fernandes, Paulo R Bueno, and Jason J Davis. Label-free capacitive diagnostics: exploiting local redox probe state occupancy. *Analytical chemistry*, 86(5):2559–2564, 2014.

[31] J Ross Macdonald and E Barsoukov. Impedance spectroscopy: theory, experiment, and applications. *History*, 1(8), 2005.

[32] NL Kusters and O Petersons. A transformer-ratio-arm bridge for high-voltage capacitance measurements. *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, 82(5):606–611, 1963.

[33] Phillip E Allen and Douglas R Holberg. *CMOS analog circuit design*. Oxford Univ. Press, 2002.

[34] Fu-Liang Yang, Di-Hong Lee, Hou-Yu Chen, Chang-Yun Chang, Sheng-Da Liu, Cheng-Chuan Huang, Tang-Xuan Chung, Hung-Wei Chen, Chien-Chao Huang, Yi-Hsuan Liu, et al. 5nm-gate nanowire finfet. In *VLSI Technology, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 196–197. IEEE, 2004.

[35] Aaron A Rowe, Andrew J Bonham, Ryan J White, Michael P Zimmer, Ramsin J Yadgar, Tony M Hobza, Jim W Honea, Ilan Ben-Yaacov, and Kevin W Plaxco. Cheapstat: an open-source,"do-it-yourself" potentiostat for analytical and educational applications. *PloS one*, 6(9):e23783, 2011.

[36] Sami Kirolos, Jason Laska, Michael Wakin, Marco Duarte, Dror Baron, Tamer Ragheb, Yehia Massoud, and Richard Baraniuk. Analog-to-information conversion via random demodulation. In *Design, Applications, Integration and Software, 2006 IEEE Dallas/CAS Workshop on*, pages 71–74. IEEE, 2006.

[37] Walt Kester. Which adc architecture is right for your application. In *EDA Tech Forum*, volume 2, pages 22–25, 2005.

[38] Oleg A Mukhanov and Alexander F Kirichenko. Implementation of a fft radix 2 butterfly using serial rsfq multiplier-adders. *IEEE Transactions on Applied Superconductivity*, 5(2):2461–2464, 1995.

[39] Dennis V Perepelitsa. Johnson noise and shot noise. *Analysis*, (2):2–5, 2006.

[40] Jimmin Chang, AA Abidi, and CR Viswanathan. Flicker noise in cmos transistors from subthreshold to strong inversion at various temperatures. *IEEE Transactions on Electron Devices*, 41(11):1965–1971, 1994.

[41] Hongjiang He and Hui Guo. The realization of fft algorithm based on fpga co-processor. In *Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on*, volume 3, pages 239–243. IEEE, 2008.

[42] Peter A Ruetz and MikeM Cai. A real time fft chip set: architectural issues. In *Pattern Recognition, 1990. Proceedings., 10th International Conference on*, volume 2, pages 385–388. IEEE, 1990.

[43] Glen Sunada, Jain Jin, Matt Berzins, and Tom Chen. Cobra: an 1.2 million transistor expandable column fft chip. In *Computer Design: VLSI in Computers and Processors, 1994. ICCD'94. Proceedings., IEEE International Conference on*, pages 546–550. IEEE, 1994.

[44] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold. A 64-point fourier transform chip for high-speed wireless lan application using ofdm. *IEEE Journal of Solid-State Circuits*, 39(3):484–493, 2004.

[45] M Hasan, Tughrul Arslan, and John S Thompson. A novel coefficient ordering based low power pipelined radix-4 fft processor for wireless lan applications. *IEEE Transactions on Consumer Electronics*, 49(1):128–134, 2003.

[46] Andrew J Miller. Fpga implemented bit-serial multiplier and infinite impulse response, August 20 2002. US Patent 6,438,570.

[47] Kenneth C Laprade. N-clock, n-bit-serial multiplier, June 13 1989. US Patent 4,839,847.

[48] LE Bowman and AP Wade. An introduction to digital signal processors, 1993.

[49] Yuxiang Yang, Fu Zhang, Kun Tao, Lianhuan Wang, He Wen, and Zhaosheng Teng. Multi-frequency simultaneous measurement of bioimpedance spectroscopy based on a low crest factor multisine excitation. *Physiological measurement*, 36(3):489, 2015.

[50] Chin-Teng Lin, Yuan-Chu Yu, and Lan-Da Van. A low-power 64-point fft/ifft design for ieee 802.11 a wlan application. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pages 4–pp. IEEE, 2006.

[51] C Chiu, Wing Hui, Tiong Jiu Ding, and JV McCanny. A 64-point fourier transform chip for video motion compensation using phase correlation. *IEEE Journal of solid-state circuits*, 31(11):1751–1761, 1996.

[52] Jen-Chih Kuo, Ching-Hua Wen, and An-Yeu Wu. Implementation of a programmable 64/spl sim/2048-point fft/ifft processor for ofdm-based communication systems. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, volume 2, pages II–II. IEEE, 2003.

[53] Matthias Wosnitza, Marco Cavadini, Markus Thaler, and Gerhard Troster. A high precision 1024-point fft processor for 2d convolution. In *Solid-State Circuits Conference, 1998. Digest of Technical Papers. 1998 IEEE International*, pages 118–119. IEEE, 1998.

[54] Frank BJ Leferink and Marcel JCM van Doom. Inductance of printed circuit board ground planes. In *1993 International Symposium on Electromagnetic Compatibility*, pages 327–329. IEEE, 1993.

[55] HN Phyu, Er-Ping Li, and Weiliang Yuan. Analysis of electromagnetic susceptibility on high speed circuits located in a shielded enclosure. In *2006 17th International Zurich Symposium on Electromagnetic Compatibility*, pages 312–315. IEEE, 2006.

[56] John B Wydallis, Rachel M Feeny, William Wilson, Tucker Kern, Tom Chen, Stuart Tobet, Melissa M Reynolds, and Charles S Henry. Spatiotemporal norepinephrine mapping using a high-density cmos microelectrode array. *Lab on a Chip*, 15(20):4075–4082, 2015.

[57] Kevin T Schomacker, Joan K Frisoli, Carolyn C Compton, Thomas J Flotte, James M Richter, Norman S Nishioka, and Thomas F Deutsch. Ultraviolet laser-induced fluorescence of colonic tissue: basic biology and diagnostic potential. *Lasers in surgery and medicine*, 12(1):63–78, 1992.

[58] Warren R Zipfel, Rebecca M Williams, Richard Christie, Alexander Yu Nikitin, Bradley T Hyman, and Watt W Webb. Live tissue intrinsic emission microscopy using multiphoton-excited native fluorescence and second harmonic generation. *Proceedings of the National Academy of Sciences*, 100(12):7075–7080, 2003.

[59] Yasuyuki Amoh, Kensei Katsuoka, and Robert M Hoffman. Color-coded fluorescent protein imaging of angiogenesis: The angiomouse. *Current pharmaceutical design*, 14(36):3810–3819, 2008.

[60] C Köbbert, R Apps, I Bechmann, José Luis Lanciego, J Mey, and S Thanos. Current concepts in neuroanatomical tracing. *Progress in neurobiology*, 62(4):327–351, 2000.

[61] Larry J Kricka and Paolo Fortina. Analytical ancestry:"firsts" in fluorescent labeling of nucleosides, nucleotides, and nucleic acids. *Clinical Chemistry*, 55(4):670–683, 2009.

[62] Warren Pettine, Matthew Jibson, Tom Chen, Stuart Tobet, Phil Nikkel, and Charles S Henry. Characterization of novel microelectrode geometries for detection of neurotransmitters. *IEEE sensors journal*, 12(5):1187–1192, 2011.

[63] William Tedjo, Jasmine E Nejad, Rachel Feeny, Lang Yang, Charles S Henry, Stuart Tobet, and Tom Chen. Electrochemical biosensor system using a cmos microelectrode array provides high spatially and temporally resolved images. *Biosensors and Bioelectronics*, 114:78–88, 2018.

[64] Lei Wang, Milena Veselinovic, Lang Yang, Brian J Geiss, David S Dandy, and Tom Chen. A sensitive dna capacitive biosensor using interdigitated electrodes. *Biosensors and Bioelectronics*, 87:646–653, 2017.

[65] Atsushi Nishino, Akihiko Yoshida, and Ichiroh Tanahashi. Electric double layer capacitor, December 31 1985. US Patent 4,562,511.

[66] Lee M Fischer, Maria Tenje, Arto R Heiskanen, Noriyuki Masuda, Jaime Castillo, Anders Bentien, Jenny Émneus, Mogens H Jakobsen, and Anja Boisen. Gold cleaning methods for electrochemical detection applications. *Microelectronic engineering*, 86(4-6):1282–1285, 2009.

[67] Colin D Bain, Joe Evall, and George M Whitesides. Formation of monolayers by the coadsorption of thiols on gold: variation in the head group, tail group, and solvent. *Journal of the American Chemical Society*, 111(18):7155–7164, 1989.

[68] Mika M Kohonen, Marilyn E Karaman, and Richard M Pashley. Debye length in multivalent electrolyte solutions. *Langmuir*, 16(13):5749–5753, 2000.

[69] Kouichi Morita, Tatsuo Maemoto, Shoji Honda, Kenji Onishi, Misako Murata, Mariko Tanaka, and Akira Igarashi. Rapid detection of virus genome from imported dengue fever and dengue hemorrhagie fever patients by direct polymerase chain reaction. *Journal of medical virology*, 44(1):54–58, 1994.

[70] G Moureau, S Temmam, JP Gonzalez, RN Charrel, G Grard, and X De Lamballerie. A real-time rt-pcr method for the universal detection and identification of flaviviruses. *Vector-Borne and Zoonotic Diseases*, 7(4):467–478, 2007.

[71] Yuan-Tih Ko, Marion Man-Ying Chan, Susan E Ford, and Dunne Fong. A pcr-elisa method for direct detection of the oyster pathogen haplosporidium nelsoni. *Marine Biotechnology*, 1(2):147–154, 1999.

[72] Giovanni Tagliabue, Valentina Faoro, Serena Rizzo, Daniele Sblattero, Andrea Saccani, Gabriele Riccio, Tommaso Bellini, Matteo Salina, Marco Buscaglia, and Alessandro Marcello. A label-free immunoassay for flavivirus detection by the reflective phantom interface technology. *Biochemical and biophysical research communications*, 492(4):558–564, 2017.

[73] Anjum Qureshi, Javed H Niazi, Saravan Kallempudi, and Yasar Gurbuz. Label-free capacitive biosensor for sensitive detection of multiple biomarkers using gold interdigitated capacitor arrays. *Biosensors and Bioelectronics*, 25(10):2318–2323, 2010.

[74] Ha-Wook Jung, Young Wook Chang, Ga-yeon Lee, Sungbo Cho, Min-Jung Kang, and Jae-Chul Pyun. A capacitive biosensor based on an interdigitated electrode with nanoislands. *Analytica chimica acta*, 844:27–34, 2014.

[75] Ganna Chornokur, Sunil K Arya, Catherine Phelan, Richard Tanner, and Shekhar Bhansali. Impedance-based miniaturized biosensor for ultrasensitive and fast prostate-specific antigen detection. *Journal of Sensors*, 2011, 2011.

[76] Anne U Holland, Carsten Munk, Ginger R Lucero, Lucia D Nguyen, and Nathaniel R Landau. α-complementation assay for hiv envelope glycoprotein-mediated fusion. *Virology*, 319(2):343–352, 2004.

[77] Alister En Kai Peh and Sam Fong Yau Li. Dengue virus detection using impedance measured across nanoporous aluminamembrane. *Biosensors and Bioelectronics*, 42:391–396, 2013.

[78] Robert B Channon, Yuanyuan Yang, Kristen M Feibelman, Brian J Geiss, David S Dandy, and Charles S Henry. Development of an electrochemical paper-based analytical device for trace detection of virus particles. *Analytical chemistry*, 90(12):7777–7783, 2018.

[79] Xiaohui Geng, Fanglin Zhang, Qiang Gao, and Yingfeng Lei. Sensitive impedimetric immunoassay of japanese encephalitis virus based on enzyme biocatalyzed precipitation on a gold nanoparticle-modified screen-printed carbon electrode. *Analytical Sciences*, 32(10):1105–1109, 2016.

[80] Binh Thi Thanh Nguyen, Alister En Kai Peh, Celine Yue Ling Chee, Katja Fink, Vincent TK Chow, Mary ML Ng, and Chee-Seng Toh. Electrochemical impedance spectroscopy characterization of nanoporous alumina dengue virus biosensor. *Bioelectrochemistry*, 88:15–21, 2012.

[81] Gary S May and Costas J Spanos. *Fundamentals of semiconductor manufacturing and process control*. Wiley Online Library, 2006.

[82] B-H Jo, Linda M Van Lerberghe, Kathleen M Motsegood, and David J Beebe. Three-dimensional micro-channel fabrication in polydimethylsiloxane (pdms) elastomer. *Journal of microelectromechanical systems*, 9(1):76–81, 2000.

[83] Jessica Filer. *Development of electrochemical assays and biosensors for detection of Zika virus*. PhD thesis, Colorado State University, 2019.

[84] Carmen-Marinela Mihailescu, Dana Stan, Rodica Iosub, Carmen Moldovan, and Mihaela Savin. A sensitive capacitive immunosensor for direct detection of human heart fatty acid-binding protein (h-fabp). *Talanta*, 132:37–43, 2015.

[85] Yeon-Kyung Choi and Keun-Gyu Park. Targeting glutamine metabolism for cancer treatment. *Biomolecules & therapeutics*, 26(1):19, 2018.

[86] Tineke WH Meijer, Wenny JM Peeters, Ludwig J Dubois, Marike W van Gisbergen, Rianne Biemans, Jan-Hendrik Venhuizen, Paul N Span, and Johan Bussink. Targeting glucose and glutamine metabolism combined with radiation therapy in non-small cell lung cancer. *Lung Cancer*, 126:32–40, 2018.

[87] Matthias Bäcker, D Rakowski, Arshak Poghossian, Manfred Biselli, Patrick Wagner, and Michael J Schöning. Chip-based amperometric enzyme sensor system for monitoring of bioprocesses by flow-injection analysis. *Journal of biotechnology*, 163(4):371–376, 2013.

[88] Ke-Jing Huang, De-Jun Niu, Xue Liu, Zhi-Wei Wu, Yang Fan, Ya-Fang Chang, and Ying-Ying Wu. Direct electrochemistry of catalase at amine-functionalized graphene/gold nanoparticles composite film for hydrogen peroxide sensor. *Electrochimica Acta*, 56(7):2947–2953, 2011.

Appendix A

Material and Reagents for Glutamine Enzymatic Sensor

All chemicals were at least of ACS grade and used as received without additional purification. Glutamine, glutaraldehyde, bovine serum albumin (BSA), tween-20 and phosphate buffered saline (PBS) were purchased from Sigma-Aldrich (St. Louis, MO, USA). L-glutamate oxidase (GLOD) (EC 1.4.3.11) from Streptomyces sp (>5UN/mg), and glutaminase (GLMN) (EC 3.5.1.2) is from Escherichia coli(50-200UN/mg) were also purchased from Sigma-Aldrich. Sylgard 184 polydimethyl-siloxane (PDMS) oligomer and cross-linker were obtained from Dow Corning (Midland, MI, USA). The gold and platinum electrodes were purchased from Metrohm dropsens (Llanera, Asturias, Spain).

Appendix B

Partial Java Code for Android GUI

B.1 MainActivity.java

```
package com.yang.eis;


import android.Manifest;
import android.app.ActionBar;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.Context;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.graphics.drawable.ColorDrawable;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
```

```
import android.view.Window;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.Toast;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {
   private boolean mScanning;
   private Handler mHandler;
   private ListView scanningDeviceListView;
   private ScannedDeviceAdapter scannedDeviceAdapter;
   private BluetoothAdapter mBluetoothAdapter;
   private final static int REQUEST_ENABLE_BT = 12;
   private static final long SCAN_PERIOD = 10000;
   private static final int MY_PERMISSIONS_LOCATIONS_COARSE =
      1561;
   private static final int MY_PERMISSIONS_WRITE_STORAGE = 7514;
   private static final int MY_PERMISSIONS_SEND_SMS = 2215;
   private static final int MY_PERMISSIONS_INTERNET = 1154;
   private static final int MULTIPLE_PERMISSIONS_REQUEST = 134;
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_device_scan);
      //Set Orientation in Landscape
```

```
setRequestedOrientation(ActivityInfo.
   SCREEN_ORIENTATION_LANDSCAPE);
//Set up action bar:
if(getActionBar()!=null){
   getActionBar().setDisplayHomeAsUpEnabled(false);
}
ActionBar actionBar = getActionBar();
//Flag to keep screen on (stay-awake):
getWindow().addFlags(WindowManager.LayoutParams.
   FLAG_KEEP_SCREEN_ON);
//Set up timer Handler
mHandler = new Handler();
//Initialize scanningDeviceListView Adapter:
scanningDeviceListView = (ListView) findViewById(R.id.
   scanningList);
//Check for BLE Support
if (!getPackageManager().hasSystemFeature(PackageManager.
   FEATURE_BLUETOOTH_LE)) {
   Toast.makeText(this, R.string.ble_not_supported, Toast.
      LENGTH_SHORT)
          .show();
   finish();
}
//Initialize Bluetooth manager
BluetoothManager bluetoothManager = (BluetoothManager)
   getSystemService(Context.BLUETOOTH_SERVICE);
mBluetoothAdapter = bluetoothManager.getAdapter();
```

```
if(mBluetoothAdapter == null) {
   Toast.makeText(this, R.string.
      error_bluetooth_not_supported,
         Toast.LENGTH_SHORT).show();
   finish();
   return;
}
int permissionCheck = ContextCompat.checkSelfPermission(
   MainActivity.this, Manifest.permission.
   ACCESS_COARSE_LOCATION);
int permissionCheck2 = ContextCompat.checkSelfPermission(
   MainActivity.this, Manifest.permission.
   WRITE_EXTERNAL_STORAGE);
int permissionCheck3 = ContextCompat.checkSelfPermission(
   MainActivity.this, Manifest.permission.SEND_SMS);
int permissionCheck4 = ContextCompat.checkSelfPermission(
   MainActivity.this, Manifest.permission.INTERNET);
if(permissionCheck!=PackageManager.PERMISSION_GRANTED &&
   permissionCheck2!=PackageManager.PERMISSION_GRANTED &&
   permissionCheck3!=PackageManager.PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
       String[]{Manifest.permission.ACCESS_COARSE_LOCATION,
         Manifest.permission.WRITE_EXTERNAL_STORAGE,
            Manifest.permission.SEND_SMS, Manifest.
            permission.INTERNET},
            MULTIPLE_PERMISSIONS_REQUEST);
```

```
} else if (permissionCheck!=PackageManager.
   PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
       String[]{Manifest.permission.ACCESS_COARSE_LOCATION
      }, MY_PERMISSIONS_LOCATIONS_COARSE);
} else if (permissionCheck2!=PackageManager.
   PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
       String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE
      }, MY_PERMISSIONS_WRITE_STORAGE);
} else if (permissionCheck3!=PackageManager.
   PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
       String[]{Manifest.permission.SEND_SMS},
      MY_PERMISSIONS_SEND_SMS);
} else if (permissionCheck4!=PackageManager.
   PERMISSION_GRANTED) {
   ActivityCompat.requestPermissions(MainActivity.this, new
       String[]{Manifest.permission.INTERNET},
      MY_PERMISSIONS_INTERNET);
}
//Initialize list view adapter:
scannedDeviceAdapter = new ScannedDeviceAdapter(this, R.
   layout.scanning_item, new ArrayList<ScannedDevice>());
scanningDeviceListView.setAdapter(scannedDeviceAdapter);
// Click Item Listener:
```

```
    scanningDeviceListView.setOnItemClickListener(new
       AdapterView.OnItemClickListener() {
       @Override
       public void onItemClick(AdapterView<?> adapterView, View
           view, int position, long id) {
          ScannedDevice item = scannedDeviceAdapter.getItem(
             position);
          if (item!=null) {
             final Intent intent = new Intent(MainActivity.this
                , EISactivity.class);
             intent.putExtra(AppConstant.EXTRAS_DEVICE_NAME,
                item.getDisplayName());
             intent.putExtra(AppConstant.EXTRAS_DEVICE_ADDRESS,
                 item.getDeviceMac());
             if(mScanning) {
                mBluetoothAdapter.stopLeScan(mLeScanCallback);
                mScanning = false;
             }
             startActivity(intent);
          }
       }
    });
}
private BluetoothAdapter.LeScanCallback mLeScanCallback = new
   BluetoothAdapter.LeScanCallback() {
   @Override
   public void onLeScan(final BluetoothDevice device,
```

```
                    final int rssi,
                    final byte[] scanRecord) {
        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                scannedDeviceAdapter.update(device, rssi,
                    scanRecord);
                scannedDeviceAdapter.notifyDataSetChanged();
            }
        });
    }
};
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_device_scan, menu);
    if (!mScanning) {
        menu.findItem(R.id.menu_stop).setVisible(false);
        menu.findItem(R.id.menu_scan).setVisible(true);
        //menu.findItem(R.id.menu_refresh).setActionView(null);
    } else {
        menu.findItem(R.id.menu_stop).setVisible(true);
        menu.findItem(R.id.menu_scan).setVisible(false);
        //menu.findItem(R.id.menu_refresh).setActionView(R.
            layout.actionbar_progress);
    }
    return true;
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.menu_scan:
         scannedDeviceAdapter.clear();
         scanLeDevice(true);
         break;
      case R.id.menu_stop:
         scanLeDevice(false);
         break;
   }
   return true;
}
@Override
protected void onResume() {
   super.onResume();

   /**
    * Ensures Bluetooth is enabled on the device - if not
       enabled - fire intent to display a
    * dialog to ask permission to enable
    */

   if (!mBluetoothAdapter.isEnabled()) {
      if (!mBluetoothAdapter.isEnabled()) {
         Intent enableBt = new Intent(BluetoothAdapter.
            ACTION_REQUEST_ENABLE);
```

```
            startActivityForResult(enableBt, REQUEST_ENABLE_BT);
        }
    }
    scanLeDevice(true);
}
@Override
protected void onActivityResult(int requestCode, int
    resultCode, Intent data) {
    //if user chose not to enable BT
    if (requestCode == REQUEST_ENABLE_BT && resultCode ==
        Activity.RESULT_CANCELED) {
        finish();
        return;
    }
    super.onActivityResult(requestCode, resultCode, data);
}
private void scanLeDevice(final boolean enable) {
    if(enable) {
        //stops scanning after ~seconds
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                mScanning = false;
                mBluetoothAdapter.stopLeScan(mLeScanCallback);
                invalidateOptionsMenu();
            }
        }, SCAN_PERIOD);
```

```
            mScanning = true;
            mBluetoothAdapter.startLeScan(mLeScanCallback);
        } else {
            mScanning = false;
            mBluetoothAdapter.stopLeScan(mLeScanCallback);
        }
        invalidateOptionsMenu();
    }
    @Override
    protected void onPause() {
        super.onPause();
        scanLeDevice(false);
        scannedDeviceAdapter.clear();
    }

}
```

B.2 EISactivity.java

```
package com.yang.eis;


import android.Manifest;
import android.app.ActionBar;
import android.app.Activity;
import android.app.Dialog;
import android.app.NotificationManager;
import android.app.PendingIntent;
```

```
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.BluetoothProfile;
import android.content.Context;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.graphics.Paint;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.os.Handler;
import android.provider.ContactsContract;
import android.support.annotation.NonNull;
import android.support.v4.app.NotificationCompat;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
```

```
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;
import android.widget.ToggleButton;
import com.beele.BluetoothLe;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.lang.Math.*;
import com.jjoe64.graphview.DefaultLabelFormatter;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.LegendRenderer;
import com.jjoe64.graphview.helper.StaticLabelsFormatter;
import com.jjoe64.graphview.series.BaseSeries;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;
import com.jjoe64.graphview.GridLabelRenderer;
import com.jjoe64.graphview.series.OnDataPointTapListener;
import com.jjoe64.graphview.series.PointsGraphSeries;
import com.jjoe64.graphview.series.Series;
import com.jjoe64.graphview.series.DataPointInterface;
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
import com.beele.BluetoothLe;
import com.opencsv.CSVWriter;
```

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.SimpleDateFormat;
import java.util.Arrays;
import java.util.List;
import java.util.*;
import java.util.UUID;
import android.support.v4.app.NotificationCompat;

public class EISactivity extends Activity implements BluetoothLe.
   BluetoothLeListener{

   private final static String TAG = EISactivity.class.
      getSimpleName();
   //LocalVars
   private String mDeviceName;
   private String mDeviceAddress;
   private boolean mConnected;
   //Class instance variable
   private BluetoothLe mBluetoothLe;
   private BluetoothGattCharacteristic mCharacteristic;
   private BluetoothGattCharacteristic characteristicTX;
   private BluetoothGattCharacteristic characteristicRX;
   private BluetoothManager mBluetoothManager = null;
   private BluetoothGatt mBluetoothGatt = null;
```

```
private BluetoothDevice mBluetoothDevice;
private TextView mRssi;
private TextView mRead;
private TextView mCon;
private TextView mStatus;
private TextView mReadX;
private TextView mReadY;
private TextView mReadFreq;
private TextView mID1;
private TextView mID2;
private TextView mID3;
private String strBLE = "␣";
private Menu menu;
private static final int RSSI_UPDATE_TIME_INTERVAL = 2000;
private Handler mTimerHandler = new Handler();
private boolean mTimerEnabled = false;
//Data Variables:
private float dataY;
//setup buttons
private ToggleButton Disconnected,NyquistPlot,PlotScale;
private static Button mExportButton,Capacitance,Impedance,
   Phase,Export,RefreshCV,CVHigh;
//Inherit legacy parameters
public static float[] DataY= new float[96];
public static float[] DataX= new float[96];
public static float[] DataXCV= new float[1024];
public static float[] DataYCV= new float[1024];
```

```
//public static char characteristic;
public static float[][] matrixData = new float[2][];
float[] frequency2={0.162f,0.19f,0.223f,0.262f,0.307f,0.362f
   ,0.424f,0.498f,0.583f,
   0.685f,0.803f,0.942f,1.1f,1.3f,1.52f,1.78f,2.09f,2.45f,2.9f
      ,3.36f,3.9f,4.6f,5.4f,
   6.3f,7.35f,8.58f,10f,11.7f,13.6f,15.8f,18.4f,21.3f,24.6f
      ,28.4f,32.7f,37.8f,43.2f,
   49.2f,55.5f,62.6f,70.5f,79.1f,88f,96.7f,107.4f,113.7f,117f
      ,120.5f,124.5f,128.7f,
   133f,137.75f,142.79f,148.2f,154.1f,160.2f,167.3f,174.8f,183
      f,192f,201.94f,212.98f,
   225.25f,239.6f,471.92f*422/472,707.88f*422/472,943.84f
      *422/472,1179.8f*422/472,
   1415.8f*422/472,1651.7f*422/472,1887.7f*422/472,2123.6f
      *422/472,2359.6f*422/472,
   2595.6f*422/472,2831.5f*422/472,3067.5f*422/472,3303.4f
      *422/472,3539.4f*422/472,
   3775.4f*422/472,4011.3f*422/472,4247.3f*422/472,4483.2f
      *422/472,4719.2f*422/472,
   4955.2f*422/472,5191.1f*422/472,5427.1f*422/472,5663f
      *422/472,6135f*422/472,
   6842.8f*422/472,7786.7f*422/472,8730.5f*422/472,9674.4f
      *422/472,10854f*422/472,
   12034f*422/472,13450*422/472,15101*422/472};
static float[] frequency
   ={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
```

```
    21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,
    39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,
    57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,
    75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
    93,94,95,96};
static String[] FrequencyString={};
//Set flags
private boolean BeginRecord=false;
private boolean BeginPlot=false;
private boolean CVmode=false;
private boolean EISmode=false;
private boolean FSCV=false;
private boolean SSCV=false;
private boolean BusyDevice=false;
private String MeasureStatus="Impedance";
private int Xindex = 0;
private int Yindex = 0;
private int index = 0;
private String Cmd;
private int BeginCV=1;
private int EndCV=4095;
private int StepCV=8;
private int CountCV=0;
// Notifications
NotificationCompat.Builder notification;
private static final int uniqueID = 45612;
private LinearLayout GraphView;
```

```
private GraphView graphView;
//Data Variables:
private float dataVoltage;
//private int batteryWarning = 20;//%
Dialog myDialog;
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   //Set background color
   LinearLayout background = (LinearLayout)findViewById(R.id.
      activity_main);
   background.setBackgroundColor(Color.BLACK);
   //NormailizeFreq();
   myDialog = new Dialog(this);
   myDialog.getWindow().requestFeature(Window.FEATURE_NO_TITLE
      );
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M &&
      checkSelfPermission(Manifest.permission.
      WRITE_EXTERNAL_STORAGE) != PackageManager.
      PERMISSION_GRANTED) {
      requestPermissions(new String[]{Manifest.permission.
         WRITE_EXTERNAL_STORAGE},1000);
   }
   //Set orientation of device based on screen type/size:
      setRequestedOrientation(ActivityInfo.
         SCREEN_ORIENTATION_LANDSCAPE);
```

```
//Recieve Intents:
Intent intent = getIntent();
mDeviceName = intent.getStringExtra(AppConstant.
   EXTRAS_DEVICE_NAME);
mDeviceAddress = intent.getStringExtra(AppConstant.
   EXTRAS_DEVICE_ADDRESS);
//Flag to keep screen on (stay-awake):
getWindow().addFlags(WindowManager.LayoutParams.
   FLAG_KEEP_SCREEN_ON);
//Set up TextViews
mRssi = (TextView) findViewById(R.id.textViewRssi);
mRead = (TextView) findViewById(R.id.textViewRead);
mStatus = (TextView) findViewById(R.id.textViewSta);
mCon = (TextView) findViewById(R.id.textViewCon);
mReadX =(TextView) findViewById(R.id.textViewReadX);
mReadY = (TextView) findViewById(R.id.textViewRead);
mReadFreq = (TextView) findViewById(R.id.textViewFreq);
mID1 =(TextView) findViewById(R.id.textViewID1);
mID2 =(TextView) findViewById(R.id.textViewID2);
mID3 =(TextView) findViewById(R.id.textViewID3);
mStatus.setText("Stand␣by");
mStatus.setTextColor(getResources().getColor(R.color.green)
   );
initialize();
GenerateCVdataX();
//Initial plot view
graphView = (GraphView) findViewById(R.id.graph);
```

```
graphView.removeAllSeries();
DataPoint[] points = new DataPoint[96];
for (int i = 0; i < points.length; i++) {
   //points[i] = new DataPoint(frequency[i], DataY[i]);
   points[i] = new DataPoint(DataX[i], DataY[i]);
}
LineGraphSeries<DataPoint> series = new LineGraphSeries<>(
   points);
graphView.getGridLabelRenderer().setHorizontalLabelsColor(
   Color.WHITE);
graphView.getGridLabelRenderer().setVerticalLabelsColor(
   Color.WHITE);
graphView.getGridLabelRenderer().setVerticalAxisTitle("
   Impedance␣(Îl')");
graphView.getGridLabelRenderer().setHorizontalAxisTitle("
   Frequency␣(Hz)");
graphView.getGridLabelRenderer().setVerticalAxisTitleColor(
   Color.WHITE);
graphView.getGridLabelRenderer().
   setHorizontalAxisTitleColor(Color.WHITE);
graphView.getGridLabelRenderer().setGridColor(Color.WHITE);
graphView.getViewport().setScalable(false);
graphView.getViewport().setScalableY(false);
graphView.getViewport().setScrollableY(false);
graphView.getViewport().setMaxX(96);
graphView.getViewport().setMinX(0);
graphView.getLegendRenderer().setVisible(true);
```

```
    graphView.getLegendRenderer().setFixedPosition(0,0);
    graphView.getLegendRenderer().setTextColor(Color.WHITE);
    graphView.getLegendRenderer().setWidth(120);
    ButtonInit();
}
// Initialize buttons
void ButtonInit() {
    Capacitance = (Button) findViewById(R.id.Capacitance);
    Capacitance.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (mConnected) {
                if(BusyDevice==false) {
                    graphView.getGridLabelRenderer().
                        setVerticalAxisTitle("Current␣(uA)");
                    graphView.getGridLabelRenderer().
                        setHorizontalAxisTitle("Voltage␣(mV)");
                    MeasureStatus="Capacitance";
                    Cmd = "Capacitance";
                    mID1.setText("I:␣");
                    mID2.setText("V:␣");
                    mID3.setText("---");
                    mReadX.setText("---");
                    mReadY.setText("---");
                    mReadFreq.setText("---");
                    Toast.makeText(getApplicationContext(), "CV␣
                        startedïijĄ", Toast.LENGTH_SHORT).show();
```

```
                mStatus.setText("busy");
                mStatus.setTextColor(getResources().getColor(R.
                    color.red));
                CVmode = true;
                EISmode = false;
                SSCV = true;
                FSCV = false;
                BusyDevice=true;
                DataYCV = new float[1024];
                send("V");
                Log.d(TAG, "CV␣mode␣started");
            }
            else
                Toast.makeText(getApplicationContext(), "Device
                    ␣is␣busy", Toast.LENGTH_SHORT).show();
        }
        else
            Toast.makeText(getApplicationContext(), "Device␣
                not␣found", Toast.LENGTH_SHORT).show();
    }
});

CVHigh = (Button) findViewById(R.id.CVHigh);
CVHigh.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mConnected) {
```

```
if(BusyDevice==false) {
   graphView.getGridLabelRenderer().
      setVerticalAxisTitle("Current␣(uA)");
   graphView.getGridLabelRenderer().
      setHorizontalAxisTitle("Voltage␣(mV)");
   MeasureStatus="Capacitance";
   Cmd = "Capacitance";
   mID1.setText("I:␣");
   mID2.setText("V:␣");
   mID3.setText("---");
   mReadX.setText("---");
   mReadY.setText("---");
   mReadFreq.setText("---");
   Toast.makeText(getApplicationContext(), "FSCV␣
      startedïijĄ", Toast.LENGTH_SHORT).show();
   mStatus.setText("busy");
   mStatus.setTextColor(getResources().getColor(R.
      color.red));
   CVmode = true;
   EISmode = false;
   FSCV = true;
   SSCV = false;
   BusyDevice=true;
   DataYCV = new float[1024];
   send("F");
   Log.d(TAG, "FSCV␣mode␣started");
}
```

```
            else
                Toast.makeText(getApplicationContext(), "Device
                    ␣is␣busy", Toast.LENGTH_SHORT).show();
        }
        else
            Toast.makeText(getApplicationContext(), "Device␣
                not␣found", Toast.LENGTH_SHORT).show();
    }
});
Impedance = (Button) findViewById(R.id.Impedance);
Impedance.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mConnected) {
            if(BusyDevice==false) {
                ShowPop(v);
            }
            else
                Toast.makeText(getApplicationContext(), "Device
                    ␣is␣busy", Toast.LENGTH_SHORT).show();
        }
        else
            Toast.makeText(getApplicationContext(), "Device␣
                not␣found", Toast.LENGTH_SHORT).show();

    }
});
```

```
RefreshCV = (Button) findViewById(R.id.RefreshCV);
RefreshCV.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      graphView.getGridLabelRenderer().setVerticalAxisTitle
         ("Current␣(uA)");
      graphView.getGridLabelRenderer().
         setHorizontalAxisTitle("Voltage␣(mV)");
      Toast.makeText(getApplicationContext(), "Refresh␣
         CVïijĄ", Toast.LENGTH_SHORT).show();
      mID1.setText("I:␣");
      mID2.setText("V:␣");
      mID3.setText("---");
      mStatus.setTextColor(getResources().getColor(R.color.
         green));
      mStatus.setText("Stand␣by");
      plotCV(DataYCV, DataXCV);
   }
});
Disconnected = (ToggleButton)findViewById(R.id.Disconnected
   );
Disconnected.setOnClickListener(new View.OnClickListener()
   {
   @Override
   public void onClick(View v) {
      if(BusyDevice==false) {
```

```
if (Disconnected.isChecked() && mBluetoothLe !=
   null) {
   initialize();
   mConnected = true;
   Log.i(TAG, "Connected");
   updateConnectionState(getString(R.string.
      connected));
   startMonitoringRssiValue();
   Toast.makeText(getApplicationContext(), "
      Connected!", Toast.LENGTH_SHORT).show();
} else {
   if (mBluetoothLe != null) {
      if (mBluetoothGatt != null) {
         mBluetoothGatt.close();
         mRssi.setText("---");
         mConnected = false;
         updateConnectionState(getString(R.string.
            disconnected));
         stopMonitoringRssiValue();
         invalidateOptionsMenu();
         Toast.makeText(getApplicationContext(), "
            Disconnected!", Toast.LENGTH_SHORT).
            show();
         mCon.setText("Disconnected");
         mCon.setTextColor(getResources().getColor
            (R.color.red));
      }
```

```
            }
        }
    }else{
        Toast.makeText(getApplicationContext(), "Device␣is
            ␣busy", Toast.LENGTH_SHORT).show();
    }
  }
});
NyquistPlot = (ToggleButton)findViewById(R.id.PlotData);
NyquistPlot.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
    if(NyquistPlot.isChecked()){
        mID1.setText("Z:␣");
        mID2.setText("Îÿ:␣");
        mID3.setText("Freq:␣");
        plotEIS(DataY, DataX);
        Toast.makeText(getApplicationContext(), "Show␣Bode
            ␣plot", Toast.LENGTH_SHORT).show();
    }else{
        mID1.setText("Imag:␣");
        mID2.setText("Real:␣");
        mID3.setText("Freq:␣");
        PlotNyquist(DataY, DataX);
            Toast.makeText(getApplicationContext(), "Show␣
                Nyquist␣plot", Toast.LENGTH_SHORT).show();
    }
```

```
        }
    });
    PlotScale = (ToggleButton)findViewById(R.id.PlotDataLog);
    PlotScale.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(PlotScale.isChecked()){
                mID1.setText("Z:␣");
                mID2.setText("Îÿ:␣");
                mID3.setText("Freq:␣");

                plotEIS(DataY, DataX);
                Toast.makeText(getApplicationContext(), "Linear␣
                    Scale", Toast.LENGTH_SHORT).show();
            }else{
                mID1.setText("Z:␣");
                mID2.setText("Îÿ:␣");
                mID3.setText("Freq:␣");

                plotEISLog(DataY, DataX);
                Toast.makeText(getApplicationContext(), "Logscale"
                    , Toast.LENGTH_SHORT).show();
            }
        }
    });
 }
// Set up different plot
```

```
void plotEIS(float dataY1[],float dataY2[]){
   graphView.removeAllSeries();
   GraphView graph = (GraphView) findViewById(R.id.graph);
   graph.getSecondScale().removeAllSeries();
   graph.removeAllSeries();
   DataPoint[] points1 = new DataPoint[96];
   DataPoint[] points2 = new DataPoint[96];
   float[] TempData2=new float[96];
   float[] TempData1=new float[96];
   if(MeasureStatus.equals("Impedance"))
   {
      for (int i = 0; i < points1.length; i++) {
         points1[i] = new DataPoint(frequency2[i], dataY1[i]);
         points2[i] = new DataPoint(frequency2[i], dataY2[i]);
         TempData1[i] = dataY1[i];
         TempData2[i] = dataY2[i];
      }

      Arrays.sort(TempData1);
      Arrays.sort(TempData2);
      LineGraphSeries<DataPoint> series1 = new LineGraphSeries
         <>(points1);
      LineGraphSeries<DataPoint> series2 = new LineGraphSeries
         <>(points2);
      graphView.getGridLabelRenderer().setVerticalAxisTitle("
         Impedance␣(Îl')");
```

```
graphView.getGridLabelRenderer().setHorizontalAxisTitle(
   "Frequency␣(Hz)");
graph.getViewport().setMaxX(frequency2[95]);
graph.getViewport().setMinX(frequency2[0]);
graph.getViewport().setMaxY(TempData1[TempData1.length
   -1]);
graph.getViewport().setMinY(TempData1[0]);
graph.getSecondScale().setMinY(TempData2[0]);
graph.getSecondScale().setMaxY(TempData2[TempData1.
   length-1]);
graph.getViewport().setScalable(true);
graph.getViewport().setScrollable(true);
graph.getViewport().setScalableY(true);
graph.getViewport().setScrollableY(true);
series2.setColor(Color.RED);
graph.getGridLabelRenderer().
   setVerticalLabelsSecondScaleColor(Color.GREEN);
series1.setDrawDataPoints(true);
series1.setDataPointsRadius(8);
series2.setDataPointsRadius(8);
series1.setColor(Color.parseColor("red"));
series1.setTitle("Imp␣(Îl')");
series2.setTitle("Pha␣(degree)");
graph.addSeries(series1);
graph.getSecondScale().removeAllSeries();
graph.getSecondScale().addSeries(series2);
series2.setColor(Color.GREEN);
```

```
series1.setOnDataPointTapListener(new
   OnDataPointTapListener() {
   @Override
   public void onTap(Series series, DataPointInterface
      dataPoint) {
      dataPoint.getX();
      mReadFreq.setText(getSignificant(dataPoint.getX()
         ,5)+"␣Hz");
      mReadY.setText(getSignificant(dataPoint.getY(),5)+
         "␣Îl'");
      if(find(frequency2,(float)dataPoint.getX()) != -1)
         {
         mReadX.setText(getSignificant(DataX[find(
            frequency2,(float)dataPoint.getX())],5)+"␣Âř
            ");
      }
      else
         mReadX.setText("----");
   }
});
series2.setOnDataPointTapListener(new
   OnDataPointTapListener() {
   @Override
   public void onTap(Series series, DataPointInterface
      dataPoint) {
      dataPoint.getX();
```

```
            mReadFreq.setText(getSignificant(dataPoint.getX()
               ,5)+"␣Hz");
            mReadX.setText(getSignificant(dataPoint.getY(),5)+
               "␣Âř");
            if(find(frequency2,(float)dataPoint.getX()) != -1)
               {
               mReadY.setText(getSignificant(DataY[find(
                  frequency2,(float)dataPoint.getX())],5)+"␣
                  Îl'");
            }
            else
               mReadY.setText("----");
         }
      });
      Log.d(TAG, "plot␣started");
   }

}
void plotCV(float data[],float dataX[]){
   graphView.removeAllSeries();
   GraphView graph = (GraphView) findViewById(R.id.graph);
   graph.removeAllSeries();
   graph.getSecondScale().removeAllSeries();
   graph.getGridLabelRenderer().
      setVerticalLabelsSecondScaleColor(Color.BLACK);
   float[] TempData2=new float[1024];
   float[] TempData1=new float[1024];
```

```
DataPoint[] points = new DataPoint[data.length];
if(MeasureStatus.equals("Capacitance"))
{
   for (int i = 0; i < points.length; i++) {
      points[i] = new DataPoint(dataX[i], data[i]);
      TempData1[i] = data[i];
      TempData2[i] = dataX[i];
   }
   Arrays.sort(TempData1);
   Arrays.sort(TempData2);
   PointsGraphSeries<DataPoint> series = new
      PointsGraphSeries<>(points);
   graph.getViewport().setMaxX(1100);
   graph.getViewport().setMinX(-1100);
   graph.getViewport().setMaxY(TempData1[TempData1.length
      -1]);
   graph.getViewport().setMinY(TempData1[0]);
   graphView.getGridLabelRenderer().setVerticalAxisTitle("
      Current␣(uA)");
   graphView.getGridLabelRenderer().setHorizontalAxisTitle(
      "Voltage␣(mV)");
   graph.getViewport().setScalable(true);
   graph.getViewport().setScrollable(true);
   graph.getViewport().setScalableY(true);
   graph.getViewport().setScrollableY(true);
   series.setColor(Color.parseColor("red"));
   if(FSCV==true && SSCV == false) {
```

```
            series.setTitle("100␣V/s␣CV");
        }else if(FSCV==false && SSCV == true)
        {
            series.setTitle("100␣mV/s␣CV");
        }
        series.setSize(2);
        graph.addSeries(series);
        series.setColor(Color.RED);
        series.setOnDataPointTapListener(new
            OnDataPointTapListener() {
            @Override
            public void onTap(Series series, DataPointInterface
                dataPoint) {
                                dataPoint.getX();
                mReadX.setText(getSignificant(dataPoint.getX(),5)+
                    "␣mV");
                mReadY.setText(getSignificant(dataPoint.getY(),5)+
                    "␣uA");
            }
        });
        Log.d(TAG, "plot␣started");
    }
}
@Override
protected void onPause() {
    super.onPause();
}
```

```
@Override
public void onResume() {
   super.onResume(); // Always call the superclass method
      first
}
@Override
protected void onDestroy() {
   stopMonitoringRssiValue();
   mBluetoothLe.disconnect(mBluetoothGatt);
   mConnected = false;
   super.onDestroy();
}
//Initialize Bluetooth
private void initialize() {
   mBluetoothManager = (BluetoothManager) getSystemService(
      Context.BLUETOOTH_SERVICE);
   mBluetoothDevice = mBluetoothManager.getAdapter().
      getRemoteDevice(mDeviceAddress);
   mBluetoothLe = new BluetoothLe(this, mBluetoothManager,
      this);
   mBluetoothGatt = mBluetoothLe.connect(mBluetoothDevice,
      false);
}
private void updateConnectionState(final String status) {
   runOnUiThread(new Runnable() {
      @Override
      public void run() {
```

```
            if(status.equals(getString(R.string.connected))) {
               Toast.makeText(getApplicationContext(), "Connected
                  ␣to:␣"+mDeviceName, Toast.LENGTH_SHORT).show();
               mCon.setText("Connected");
               mCon.setTextColor(getResources().getColor(R.color.
                  green));
            } else if (status.equals(getString(R.string.
               disconnected))) {
               Toast.makeText(getApplicationContext(), "Device␣
                  Disconnected!", Toast.LENGTH_SHORT).show();
               mCon.setText("DisConnected");
               mCon.setTextColor(getResources().getColor(R.color.
                  red));
            }
         }
      });
   }
   public void startMonitoringRssiValue() {
      readPeriodicallyRssiValue(true);
   }
   public void stopMonitoringRssiValue() {
      readPeriodicallyRssiValue(false);
   }
   public void readPeriodicallyRssiValue(final boolean repeat) {
      mTimerEnabled = repeat;
      // check if we should stop checking RSSI value
```

```
    if(!mConnected || mBluetoothGatt == null || !mTimerEnabled)
        {
       mTimerEnabled = false;
       return;
    }
    mTimerHandler.postDelayed(new Runnable() {
       @Override
       public void run() {
          if(mBluetoothGatt == null ||
                !mConnected)
          {
             mTimerEnabled = false;
             return;
          }

          // request RSSI value
          mBluetoothGatt.readRemoteRssi();
          // add call it once more in the future
          readPeriodicallyRssiValue(mTimerEnabled);
       }
    }, RSSI_UPDATE_TIME_INTERVAL);
}
 // Setup BluetoothListener methods
@Override
public void onServicesDiscovered(BluetoothGatt gatt, int
   status) {
   Log.i(TAG, "onServicesDiscovered");
```

```
    List<BluetoothGattCharacteristic> characteristics;
    if(status == BluetoothGatt.GATT_SUCCESS) {
       for (BluetoothGattService service : gatt.getServices())
          {
          if ((service == null) || (service.getUuid()==null)) {
             continue;
          }
          if(AppConstant.SERVICE_EMG_SIGNAL.equals(service.
             getUuid())) {
             //Set notification for EMG signal:
             mBluetoothLe.setCharacteristicNotification(gatt,
                service.getCharacteristic(AppConstant.
                CHAR_EMG_SIGNAL), true);
             characteristics = service.getCharacteristics();

          }

       }
    }

}
@Override
public void onReadRemoteRssi(BluetoothGatt gatt, int rssi, int
    status) {
    uiRssiUpdate(rssi);
}
@Override
```

```
public void onConnectionStateChange(BluetoothGatt gatt, int
   status, int newState) {
   switch (newState) {
      case BluetoothProfile.STATE_CONNECTED:
         mConnected = true;
         Log.i(TAG, "Connected");
         updateConnectionState(getString(R.string.connected));
         invalidateOptionsMenu();
         //Start the service discovery:
         gatt.discoverServices();
         startMonitoringRssiValue();
         break;
      case BluetoothProfile.STATE_DISCONNECTED:
         mConnected = false;
         Log.i(TAG, "Disconnected");
         updateConnectionState(getString(R.string.disconnected
            ));
         stopMonitoringRssiValue();
         invalidateOptionsMenu();
         break;
      default:
         break;
   }
}
@Override
public void onCharacteristicChanged(BluetoothGatt gatt,
   BluetoothGattCharacteristic characteristic) {
```

```
//Log.i(TAG, "onCharacteristicChanged");
if(AppConstant.CHAR_EMG_SIGNAL.equals(characteristic.
   getUuid())) {
   if(EISmode==true && CVmode==false) {
   /*TODO: SHOULD THIS BE SIGNED INT*/
      final byte[] data = characteristic.getValue();
      if (data != null && data.length > 0) {
         final StringBuilder stringBuilder = new
            StringBuilder(data.length);
         for (byte byteChar : data)
            stringBuilder.append(String.format("%02X␣",
               byteChar));
         final String ShowData = new String(data);
         if (isValidFloat(ShowData)) {
            final float f = Float.parseFloat(ShowData);
            Log.d(TAG, String.format("IndexY:␣%d", Yindex))
               ;
            Log.d(TAG, String.format("IndexX:␣%d", Xindex))
               ;
         /*To record data*/
            if (index <= 96 * 2 - 1) {
               if (index % 2 == 0) {
                  DataY[Yindex] = f;
                  Log.d(TAG, String.format("Ydata=␣%f",
                     DataY[Yindex]));
                  Yindex = Yindex + 1;
               } else if (index % 2 == 1) {
```

```
                DataX[Xindex] = f;
                Log.d(TAG, String.format("Xdata=␣%f",
                    DataX[Xindex]));
                updateEISstatus(DataY[Xindex],DataX[
                    Xindex],Xindex);
                Xindex = Xindex + 1;
            }
            index = index + 1;
            Log.d(TAG, String.format("Tracting␣plot␣
                Index:␣%d", index));
            if (Xindex == 96 && Yindex == 96 && index ==
                 96 * 2) {
                Xindex = 0;
                Yindex = 0;
                index = 0;
                saveTextAsFileEIS(DataY,DataX);
                plotEIS(DataY, DataX);
                updateEMGState(f);
                BusyDevice=false;
                Notify();
                Log.d(TAG, String.format("EIS␣Plot␣
                    successful"));
            }
        }
    } else {
        Log.d(TAG, String.format("Received␣heart␣rate␣
            string␣%s:", ShowData));
```

```
            if (ShowData.equals("start"))
                Log.d(TAG, String.format("Matched␣%s",
                    ShowData));
                index = 0;
            }
        }
    }
    else if(CVmode==true && EISmode==false){
        final byte[] data = characteristic.getValue();
        if (data != null && data.length > 0) {
            final StringBuilder stringBuilder = new
                StringBuilder(data.length);
            for (byte byteChar : data)
                stringBuilder.append(String.format("%02X␣",
                    byteChar));
            final String ShowData = new String(data);
            if (isValidFloat(ShowData)) {
                final float f = Float.parseFloat(ShowData);
                Log.d(TAG, String.format("IndexY:␣%d", Yindex))
                    ;
                Log.d(TAG, String.format("IndexX:␣%d", Xindex))
                    ;
            /*To record data*/
                if (index < 1024) {
                    DataYCV[index] = f;
                    Log.d(TAG, String.format("Ydata=␣%f", DataY[
                        Yindex]));
```

```
                index = index + 1;
                Log.d(TAG, String.format("Tracting␣plot␣
                   Index:␣%d", index));
                ;
             }
                if (index ==1024) {
                   index = 0;
                   saveTextAsFileCV(DataXCV,DataYCV);
                   plotCV(DataYCV, DataXCV);
                   updateEMGState(f);
                   BusyDevice=false;
                   Notify();
                   Log.d(TAG, String.format("CV␣Plot␣
                      successful"));
                }
          } else {
             Log.d(TAG, String.format("Received␣heart␣rate␣
                string␣%s:", ShowData));
             if (ShowData.equals("start"))
                Log.d(TAG, String.format("Matched␣%s",
                   ShowData));
             index = 0;
          }
       }
    }
```

```
    }

}
private void updateEMGState(final float value) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStatus.setTextColor(getResources().getColor(R.color.
                green));
            mStatus.setText("Stand␣by");
        }
    });
}

private void updateEISstatus(final float Imp,final float phase
    ,final int num) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mStatus.setTextColor(getResources().getColor(R.color.
                green));
            mReadFreq.setText(getSignificant(frequency2[num],3)+"
                ␣Hz");
            mReadY.setText(getSignificant(Imp,5)+"␣Îl’");
            mReadX.setText(getSignificant(phase,5)+"␣Âř");
            mStatus.setText(Integer.toString(num+1)+"/96" );
        }
```

```
    });
}
@Override
public void onCharacteristicRead(BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic, int status) {
    Log.i(TAG, "onCharacteristicRead");
}
@Override
public void onCharacteristicWrite(BluetoothGatt gatt,
    BluetoothGattCharacteristic characteristic, int status) {
    Log.i(TAG, "onCharacteristicWrite␣::␣Status::␣" + status);
}
@Override
public void onDescriptorRead(BluetoothGatt gatt,
    BluetoothGattDescriptor descriptor, int status) {
}
@Override
public void onDescriptorWrite(BluetoothGatt gatt,
    BluetoothGattDescriptor descriptor, int status) {
    Log.i(TAG, "onDescriptorRead␣::␣Status::␣" + status);
}
@Override
public void onError(String errorMessage) {
    Log.e(TAG, "Error::␣" + errorMessage);
}
private void uiRssiUpdate(final int rssi) {
    runOnUiThread(new Runnable() {
```

```
        @Override
        public void run() {
            final String valueOfRSSI = String.valueOf(rssi)+"␣dB"
                ;
            mRssi.setText(valueOfRSSI);
            if(mConnected) {
                String newStatus = "Status:␣" + getString(R.string
                    .connected);
            } else {
                String newStatus = "Status:␣" + getString(R.string
                    .disconnected);
            }
        }
    });
}
private void uiReadUpdate(final String read) {

    runOnUiThread(new Runnable() {
        @Override
        public void run() {
        }
    });
}

private void uiReadUpdateCV(final String read) {
    runOnUiThread(new Runnable() {
        @Override
```

```
        public void run() {
        }
    });
}
public static Boolean isValidFloat(String value) {
    try {
        float temp = Float.parseFloat(value);
        return true;
    } catch (NumberFormatException e) {
        return false;
    }
}
// Write data functions
public void send(String sendVar) {
    strBLE=sendVar;
    Log.d(TAG, "Sending result=" + strBLE);
    final byte[] tx = strBLE.getBytes();
    Log.d(TAG, "Sending result=" + tx);
        if (mConnected) {
            BluetoothGattService mCustomService = mBluetoothGatt.
                getService(AppConstant.SERVICE_EMG_SIGNAL);
            BluetoothGattCharacteristic TXCharacteristic =
                mCustomService.getCharacteristic(AppConstant.
                CHAR_EMG_SIGNAL);
            mBluetoothLe.writeCharacteristic(mBluetoothGatt,
                TXCharacteristic,tx);
        }
```

```
}
// Read data functions
// Write data functions
public void GenerateCVdataX(){
   int i_up=BeginCV;
   int i_down=EndCV;
   while(i_up<EndCV+1){
      DataXCV[CountCV]=(i_up-2048)*2.2f/4.096f-13.5f;
      i_up+=StepCV;
      CountCV+=1;
      }
   while(i_down>BeginCV+1){
      DataXCV[CountCV]=(i_down-2048)*2.2f/4.096f-9;
      i_down-=StepCV;
      CountCV+=1;
   }
}

private void PlotNyquist(float[] dataY1,float[] dataY2){
   //GraphView graph = (GraphView) findViewById(R.id.graph);
   graphView.removeAllSeries();
   GraphView graph = (GraphView) findViewById(R.id.graph);
   graph.removeAllSeries();
   graph.getSecondScale().removeAllSeries();
   graph.getGridLabelRenderer().
      setVerticalLabelsSecondScaleColor(Color.BLACK);
   final float[] Real=new float[96];
```

```
final float[] Imag=new float[96];
float[] TempData2=new float[96];
float[] TempData1=new float[96];
for(int i = 0; i < 96; ++i){
   Real[i]=dataY1[i]*(float)Math.cos(Math.toRadians(dataY2[
      i]));
   Imag[i]=(-1)*dataY1[i]*(float)Math.sin(Math.toRadians(
      dataY2[i]));
   TempData2[i]=Imag[i];
   TempData1[i]=Real[i];
}
   DataPoint[] points1 = new DataPoint[96];
if(MeasureStatus.equals("Impedance"))
{
   for (int i = 0; i < points1.length; i++) {
      points1[i] = new DataPoint(Real[i], Imag[i]);
   }
   Arrays.sort(TempData1);
   Arrays.sort(TempData2);
   PointsGraphSeries<DataPoint> series = new
      PointsGraphSeries<>(points1);
   graph.getViewport().setMaxX(TempData1[TempData1.length
      -1]);
   graph.getViewport().setMinX(TempData1[0]);
   graph.getViewport().setMaxY(TempData2[TempData2.length
      -1]);
   graph.getViewport().setMinY(TempData2[0]);
```

```
graph.getViewport().setScalable(true);
graph.getViewport().setScalableY(true);
series.setSize(4);
series.setColor(Color.parseColor("red"));
series.setTitle("Nyquist");
graph.getViewport().setScalable(false);
graph.getViewport().setScrollable(false);
graph.getViewport().setScalableY(false);
graph.getViewport().setScrollableY(false);
graph.getGridLabelRenderer().setVerticalAxisTitle("-Imag
   ␣(Îl')");
graph.getGridLabelRenderer().setHorizontalAxisTitle("
   Real␣(Îl')");
graph.addSeries(series);
series.setColor(Color.YELLOW);
series.setOnDataPointTapListener(new
   OnDataPointTapListener() {
   @Override
   public void onTap(Series series, DataPointInterface
      dataPoint) {
      dataPoint.getX();
      mReadX.setText(getSignificant(dataPoint.getX(),5)+
         "␣Îl'");
      mReadY.setText("-"+getSignificant(dataPoint.getY()
         ,5)+"␣Îl'");
      if(find(Real,(float)dataPoint.getX()) != -1){
```

```
                mReadFreq.setText(getSignificant(frequency2[
                    find(Real,(float)dataPoint.getX())],5)+"␣Hz"
                    );
            }
            else
                mReadFreq.setText("----");
        }
    });

    Log.d(TAG, "plot␣started");
  }
}
// Set up different plot
void plotEISLog(float dataY1[],float dataY2[]){
  GraphView graph = (GraphView) findViewById(R.id.graph);
  graph.removeAllSeries();
  graph.getSecondScale().removeAllSeries();
  DataPoint[] points1 = new DataPoint[96];
  DataPoint[] points2 = new DataPoint[96];
  float[] TempData2=new float[96];
  float[] TempData1=new float[96];
  if(MeasureStatus.equals("Impedance"))
  {
    for (int i = 0; i < points1.length; i++) {
      points1[i] = new DataPoint(Math.log10(frequency2[i]),
          dataY1[i]);
```

```
    points2[i] = new DataPoint(Math.log10(frequency2[i]),
        dataY2[i]);
    TempData1[i] = dataY1[i];
    TempData2[i] = dataY2[i];
}
Arrays.sort(TempData1);
Arrays.sort(TempData2);
LineGraphSeries<DataPoint> series1 = new LineGraphSeries
    <>(points1);
LineGraphSeries<DataPoint> series2 = new LineGraphSeries
    <>(points2);
graph.getViewport().setMaxX(4.2);
graph.getViewport().setMinX(-0.8);
graph.getViewport().setMaxY(TempData1[TempData1.length
    -1]);
graph.getViewport().setMinY(TempData1[0]);
graph.getSecondScale().setMinY(TempData2[0]);
graph.getSecondScale().setMaxY(TempData2[TempData2.
    length-1]);
graph.getGridLabelRenderer().setVerticalAxisTitle("
    Impedance␣(Îl')");
graph.getGridLabelRenderer().setHorizontalAxisTitle("
    Frequency␣(lg␣Hz)");
graph.getViewport().setScalable(true);
graph.getViewport().setScrollable(true);
graph.getViewport().setScalableY(true);
graph.getViewport().setScrollableY(true);
```

```
series2.setColor(Color.RED);
graph.getGridLabelRenderer().
   setVerticalLabelsSecondScaleColor(Color.GREEN);
series1.setDrawDataPoints(true);
series1.setDataPointsRadius(8);
series2.setDataPointsRadius(8);
series1.setColor(Color.parseColor("red"));
series1.setTitle("Imp␣(Îl')");
series2.setTitle("Pha␣(degree)");
graph.addSeries(series1);
graph.getSecondScale().addSeries(series2);
series2.setColor(Color.GREEN);
series1.setOnDataPointTapListener(new
   OnDataPointTapListener() {
   @Override
   public void onTap(Series series, DataPointInterface
      dataPoint) {
      dataPoint.getX();
      mReadFreq.setText(getSignificant(Math.pow(10,
         dataPoint.getX()),4)+"␣Hz");
      mReadY.setText(getSignificant(dataPoint.getY(),5)+
         "␣Îl'");
      if(find(frequency2,(float)Math.pow(10,dataPoint.
         getX())) != -1){
         mReadX.setText(getSignificant(DataX[find(
            frequency2,(float)Math.pow(10,dataPoint.getX
            ()))],5)+"␣Âř");
```

```
        }
        else
            mReadX.setText("----");
    }
});
series2.setOnDataPointTapListener(new
   OnDataPointTapListener() {
   @Override
   public void onTap(Series series, DataPointInterface
      dataPoint) {
      dataPoint.getX();
      mReadFreq.setText(getSignificant(Math.pow(10,
         dataPoint.getX()),4)+"␣Hz");
      mReadX.setText(getSignificant(dataPoint.getY(),5)+
         "␣Âř");
      if(find(frequency2,(float)Math.pow(10,dataPoint.
         getX())) != -1){
         mReadY.setText(getSignificant(DataY[find(
            frequency2,(float)Math.pow(10,dataPoint.getX
            ()))],5)+"␣Îl’");
      }
      else
         mReadY.setText("----");

   }
});
Log.d(TAG, "plot␣started");
```

```
    }
}
public static String getSignificant(double value, int sigFigs)
    {
    MathContext mc = new MathContext(sigFigs, RoundingMode.DOWN
       );
    BigDecimal bigDecimal = new BigDecimal(value, mc);
    return bigDecimal.toPlainString();
}
public static int find(float[] a,float target){
    for (int i = 0; i<a.length;i++)
       if(a[i]==target)
          return i;
    return -1;
}
private void saveTextAsFileCV(float[] Voltage,float[] Current)
    {
    SimpleDateFormat sdf = new SimpleDateFormat("
       yyyyMMdd_HHmmss");
    String currentDateandTime = sdf.format(new Date());

    SimpleDateFormat sdf2 = new SimpleDateFormat("yyyyMMdd");
    String CurrentDate = sdf2.format(new Date());
    String fileName;

    if(FSCV== false && SSCV== true) {
       fileName = "SSCV_"+currentDateandTime + ".txt";
```

```
}else{
   fileName = "FSCV_"+currentDateandTime + ".txt";
}
File Root = Environment.getExternalStorageDirectory();
File Dir = new File(Root.getAbsolutePath()+"/
   ElectrochemistryData"+"/"+CurrentDate);
if(!Dir.exists())
{
   Dir.mkdirs();
   Log.d(TAG, "Create␣the␣DIR");
}
File file = new File(Dir,fileName);
String Message1[] = new String[Voltage.length] ;
String Message2[] = new String[Current.length] ;
String temp;
for (int i=0;i<Voltage.length;++i){
   Message1[i] = Float.toString(Voltage[i]);
   Message2[i] = Float.toString(Current[i]);
}
//write file
try{
   FileOutputStream fos = new FileOutputStream(file);
   for (int i=0;i<Voltage.length;++i) {
      temp="Index:"+ i + "␣," +"Voltage:"+Message1[i]+"␣mV␣
         ,"+"Current"+Message2[i]+"␣uA␣;\n";
      fos.write(temp.getBytes());
   }
```

```
            fos.close();
        } catch(FileNotFoundException e){
            e.printStackTrace();
        } catch (IOException e){
            e.printStackTrace();
        }
    }
    private void saveTextAsFileEIS(float[] DataImp,float[] DataPha
        ){
        SimpleDateFormat sdf = new SimpleDateFormat("
            yyyyMMdd_HHmmss");
        String currentDateandTime = sdf.format(new Date());

        SimpleDateFormat sdf2 = new SimpleDateFormat("yyyyMMdd");
        String CurrentDate = sdf2.format(new Date());
        String fileName = "EIS_"+currentDateandTime + ".txt";
        //Create file
        File Root = Environment.getExternalStorageDirectory();
        File Dir = new File(Root.getAbsolutePath()+"/
            ElectrochemistryData"+"/"+CurrentDate);
        if(!Dir.exists())
        {
            Dir.mkdirs();
        }
        final float[] Real=new float[96];
        final float[] Imag=new float[96];
        float[] TempData2=new float[96];
```

```
float[] TempData1=new float[96];
for(int i = 0; i < 96; ++i){
   Real[i]=DataImp[i]*(float)Math.cos(Math.toRadians(
      DataPha[i]));
   Imag[i]=DataImp[i]*(float)Math.sin(Math.toRadians(
      DataPha[i]));
   TempData2[i]=Imag[i];
   TempData1[i]=Real[i];
}
File file = new File(Dir,fileName);
String Message1[] = new String[DataImp.length] ;
String Message2[] = new String[DataImp.length] ;
String Message3[] = new String[DataImp.length] ;
String Message4[] = new String[DataImp.length] ;
String Message5[] = new String[DataImp.length] ;
String temp;
for (int i=0;i<DataImp.length;++i){
   Message2[i] = Float.toString(DataImp[i]);
   Message3[i] = Float.toString(DataPha[i]);
   Message4[i] = Float.toString(Real[i]);
   Message5[i] = Float.toString(Imag[i]);
   Message1[i] = Float.toString(frequency2[i]);
}
//write file
try{
   FileOutputStream fos = new FileOutputStream(file);
   for (int i=0;i<DataImp.length;++i) {
```

```
            temp="Index:"+ i + "␣," +"Frequency:"+Message1[i]+"␣
               Hz␣,"+"Z:"+Message2[i]+"␣Ohms␣,"+"Phase:"+Message3
               [i]+"␣Degree␣,"+"Real:"+Message4[i]+"␣Ohms␣,"+"
               Imaginary:"+Message5[i]+"␣Ohms␣;\n";
            fos.write(temp.getBytes());
         }
         fos.close();
      } catch(FileNotFoundException e){
         e.printStackTrace();
      } catch (IOException e){
         e.printStackTrace();
      }
   }

   @Override
   public void onRequestPermissionsResult(int requestCode,
      @NonNull String[] permissions, @NonNull int[] grantResults)
      {
      switch(requestCode){
         case 1000:
            if(grantResults[0] == PackageManager.
               PERMISSION_GRANTED){
               Toast.makeText(this,"Permission␣granted!",Toast.
                  LENGTH_SHORT).show();
            }else {
               Toast.makeText(this,"Permission␣not␣granted!",
                  Toast.LENGTH_SHORT).show();
```

```
        }
    }
}

public void Notify(){
    notification = new NotificationCompat.Builder(this);
    notification.setAutoCancel(true);
    notification.setSmallIcon(R.drawable.raedy2);
    notification.setTicker("Data␣collected");
    notification.setWhen(System.currentTimeMillis());
    notification.setContentTitle("Dxotronics␣Tech");

    if( CVmode == false && EISmode == true){
       notification.setContentText("EIS␣process␣finished");
    }else if(CVmode == true && EISmode == false){
       if(FSCV== false && SSCV== true) {
          notification.setContentText("slow␣scan␣CV␣process␣
             finished");
       }else if(FSCV== true && SSCV== false){
          notification.setContentText("fast␣scan␣CV␣process␣
             finished");
       }
    }
    NotificationManager nm = (NotificationManager)
       getSystemService(NOTIFICATION_SERVICE);
    nm.notify(uniqueID,notification.build());
```

```
}
public void ShowPop(View v){
   TextView back,EISp2,EISp1,EIS0,EISn1,EISn2;
   Button Bback,BEISp2,BEISp1,BEIS0,BEISn1,BEISn2;
   myDialog.setContentView(R.layout.popeis);
   Bback = (Button) myDialog.findViewById(R.id.back);
   BEISp2 = (Button) myDialog.findViewById(R.id.EISp2);
   BEISp1 = (Button) myDialog.findViewById(R.id.EISp1);
   BEIS0 = (Button) myDialog.findViewById(R.id.EIS0);
   BEISn1 = (Button) myDialog.findViewById(R.id.EISn1);
   BEISn2 = (Button) myDialog.findViewById(R.id.EISn2);
   Bback.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         myDialog.dismiss();
      }
   });
   BEIS0.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
         if (mConnected) {
            if(BusyDevice==false) {
               graphView.getGridLabelRenderer().
                  setVerticalAxisTitle("Impedance␣(Îl')");
               graphView.getGridLabelRenderer().
                  setHorizontalAxisTitle("Frequency␣(Hz)");
```

```
            graphView.getGridLabelRenderer().
               setVerticalLabelsSecondScaleColor(Color.RED)
               ;
            MeasureStatus="Impedance";
            mID1.setText("Z:␣");
            mID2.setText("Îÿ:␣");
            mID3.setText("Freq:␣");
            mReadX.setText("---");
            mReadY.setText("---");
            mReadFreq.setText("---");
            CVmode=false;
            EISmode=true;
            FSCV = false;
            SSCV = false;
            BusyDevice=true;
            DataY= new float[96];
            DataX= new float[96];
            myDialog.dismiss();
            Toast.makeText(getApplicationContext(), "0V␣EIS
               ␣startedïijĄ", Toast.LENGTH_SHORT).show();
            mStatus.setText("busy");
            mStatus.setTextColor(getResources().getColor(R.
               color.red));
            send("E");
        }
        else
```

```
                Toast.makeText(getApplicationContext(), "Device
                    ␣is␣busy", Toast.LENGTH_SHORT).show();
            }
            else
                Toast.makeText(getApplicationContext(), "Device␣
                    not␣found", Toast.LENGTH_SHORT).show();
        }
    });
    BEISp2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (mConnected) {
                if(BusyDevice==false) {
                    graphView.getGridLabelRenderer().
                        setVerticalAxisTitle("Impedance␣(Îl')");
                    graphView.getGridLabelRenderer().
                        setHorizontalAxisTitle("Frequency␣(Hz)");
                    graphView.getGridLabelRenderer().
                        setVerticalLabelsSecondScaleColor(Color.RED)
                        ;
                    MeasureStatus="Impedance";
                    mID1.setText("Z:␣");
                    mID2.setText("Îÿ:␣");
                    mID3.setText("Freq:␣");
                    mReadX.setText("---");
                    mReadY.setText("---");
                    mReadFreq.setText("---");
```

```
                CVmode=false;
                EISmode=true;
                FSCV = false;
                SSCV = false;
                BusyDevice=true;
                DataY= new float[96];
                DataX= new float[96];
                myDialog.dismiss();
                Toast.makeText(getApplicationContext(), "+0.2␣V
                   ␣EIS␣startedïijĄ", Toast.LENGTH_SHORT).show
                   ();
                mStatus.setText("busy");
                mStatus.setTextColor(getResources().getColor(R.
                   color.red));
                send("A");
             }
             else
                Toast.makeText(getApplicationContext(), "Device
                   ␣is␣busy", Toast.LENGTH_SHORT).show();
          }
          else
             Toast.makeText(getApplicationContext(), "Device␣
                not␣found", Toast.LENGTH_SHORT).show();
       }
    });

    BEISp1.setOnClickListener(new View.OnClickListener() {
```

```
@Override
public void onClick(View v) {
   if (mConnected) {
      if(BusyDevice==false) {
         graphView.getGridLabelRenderer().
            setVerticalAxisTitle("Impedance␣(Îl')");
         graphView.getGridLabelRenderer().
            setHorizontalAxisTitle("Frequency␣(Hz)");
         graphView.getGridLabelRenderer().
            setVerticalLabelsSecondScaleColor(Color.RED)
            ;
         MeasureStatus="Impedance";
         mID1.setText("Z:␣");
         mID2.setText("Îÿ:␣");
         mID3.setText("Freq:␣");
         mReadX.setText("---");
         mReadY.setText("---");
         mReadFreq.setText("---");
         CVmode=false;
         EISmode=true;
         FSCV = false;
         SSCV = false;
         BusyDevice=true;
         DataY= new float[96];
         DataX= new float[96];
         myDialog.dismiss();
```

```
            Toast.makeText(getApplicationContext(), "+0.1␣V
               ␣EIS␣startedïijĄ", Toast.LENGTH_SHORT).show
               ();
            mStatus.setText("busy");
            mStatus.setTextColor(getResources().getColor(R.
               color.red));
            send("B");
         }
         else
            Toast.makeText(getApplicationContext(), "Device
               ␣is␣busy", Toast.LENGTH_SHORT).show();
      }
      else
         Toast.makeText(getApplicationContext(), "Device␣
            not␣found", Toast.LENGTH_SHORT).show();
   }
});
BEISn1.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {



      if (mConnected) {
         if(BusyDevice==false) {
            graphView.getGridLabelRenderer().
               setVerticalAxisTitle("Impedance␣(Îl')");
```

```
graphView.getGridLabelRenderer().
   setHorizontalAxisTitle("Frequency␣(Hz)");
graphView.getGridLabelRenderer().
   setVerticalLabelsSecondScaleColor(Color.RED)
   ;
MeasureStatus="Impedance";
mID1.setText("Z:␣");
mID2.setText("Îÿ:␣");
mID3.setText("Freq:␣");
mReadX.setText("---");
mReadY.setText("---");
mReadFreq.setText("---");
CVmode=false;
EISmode=true;
FSCV = false;
SSCV = false;
BusyDevice=true;
DataY= new float[96];
DataX= new float[96];
myDialog.dismiss();
Toast.makeText(getApplicationContext(), "-0.1␣V
   ␣EIS␣startedïijĄ", Toast.LENGTH_SHORT).show
   ();
mStatus.setText("busy");
mStatus.setTextColor(getResources().getColor(R.
   color.red));
send("C");
```

```
            }
            else
                Toast.makeText(getApplicationContext(), "Device
                    ␣is␣busy", Toast.LENGTH_SHORT).show();
        }
        else
            Toast.makeText(getApplicationContext(), "Device␣
                not␣found", Toast.LENGTH_SHORT).show();
    }
});
BEISn2.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mConnected) {
            if(BusyDevice==false) {
                graphView.getGridLabelRenderer().
                    setVerticalAxisTitle("Impedance␣(Îl')");
                graphView.getGridLabelRenderer().
                    setHorizontalAxisTitle("Frequency␣(Hz)");
                graphView.getGridLabelRenderer().
                    setVerticalLabelsSecondScaleColor(Color.RED)
                    ;
                MeasureStatus="Impedance";
                mID1.setText("Z:␣");
                mID2.setText("Îÿ:␣");
                mID3.setText("Freq:␣");
                mReadX.setText("---");
```

```
                mReadY.setText("---");
                mReadFreq.setText("---");
                CVmode=false;
                EISmode=true;
                FSCV = false;
                SSCV = false;
                BusyDevice=true;
                DataY= new float[96];
                DataX= new float[96];
                myDialog.dismiss();
                Toast.makeText(getApplicationContext(), "-0.2␣V
                   ␣EIS␣startedïijĄ", Toast.LENGTH_SHORT).show
                   ();
                mStatus.setText("busy");
                mStatus.setTextColor(getResources().getColor(R.
                   color.red));
                send("D");
            }
            else
                Toast.makeText(getApplicationContext(), "Device
                   ␣is␣busy", Toast.LENGTH_SHORT).show();
        }
        else
            Toast.makeText(getApplicationContext(), "Device␣
               not␣found", Toast.LENGTH_SHORT).show();
    }
});
```

```
      myDialog.show();
   }
   public void NormailizeFreq(){
      for (int i=0;i<32;++i){
         frequency2[i+63]=frequency2[63+i]*422/472;
      }
   }
}
```

Appendix C

C Code for MCU on Second Generation Device

```
#include <fix_fft.h>
#include <DueTimer.h>
#include <DueFlashStorage.h>
#include <efc.h>
#include <flash_efc.h>
#define Bluetooth Serial3
#include <SPI.h>
#define AveSize 1024
#define AveEIS 4
#define AveEISLow 3
#define AveEISHigh 4
#include <math.h>
#if defined(ARDUINO) && ARDUINO >= 100
  #include "Arduino.h"
  #else
  #include "WProgram.h"
  #endif
#define Samplepoints 256
#define MeasurePoints 32
#define WaitforMS 25
DueFlashStorage dueFlashStorage;
int DACin[Samplepoints];
const byte DATA[]={0,1,2,3,4,5,6,7,8,9,10,11};
const float pi = 3.1415926;
```

```
int Repeat=1;
int FirstFreq=0;
short im_in[AveEIS][Samplepoints];
short real_in[AveEIS][Samplepoints];
float real_in_Smooth[Samplepoints];
short real_in_mapto16Bit[AveEIS][Samplepoints];
short test[Samplepoints];
int val_in=0;
float mag_in[AveEIS][Samplepoints];
float phase_in[AveEIS][Samplepoints];
short im_out[AveEIS][Samplepoints];
short real_out[AveEIS][Samplepoints];
float real_out_Smooth[Samplepoints];
short real_out_mapto16Bit[AveEIS][Samplepoints];
int val_out=0;
float mag_out[AveEIS][Samplepoints];
float phase_out[AveEIS][Samplepoints];
int Vref;
float Rref
   []={50,500,5000,50000,500000,3000000,20000000,100000000};
int RrefMode[]={0,2,6,19,52,142,386,1023};
int DefaultRefMode=7;
float swing=0.10;
float Rtest;
float impedance[96];
float phase[96];
float RealPrint[96];
```

```
float ImagPrint[96];
float DataEIS[96*2]={};
float SineTone[]={2048,2048,2049,2049,2050,2051
,2051,2052,2052,2053,2054,2054,2055,2055,2056,
2056,2057,2058,2058,2059,2059,2060,2060,2061,
2061,2062,2062,2063,2063,2064,2064,2065,2065,
2065,2066,2066,2067,2067,2067,2068,2068,2068,
2069,2069,2069,2070,2070,2070,2070,2071,2071,2071,
2071,2071,2072,2072,2072,2072,2072,2072,2072,2072,
2072,2072,2072,2072,2072,2072,2072,2072,2072,2072,
2072,2072,2072,2071,2071,2071,2071,2071,2070,2070,
2070,2070,2069,2069,2069,2068,2068,2068,2067,2067,
2067,2066,2066,2065,2065,2065,2064,2064,2063,2063,
2062,2062,2061,2061,2060,2060,2059,2059,2058,2058,
2057,2056,2056,2055,2055,2054,2054,2053,2052,2052,
2051,2051,2050,2049,2049,2048,2048,2048,2047,2047,
2046,2045,2045,2044,2044,2043,2042,2042,2041,2041,
2040,2040,2039,2038,2038,2037,2037,2036,2036,2035,
2035,2034,2034,2033,2033,2032,2032,2031,2031,2031,
2030,2030,2029,2029,2029,2028,2028,2028,2027,2027,
2027,2026,2026,2026,2026,2025,2025,2025,2025,2025,
2024,2024,2024,2024,2024,2024,2024,2024,2024,2024,
2024,2024,2024,2024,2024,2024,2024,2024,2024,2024,
2024,2025,2025,2025,2025,2025,2026,2026,2026,2026,
2027,2027,2027,2028,2028,2028,2029,2029,2029,2030,
2030,2031,2031,2031,2032,2032,2033,2033,2034,2034,
2035,2035,2036,2036,2037,2037,2038,2038,2039,2040,
```

```
2040,2041,2041,2042,2042,2043,2044,2044,2045,2045,
2046,2047,2047,2048};
float SineToneHigh[32][256]=...; \\hiding to reduce content
double CapChoice;
double frequency[]={0.162,0.19,0.223,0.262,0.307,
0.362,0.424,0.498,0.583,0.685,0.803,0.942,1.1,1.3,
1.52,1.78,2.09,2.45,2.9,3.36,3.9,4.6,5.4,6.3,7.35,
8.58,10,11.7,13.6,15.8,18.4,21.3,24.6,28.4,32.7,
37.8,43.2,49.2,55.5,62.6,70.5,79.1,88,96.7,107.4,
113.7,117,120.5,124.5,128.7,133,137.75,142.79,148.2,
154.1,160.2,167.3,174.8,183,192,201.94,212.98,225.25,
239.6,471.92,707.88,943.84,1179.8,1415.8,1651.7,
1887.7,2123.6,2359.6,2595.6,2831.5,3067.5,3303.4,
3539.4,3775.4,4011.3,4247.3,4483.2,4719.2,4955.2,
5191.1,5427.1,5663,6135,6842.8,7786.7,8730.5,9674.4,
10854,12034,13450,15101};
float DelayTime[]={24000,20450,17424,14846,12650,
10778,9184,7825,6667,5681,4840,4124,3514,2994,2551,
2173,1852,1578,1344,1145,976,831,708,603,514,438,
373,317,270,230,196,167,142,121,103,87,74,63,54,
46,39,33,28,24,20,18,17,16,15,14,13,12,11,10,9,
8,7,6,5,4,3,2,1,0};
#define DACOUT DAC0
#define Vrefpin DAC1
#define TIAInput A0
#define TIAOutput A1
//Parameter setup for potentiometer
```

```
 const int csPinCWF = 10;
 const byte enableUpdateMSB = 0x1C; //B00011100
 const byte enableUpdateLSB = 0x02; //B00000010
 const byte command = 0x04; //B00000100
//Parameter for muxing
const int S0C = 12;
const int S1C = 32;
const int S2C = 33;
const int S0R = 37;
const int S1R = 38;
const int S2R = 39;
const int EN_C = 41;
const int EN_R = 40;
const int EN_DIFF = 45;
const int MODE = 44;
const int ENPW = 47;
int CapModetemp;
int capMode;
int WindowSize=9;
int EISSmoothEnable=0;
const int BeginCV=1;
const int EndCV=4095;
const int StepCV=8;
const int CycleCV=1;
float CVV[1024];
float CVI[1024];
float CVI_Smoothed[1024];
```

```
float real_in_CV[1024];
float real_out_CV[1024];
int EnhanceFactor=3;
int ReadOutWindow[AveSize];
float ReadOutSum;
float ReadOutAve;
int CommonMode=2048;
int Bias=0;
float Coefficient=10;
void setup() {
  Serial.begin(115200);
  Bluetooth.begin(115200);
//Setup for muxing capacitor
  pinMode(S0R,OUTPUT);
  pinMode(S1R,OUTPUT);
  pinMode(S2R,OUTPUT);
  pinMode(S0C,OUTPUT);
  pinMode(S1C,OUTPUT);
  pinMode(S2C,OUTPUT);
  pinMode(EN_C,OUTPUT);
  pinMode(EN_R,OUTPUT);
  pinMode(MODE,OUTPUT);
  pinMode(ENPW,OUTPUT);
  pinMode(EN_DIFF,OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
```

```
  selectR(0);
  selectC(0);
     //Normalize frequency
    for (int i=0;i<32;i++){
     frequency[63+i]=frequency[63+i]*422/472;
    }
    //Enhance input Signal
    for (int i=0;i<Samplepoints;++i){
     SineTone[i]=(SineTone[i]-2048)*EnhanceFactor+2048;
    }
    for (int j=0;j<32;++j){
     for (int i=0;i<Samplepoints;++i){
       SineToneHigh[j][i]=(SineToneHigh[j][i]-2048)*EnhanceFactor
          +2048;
    }
    }
    digitalWrite(MODE, HIGH);
    digitalWrite(ENPW, HIGH);
}
void loop() {
  analogWriteResolution(12);
  analogReadResolution(12);
  analogWrite(Vrefpin,3072);
  analogWrite(DACOUT,2048);
  selectC(6);
  selectR(1);
  real_out[0][0]=analogRead(TIAOutput);
```

```
real_in[0][0]=analogRead(TIAInput);
ReadOutSum=0;
for(int n =0;n<1024;++n){
  ReadOutWindow[n]=analogRead(TIAOutput);
  ReadOutSum+=ReadOutWindow[n];
}
ReadOutAve=ReadOutSum/1024;
if (Bluetooth.available()){
char inByte = Bluetooth.read();
  if (inByte=='V'){
    DefaultRefMode=7;
    selectR(DefaultRefMode);
    int count=0;
    int count2=0;
    int i_up=BeginCV;
    int i_down=EndCV;
    int sumtemp=0;
    int avetemp=0;
    for (int h=0;h<8;++h){
         ReadOutSum=0;
         selectR(h);
         CapModetemp=ChooseCapCV(h);
         delay(200);
      for(int l =0;l<AveSize;++l){
         ReadOutWindow[l]=analogRead(TIAOutput);
         ReadOutSum+=ReadOutWindow[l];
       }
```

```
      avetemp=ReadOutSum/AveSize;
      if(avetemp>(4096*0.1)){
      continue;
      }
      if(avetemp>(4096*0.1)){
        DefaultRefMode=h;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        adjust_com();
        delay(5000);
        break;
      }else{
        DefaultRefMode=h-1;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        adjust_com();
        delay(5000);
        break;
      }
    }
selectR(DefaultRefMode);
CapModetemp=ChooseCapCV(DefaultRefMode);
while(i_up<EndCV+1){
  analogWrite(DACOUT,i_up);
  ReadOutSum=0;
  delay(28);
  real_in_CV[count]=analogRead(TIAInput);
```

```
    for(int n =0;n<AveSize;++n){
       ReadOutWindow[n]=analogRead(TIAOutput);
       ReadOutSum+=ReadOutWindow[n];
      }
    ReadOutAve=ReadOutSum/AveSize;
    real_out_CV[count]=ReadOutAve;
    if(i_up>4096*0.1 &&i_up <4096*0.9){
    if(real_out_CV[count]>=4096*(1-2*swing)||real_out_CV[count
       ]<=4096*(swing) && DefaultRefMode>=1){
      i_up=BeginCV;
      DefaultRefMode=DefaultRefMode-1;
      count=0;
      analogWrite(DACOUT,2048);
      selectR(DefaultRefMode);
      CapModetemp=ChooseCapCV(DefaultRefMode);
      delay(100);
      continue;
      }
    }
    i_up=i_up+StepCV;
    CVV[count]=float(2048-real_in_CV[count])*2/4096*1000;
    CVI[count]=float(real_out_CV[count]-2048)*3.3/4096/Rref[
       DefaultRefMode]*1000000;
    count=count+1;
  }
int countUp=count;
while(i_down>BeginCV-1){
```

```
analogWrite(DACOUT,i_down);
ReadOutSum=0;
delay(28);
real_in_CV[count]=analogRead(TIAInput);
for(int n =0;n<AveSize;++n){
   ReadOutWindow[n]=analogRead(TIAOutput);
   ReadOutSum+=ReadOutWindow[n];
  }
ReadOutAve=ReadOutSum/AveSize;
real_out_CV[count]=ReadOutAve;
if(i_down>4096*0.1 && i_down<4096*0.9){
if(real_out_CV[count]>=4096*(1-2*swing)||real_out_CV[count
   ]<=4096*(swing) && DefaultRefMode>=1){
  i_down=EndCV;
  DefaultRefMode=DefaultRefMode-1;
  count=countUp;
  analogWrite(DACOUT,2048);
  selectR(DefaultRefMode);
  CapModetemp=ChooseCapCV(DefaultRefMode);
  delay(100);
    continue;
}
}
i_down=i_down-StepCV;
CVV[count]=float(2048-real_in_CV[count])*2.2/4096*1000;
CVI[count]=float(real_out_CV[count]-2048)*3.3/4096/Rref[
   DefaultRefMode]*1000000;
```

```
  count=count+1;
}
analogWrite(DACOUT,2048);
//Smoothing
for (int i = 0; i < count; i++) {
  CVI_Smoothed[i] = CVI[i];
}
if(WindowSize==5){
  for(int i=2;i<count-2;++i){
    CVI_Smoothed[i]=((-3)*CVI[i-2]+12*CVI[i-1]+17*CVI[i-0]+12*
       CVI[i+1]-3*CVI[i+2])/35;
  }
}
else if (WindowSize==7){
  for(int i=3;i<count-3;++i){
    CVI_Smoothed[i]=((-2)*CVI[i-3]+(3)*CVI[i-2]+6*CVI[i-1]+7*
       CVI[i]+6*CVI[i+1]+3*CVI[i+2]-2*CVI[i+3])/21;
  }
}
else if (WindowSize==9){
  for(int i=4;i<count-4;++i){
    CVI_Smoothed[i]=((-21)*CVI[i-4]+(14)*CVI[i-3]+39*CVI[i
       -2]+54*CVI[i-1]+59*CVI[i]+54*CVI[i+1]+39*CVI[i+2]+14*CVI
       [i+3]-21*CVI[i+4])/231;
  }
}
else if (WindowSize==25){
```

```
    for(int i=12;i<count-12;++i){
      CVI_Smoothed[i]=((-21)*CVI[i-4]+(14)*CVI[i-3]+39*CVI[i
          -2]+54*CVI[i-1]+59*CVI[i]+54*CVI[i+1]+39*CVI[i+2]+14*CVI
          [i+3]-21*CVI[i+4])/231;
    }
  }
  for (int i=0;i<count;++i){
       Bluetooth.print(CVI_Smoothed[i],5);
       delay(WaitforMS);
  }
}
else if (inByte=='F'){
   DefaultRefMode=7;
   selectR(DefaultRefMode);
   int count=0;
   int count2=0;
   int i_up=BeginCV;
   int i_down=EndCV;
   int sumtemp=0;
   int avetemp=0;
   for (int h=0;h<8;++h){
        analogWrite(DACOUT,i_up+4096*0.1);
        ReadOutSum=0;
        selectR(h);
        CapModetemp=ChooseCapCV(h);
        delay(200);
     for(int l =0;l<AveSize;++l){
```

```
      ReadOutWindow[l]=analogRead(TIAOutput);
      ReadOutSum+=ReadOutWindow[l];
    }
      avetemp=ReadOutSum/AveSize;
      if(avetemp>(4096*0.1)){
      continue;
      }
      if(avetemp>(4096*0.1)){
        DefaultRefMode=h;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        adjust_com();
        delay(5000);
        break;
      }else{
        DefaultRefMode=h-1;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        adjust_com();
        delay(5000);
        break;
      }
    }
selectR(DefaultRefMode);
CapModetemp=ChooseCapCVFS(DefaultRefMode);
while(i_up<EndCV+1){
  analogWrite(DACOUT,i_up);
```

```
    real_in_CV[count]=analogRead(TIAInput);
    ReadOutWindow[0]=analogRead(TIAOutput);
    ReadOutWindow[1]=analogRead(TIAOutput);
    ReadOutWindow[2]=analogRead(TIAOutput);
    ReadOutWindow[3]=analogRead(TIAOutput);
    ReadOutAve=(ReadOutWindow[0]+ReadOutWindow[1]+ReadOutWindow
       [2]+ReadOutWindow[3])/4;
    real_out_CV[count]=ReadOutAve;
    if(i_up>4096*0.1 &&i_up <4096*0.9){
    if(real_out_CV[count]>=4096*(1-2*swing)||real_out_CV[count
       ]<=4096*(swing) && DefaultRefMode>=1){
      i_up=BeginCV;
      DefaultRefMode=DefaultRefMode-1;
      count=0;
      analogWrite(DACOUT,2048);
      selectR(DefaultRefMode);
      CapModetemp=ChooseCapCVFS(DefaultRefMode);
      delay(100);
      continue;
      }
    }
    i_up=i_up+StepCV;
    count=count+1;
  }
int countUp=count;
while(i_down>BeginCV-1){
  analogWrite(DACOUT,i_down);
```

```
  real_in_CV[count]=analogRead(TIAInput);
  ReadOutWindow[0]=analogRead(TIAOutput);
  ReadOutWindow[1]=analogRead(TIAOutput);
  ReadOutWindow[2]=analogRead(TIAOutput);
  ReadOutWindow[3]=analogRead(TIAOutput);
  ReadOutAve=(ReadOutWindow[0]+ReadOutWindow[1]+ReadOutWindow
     [2]+ReadOutWindow[3])/4;
  real_out_CV[count]=ReadOutAve;
  if(i_down>4096*0.1 && i_down<4096*0.9){
  if(real_out_CV[count]>=4096*(1-2*swing)||real_out_CV[count
     ]<=4096*(swing) && DefaultRefMode>=1){
    i_down=EndCV;
    DefaultRefMode=DefaultRefMode-1;
    count=countUp;
    analogWrite(DACOUT,2048);
    selectR(DefaultRefMode);
    CapModetemp=ChooseCapCVFS(DefaultRefMode);
    delay(100);
      continue;
  }
  }
  i_down=i_down-StepCV;
  count=count+1;
}
analogWrite(DACOUT,2048);
//Smoothing
for (int i = 0; i < count; i++) {
```

```
  CVI[i]=float(real_out_CV[i]-2048)*3.3/4096/Rref[
     DefaultRefMode]*1000000;
  CVI_Smoothed[i] = CVI[i];
 }
 if(WindowSize==5){
  for(int i=2;i<count-2;++i){
   CVI_Smoothed[i]=((-3)*CVI[i-2]+12*CVI[i-1]+17*CVI[i-0]+12*
      CVI[i+1]-3*CVI[i+2])/35;
  }
 }
 else if (WindowSize==7){
  for(int i=3;i<count-3;++i){
   CVI_Smoothed[i]=((-2)*CVI[i-3]+(3)*CVI[i-2]+6*CVI[i-1]+7*
      CVI[i]+6*CVI[i+1]+3*CVI[i+2]-2*CVI[i+3])/21;
  }
 }
 else if (WindowSize==9){
  for(int i=4;i<count-4;++i){
   CVI_Smoothed[i]=((-21)*CVI[i-4]+(14)*CVI[i-3]+39*CVI[i
      -2]+54*CVI[i-1]+59*CVI[i]+54*CVI[i+1]+39*CVI[i+2]+14*CVI
      [i+3]-21*CVI[i+4])/231;
  }
 }
 else if (WindowSize==25){
  for(int i=12;i<count-12;++i){
```

```
      CVI_Smoothed[i]=((-21)*CVI[i-4]+(14)*CVI[i-3]+39*CVI[i
         -2]+54*CVI[i-1]+59*CVI[i]+54*CVI[i+1]+39*CVI[i+2]+14*CVI
         [i+3]-21*CVI[i+4])/231;
     }
   }
   for (int i=0;i<count;++i){
        Bluetooth.print(CVI_Smoothed[i],5);
        delay(WaitforMS);
   }
  }
  //EIS part
else if(inByte=='E' ||inByte=='A' ||inByte=='B'||inByte=='C'||
   inByte=='D'){
   if(inByte=='E'){
     Bias=0;
     CommonMode=2048;
   }
   else if(inByte=='A'){
     Bias=2*186;
     CommonMode=2048+2*186;
   }
      else if(inByte=='B'){
     Bias=186;
     CommonMode=2048+186;
   }
      else if(inByte=='C'){
     Bias=-186;
```

```
    CommonMode=2048-186;
  }
     else if(inByte=='D'){
    Bias=(-2)*186;
    CommonMode=2048-186*2;
  }
  analogWrite(DACOUT,CommonMode-72);
  delay(3000);
//******* Low band************//
  int avetemp=0;
  int k=0;
  DefaultRefMode=7;
  for (int h=0;h<8;++h){
        //Serial.print(h);
        analogWrite(DACOUT,CommonMode-72);
        //selectR(0);
        ReadOutSum=0;
        selectR(h);
        CapModetemp=ChooseCapCV(h);
        delay(200);
        for(int l =0;l<AveSize;++l){
        ReadOutWindow[l]=analogRead(TIAOutput);
        ReadOutSum+=ReadOutWindow[l];
        }
        avetemp=ReadOutSum/AveSize;
        if(avetemp>(4096*0.1) && avetemp<(4096*0.8)){
        continue;
```

```
      }
      if(avetemp>(4096*0.1) && avetemp<(4096*0.8)){
        analogWrite(DACOUT,CommonMode);
        DefaultRefMode=h;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        delay(10000);
        break;
      }else{
        analogWrite(DACOUT,CommonMode);
        DefaultRefMode=h-1;
        selectR(DefaultRefMode);
        CapModetemp=ChooseCapCV(DefaultRefMode);
        delay(10000);
        break;
      }
    }
for (int k=FirstFreq;k<=95;++k){
  if (k<=63) {
  selectR(DefaultRefMode);
  CapModetemp=ChooseCap(DefaultRefMode,k);
  delay(10);
  int capMode;
    if(k==0){
    }
    for (int j=0;j<=7;++j){
    if(k==0){
```

```
      for (int L=0;L<1;++L) {
        for (int i=0;i<Samplepoints;++i) {
          analogWrite(DACOUT,SineTone[i]+Bias);
          delayMicroseconds(DelayTime[k]);
          real_in[0][i]=analogRead(TIAInput);
          real_out[0][i]=analogRead(TIAOutput);
          }
       }
      }
      else{
       for (int L=0;L<1;++L) {
        for (int i=0;i<Samplepoints;++i) {
          analogWrite(DACOUT,SineTone[i]+Bias);
          delayMicroseconds(DelayTime[k]);
          real_in[0][i]=analogRead(TIAInput);
          real_out[0][i]=analogRead(TIAOutput);
         }
       }
      }
  //Gain control Version two!
for (int L=0;L<1;++L) {
for (int i=0;i<Samplepoints;++i) {
  im_in[L][i]=0;
  im_out[L][i]=0;
  real_in_mapto16Bit[L][i]=real_in[L][i]<<3;
  real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
     b0111111111111111;
```

```
  real_out_mapto16Bit[L][i]=real_out[L][i]<<3;
  real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
     b0111111111111111;
}
}
  // 256 points FFT
 for (int L=0;L<1;++L) {
fix_fft(real_in_mapto16Bit[L],im_in[L],8,0);
fix_fft(real_out_mapto16Bit[L],im_out[L],8,0);
 }
// Calculate Magnitude and phase: Method 2, adds up aliasing
for (int L=0;L<1;++L) {
for (int i=0;i<=32;++i) {
  mag_in[L][i] = sqrt((long)real_in_mapto16Bit[L][i] * (long)
     real_in_mapto16Bit[L][i] + (long)im_in[L][i] * (long)im_in[
     L][i])/32768.0*3.3;
  phase_in[L][i] = atan((float)im_in[L][i]/(float)
     real_in_mapto16Bit[L][i])*57.32484;
  mag_out[L][i] = sqrt((long)real_out_mapto16Bit[L][i] * (long)
     real_out_mapto16Bit[L][i] + (long)im_out[L][i] * (long)
     im_out[L][i])/32768.0*3.3;
  phase_out[L][i] = atan((float)im_out[L][i]/(float)
     real_out_mapto16Bit[L][i])*57.32484;
}
}
  // Correct phase
for (int L=0;L<1;++L) {
```

```
for (int i=0;i<=32;++i) {
  if (real_in_mapto16Bit[L][i]<0)
    phase_in[L][i]=phase_in[L][i]+180;
  if (real_out_mapto16Bit[L][i]<0)
    phase_out[L][i]=phase_out[L][i]+180;
}
  for (int i=1;i<=32;++i) {
  mag_in[L][i] = mag_in[L][i]+mag_in[L][Samplepoints-i];
  mag_out[L][i] = mag_out[L][i]+mag_out[L][Samplepoints-i];
}
}
 if (mag_out[0][1]>0.5-(abs(Bias)/1860.0))
 {
  DefaultRefMode=DefaultRefMode-1;
  analogWrite(DACOUT,CommonMode);
  selectR(DefaultRefMode);
  CapModetemp=ChooseCap(DefaultRefMode,k);
 }
 else if(mag_out[0][1]<0.01)
 {DefaultRefMode=DefaultRefMode+1;
  analogWrite(DACOUT,CommonMode);
  selectR(DefaultRefMode);
  CapModetemp=ChooseCap(DefaultRefMode,k);
 }
  else break;
}
```

```
  for (int L=0;L<AveEISLow;++L) {
  for (int i=0;i<Samplepoints;++i) {
    analogWrite(DACOUT,SineTone[i]+Bias);
    delayMicroseconds(DelayTime[k]);
    real_in[L][i]=analogRead(TIAInput);
    real_out[L][i]=analogRead(TIAOutput);
  }
}
for (int L=0;L<AveEISLow;++L) {
for (int i=0;i<Samplepoints;++i) {
  im_in[L][i]=0;
  im_out[L][i]=0;
  real_in_mapto16Bit[L][i]=real_in[L][i]<<3;
  real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
     b0111111111111111;
  real_out_mapto16Bit[L][i]=real_out[L][i]<<3;
  real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
     b0111111111111111;
}
}
  // 256 points FFT
 for (int L=0;L<AveEISLow;++L) {
fix_fft(real_in_mapto16Bit[L],im_in[L],8,0);
fix_fft(real_out_mapto16Bit[L],im_out[L],8,0);
 }
// Calculate Magnitude and phase: Method 2, adds up aliasing
for (int L=0;L<AveEISLow;++L) {
```

```
for (int i=0;i<=32;++i) {
  mag_in[L][i] = sqrt((long)real_in_mapto16Bit[L][i] * (long)
     real_in_mapto16Bit[L][i] + (long)im_in[L][i] * (long)im_in[
     L][i])/32768.0*3.3;
  phase_in[L][i] = atan((float)im_in[L][i]/(float)
     real_in_mapto16Bit[L][i])*57.32484;
  mag_out[L][i] = sqrt((long)real_out_mapto16Bit[L][i] * (long)
     real_out_mapto16Bit[L][i] + (long)im_out[L][i] * (long)
     im_out[L][i])/32768.0*3.3;
  phase_out[L][i] = atan((float)im_out[L][i]/(float)
     real_out_mapto16Bit[L][i])*57.32484;
}
}
  // Correct phase
for (int L=0;L<AveEISLow;++L) {
for (int i=0;i<=32;++i) {
  if (real_in_mapto16Bit[L][i]<0)
    phase_in[L][i]=phase_in[L][i]+180;
  if (real_out_mapto16Bit[L][i]<0)
    phase_out[L][i]=phase_out[L][i]+180;
}
  for (int i=1;i<=32;++i) {
  mag_in[L][i] = mag_in[L][i]+mag_in[L][Samplepoints-i];
  mag_out[L][i] = mag_out[L][i]+mag_out[L][Samplepoints-i];
}
}
impedance[k]=0;
```

```
phase[k];
float tempPhase=0;
for (int L=0;L<AveEISLow;++L){
  impedance[k]+=mag_in[L][1]*Rref[DefaultRefMode]/mag_out[L][1];
  tempPhase=phase_out[L][1]-phase_in[L][1];
  if (tempPhase>170) {
    tempPhase=tempPhase-360;
  }
  if (tempPhase<-180) {
    tempPhase=tempPhase+360;
  }
  phase[k]+=tempPhase*(-1);
}
impedance[k]=impedance[k]/AveEISLow;
phase[k]=phase[k]/AveEISLow;
  RealPrint[k]=impedance[k]*cos(phase[k]*pi/180);
  ImagPrint[k]=impedance[k]*sin(phase[k]*pi/180);
  DataEIS[2*k]=impedance[k];
  DataEIS[2*k+1]=phase[k];
  Bluetooth.print(DataEIS[2*k]);
  delay(WaitforMS);
  Bluetooth.print(DataEIS[2*k+1]);
}


  else {
    selectR(DefaultRefMode);
    CapModetemp=ChooseCap(DefaultRefMode,k);
```

```
  int capMode;
  for (int j=0;j<=7;++j){
   delay(10);
   for (int L=0;L<2;++L) {
    for (int i=0;i<Samplepoints;++i) {
     analogWrite(DACOUT,SineToneHigh[k-64][i]+Bias);
     real_in[0][i]=analogRead(TIAInput);
     real_out[0][i]=analogRead(TIAOutput);
    }
   }
  /*This is to recognize proper gain*/
int Maximum=real_out[0][0];
int Minimum=real_out[0][0];
for (int i=1;i<Samplepoints;++i){
 if (real_out[0][i]>Maximum)
 {
  Maximum=real_out[0][i];
 }
 if (real_out[0][i]<Minimum)
 {
  Minimum=real_out[0][i];
 }
}
if (Maximum>=4096*(1-2*swing)||Minimum<=4096*(swing))
{

 DefaultRefMode=DefaultRefMode-1;
```

```
   analogWrite(DACOUT,CommonMode);
   selectR(DefaultRefMode);
   CapModetemp=ChooseCap(DefaultRefMode,k);
   delay(100);
  }
   else break;
 }
    for (int L=0;L<AveEISHigh;L++){
      for (int i=0;i<Samplepoints;++i) {
      analogWrite(DACOUT,SineToneHigh[k-64][i]+Bias);
      real_in[L][i]=analogRead(TIAInput);
      real_out[L][i]=analogRead(TIAOutput);
      }
    }

 //make 12 bit into 16 bit
for (int L=0;L<AveEISHigh;++L) {
 for (int i=0;i<Samplepoints;++i) {
   im_in[L][i]=0;
   im_out[L][i]=0;
   real_in_mapto16Bit[L][i]=real_in[L][i]<<3;
   real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
      b0111111111111111;
   real_out_mapto16Bit[L][i]=real_out[L][i]<<3;
   real_in_mapto16Bit[L][i]=real_in_mapto16Bit[L][i] & 0
      b0111111111111111;
 }
```

```
}


  // 256 points FFT
 for (int L=0;L<AveEISHigh;++L) {
fix_fft(real_in_mapto16Bit[L],im_in[L],8,0);
fix_fft(real_out_mapto16Bit[L],im_out[L],8,0);
 }
// Calculate Magnitude and phase: Method 2, adds up aliasing
for (int L=0;L<AveEISHigh;++L) {
for (int i=0;i<=64;++i) {
  mag_in[L][i] = sqrt((long)real_in_mapto16Bit[L][i] * (long)
     real_in_mapto16Bit[L][i] + (long)im_in[L][i] * (long)im_in[
     L][i])/32768.0*3.3;
  phase_in[L][i] = atan((float)im_in[L][i]/(float)
     real_in_mapto16Bit[L][i])*57.32484;
  mag_out[L][i] = sqrt((long)real_out_mapto16Bit[L][i] * (long)
     real_out_mapto16Bit[L][i] + (long)im_out[L][i] * (long)
     im_out[L][i])/32768.0*3.3;
  phase_out[L][i] = atan((float)im_out[L][i]/(float)
     real_out_mapto16Bit[L][i])*57.32484;
}
}
  // Correct phase
for (int L=0;L<AveEISHigh;++L) {
for (int i=0;i<=64;++i) {
  if (real_in_mapto16Bit[L][i]<0)
    phase_in[L][i]=phase_in[L][i]+180;
```

```
  if (real_out_mapto16Bit[L][i]<0)
    phase_out[L][i]=phase_out[L][i]+180;
}
  for (int i=1;i<=64;++i) {
  mag_in[L][i] = mag_in[L][i]+mag_in[L][Samplepoints-i];
  mag_out[L][i] = mag_out[L][i]+mag_out[L][Samplepoints-i];
}
}
impedance[k]=0;
phase[k];
float tempPhase=0;
for (int L=0;L<AveEISHigh;++L){
  impedance[k]+=mag_in[L][HighfreqIndex[k-64]]*Rref[
     DefaultRefMode]/mag_out[L][HighfreqIndex[k-64]];
  tempPhase=phase_out[L][HighfreqIndex[k-64]]-phase_in[L][
     HighfreqIndex[k-64]];
  if (tempPhase>170) {
    tempPhase=tempPhase-360;
  }
  if (tempPhase<-180) {
    tempPhase=tempPhase+360;
  }
  phase[k]+=tempPhase*(-1);
}
impedance[k]=impedance[k]/AveEISHigh;
phase[k]=phase[k]/AveEISHigh;
  DataEIS[2*k]=impedance[k];
```

```
   DataEIS[2*k+1]=phase[k];
   Bluetooth.print(DataEIS[2*k]);
   delay(WaitforMS);
   Bluetooth.print(DataEIS[2*k+1]);
   }
}
  }
  }
}

void selectR(int Res) {
  digitalWrite(EN_R, LOW);
  switch (Res) {
   case 0:
   {
     digitalWrite(S0R, HIGH);
     digitalWrite(S1R, HIGH);
     digitalWrite(S2R, LOW);
     break;
   }
   case 1:
   {
     digitalWrite(S0R, LOW);
     digitalWrite(S1R, HIGH);
     digitalWrite(S2R, LOW);
     break;
   }
```

```
 case 2:
{
  digitalWrite(S0R, HIGH);
  digitalWrite(S1R, LOW);
  digitalWrite(S2R, LOW);
  break;
}
   case 3:
{
  digitalWrite(S0R, LOW);
  digitalWrite(S1R, LOW);
  digitalWrite(S2R, LOW);
  break;
}
   case 4:
{
  digitalWrite(S0R, LOW);
  digitalWrite(S1R, LOW);
  digitalWrite(S2R, HIGH);
  break;
}
   case 5:
{
  digitalWrite(S0R, HIGH);
  digitalWrite(S1R, LOW);
  digitalWrite(S2R, HIGH);
  break;
```

```
   }
      case 6:
   {
     digitalWrite(S0R, LOW);
     digitalWrite(S1R, HIGH);
     digitalWrite(S2R, HIGH);
     break;
   }
      case 7:
   {
     digitalWrite(S0R, HIGH);
     digitalWrite(S1R, HIGH);
     digitalWrite(S2R, HIGH);
     break;
   }
  }
}


void selectC(int Cap) {
  digitalWrite(EN_C, LOW);
  switch (Cap) {
    case 0:
   {
     digitalWrite(S0C, HIGH);
     digitalWrite(S1C, HIGH);
     digitalWrite(S2C, LOW);
     break;
```

```
}
 case 1:
{
  digitalWrite(S0C, LOW);
  digitalWrite(S1C, HIGH);
  digitalWrite(S2C, LOW);
  break;
}
 case 2:
{
  digitalWrite(S0C, HIGH);
  digitalWrite(S1C, LOW);
  digitalWrite(S2C, LOW);
  break;
}
   case 3:
{
  digitalWrite(S0C, LOW);
  digitalWrite(S1C, LOW);
  digitalWrite(S2C, LOW);
  break;
}
   case 4:
{
  digitalWrite(S0C, LOW);
  digitalWrite(S1C, LOW);
  digitalWrite(S2C, HIGH);
```

```
    break;
  }
     case 5:
  {
    digitalWrite(S0C, HIGH);
    digitalWrite(S1C, LOW);
    digitalWrite(S2C, HIGH);
    break;
  }
     case 6:
  {
    digitalWrite(S0C, LOW);
    digitalWrite(S1C, HIGH);
    digitalWrite(S2C, HIGH);
    break;
  }
     case 7:
  {
    digitalWrite(S0C, HIGH);
    digitalWrite(S1C, HIGH);
    digitalWrite(S2C, HIGH);
    break;
  }
 }
}
 int ChooseCap (int modeR,int indexF){
      //This is to determine what should be the filter cap size
```

```
CapChoice=1/(2*3.14159265*Rref[modeR]*50*frequency[indexF])
   *1000000000;
if (CapChoice>=10000)
    { selectC(7);
    capMode=8;
   }
else if (CapChoice<10000&&CapChoice>=1000)
   { selectC(6);
   capMode=7;
  }
     else if (CapChoice<1000&&CapChoice>=100)
   { selectC(5);
   capMode=6;
  }
     else if (CapChoice<100&&CapChoice>=10)
   { selectC(4);
   capMode=5;
  }
    else if (CapChoice<10&&CapChoice>=1)
   { selectC(3);
   capMode=4;
  }
     else if (CapChoice<1&&CapChoice>=0.1)
   { selectC(2);
   capMode=3;
  }
     else if (CapChoice<0.1&&CapChoice>=0.02)
```

```
    { selectC(1);
    capMode=2;
   }
     else if (CapChoice<0.02)
    { selectC(0);
    capMode=1;
   }
   return capMode ;
}

 int ChooseCapCV(int modeR){
    //This is to determine what should be the filter cap size
 CapChoice=1/(2*3.14159265*Rref[modeR]*0.5)*1000000000;
 if (CapChoice>=10000)
    { selectC(7);
    capMode=8;
   }
 else if (CapChoice<10000&&CapChoice>=1000)
    { selectC(6);
    capMode=7;
   }
     else if (CapChoice<1000&&CapChoice>=100)
    { selectC(5);
    capMode=6;
   }
     else if (CapChoice<100&&CapChoice>=10)
    { selectC(4);
```

```
    capMode=5;
   }
    else if (CapChoice<10&&CapChoice>=1)
    { selectC(3);
    capMode=4;
   }
      else if (CapChoice<1&&CapChoice>=0.1)
    { selectC(2);
    capMode=3;
   }
      else if (CapChoice<0.1&&CapChoice>=0.02)
    { selectC(1);
    capMode=2;
   }
      else if (CapChoice<0.02)
    { selectC(0);
    capMode=1;
   }
   return capMode ;
}
   int ChooseCapCVFS(int modeR){
    //This is to determine what should be the filter cap size
 CapChoice=1/(2*3.14159265*Rref[modeR]*800)*1000000000;
 if (CapChoice>=10000)
     { selectC(7);
     capMode=8;
    }
```

```
else if (CapChoice<10000&&CapChoice>=1000)
   { selectC(6);
   capMode=7;
  }
     else if (CapChoice<1000&&CapChoice>=100)
   { selectC(5);
   capMode=6;
  }
     else if (CapChoice<100&&CapChoice>=10)
   { selectC(4);
   capMode=5;
  }
    else if (CapChoice<10&&CapChoice>=1)
   { selectC(3);
   capMode=4;
  }
     else if (CapChoice<1&&CapChoice>=0.1)
   { selectC(2);
   capMode=3;
  }
     else if (CapChoice<0.1&&CapChoice>=0.02)
   { selectC(1);
   capMode=2;
  }
     else if (CapChoice<0.02)
   { selectC(0);
   capMode=1;
```

```
    }
    return capMode ;
}
void adjust_com(){
  analogWrite(DACOUT,2048);
  selectR(1);
  delay(1000);
}
```

Appendix D

Python Code for PC GUI

```
# A window with a button
import sys
from PyQt5 import QtGui,QtCore
from PyQt5 import QtWidgets
from PyQt5.QtWidgets import QApplication, QMainWindow, QDialog,
   QGridLayout, QGroupBox, QVBoxLayout,QMessageBox, QPushButton
import pyqtgraph as pg
import numpy as np
import serial.tools.list_ports
import serial
import xlsxwriter
import time
import datetime
import xlwt
from xlwt import Workbook


## Serial parameters
ser = serial.Serial(timeout=1000)
ser.baudrate=500000
## Device parameter for calibration
Clock_frequency=50000000
Serial_delay =4262 ## In clock cycle
Amp_latancy=200 ##in micro seconds
R_para=41
```

```
Gain_1 = [10+R_para,100+R_para,1000+R_para,10000+R_para
   ,100000,1000000,10000000,100000000]
Gain_2 = [2+R_para/500,4+R_para/500,8+R_para/500,16+R_para
   /500,32+R_para/500,64+R_para/500,100,200]
Vout_offset=0.024078
Base_out_Full_scale=5.0 ## full scale voltage +- range



## Some flags
Device_flag=0
Mode='AMP'
x=0
VID_PID='29DD:8001' ##2341:003D for arduino// 29DD:8001 for this
   device
data1 = []
data2 = []
datatime = []
time_now=0
workbook=0


##Settings for different module
DAC_bit=12
DAC_fs=3.0 ## full scale voltage +- range
ADC_bit=14
ADC_fs=5.0 ## full scale voltage +- range


DAC_MSB=2**DAC_bit
```

```
ADC_MSB=2**ADC_bit
class Window(QMainWindow):
   def __init__(self):
      super().__init__()
      self.title = "PyQt5␣GridLayout"
      self.setWindowTitle('Main␣Window')
      self.top = 100
      self.left = 100
      self.width = 1500
      self.height = 500
      self.init_ui()
   def init_ui(self):
      self.setGeometry(self.top, self.left, self.width, self.
         height)
      self.set_plot_amp()
   def style_choice(self,text):
      if text=="Amperametry":
            print(text)
            self.set_plot_amp()
      elif text=="EIS":
            print(text)
            self.set_plot_eis()
      elif text=="CV":
            print(text)
            self.set_plot_cv()
   def set_plot_amp(self):
      global Mode,curve1, curve2,Device_flag
```

```
Mode='AMP'
self.v_box1 = QtWidgets.QGridLayout()
##Clear layout
for i in reversed(range(self.v_box1.count())):
   self.v_box1.itemAt(i).widget().setParent(None)
##Initial Labels
self.t0 = QtWidgets.QLabel('Method')
self.t0.setAlignment(QtCore.Qt.AlignCenter)
self.t1 = QtWidgets.QLabel('Potential (V)')
self.t2 = QtWidgets.QLabel('SampleRate (Sample/s)')
self.t3 = QtWidgets.QLabel('Current Range (uA)')
##Initial LineEditor
self.l1 = QtWidgets.QLineEdit('-3V~+3V range')
self.l2 = QtWidgets.QLineEdit('0.02 to 11000')
self.l3 = QtWidgets.QLineEdit('0.00025 to 10000')
##Initial comboBox
self.comboBox= QtWidgets.QComboBox()
self.comboBox.addItem("Amperametry")
self.comboBox.addItem("EIS")
self.comboBox.addItem("CV")
self.comboBox.addItem("DPV")
self.comboBox.activated[str].connect(self.style_choice)
##Initial push button
self.b1 = QtWidgets.QPushButton('Connect')
self.b1.clicked.connect(self.connect_serial)
self.b2 = QtWidgets.QPushButton('Disconnect')
self.b2.clicked.connect(self.disconnect_serial)
```

```
self.b3 = QtWidgets.QPushButton('Run')
self.b3.setIcon(QtGui.QIcon('Run.jpg'))
if Device_flag:
   self.b3.setStyleSheet("background-color:#00ff00")
self.b3.setCheckable(True)
self.b3.clicked.connect(self.run_amp)
self.b4 = QtWidgets.QPushButton('Reset')
self.b4.clicked.connect(self.reset)
##Initial plot widget
pg.setConfigOption('background',pg.mkColor(0.9))
self.p1 = pg.PlotWidget()
self.p2 = pg.PlotWidget()
self.p1.addLegend()
self.p2.addLegend()
self.p1.showGrid(x=True, y=True, alpha=0.8)
self.p2.showGrid(x=True, y=True, alpha=0.8)
self.p1.setLabel('left', 'Re␣voltage','V')
self.p2.setLabel('left', 'We␣Current','A')
self.p1.setLabel('bottom', 'Time','s')
self.p2.setLabel('bottom', 'Time','s')
curve1 = self.p1.plot(pen=pg.mkPen('y', width=5), name="
   Voltage")
curve2 = self.p2.plot(pen='r', name="Current")
##Add widgets to layout
self.v_box1.addWidget(self.b1, 0, 0)
self.v_box1.addWidget(self.b2, 0, 1)
self.v_box1.addWidget(self.t0, 1, 0)
```

```
        self.v_box1.addWidget(self.comboBox, 1, 1)
        self.v_box1.addWidget(self.b3, 2, 0)
        self.v_box1.addWidget(self.b4, 2, 1)
        self.v_box1.addWidget(self.t1, 4, 0)
        self.v_box1.addWidget(self.l1, 4, 1)
        self.v_box1.addWidget(self.t2, 5, 0)
        self.v_box1.addWidget(self.l2, 5, 1)
        self.v_box1.addWidget(self.t3, 6, 0)
        self.v_box1.addWidget(self.l3, 6, 1)
        self.v_box1.addWidget(self.p1, 0,2,4,10)
        self.v_box1.addWidget(self.p2, 4,2,4,10)
        ##Other layout settings
        self.setWindowTitle('Lang')
        self.v_box1.setColumnStretch(0,1)
        self.v_box1.setColumnStretch(1,1)
        self.v_box1.setColumnStretch(2,8)
        ##Update widget and layout
        self.amp_widget=QtWidgets.QWidget()
        self.amp_widget.setLayout(self.v_box1)
        self.setCentralWidget(self.amp_widget)
    def set_plot_eis(self):
        global Mode,curve1, curve2,Device_flag
        Mode='EIS'
        self.v_box2 = QtWidgets.QGridLayout()
        print(self.v_box2.count())
        for i in reversed(range(self.v_box2.count())):
            self.v_box2.itemAt(i).widget().setParent(None)
```

```
self.t1 = QtWidgets.QLabel('P-P␣Amplitude␣(mV)')
self.l1 = QtWidgets.QLineEdit('1.5mV~6V␣range')
self.t2 = QtWidgets.QLabel('DC␣Bias␣(mV)')
self.l2 = QtWidgets.QLineEdit('-500mV~+500mV␣range')
self.t3 = QtWidgets.QLabel('Start␣frequency')
self.l3 = QtWidgets.QLineEdit()
self.t4 = QtWidgets.QLabel('End␣frequency')
self.l4 = QtWidgets.QLineEdit('')
self.t5 = QtWidgets.QLabel('Number␣of␣points')
self.l5 = QtWidgets.QLineEdit('up␣to␣256')
self.v_box2.addWidget(self.t1, 4, 0)
self.v_box2.addWidget(self.l1, 4, 1)
self.v_box2.addWidget(self.t2, 5, 0)
self.v_box2.addWidget(self.l2, 5, 1)
self.v_box2.addWidget(self.t3, 6, 0)
self.v_box2.addWidget(self.l3, 6, 1)
self.v_box2.addWidget(self.t4, 7, 0)
self.v_box2.addWidget(self.l4, 7, 1)
self.v_box2.addWidget(self.t5,8, 0)
self.v_box2.addWidget(self.l5, 8, 1)
self.t0 = QtWidgets.QLabel('Method')
self.t0.setAlignment(QtCore.Qt.AlignCenter)
self.comboBox= QtWidgets.QComboBox()
self.comboBox.addItem("EIS")
self.comboBox.addItem("Amperametry")
self.comboBox.addItem("CV")
self.comboBox.addItem("DPV")
```

```
self.comboBox.activated[str].connect(self.style_choice)
self.b1 = QtWidgets.QPushButton('Connect')
self.b2 = QtWidgets.QPushButton('Disconnect')
self.b1.clicked.connect(self.connect_serial)
self.b2.clicked.connect(self.disconnect_serial)
self.b3 = QtWidgets.QPushButton('Run')
self.b3.setIcon(QtGui.QIcon('Run.jpg'))
if Device_flag:
   self.b3.setStyleSheet("background-color:#00ff00")
self.b4 = QtWidgets.QPushButton('Reset')
self.b3.setCheckable(True)
self.b4.clicked.connect(self.reset)
self.v_box2.addWidget(self.b1, 0, 0)
self.v_box2.addWidget(self.b2, 0, 1)
self.v_box2.addWidget(self.t0, 1, 0)
self.v_box2.addWidget(self.comboBox, 1, 1)
self.v_box2.addWidget(self.b3, 2, 0)
self.v_box2.addWidget(self.b4, 2, 1)
pg.setConfigOption('background',pg.mkColor(0.9))
self.p1 = pg.PlotWidget()
self.p2 = pg.PlotWidget()
self.p1.setRange(yRange=[0,100000000])
self.p2.setRange(yRange=[-180, 180])
self.p1.addLegend()
self.p2.addLegend()
self.p1.showGrid(x=True, y=True, alpha=0.8)
self.p2.showGrid(x=True, y=True, alpha=0.8)
```

```
        self.p1.setLabel('left', 'Impedance','Ohms')
        self.p2.setLabel('left', 'Phase','Degree')
        self.p1.setLabel('bottom', 'Frequency','Hz')
        self.p2.setLabel('bottom', 'Frequency','Hz')
        self.v_box2.addWidget(self.p1, 0, 2,4,3)
        self.v_box2.addWidget(self.p2, 4, 2,6,3)
        self.v_box2.setColumnStretch(0,1)
        self.v_box2.setColumnStretch(1,1)
        self.v_box2.setColumnStretch(2,8)
        self.eis_widget=QtWidgets.QWidget()
        self.eis_widget.setLayout(self.v_box2)
        self.setCentralWidget(self.eis_widget)
        curve1 = self.p1.plot(pen='y', name="Impedance")
        curve2 = self.p2.plot(pen='r', name="Phase")
    def set_plot_cv(self):
        global Mode,curve1, curve2,Device_flag
        Mode='CV'
        self.v_box3 = QtWidgets.QGridLayout()
        for i in reversed(range(self.v_box3.count())):
            self.v_box3.itemAt(i).widget().setParent(None)
        ##Initial Labels
        self.t0 = QtWidgets.QLabel('Method')
        self.t0.setAlignment(QtCore.Qt.AlignCenter)
        self.t1 = QtWidgets.QLabel('V_min␣(mV)')
        self.t2 = QtWidgets.QLabel('V_max␣(mV)')
        self.t3 = QtWidgets.QLabel('Step␣(mV)')
        self.t4 = QtWidgets.QLabel('Rate␣(V/s)')
```

```
self.t5 = QtWidgets.QLabel('Cycles')
self.t6 = QtWidgets.QLabel('Range (uA)')
##Initial LineEditor
self.l1 = QtWidgets.QLineEdit()
self.l2 = QtWidgets.QLineEdit()
self.l3 = QtWidgets.QLineEdit()
self.l4 = QtWidgets.QLineEdit('')
self.l5 = QtWidgets.QLineEdit('up to 256')
self.l6 = QtWidgets.QLineEdit('up to 256')
##Initial comboBox
self.comboBox= QtWidgets.QComboBox()
self.comboBox.addItem("CV")
self.comboBox.addItem("Amperametry")
self.comboBox.addItem("EIS")
self.comboBox.addItem("DPV")
self.comboBox.activated[str].connect(self.style_choice)
##Initial push button
self.b1 = QtWidgets.QPushButton('Connect')
self.b1.clicked.connect(self.connect_serial)
self.b2 = QtWidgets.QPushButton('Disconnect')
self.b2.clicked.connect(self.disconnect_serial)
self.b3 = QtWidgets.QPushButton('Run')
self.b3.setIcon(QtGui.QIcon('Run.jpg'))
if Device_flag:
   self.b3.setStyleSheet("background-color:#00ff00")
self.b3.setCheckable(True)
self.b3.clicked.connect(self.run_amp)
```

```
self.b4 = QtWidgets.QPushButton('Reset')
self.b4.clicked.connect(self.reset)
##Initial plot widget
pg.setConfigOption('background',pg.mkColor(0.9))
self.p1 = pg.PlotWidget()
self.p1.enableAutoRange(True)
self.p1.setRange(yRange=[-0.00001, 0.00001])
self.p1.addLegend()
self.p1.showGrid(x=True, y=True, alpha=0.8)
self.p1.setLabel('left', 'Current','A')
self.p1.setLabel('bottom', 'Voltage','V')
curve1 = self.p1.plot(pen='y', name="CV")
##Add widgets to layout
self.v_box3.addWidget(self.b1, 0, 0)
self.v_box3.addWidget(self.b2, 0, 1)
self.v_box3.addWidget(self.t0, 1, 0)
self.v_box3.addWidget(self.comboBox, 1, 1)
self.v_box3.addWidget(self.b3, 2, 0)
self.v_box3.addWidget(self.b4, 2, 1)
self.v_box3.addWidget(self.t1, 4, 0)
self.v_box3.addWidget(self.l1, 4, 1)
self.v_box3.addWidget(self.t2, 5, 0)
self.v_box3.addWidget(self.l2, 5, 1)
self.v_box3.addWidget(self.t3, 6, 0)
self.v_box3.addWidget(self.l3, 6, 1)
self.v_box3.addWidget(self.t4, 7, 0)
self.v_box3.addWidget(self.l4, 7, 1)
```

```
        self.v_box3.addWidget(self.t5, 8, 0)
        self.v_box3.addWidget(self.l5, 8, 1)
        self.v_box3.addWidget(self.t6, 9, 0)
        self.v_box3.addWidget(self.l6, 9, 1)
        self.v_box3.addWidget(self.p1, 0, 2,10,3)
        ##Other layout settings
        self.setWindowTitle('Lang')
        self.v_box3.setColumnStretch(0,1)
        self.v_box3.setColumnStretch(1,1)
        self.v_box3.setColumnStretch(2,8)
        ##Update widget and layout
        self.cv_widget=QtWidgets.QWidget()
        self.cv_widget.setLayout(self.v_box3)
        self.setCentralWidget(self.cv_widget)
    def set_plot_dpv(self):
        global Mode,curve1, curve2
        Mode='DPV'
        self.v_box4 = QtWidgets.QGridLayout()
        self.t1 = QtWidgets.QLabel('V_min (mV)')
        self.l1 = QtWidgets.QLineEdit()
        self.t2 = QtWidgets.QLabel('V_max (mV)')
        self.l2 = QtWidgets.QLineEdit()
        self.t3 = QtWidgets.QLabel('Step (mV)')
        self.l3 = QtWidgets.QLineEdit()
        self.t4 = QtWidgets.QLabel('Rate (V/s)')
        self.l4 = QtWidgets.QLineEdit('')
        self.t5 = QtWidgets.QLabel('Amplitude (mV)')
```

```
self.l5 = QtWidgets.QLineEdit('')
self.v_box4.addWidget(self.t1, 4, 0)
self.v_box4.addWidget(self.l1, 4, 1)
self.v_box4.addWidget(self.t2, 5, 0)
self.v_box4.addWidget(self.l2, 5, 1)
self.v_box4.addWidget(self.t3, 6, 0)
self.v_box4.addWidget(self.l3, 6, 1)
self.v_box4.addWidget(self.t4, 7, 0)
self.v_box4.addWidget(self.l4, 7, 1)
self.v_box4.addWidget(self.t5, 8, 0)
self.v_box4.addWidget(self.l5, 8, 1)
self.t0 = QtWidgets.QLabel('Method')
self.t0.setAlignment(QtCore.Qt.AlignCenter)
self.comboBox= QtWidgets.QComboBox()
self.comboBox.addItem("DPV")
self.comboBox.addItem("Amperametry")
self.comboBox.addItem("EIS")
self.comboBox.addItem("CV")
self.comboBox.activated[str].connect(self.style_choice)
self.b1 = QtWidgets.QPushButton('Connect')
self.b2 = QtWidgets.QPushButton('Disconnect')
self.b1.clicked.connect(self.connect_serial)
self.b2.clicked.connect(self.disconnect_serial)
self.v_box4.addWidget(self.b1, 0, 0)
self.v_box4.addWidget(self.b2, 0, 1)
self.v_box4.addWidget(self.t0, 1, 0)
self.v_box4.addWidget(self.comboBox, 1, 1)
```

```
    pg.setConfigOption('background', 'w')
    self.p1 = pg.PlotWidget()
    self.p1.setRange(yRange=[-0.00001, 0.00001])
    self.p1.addLegend()
    self.p1.showGrid(x=True, y=True, alpha=0.8)
    self.p1.setLabel('left', 'Current','V')
    self.v_box4.addWidget(self.p1, 0, 0,1,3)
    self.dpv_widget=QtWidgets.QWidget()
    self.dpv_widget.setLayout(self.v_box4)
    curve1 = self.p1.plot(pen='y', name="Voltage")
def connect_serial(self):
    global ser,Device_flag,curve1,curve2
    if (Device_flag==0):
        for p in serial.tools.list_ports.comports():
            if VID_PID in p[2] :
                ser.port = p[0]
                ##ser.open()
                Device_flag=1
                self.b3.setStyleSheet("background-color:#00ff00")
                print("Device␣Found")
                break
            else:
                Device_flag=0
        if Device_flag==0:
            print("Device␣not␣found");
    else:
        QMessageBox.about(self, "Msg", "Device␣is␣connected")
```

```
def disconnect_serial(self):
    global ser,Device_flag
    Device_flag=0;
    self.b3.setStyleSheet("background-color:None")
    ser.close()
    print("Disconnected")
def print_test(self):
    print("Testing")
def run_amp(self):
    global workbook,Amp_latancy,Serial_delay,ser,time_now,x,
        data1,data2,datatime
    Amp_bias=self.l1.text()
    Amp_Sample=self.l2.text()
    Amp_Range=self.l3.text()
    AMP_paraCheck=0
    todays_date ='Amp_'+ str(datetime.datetime.now().strftime("
        %Y_%m_%d_%H_%M") )+ '.xlsx'
    try :
        a=float(Amp_bias)
        print(Amp_bias)
        AMP_offset=int(round((float(Amp_bias)+DAC_fs*1000)/(
            DAC_fs*2000.0)*((2**DAC_bit)-0.6)))
        print(AMP_offset)
    except:
        QMessageBox.about(self, "Error!!", "Bias␣must␣be␣a␣
            number␣\n␣(from␣-3000␣to␣3000)")
```

```
try :
    a=float(Amp_Sample)
    print(Amp_Sample)
    if float(Amp_Sample)>11000:
        Amp_Sample=11000.0
        self.l2.setText('11000')
    if float(Amp_Sample)<0.02:
        Amp_Sample=0.02
        self.l2.setText('0.02')
    Amp_delay=int(round((Clock_frequency*1.0/float(
        Amp_Sample)-Serial_delay-255)))
    print(Amp_delay)
except:
    QMessageBox.about(self, "Error!!", "Sample␣rate␣must␣a␣
        positive␣number␣\n␣(<11000␣sample/s)")
    print(Amp_Sample)
try :
    a=float(Amp_Range)
    print(float(Amp_Range))
    AMP_gain_1,AMP_gain_2=self.amp_find_gain(float(Amp_Range
        ))
    print(AMP_gain_1)
    print(AMP_gain_2)
    AMP_paraCheck=1
except:
    QMessageBox.about(self, "Error!!", "Range␣be␣a␣positive␣
        number␣(0.0001␣uA␣to␣10000␣uA)")
```

```
            print("Must␣be␣a␣number␣(from␣0.001␣to␣10000)")
        print(int(self.b3.isChecked()))
        if AMP_paraCheck==1:
            AMP_gain_C=4;
            print('AMP_offset=',AMP_offset)
            print('AMP_gain_1=',AMP_gain_1)
            print('AMP_gain_2=',AMP_gain_2)
            print('AMP_gain_C=',AMP_gain_C)
            print('Amp_delay=',Amp_delay)
            if self.b3.isChecked():
                if Device_flag==1:
                    x=0;
                    self.b3.setText('Stop')
                    self.b3.setStyleSheet("background-color:#ed2939")
                    ##setup for wrting files
                    now = datetime.datetime.now()
                    workbook = Workbook()
                    worksheet = workbook.add_sheet('Data')
                    worksheet.write(0, 0, "Index")
                    worksheet.write(0, 1, "Time")
                    worksheet.write(0, 2, "Voltage")
                    worksheet.write(0, 3, "Current")
                    ser.open()
                    ser.write(b'modeApara')
                    ser.write(AMP_offset.to_bytes(2,byteorder='big'))
                    ser.write(AMP_gain_1.to_bytes(1,byteorder='big'))
                    ser.write(AMP_gain_2.to_bytes(1,byteorder='big'))
```

```
            ser.write(AMP_gain_C.to_bytes(1,byteorder='big'))
            ser.write(Amp_delay.to_bytes(4,byteorder='big'))
            time_now=time.time()
            time_last_update=time.time()
            data1 = []
            data2 = []
            datatime = []
            while self.b3.isChecked():
               self.update(Mode,worksheet,AMP_gain_1,
                  AMP_gain_2)
               time_update=time.time()
               print(time_update)
               print(time_last_update)
               if(time_update-time_last_update>=20):
                  workbook.save(todays_date)
                  print(todays_date)
                  time_last_update=time_update
         else:
            QMessageBox.about(self, "Error!!", "Connect␣the␣
               device␣first!")
      else :
         if Device_flag==1:
            self.b3.setText('Run')
            self.b3.setIcon(QtGui.QIcon('run.jpg'))
            self.b3.setStyleSheet("background-color:#00ff00")
            ser.write(b'RESET')
            ser.reset_input_buffer()
```

```
            ser.reset_output_buffer()
            ser.close()
            workbook.save(todays_date)
         else:
            QMessageBox.about(self, "Error!!", "Connect␣the␣
               device␣first!")
   else:
      self.b3.setText('Run')
      self.b3.setIcon(QtGui.QIcon('run.jpg'))
      self.b3.toggle()
def amp_find_gain(self,rangeI):
   global ADC_fs,Gain_1,Gain_2
   current_rangeI=np.zeros((8, 8))
   for i in range (0,8):
      for j in range (0,8):
         current_rangeI[i,j]=ADC_fs*1000000/(Gain_1[i]*Gain_2[
            j])
   X = np.abs(current_rangeI-rangeI)
   idx = np.where( X == X.min() )
   print(current_rangeI.max())
   print(current_rangeI.min())
   self.l3.setText(str(format(float(current_rangeI[idx[0],idx
      [1]]),'.5f')))
   return int(idx[0]),int(idx[1])
def reset(self):
   global Device_flag
   if Device_flag==1:
```

```
            ser.open()
            ser.write(b'RESET')
            ser.close()
        else:
            QMessageBox.about(self, "Error!!", "Connect␣the␣device␣
               first!")
##Update read data and update plot
    def update(self,mode,worksheet,gain1,gain2):
        global ADC_fs,Gain_1,Gain_2,ADC_MSB,curve1,curve2,x, ser,
           size, buffersize,data1,data2,datatime,time_now
        if mode=='AMP' and (ser != None and ser.is_open):
            reading = ser.readline(4)
            C1=int.from_bytes(reading[:2], byteorder='big')
            C2=int.from_bytes(reading[2:4], byteorder='big')
            voltage=(2*C1-ADC_MSB)/float(ADC_MSB)*ADC_fs
            current=((2*C2-ADC_MSB)/float(ADC_MSB)*ADC_fs-
               Vout_offset)*1000000.0/(Gain_1[gain1]*Gain_2[gain2])
            time_data=time.time()
            x += 1
            data1.append(float(voltage))
            data2.append(float(current))
            delta_time=time_data-time_now
            worksheet.write(x, 0, x)
            worksheet.write(x, 1, delta_time)
            worksheet.write(x, 2, voltage)
            worksheet.write(x, 3, current)
            datatime.append(delta_time)
```

```
            curve1.setData(datatime,data1)
            curve2.setData(datatime,data2)
            app.processEvents()
app = QtWidgets.QApplication(sys.argv)
a_window = Window()
a_window.show()
a_window.raise_()
sys.exit(app.exec_())
```

Appendix E

Partial Verilog Code for FPGA

E.1 Verilog Top Module for Base Unit

```
`timescale 1ns / 1ps
module Base_top(
   // 50MHz clock input
   input clk,
   // Input from reset button (active low)
   input rst_n,
   // cclk input from AVR, high when AVR is ready
   input cclk,
       input ADC_chan_A,
       input ADC_chan_B,
       input adc_busy,
       output [2:0] BASE_PREMUX_A,
       output [2:0] BASE_AMPMUX_A,
       output [2:0] BASE_AMPMUXC_A,
       output dac_LDAC_bar,
       output dac_mosi,
       output dac_cs,
       output dac_sclk,
       output dac_clr,
       output dac_rst,
       output adc_CNVST_bar,
       output adc_slck,
```

```
        output adc_cs,
    // Outputs to the 8 onboard LEDs
    output reg [3:0] led,
        //output toprst,
    // AVR SPI connections
    output spi_miso,
    input spi_ss,
    input spi_mosi,
    input spi_sck,
    // AVR ADC channel select
    output [3:0] spi_channel,
    // Serial connections
    input avr_tx, // AVR Tx => FPGA Rx
    output avr_rx, // AVR Rx => FPGA Tx
    input avr_rx_busy, // AVR Rx buffer full
        // Base Module control
        output ModuleLED, //Base Module LED
        output BaseRelay_on, // Base module relay
        output BASE_MUX_EN,
        output EN_PW
    );
localparam STATE_SIZE = 4;
localparam IDLE = 4'd0,
                        WAIT = 4'd1,
                        RUN = 4'd2,
                        Reset1=4'd3,
                        Reset2=4'd4,
```

```
                        Reset3=4'd5,
                        Reset4=4'd6,
                        Reset5=4'd7;
wire rst = ~rst_n; // make reset active high
wire [7:0] tx_data;
wire new_tx_data;
wire tx_busy;
wire [7:0] rx_data;
wire new_rx_data;
//wire toprst;
reg toprst_d,toprst_q,toprst;
reg [STATE_SIZE-1:0] state_q,state_d,bufferstage;
reg [24:0] ctr_d,ctr_q;
reg blink;
reg EIS_start;
// these signals should be high-z when not used
assign spi_miso = 1'bz;
assign BASE_MUX_EN=1'b0;
assign spi_channel = 4'bzzzz;
assign ModuleLED = 1;
assign dac_LDAC_bar=0;
assign EN_PW=1;
assign dac_clr=1;
assign dac_rst=1;
      initial begin
            // Initialize Inputs
            state_d=IDLE;
```

```
            ctr_d=0;
            toprst=1;
            // Wait 100 ns for global reset to finish
            // Add stimulus here
      end
BaseModule_CV_EIS_simple BaseModule_CV_EIS(
      .clk(clk), // input clock--------------------port
      .rst(toprst),  //input reset
      .adc_miso_A(ADC_chan_A), // input SPI ADC_A
         ---------------------port
      .adc_miso_B(ADC_chan_B), // input SPI ADC_B
         ----------------------------port
      .DAC_ctrlEn(0), //input DAC_ctrlReg setting enable,leave as
          0 for default setting
      .adc_busy (adc_busy), //input ADC busy signal from circuit
         ------------------------port
      .dac_mosi(dac_mosi), //output for DAC mosi
         ---------------------port
      .dac_cs(dac_cs), //output for DAC SPI cs
         ---------------------port
      .dac_slck(dac_sclk), //output for DAC SPI clock
         ---------------------port
      .adc_CNVST_bar(adc_CNVST_bar), //output for ADC conversion
         signal---------------------port
      .adc_slck(adc_slck), //output for ADC SPI clock
         ---------------------port
```

```
      .adc_cs(adc_cs), //output for ADC SPI cs
         ---------------------port
      .done(EIS_single_done), //output done for single EIS
      .Measure_done(), //output done for single EIS measurement
         part
      //AVR part
      .avr_tx(avr_tx),
      .cclk(cclk),
      .avr_rx(avr_rx),
      .avr_rx_busy(avr_rx_busy),
      .spi_miso(spi_miso),
  .spi_ss(spi_ss),
  .spi_mosi(spi_mosi),
  .spi_sck(spi_sck),
      .spi_channel(spi_channel),
      .rx_data(rx_data),
  .new_rx_data(new_rx_data),
      //Brd ctrl
      .BaseRelay_on(BaseRelay_on),
      .BASE_PREMUX_A(BASE_PREMUX_A),
      .BASE_AMPMUX_A(BASE_AMPMUX_A),
      .BASE_AMPMUXC_A(BASE_AMPMUXC_A)
);
always @ (*) begin
      blink = ctr_q[24];
  ctr_d = ctr_q ;
      state_d=state_q;
```

```
toprst_d=toprst_q;
toprst=1;
toprst_d=toprst_q;
led=4'b0000;
EIS_start=0;
case (state_q)
      IDLE: begin
            led=4'b0001;
            toprst=1;
            ctr_d=ctr_q+1;
            if(blink) begin
                  state_d=WAIT;
                  ctr_d=0;
                  toprst=1;
            end
      end
      WAIT: begin
            led=4'b0011;
            ctr_d=ctr_q+1;
            toprst=1;
            if(blink) begin
                  state_d=RUN;
                  ctr_d=0;
                  toprst=0;
            end
      end
      RUN: begin
```

```
      led=4'b1011;
      toprst=0;
      state_d=Reset1;
      EIS_start=1;
end
Reset1:begin
      led=4'b1011;
      toprst=0;
  if (new_rx_data && rx_data == "R")
                  state_d = Reset2;
      else if(new_rx_data && rx_data != "R")
            state_d = Reset1;
end
Reset2:begin
      led=4'b1011;
      toprst=0;
  if (new_rx_data && rx_data == "E")
                  state_d = Reset3;
      else if(new_rx_data && rx_data != "E")
            state_d = Reset1;
end
Reset3:begin
      led=4'b1011;
      toprst=0;
  if (new_rx_data && rx_data == "S")
                  state_d = Reset4;
      else if(new_rx_data && rx_data != "S")
```

```
                        state_d = Reset1;
            end
            Reset4:begin
                  led=4'b1011;
                  toprst=0;
              if (new_rx_data && rx_data == "E")
                              state_d = Reset5;
                  else if(new_rx_data && rx_data != "E")
                        state_d = Reset1;
            end
            Reset5:begin
                  led=4'b1011;
                  toprst=0;
              if (new_rx_data && rx_data == "T")
                              state_d = IDLE;
                  else if(new_rx_data && rx_data != "T")
                        state_d = Reset1;
            end
            default: state_d=IDLE;
      endcase
end
always @(posedge clk) begin
      if (rst==1) begin
            state_q <= IDLE;
            ctr_q<=0;
            toprst_q<=1;
      end
```

```
      else begin
            state_q <= state_d;
            ctr_q<=ctr_d;
            toprst_q<=toprst_d;
      end
end

endmodule
```

E.2 Main Control Module for Base Unit

```
module BaseModule_CV_EIS_simple (
      input clk,
      input adc_miso_A,
      input adc_miso_B,
      input DAC_ctrlEn,
      input rst,
      input adc_busy,
      output dac_mosi,
      output dac_cs,
      output dac_slck,
      output adc_CNVST_bar,
      output adc_slck,
      output adc_cs,
      output done,
      output Measure_done,
      input avr_tx,
```

```
      input cclk,
      output avr_rx,
      input avr_rx_busy,
      output spi_miso,
  input spi_ss,
  input spi_mosi,
  input spi_sck,
      output [3:0] spi_channel,
      output [7:0] rx_data,
      output new_rx_data,
      output reg BaseRelay_on,
      output [2:0] BASE_PREMUX_A,
      output [2:0] BASE_AMPMUX_A,
      output [2:0] BASE_AMPMUXC_A
);
localparam Debuger_mode=1'b0;
localparam OSR_reduce=8'd16;
localparam ACG_th_l2=14'd1800;
localparam ACG_th_h2=14'd7500;
localparam ACG_delay_begin=32'd1000; // 100ms delay for each 32'
   d5000000 =32'd1000
localparam ACG_th_l=14'd300;
localparam ACG_th_h=14'd6500;
localparam Common_mode = 14'd2048;
localparam Common_mode_EIS = 14'd8192;
localparam CV_delay_ACG = 32'd1000; // 100ms delay for each 32'
   d500000 = 32'd25000
```

```
localparam STATE_SIZE = 3'd6;
localparam Major_tone=8'd0;
localparam Buffersize =6'd32;
localparam number_of_characters=6'd4;
localparam IDLE = 6'd0,
                  ModeSelect = 6'b1,
                  //CV states beginl
                  IDLE_CV=6'd2,
                  DAC_CV=6'd3,
                  ADC_CV=6'd4,
                  ADC_CV_delay=6'd5,
                  Send_out_CV=6'd6,
                  Send_out_CV_final=6'd7,
                  Send_CV_end=6'd8,
                  //CV states end
                  //EIS states begin
                  IDLE_EIS=6'd9,
                  DAC_EIS=6'd10,
                  ADC_EIS=6'd11,
                  ADC_EIS_delay=6'd12,
                  wait_ave_EIS=6'd13,
                  wait_ADC_ram=6'd14,
                  EIS_signle_done=6'd15,
                  EIS_FFT_collect=6'd16,
                  EIS_FFT_Finish=6'd17,
                  EIS_FFT_Calculate_1=6'd18,
                  EIS_FFT_Calculate_finish_1=6'd19,
```

```
EIS_FFT_Calculate_2=6'd20,
EIS_FFT_Calculate_finish_2 =6'd21,
EIS_Imp_cal=6'd22,
Send_out_EIS_imp=6'd23,
Send_out_EIS_imp_done=6'd24,
Send_out_EIS_pha=6'd25,
Send_out_EIS_Gain=6'd26,
Send_EIS_end=6'd27,
Wait_delay_cal_EIS=6'd36,
Send_out_EIS_A=6'd37,
Send_out_EIS_B=6'd38,
IDLE_CV_ACG=6'd28,
IDLE_CV_ACG2=6'd47,
DAC_CV_ACG=6'd29,
ADC_CV_ACG=6'd30,
ADC_CV_delay_ACG=6'd31,
Decide_ACG_1=6'd32,
Decide_ACG_2=6'd33,
CV_ACG_Rdy=6'd34,
//CV ACG end
Print=6'd35,
//EIS ACG begin
EIS_ACG_init=6'd39,
EIS_ACG_assign=6'd40,
EIS_ACG_assign_wait=6'd41,
EIS_ACG_compare=6'd42,
EIS_ACG_decide1=6'd43,
```

```
                    EIS_ACG_decide2=6'd44,
                    EIS_Decide_ACG_1=6'd45,
                    EIS_Decide_ACG_2=6'd46,
                    //EIS ACG end
                    Reset_CM=6'd48,
                    //AMP states;
                    IDLE_AMP=6'd49,
                    DAC_AMP=6'd50,
                    ADC_AMP=6'd51,
                    END_AMP=6'd52,
                    ADC_AMP_delay=6'd53,
                    Send_out_AMP=6'd54;
reg [STATE_SIZE-1:0] state_d, state_q,buffer_state_d,
   buffer_state_q;
reg adc_rst;
//wire DAC_ctrlEn;
reg DAC_START_d,DAC_START_q,DAC_START,ADC_MeasuEN_d,ADC_MeasuEN_q
   ,ADC_MeasuEN;
reg [7:0] DACindex_d,DACindex_q=0;
reg en_adcmem_w,en_adcmem;
reg done_d,done_q;
reg Measure_done_d,Measure_done_q;
reg [Buffersize-1:0] ctr_q,ctr_d;
reg [9:0] Read_mem_d,Read_mem_q;
reg start_FFT;
reg [7:0] xn_index_Mem;
reg en_fft_mem_w;
```

```
reg tan_nd, tan_ce,tan_sclr,sqrt_nd,sqrt_ce,sqrt_sclr;
//reg [7:0] addr_rd_fft_q,addr_rd_fft_d;
reg [7:0] addr_rd_fft;
reg [31:0] rd_fft_A_Mag_squared_d,rd_fft_A_Mag_squared_q,
   rd_fft_B_Mag_squared;
reg [7:0] DataReady;
reg en_Cal_mem_A,en_Cal_mem_B;
reg [15:0] rd_fft_A_re_buff_q,rd_fft_A_im_buff_q,
   rd_fft_B_re_buff_q,rd_fft_B_im_buff_q;
reg [15:0] rd_fft_A_re_buff_d,rd_fft_A_im_buff_d,
   rd_fft_B_re_buff_d,rd_fft_B_im_buff_d;
reg [15:0] phase_out_A_buff_d,phase_out_B_buff_d;
reg [15:0] phase_out_A_buff_q,phase_out_B_buff_q;
reg arctan_inv_flag_A,arctan_inv_flag_B;
reg Divider_ce,Divider_nd;
reg [16:0] rd_fft_Mag_Dividend_d,rd_fft_Mag_Divisor_d;
reg [16:0] rd_fft_Mag_Dividend_q,rd_fft_Mag_Divisor_q;
reg [11:0] data_reg_d,data_reg_q;
// Average buff init
reg [8:0] ave_index_q,ave_index_d;
reg ave_en;
reg [13:0] data_in_A_ave_q,data_in_B_ave_q;
reg [13:0] data_in_A_ave_d,data_in_B_ave_d;
wire [13:0] data_out_ave_A,data_out_ave_B;
reg ave_final;
reg print_rst;
reg startPrint;
```

```
reg dac_rst;
// CV reg
reg direc_d,direc_q;
reg [7:0] CV_cycle_count_d,CV_cycle_count_q;
// Mode_parameter
wire [3:0] Mode;
wire [7:0] CV_step,CV_cycle;
wire [31:0] CV_delay;
wire [11:0] CV_up,CV_down;
wire [15:0] xn0_im,xn1_im;
wire [13:0] ADC_B,ADC_A;
reg [13:0] ADC_B_true,ADC_A_true;
reg [7:0] EIS_DACcount_d,EIS_DACcount_q;
wire [15:0] Pha;
wire [31:0] Imp;
wire [15:0] mem_A,mem_B;
wire [15:0] ADC_A_FFT_re,ADC_B_FFT_re,ADC_A_FFT_im,ADC_B_FFT_im;
wire [7:0] xn_index,xk_index;
wire [15:0] rd_fft_A_re,rd_fft_B_re,rd_fft_A_im,rd_fft_B_im,
   phase_out_A,phase_out_B;
wire [16:0] rd_fft_A_Mag,rd_fft_B_Mag,rd_fft_Imp_int;
wire [15:0] rd_fft_Imp_frac;
//Avr serial
reg [number_of_characters*8:1] debug_msg_d,debug_msg_q;
wire [7:0] tx_data;
wire new_tx_data;
wire tx_busy;
```

```
wire [7:0] rx_data;
wire new_rx_data;
// EIS init
wire [11:0] EIS_offset,EIS_amp;
wire [15:0] EIS_freq_delay;
wire [23:0] EIS_freq_delay_end;
wire [15:0] EIS_freq_delay_start, EIS_freq_delay_multiplier;
wire [7:0] EIS_num_points;
wire [23:0] EIS_delay_q;
reg [7:0] EIS_index_q,EIS_index_d;
reg [15:0] multi_a,multi_b;
reg [31:0] multi_out;
wire [11:0] EIS_reg_sine;
wire [23:0] EIS_freq;
wire [7:0] EIS_freq_index;
reg [31:0] ctr_EIS_q,ctr_EIS_d;
wire [7:0] AMP_Gain_2,AMP_Gain_1,AMP_Gain_C;
wire [31:0] AMP_delay;
//ACG inti
assign BASE_PREMUX_A=BASE_PREMUX_A_q;
assign BASE_AMPMUX_A=BASE_AMPMUX_A_q;
assign BASE_AMPMUXC_A=BASE_AMPMUXC_A_q;
reg [2:0] BASE_PREMUX_A_q,BASE_AMPMUX_A_q,BASE_AMPMUXC_A_q;
reg [2:0] BASE_PREMUX_A_d,BASE_AMPMUX_A_d,BASE_AMPMUXC_A_d;
reg [13:0] ACG_down_temp_q,ACG_up_temp_q,ACG_down_temp_d,
   ACG_up_temp_d;
```

```
reg [13:0] mag_max_d,mag_max_q,mag_min_d,mag_min_q,mag_eis_acg_q,
   mag_eis_acg_d,EIS_up_temp,EIS_down_temp;
assign done=done_q;
assign Measure_done=Measure_done_q;
      message_printer #(.number_of_characters(
         number_of_characters)) send_CV(
   .clk(clk),
   .rst(print_rst),
   .tx_data(tx_data),
       .msg(debug_msg_q),
   .new_tx_data(new_tx_data),
   .tx_busy(tx_busy),
   .rx_data(rx_data),
   .new_rx_data(new_rx_data),
       .done(done_print),
       .startPrint(startPrint)
 );
 avr_interface #(.CLK_RATE(50000000), .SERIAL_BAUD_RATE(500000))
     avr_interface (
   .clk(clk),
   .rst(rst),
   .cclk(cclk),
   .spi_miso(spi_miso),
   .spi_mosi(spi_mosi),
   .spi_sck(spi_sck),
   .spi_ss(spi_ss),
   .spi_channel(spi_channel),
```

```
   .tx(avr_rx), // FPGA tx goes to AVR rx
   .rx(avr_tx),
   .channel(4'd15), // invalid channel disables the ADC
   .new_sample(),
   .sample(),
   .sample_channel(),
   .tx_data(tx_data),
   .new_tx_data(new_tx_data),
   .tx_busy(tx_busy),
   .tx_block(avr_rx_busy),
   .rx_data(rx_data),
   .new_rx_data(new_rx_data)
  );
freq_mem freq_mem (
  .clka(clk), // input clka
  .wea(EIS_mem_w), // input [0 : 0] wea
  .addra(EIS_freq_index), // input [7 : 0] addra
  .dina(EIS_freq), // input [23 : 0] dina
  .clkb(~clk), // input clkb
  .addrb(EIS_index_q), // input [7 : 0] addrb
  .doutb(EIS_delay_q) // output [23 : 0] doutb
);
xfft fft_base (
  .clk(clk), // input clk
  .start(start_FFT), // input start
  //.start(0), // input start
  .xn0_re(mem_A), // input [15 : 0] xn0_re
```

```
  .xn0_im(16'b0), // input [15 : 0] xn0_im
  .xn1_re(mem_B), // input [15 : 0] xn1_re
  .xn1_im(16'b0), // input [15 : 0] xn1_im
  .fwd_inv0(1'b1), // input fwd_inv0
  .fwd_inv0_we(1'b1), // input fwd_inv0_we
  .fwd_inv1(1'b1), // input fwd_inv1
  .fwd_inv1_we(1'b1), // input fwd_inv1_we
  .scale_sch0(16'b0101010101010101), // input [15 : 0] scale_sch0
      16'b0101010101010101
  .scale_sch0_we(1'b1), // input scale_sch0_we
  .scale_sch1(16'b0101010101010101), // input [15 : 0] scale_sch1
      16'b0101010101010101
  .scale_sch1_we(1'b1), // input scale_sch1_we
  .rfd(FFT_rfd), // output rfd
  .xn_index(xn_index), // output [7 : 0] xn_index (Index of input
      data.)
  .busy(FFT_busy), // output busy
  .edone(edone_FFT), // output edone
  .done(done_FFT), // output done
  .dv(dv_FFT), // output dv
  .xk_index(xk_index), // output [7 : 0] xk_index (Index of
     output data.)
  .xk0_re(ADC_A_FFT_re), // output [15 : 0] xk0_re
  .xk0_im(ADC_A_FFT_im), // output [15 : 0] xk0_im
  .xk1_re(ADC_B_FFT_re), // output [15 : 0] xk1_re
  .xk1_im(ADC_B_FFT_im) // output [15 : 0] xk1_im
);
```

```
adc_memory adc_mem_A (
  .clka(clk), // input clka
  .wea(en_adcmem_w), // input [0 : 0] wea
  .addra({EIS_DACcount_q}), // input [9 : 0] addra
  .dina({2'b00,data_out_ave_A}), // input [15 : 0] dina
  .clkb(~clk), // input clkb
  .addrb({xn_index_Mem}), // input [9 : 0] addrb
  .doutb(mem_A) // output [15 : 0] doutb
);
adc_memory adc_mem_B (
  .clka(clk), // input clka
  .wea(en_adcmem_w), // input [0 : 0] wea
  .addra({EIS_DACcount_q}), // input [9 : 0] addra
  .dina({2'b00,data_out_ave_B}), // input [15 : 0] dina
  .clkb(~clk), // input clkb
  .addrb({xn_index_Mem}), // input [9 : 0] addrb
  .doutb(mem_B) // output [15 : 0] doutb
);
adc_memory fft_mem_a_re (
  .clka(clk), // input clka
  .wea(en_fft_mem_w), // input [0 : 0] wea
  .addra({xk_index}), // input [9 : 0] addra
  .dina(ADC_A_FFT_re), // input [15 : 0] dina
  .clkb(clk), // input clkb
  .addrb(addr_rd_fft), // input [9 : 0] addrb
  .doutb(rd_fft_A_re) // output [15 : 0] doutb
);
```

```
adc_memory fft_mem_a_im (
  .clka(clk), // input clka
  .wea(en_fft_mem_w), // input [0 : 0] wea
  .addra({xk_index}), // input [9 : 0] addra
  .dina(ADC_A_FFT_im), // input [15 : 0] dina
  .clkb(clk), // input clkb
  .addrb(addr_rd_fft), // input [9 : 0] addrb
  .doutb(rd_fft_A_im) // output [15 : 0] doutb
);
adc_memory fft_mem_b_re (
  .clka(clk), // input clka
  .wea(en_fft_mem_w), // input [0 : 0] wea
  .addra({xk_index}), // input [9 : 0] addra
  .dina(ADC_B_FFT_re), // input [15 : 0] dina
  .clkb(clk), // input clkb
  .addrb(addr_rd_fft), // input [9 : 0] addrb
  .doutb(rd_fft_B_re) // output [15 : 0] doutb
);
adc_memory fft_mem_b_im (
  .clka(clk), // input clka
  .wea(en_fft_mem_w), // input [0 : 0] wea
  .addra({xk_index}), // input [9 : 0] addra
  .dina(ADC_B_FFT_im), // input [15 : 0] dina
  .clkb(clk), // input clkb
  .addrb(addr_rd_fft), // input [9 : 0] addrb
  .doutb(rd_fft_B_im) // output [15 : 0] doutb
);
```

```
DACctrl base_dac (
      .ctrlReg(DAC_ctrlReg),  //11-bit Control register input
      .ctrlRegEN(0),  //Change control register input
      .DACstart(DAC_START),  //Start DAC input
      .clk(clk),  //clock input
      .rst(rst),  //reset input
      .dac_miso(),  // DAC SPI-miso input
      .dac_sclk(dac_slck),  //DAC SPI-clock output
      .dac_cs(dac_cs),  //DAC SPI-CS_bar output
      .dac_mosi(dac_mosi),  //DAC SPI-mosi output
      .done(dac_done),
      .DAC_begin(DAC_begin),
      .signle_done (signle_done),
      .data_reg(data_reg_q)
);
sine_wave_gen Base_DAC (
      .dacCount(EIS_DACcount_q),
      .offset(EIS_offset),
      .amplitude(EIS_amp),
      .data_out(EIS_reg_sine),
      .ReduceOSR(OSR_reduce)
);
Base_adc_ctrl base_adc (
      .MeasuEN(ADC_MeasuEN),  //Start measurement input
      .adc_busy(adc_busy), //ADC busy input
      .clk(clk),  //clock input
      .rst(adc_rst),  //reset input
```

```
      .adc_miso_A(adc_miso_A),  //ADC SPI-miso input for Channel
        A
      .adc_miso_B(adc_miso_B),  //ADC SPI-miso input for Channel
        B
      .adc_sclk(adc_slck),  //ADC SPI-clk output
      .adc_cs(adc_cs),  //ADC SPI-CS_bar output
      .adc_mosi(),  //ADC SPI-mosi output (not in use!)
      .CNVST_bar(adc_CNVST_bar),  //ADC start convertion trigger
        output
      .adc_channel(),  //ADC channel select (default value 2'b00)
      .adc_data_out_A(ADC_A),  //14 bit ADC channel A output
      .adc_data_out_B(ADC_B),   //14 bit ADC channel B output
      .done (adc_done)
);
ave_buff_new ave_buff(
      .rst(rst),
      .clk(clk),
      .ave_en(ave_en),
      .data_in_A(data_in_A_ave_q),  //input 14 bit
      .data_in_B(data_in_B_ave_q), // input 14 bit
      .index(ave_index_q), // input 9 bit
      .finalize(ave_final), // input finalize ave
      .data_out_ave_A(data_out_ave_A), //output 14 bit
      .data_out_ave_B(data_out_ave_B), // output 14 bit
      .done(ave_done)  //output 14 bit
   );
Set_Mode set_mode(
```

```
      .rx_data(rx_data),
      .new_rx_data(new_rx_data),
      .clk(clk),
      .rst(rst),
      .Mode(Mode)
   );
Set_Para set_para(
      .rx_data(rx_data),
      .new_rx_data(new_rx_data),
      .clk(clk),
      .rst(rst),
      .mode(Mode),
      .CV_cycle(CV_cycle), //8 bit output CV number of cycle
      .CV_up(CV_up), //12 bit output CV maximum
      .CV_down(CV_down), //12 bit output CV minimum
      .CV_step(CV_step), //12 bit output CV step
      .CV_delay(CV_delay), //32 bit output CV delay
      .EIS_amp(EIS_amp), //12 bit output EIS amplitude
      .EIS_offset(EIS_offset), //12 bit output EIS/amp offset
      .EIS_freq(EIS_freq),
      .EIS_mem_w(EIS_mem_w),
      .EIS_freq_index(EIS_freq_index),
      .EIS_num_points(EIS_num_points), //8 bit output EIS
        datapoints
      .AMP_delay(AMP_delay), // output 32 bit
      .AMP_Gain_1(AMP_Gain_1), // output 8 bit
      .AMP_Gain_2(AMP_Gain_2), // output 8 bit
```

```
      .AMP_Gain_C(AMP_Gain_C),
      .done(done_set)
   );
always @(*) begin
      state_d=state_q;
      DAC_START_d=DAC_START_q;
      DACindex_d= DACindex_q;
      done_d=done_q;
      Measure_done_d=Measure_done_q;
      ctr_d=ctr_q;
      Read_mem_d=Read_mem_q;
      DAC_START=0;
      start_FFT=0;
      en_fft_mem_w=0;
      xn_index_Mem=0;
      //addr_rd_fft_d=addr_rd_fft_q;
      addr_rd_fft=OSR_reduce;
      tan_sclr=1;
      tan_ce=0;
      tan_nd=0;
      sqrt_nd=0;
      sqrt_ce=0;
      sqrt_sclr=0;
      en_Cal_mem_A=0;
      en_Cal_mem_B=0;
      DataReady=0;
      arctan_inv_flag_A=0;
```

```
arctan_inv_flag_B=0;
Divider_ce=0;
Divider_nd=0;
en_adcmem_w=0;
ADC_MeasuEN =0;
phase_out_A_buff_d=phase_out_A_buff_q;
phase_out_B_buff_d=phase_out_B_buff_q;
rd_fft_Mag_Dividend_d=rd_fft_Mag_Dividend_q;
rd_fft_Mag_Divisor_d=rd_fft_Mag_Divisor_q;
rd_fft_A_re_buff_d=rd_fft_A_re_buff_q;
rd_fft_A_im_buff_d=rd_fft_A_im_buff_q;
rd_fft_A_Mag_squared_d=rd_fft_A_Mag_squared_q;
BaseRelay_on=1;
data_reg_d=data_reg_q;
// Average buffer
ave_index_d=ave_index_q;
data_in_A_ave_d=data_in_A_ave_q;
data_in_B_ave_d=data_in_B_ave_q;
ave_en=0;
ave_final=0;
//AVR serial
debug_msg_d=debug_msg_q;
startPrint=0;
print_rst=0;
//CV reg;
direc_d=direc_q;
CV_cycle_count_d=CV_cycle_count_qïijŻ
```

```
EIS_DACcount_d=EIS_DACcount_q;
EIS_index_d=EIS_index_q;
multi_a=0;
multi_b=0;
ctr_EIS_d=ctr_EIS_q;
//ACG REG
BASE_PREMUX_A_d=BASE_PREMUX_A_q;
BASE_AMPMUX_A_d=BASE_AMPMUX_A_q;
BASE_AMPMUXC_A_d=BASE_AMPMUXC_A_q;
ACG_down_temp_d=ACG_down_temp_q;
ACG_up_temp_d=ACG_up_temp_q;
buffer_state_d=buffer_state_q;
adc_rst=0;
dac_rst=0;
if(rst==1) dac_rst=1;
mag_max_d=mag_max_q;
mag_min_d=mag_min_q;
mag_eis_acg_d=mag_eis_acg_q;
//Convert ADC out from 2's complementary to true form
ADC_A_true[12:0]=ADC_A[12:0];
ADC_B_true[12:0]=ADC_B[12:0];
if(ADC_A[13]==0) begin
      ADC_A_true[13]=1'b1;
end
else if(ADC_A[13]==1) begin
      ADC_A_true[13]=1'b0;
end
```

```
if(ADC_B[13]==0) begin
      ADC_B_true[13]=1'b1;
end
else if(ADC_B[13]==1) begin
      ADC_B_true[13]=1'b0;
end
case (state_q)
      IDLE:begin
            BaseRelay_on=0;
            ctr_d=0;
            done_d=0;
            ctr_EIS_d=0;
            adc_rst=1;
            data_reg_d=Common_mode;
            EIS_DACcount_d=0; // for EIS
            EIS_index_d=1'd0;
            BASE_PREMUX_A_d=3'd0;
            BASE_AMPMUX_A_d=3'd0;
            BASE_AMPMUXC_A_d=3'd0;
            if(done_set==1) begin
                  state_d=ModeSelect;
            end
      end
      ModeSelect:begin
            case(Mode)
                  4'd1: begin
                        state_d=IDLE_CV_ACG;
```

```
                end
                4'd2: begin
                      state_d=IDLE_EIS;
                end
                4'd3: state_d=IDLE_AMP;
        endcase
end
//CV ACG
IDLE_CV_ACG:begin
        //Find bigger amplitude
        if(CV_down<Common_mode) begin
                ACG_down_temp_d=Common_mode-CV_down;
        end
        else if(CV_down>=Common_mode) begin
                ACG_down_temp_d=CV_down-Common_mode;
        end
        else if(CV_up>Common_mode) begin
                ACG_up_temp_d=CV_up-Common_mode;
        end
        else if(CV_up<=Common_mode) begin
                ACG_up_temp_d=Common_mode-CV_up;
        end
        state_d=IDLE_CV_ACG2;
        if(Debuger_mode) begin
                debug_msg_d={"ST01"};
                buffer_state_d=IDLE_CV_ACG2;
                state_d=Print;
```

```
        end
    end
    IDLE_CV_ACG2:begin
        ctr_d=ctr_q+1;
        //Find bigger amplitude
        if(ACG_up_temp_q>=ACG_down_temp_q) begin
            data_reg_d=CV_up;
        end
        else if(ACG_up_temp_q<ACG_down_temp_q) begin
            data_reg_d=CV_down;
        end
        if(ctr_q>ACG_delay_begin) begin
            state_d=DAC_CV_ACG;
            ctr_d=0;
            if(Debuger_mode) begin
                debug_msg_d={2'b00,ACG_up_temp_q,2'
                    b00,ACG_down_temp_q};
                buffer_state_d=DAC_CV_ACG;
                state_d=Print;
            end
        end
    end
    DAC_CV_ACG:begin
        DAC_START =1;
        if (DAC_begin==1 && signle_done==1) begin
            DAC_START =0;
            state_d=ADC_CV_ACG;
```

```
                ctr_d=0;
                ave_index_d=0;
                if(Debuger_mode) begin
                      debug_msg_d={"In",4'b0000,
                         data_reg_q};
                      buffer_state_d=ADC_CV_ACG;
                      state_d=Print;
                end
          end
    end
    ADC_CV_ACG:begin
          ADC_MeasuEN =1;
          ave_en=1;
          if (adc_done==1) begin
                adc_rst=1;
                ADC_MeasuEN =0;
                data_in_A_ave_d=ADC_A_true;
                data_in_B_ave_d=ADC_B_true;
                state_d=ADC_CV_delay_ACG;
                ctr_d=0;
                if(Debuger_mode) begin
                      debug_msg_d={2'b00,ADC_A_true,2'b00
                         ,ADC_B_true};
                      buffer_state_d=ADC_CV_delay_ACG;
                      state_d=Print;
                end
          end
```

```
end
ADC_CV_delay_ACG:begin
      ctr_d=ctr_q+1;
      ave_en=1;
      ADC_MeasuEN =1;
      if(ctr_q<CV_delay_ACG) begin //Setup CV scan
         rate
            if (adc_done==1) begin
                  adc_rst=1;
                  ave_index_d=ave_index_q+1;
                  data_in_A_ave_d=ADC_A_true;
                  data_in_B_ave_d=ADC_B_true;
                  state_d=ADC_CV_delay_ACG;
                  ADC_MeasuEN =0;
            end
      end
      else if (ctr_q>CV_delay_ACG) begin //Setup CV
         scan rate
            adc_rst=1;
            ave_final=1;
            state_d=Decide_ACG_1;
            ctr_d=0;
      end
end
Decide_ACG_1:begin
      if(ave_done==1) begin
```

```
if(data_out_ave_B<(8192+ACG_th_l) &&
   data_out_ave_B>(8192-ACG_th_l) &&
   BASE_PREMUX_A_q<3'd7) begin
      BASE_PREMUX_A_d=BASE_PREMUX_A_q+1;
      debug_msg_d={"G1+",1'b0,
         BASE_PREMUX_A_q,1'b0,
         BASE_AMPMUX_A_q};
      buffer_state_d=IDLE_CV_ACG;
      state_d=Print;
end
else if(data_out_ave_B<(8192+ACG_th_l) &&
    data_out_ave_B>(8192-ACG_th_l) &&
   BASE_PREMUX_A_q==3'd7) begin
      debug_msg_d={"1OV",1'b0,
         BASE_PREMUX_A_q,1'b0,
         BASE_AMPMUX_A_q}; //Gain
         overflow, reaching maximum gain
      buffer_state_d=Decide_ACG_2;
      state_d=Print;
end
else if (data_out_ave_B>=(8192+ACG_th_h)
   || data_out_ave_B<=(8192-ACG_th_h)&&
   BASE_PREMUX_A_q>3'd0) begin
      BASE_PREMUX_A_d=BASE_PREMUX_A_q-1;
      debug_msg_d={"G1-",1'b0,
         BASE_PREMUX_A_q,1'b0,
         BASE_AMPMUX_A_q};
```

```
                buffer_state_d=IDLE_CV_ACG;
                state_d=Print;
            end
            else if (data_out_ave_B>=(8192+ACG_th_h)
               || data_out_ave_B<=(8192-ACG_th_h)&&
               BASE_PREMUX_A_q==3'd0) begin
                debug_msg_d={"1UD",1'b0,
                   BASE_PREMUX_A_q,1'b0,
                   BASE_AMPMUX_A_q}; //Gain
                   underflow, reaching minimum gain
                buffer_state_d=Decide_ACG_2;
                state_d=Print;
            end
            else begin
                debug_msg_d={"G1=",1'b0,
                   BASE_PREMUX_A_q,1'b0,
                   BASE_AMPMUX_A_q};
                buffer_state_d=Decide_ACG_2;
                state_d=Print;
            end
        end
    end
    Decide_ACG_2:begin
        if(data_out_ave_B<(8192+ACG_th_l2) &&
           data_out_ave_B>(8192-ACG_th_l2) &&
           BASE_AMPMUX_A_q<3'd7) begin
              BASE_AMPMUX_A_d=BASE_AMPMUX_A_q+1;
```

```
            debug_msg_d={"G2+",1'b0,BASE_PREMUX_A_q
               ,1'b0,BASE_AMPMUX_A_q};
            buffer_state_d=IDLE_CV_ACG;
            state_d=Print;
         end
         else if(data_out_ave_B<(8192+ACG_th_l2) &&
            data_out_ave_B>(8192-ACG_th_l2) &&
            BASE_AMPMUX_A_q==3'd7) begin
            debug_msg_d={"2OV",1'b0,BASE_PREMUX_A_q
               ,1'b0,BASE_AMPMUX_A_q}; //Gain
               overflow, reaching maximum gain
            buffer_state_d=CV_ACG_Rdy;
            state_d=Print;
         end
         else if (data_out_ave_B>=(8192+ACG_th_h2) ||
            data_out_ave_B<=(8192-ACG_th_h2)&&
            BASE_AMPMUX_A_q>3'd0) begin
            BASE_AMPMUX_A_d=BASE_AMPMUX_A_q-1;
            debug_msg_d={"G2-",1'b0,BASE_PREMUX_A_q
               ,1'b0,BASE_AMPMUX_A_q};
            buffer_state_d=IDLE_CV_ACG;
            state_d=Print;
         end
         else if (data_out_ave_B>=(8192+ACG_th_h2) ||
            data_out_ave_B<=(8192-ACG_th_h2)&&
            BASE_AMPMUX_A_q==3'd0) begin
```

```
                debug_msg_d={"2UD",1'b0,BASE_PREMUX_A_q
                   ,1'b0,BASE_AMPMUX_A_q}; //Gain
                   underflow, reaching minimum gain
                buffer_state_d=CV_ACG_Rdy;
                state_d=Print;
          end
          else begin
                debug_msg_d={"G2=",1'b0,BASE_PREMUX_A_q
                   ,1'b0,BASE_AMPMUX_A_q};
                buffer_state_d=CV_ACG_Rdy;
                state_d=Print;
          end
    end
    CV_ACG_Rdy:begin
          debug_msg_d={"CVRd"};
          buffer_state_d=IDLE_CV;
          state_d=Print;
    end
    IDLE_CV:begin
          BaseRelay_on=1;
          ctr_d=ctr_q+1;
          direc_d=0;
          CV_cycle_count_d=0;
          if(ctr_q>2000) begin
                debug_msg_d={"G=",5'b0,BASE_PREMUX_A_q,5'
                   b0,BASE_AMPMUX_A_q};
                buffer_state_d=DAC_CV;
```

```
                state_d=Print;
                if(Debuger_mode) begin
                      debug_msg_d={"G=",5'b0,
                         BASE_PREMUX_A_q,5'b0,
                         BASE_AMPMUX_A_q};
                      buffer_state_d=DAC_CV;
                      state_d=Print;
                end
          end
          data_reg_d=CV_down;//CV lower boundary
    end
    DAC_CV:begin
          DAC_START =1;
          if (DAC_begin==1 && signle_done==1) begin
                if({1'b0,data_reg_q}+{1'b0,CV_step}<=
                   CV_up && direc_q==0) begin
                      data_reg_d=data_reg_q+CV_step; //
                         Set up CV step size
                end
                else if({1'b0,data_reg_q}>=CV_down+{1'b0,
                   CV_step} && direc_q==1) begin
                      data_reg_d=data_reg_q-CV_step; //
                         Set up CV step size
                end
                      DAC_START =0;
                      state_d=ADC_CV;
                      ctr_d=0;
```

```
                ave_index_d=0;
                if(Debuger_mode) begin
                    debug_msg_d={"In",4'b0000,
                        data_reg_q};
                    buffer_state_d=ADC_CV;
                    state_d=Print;
                end
            if ({1'b0,data_reg_q}+{1'b0,CV_step}>
               CV_up && direc_q==0) begin
                    direc_d=1;
                    data_reg_d=CV_up;
                    DAC_START =0;
            end
            else if ({1'b0,data_reg_q}<CV_down+{1'b0,
               CV_step} && direc_q==1) begin
                    direc_d=0;
                    data_reg_d=CV_down;
                    DAC_START =0;
                    CV_cycle_count_d=CV_cycle_count_q
                        +1;
            end
        end
    end
    ADC_CV:begin
        ADC_MeasuEN =1;
        ave_en=1;
        if (adc_done==1) begin
```

```
                adc_rst=1;
                ADC_MeasuEN =0;
                data_in_A_ave_d=ADC_A_true;
                data_in_B_ave_d=ADC_B_true;
                state_d=ADC_CV_delay;
                ctr_d=0;
                if(Debuger_mode) begin
                      debug_msg_d={2'b00,ADC_A_true,2'b00
                         ,ADC_B_true};
                      buffer_state_d=ADC_CV_delay;
                      state_d=Print;
                end
          end
    end
    ADC_CV_delay:begin
          ctr_d=ctr_q+1;
          ave_en=1;
          ADC_MeasuEN =1;
          if(ctr_q<CV_delay) begin //Setup CV scan rate
                if (adc_done==1) begin
                      adc_rst=1;
                      ave_index_d=ave_index_q+1;
                      data_in_A_ave_d=ADC_A_true;
                      data_in_B_ave_d=ADC_B_true;
                      state_d=ADC_CV_delay;
                      ADC_MeasuEN =0;
                      if(Debuger_mode) begin
```

```
                        debug_msg_d={2'b00,ADC_A_true
                           ,2'b10,ADC_B_true};
                        buffer_state_d=ADC_CV_delay;
                        state_d=Print;
                    end
                end
            end
            else if (ctr_q>=CV_delay) begin //Setup CV scan
                rate
                adc_rst=1;
                ave_final=1;
                state_d=Send_out_CV;
                ctr_d=0;
            end
        end
        Send_out_CV:begin
            if(ave_done==1) begin
                debug_msg_d={2'd0,data_out_ave_A,2'd0,
                   data_out_ave_B};
                state_d=Send_out_CV_final;
                print_rst=0;
            end
        end
        Send_out_CV_final:begin
            startPrint=1;
            if(done_print==1) begin
                state_d=DAC_CV;
```

```
                print_rst=1;
                startPrint=0;
                ctr_d=0;
                if(CV_cycle_count_q==CV_cycle) begin
                      state_d=Send_CV_end;
                      debug_msg_d={"CVND"};
                end
          end
    end
    Send_CV_end:begin
          startPrint=1;
          if(done_print==1) begin
                state_d=Reset_CM;
                print_rst=1;
                startPrint=0;
                ctr_d=0;
                done_d=1;
          end
    end
    //CV part end

    ///EIS part begin
    IDLE_EIS:begin
          BaseRelay_on=1;
          ctr_d=ctr_q+1;
          CV_cycle_count_d=0;
          if(ctr_q>500000) begin //5000000
```

```
                state_d=DAC_EIS;
                data_reg_d=EIS_reg_sine;//CV lower
                   boundary
                if(Debuger_mode) begin
                      debug_msg_d={"ST",4'b0,data_reg_q};
                      buffer_state_d=DAC_EIS;
                      state_d=Print;
                end
          end
    end
    DAC_EIS:begin
          data_reg_d=EIS_reg_sine;
          DAC_START =1;
          Measure_done_d=0;
          if (DAC_begin==1 && signle_done==1) begin
                DAC_START=0;
                state_d = ADC_EIS;
                ctr_d=0;
                if(Debuger_mode) begin
                      debug_msg_d={"In",4'b0,data_reg_q};
                      buffer_state_d=ADC_EIS;
                      state_d=Print;
                end
          end
    end
    ADC_EIS:begin
          ADC_MeasuEN =1;
```

```
        ave_en=1;
        if (adc_done==1) begin
              adc_rst=1;
              ADC_MeasuEN =0;
              data_in_A_ave_d=ADC_A_true;
              data_in_B_ave_d=ADC_B_true;
              state_d=ADC_EIS_delay;
              ctr_d=0;
              if(Debuger_mode) begin
                    debug_msg_d={2'b0,ADC_A_true,2'b0,
                       ADC_B_true};
                    buffer_state_d=ADC_EIS_delay;
                    state_d=Print;
              end
        end
end
ADC_EIS_delay:begin
        ctr_d=ctr_q+1;
        ave_en=1;
        ADC_MeasuEN =1;
        if(ctr_q<EIS_delay_q) begin //Setup CV scan
           rate
              if (adc_done==1) begin
                    adc_rst=1;
                    ave_index_d=ave_index_q+1;
                    ADC_MeasuEN =0;
                    data_in_A_ave_d=ADC_A_true;
```

```
                    data_in_B_ave_d=ADC_B_true;
                    state_d=ADC_EIS_delay;
                end
            end
            else if (ctr_q>=EIS_delay_q) begin //Setup CV
               scan rate
                adc_rst=1;
                ave_final=1;
                state_d=wait_ave_EIS;
                ctr_d=0;
            end
        end
        wait_ave_EIS:begin
            if(ave_done==1) begin
                state_d=wait_ADC_ram;
                if(Debuger_mode) begin
                    debug_msg_d={2'b0,data_out_ave_A,2'
                       b0,data_out_ave_B};
                    buffer_state_d=wait_ADC_ram;
                    state_d=Print;
                end
            end
        end
        wait_ADC_ram:begin
            ctr_d=ctr_q+1;
            en_adcmem_w=1;
            if (ctr_q==1) begin
```

```
        en_adcmem_w=0;
    end
    if (ctr_q>=2) begin
        en_adcmem_w=0;
        EIS_DACcount_d=EIS_DACcount_q+1;
        if (EIS_DACcount_q==255)begin
            state_d = EIS_ACG_init;
            en_adcmem_w=0;
            Measure_done_d=1;
            ctr_d=0;
            if(Debuger_mode) begin
                debug_msg_d={2'b0,
                   data_out_ave_A,2'b0,
                   data_out_ave_B};
                buffer_state_d=EIS_ACG_init;
                state_d=Print;
            end
        end
        else begin
            state_d = DAC_EIS;
            en_adcmem_w=0;
            if(Debuger_mode) begin
                debug_msg_d={2'b0,
                   data_out_ave_A,2'b0,
                   data_out_ave_B};
                buffer_state_d=DAC_EIS;
                state_d=Print;
```

```
                end
            end
        end
    end
    EIS_ACG_init:begin
        xn_index_Mem=ctr_q[7:0];
        mag_max_d=mem_B;
        mag_min_d=mem_B;
        state_d=EIS_ACG_assign;
        if(Debuger_mode) begin
            debug_msg_d={"A",ctr_q[7:0],mem_B};
            buffer_state_d=EIS_ACG_assign;
            state_d=Print;
        end
    end
    EIS_ACG_assign:begin
        xn_index_Mem=ctr_q[7:0];
        ctr_d=ctr_q+1;
        state_d=EIS_ACG_assign_wait;
        if(Debuger_mode) begin
            debug_msg_d={mag_max_q,mag_min_q};
            buffer_state_d=EIS_ACG_assign_wait;
            state_d=Print;
        end
    end
    EIS_ACG_assign_wait:begin
        xn_index_Mem=ctr_q[7:0];
```

```
            state_d=EIS_ACG_compare;
            if(Debuger_mode) begin
                  debug_msg_d={"Ad=",ctr_q[7:0]};
                  buffer_state_d=EIS_ACG_compare;
                  state_d=Print;
            end
      end
      EIS_ACG_compare:begin
            xn_index_Mem=ctr_q[7:0];
            state_d=EIS_ACG_assign;
            if(mem_B>=mag_max_q) mag_max_d=mem_B;
            if(mem_B<mag_min_q) mag_min_d=mem_B;
            if(ctr_q==255) begin
                  state_d=EIS_ACG_decide1;
                  if(Debuger_mode) begin
                        debug_msg_d={"CPND"};
                        buffer_state_d=EIS_ACG_decide1;
                        state_d=Print;
                  end
            end
            else begin
                  state_d=EIS_ACG_assign;
                  if(Debuger_mode) begin
                        debug_msg_d={mag_max_q,mag_min_q};
                        buffer_state_d=EIS_ACG_assign;
                        state_d=Print;
                  end
```

```
        end
    end
    EIS_ACG_decide1:begin
        if(mag_min_q<Common_mode_EIS) begin
            ACG_down_temp_d=Common_mode_EIS-mag_min_q
              ;
        end
        else if(mag_min_q>=Common_mode_EIS) begin
            ACG_down_temp_d=mag_min_q-Common_mode_EIS
              ;
        end
        else if(mag_max_q>Common_mode_EIS) begin
            ACG_up_temp_d=mag_max_q-Common_mode_EIS;
        end
        else if(mag_max_q<=Common_mode_EIS) begin
            ACG_up_temp_d=Common_mode_EIS-mag_max_q;
        end
        state_d=EIS_ACG_decide2;
        if(Debuger_mode) begin
                debug_msg_d={mag_max_q,mag_min_q};
                buffer_state_d=EIS_ACG_decide2;
                state_d=Print;
        end
    end
    EIS_ACG_decide2:begin
        if(ACG_up_temp_q>=ACG_down_temp_q) begin
            mag_eis_acg_d=mag_max_q;
```

```
        end
        else if(ACG_up_temp_q<ACG_down_temp_q) begin
              mag_eis_acg_d=mag_min_q;
        end
        state_d=EIS_Decide_ACG_1;
        if(Debuger_mode) begin
                    debug_msg_d={ACG_up_temp_q,
                       ACG_down_temp_q};
                    buffer_state_d=EIS_Decide_ACG_1;
                    state_d=Print;
        end
    end
    EIS_Decide_ACG_1:begin
        if(mag_eis_acg_q<(8192+ACG_th_l) &&
           mag_eis_acg_q>(8192-ACG_th_l) &&
           BASE_PREMUX_A_q<3'd7) begin
              BASE_PREMUX_A_d=BASE_PREMUX_A_q+1;
              if(Debuger_mode) begin
                    debug_msg_d={"G1+",2'b0,
                       BASE_PREMUX_A_q,BASE_AMPMUX_A_q
                       };
                    buffer_state_d=IDLE_EIS;
                    state_d=Print;
              end
              else state_d=IDLE_EIS;;
        end
```

```
else if(mag_eis_acg_q<(8192+ACG_th_l) &&
   mag_eis_acg_q>(8192-ACG_th_l) &&
   BASE_PREMUX_A_q==3'd7) begin
      if(Debuger_mode) begin
                        debug_msg_d={"1OV",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=
                           EIS_Decide_ACG_2;
                        state_d=Print;
      end
      else state_d=EIS_Decide_ACG_2;
end
else if (mag_eis_acg_q>=(8192+ACG_th_h) ||
   mag_eis_acg_q<=(8192-ACG_th_h)&&
   BASE_PREMUX_A_q>3'd0) begin
      BASE_PREMUX_A_d=BASE_PREMUX_A_q-1;
      if(Debuger_mode) begin
                        debug_msg_d={"G1-",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=IDLE_EIS
                           ;
                        state_d=Print;
      end
      else state_d=IDLE_EIS;
end
```

```
        else if (mag_eis_acg_q>=(8192+ACG_th_h) ||
           mag_eis_acg_q<=(8192-ACG_th_h)&&
           BASE_PREMUX_A_q==3'd0) begin
              if(Debuger_mode) begin
                                 debug_msg_d={"1UD",1'b0
                                    ,BASE_PREMUX_A_q,1'
                                    b0,BASE_AMPMUX_A_q};
                                 buffer_state_d=
                                    EIS_Decide_ACG_2;
                                 state_d=Print;
              end
              else state_d=EIS_Decide_ACG_2;
        end
        else begin
              if(Debuger_mode) begin
                                 debug_msg_d={"G1=",1'b0
                                    ,BASE_PREMUX_A_q,1'
                                    b0,BASE_AMPMUX_A_q};
                                 buffer_state_d=
                                    EIS_Decide_ACG_2;
                                 state_d=Print;
              end
              else state_d=EIS_Decide_ACG_2;
        end
  end
  EIS_Decide_ACG_2:begin
```

```
if(mag_eis_acg_q<(8192+ACG_th_l2) &&
   mag_eis_acg_q>(8192-ACG_th_l2) &&
   BASE_AMPMUX_A_q<3'd7) begin
      BASE_AMPMUX_A_d=BASE_AMPMUX_A_q+1;
      if(Debuger_mode) begin
                        debug_msg_d={"G2+",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=IDLE_EIS
                           ;
                        state_d=Print;
      end
      else state_d=IDLE_EIS;
end
else if(mag_eis_acg_q<(8192+ACG_th_l2) &&
   mag_eis_acg_q>(8192-ACG_th_l2) &&
   BASE_AMPMUX_A_q==3'd7) begin
      if(Debuger_mode) begin
                        debug_msg_d={"2OV",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=
                           EIS_signle_done;
                        state_d=Print;
      end
      else state_d=EIS_signle_done;
end
```

```
else if (mag_eis_acg_q>=(8192+ACG_th_h2) ||
   mag_eis_acg_q<=(8192-ACG_th_h2)&&
   BASE_AMPMUX_A_q>3'd0) begin
      BASE_AMPMUX_A_d=BASE_AMPMUX_A_q-1;
      if(Debuger_mode) begin
                        debug_msg_d={"G2-",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=IDLE_EIS
                           ;
                        state_d=Print;
      end
      else state_d=IDLE_EIS;
end
else if (mag_eis_acg_q>=(8192+ACG_th_h2) ||
   mag_eis_acg_q<=(8192-ACG_th_h2)&&
   BASE_AMPMUX_A_q==3'd0) begin
      if(Debuger_mode) begin
                        debug_msg_d={"2UD",1'b0
                           ,BASE_PREMUX_A_q,1'
                           b0,BASE_AMPMUX_A_q};
                        buffer_state_d=
                           EIS_signle_done;
                        state_d=Print;
      end
      else state_d=EIS_signle_done;
end
```

```
        else begin
              if(Debuger_mode) begin
                                    debug_msg_d={"G2=",1'b0
                                       ,BASE_PREMUX_A_q,1'
                                       b0,BASE_AMPMUX_A_q};
                                    buffer_state_d=
                                       EIS_signle_done;
                                    state_d=Print;
              end
              else state_d=EIS_signle_done;
        end
  end
  EIS_signle_done:begin
        start_FFT=1;
        if (FFT_rfd==1 && edone_FFT!=1) begin
              xn_index_Mem=xn_index+1;
        end
        else if (FFT_rfd==0) begin
              xn_index_Mem=0;
        end
        if (FFT_rfd!=1 && edone_FFT==1) begin
              state_d=EIS_FFT_collect;
              //start_FFT=1;
              xn_index_Mem=0;
              if(Debuger_mode) begin
                    debug_msg_d={"FFT1"};
                    buffer_state_d=EIS_FFT_collect;
```

```
                        state_d=Print;
                  end
            end
      end
      EIS_FFT_collect:begin
            xn_index_Mem=0;
            if (dv_FFT==1) begin
                  state_d=EIS_FFT_Finish;
                  en_fft_mem_w=1;
                  if(Debuger_mode) begin
                        debug_msg_d={"FFT2"};
                        buffer_state_d=EIS_FFT_Finish;
                        state_d=Print;
                  end
            end
      end
      EIS_FFT_Finish:begin
            en_fft_mem_w=1;
            if (dv_FFT==0) begin
                  state_d= Send_out_EIS_Gain;
                  en_fft_mem_w=0;
                  ctr_d=0;
                  if(Debuger_mode) begin
                        debug_msg_d={"FFT3"};
                        buffer_state_d=Send_out_EIS_Gain;
                        state_d=Print;
                  end
```

```
      end
end
Send_out_EIS_Gain:begin
      debug_msg_d={"G=",5'b0,BASE_PREMUX_A_q,5'b0,
         BASE_AMPMUX_A_q};
      state_d=Print;
      buffer_state_d=Send_out_EIS_A;
end
Send_out_EIS_A:begin
      debug_msg_d={rd_fft_A_re,rd_fft_A_im};
      state_d=Print;
      buffer_state_d=Send_out_EIS_B;
end
Send_out_EIS_B:begin
      debug_msg_d={rd_fft_B_re,rd_fft_B_im};
      state_d=Print;
      buffer_state_d=DAC_EIS;
      EIS_index_d=EIS_index_q+1;
      if(EIS_num_points==0) begin
            ctr_EIS_d=ctr_EIS_q+1;
            if(ctr_EIS_q>=0)
                  buffer_state_d=Send_EIS_end;
      end
      else if(EIS_num_points>0)begin
            EIS_index_d=EIS_index_q+1;
            if(EIS_index_q==EIS_num_points)
               buffer_state_d=Send_EIS_end;
```

```
                end
            end
            Send_EIS_end:begin
                        debug_msg_d="EISD";
                        state_d=Print;
                        buffer_state_d=Reset_CM;
                        dac_rst=1;
            end
////////////////////
//Amperametry mode//
////////////////////
            IDLE_AMP:begin
                  BaseRelay_on=1;
                  ctr_d=ctr_q+1;
                  BASE_PREMUX_A_d=AMP_Gain_1[2:0];
                  BASE_AMPMUX_A_d=AMP_Gain_2[2:0];
                  BASE_AMPMUXC_A_d=AMP_Gain_C[2:0];
                  if(ctr_q>200000) begin
                        state_d=DAC_AMP;
                        ctr_d=0;
                  end
                  data_reg_d=EIS_offset;//CV lower boundary
            end
            DAC_AMP:begin
                  DAC_START =1;
                  if (DAC_begin==1 && signle_done==1) begin
                        DAC_START =0;
```

```
            //ADC_MeasuEN =1;
            state_d=ADC_AMP;
            ctr_d=1;
      end
end
ADC_AMP:begin
      ADC_MeasuEN =1;
      ave_en=1;
      if (adc_done==1) begin
            adc_rst=1;
            ADC_MeasuEN =0;
            state_d=ADC_AMP_delay;
            data_in_A_ave_d=ADC_A_true;
            data_in_B_ave_d=ADC_B_true;
            ctr_d=0;
      end
end
ADC_AMP_delay:begin
      ctr_d=ctr_q+1;
      ave_en=1;
      ADC_MeasuEN =1;
      if(ctr_q<AMP_delay) begin //Setup CV scan rate
            if (adc_done==1) begin
                  adc_rst=1;
                  ave_index_d=ave_index_q+1;
                  data_in_A_ave_d=ADC_A_true;
                  data_in_B_ave_d=ADC_B_true;
```

```
                        state_d=ADC_AMP_delay;
                        ADC_MeasuEN =0;
                    end
                end
                else if (ctr_q>=AMP_delay) begin //Setup CV
                   scan rate
                    adc_rst=1;
                    ave_final=1;
                    state_d=Send_out_AMP;
                    ctr_d=0;
                end
            end
            Send_out_AMP:begin
                if(ave_done==1) begin
                    debug_msg_d={2'd0,data_out_ave_A,2'd0,
                       data_out_ave_B};
                    state_d=Print;
                    print_rst=0;
                    buffer_state_d=ADC_AMP;
                    //startPrint=1;
                end
            end
////////NOt in use
            END_AMP:begin
                debug_msg_d={"AMPD"};
                buffer_state_d=IDLE;
                state_d=Print;
```

```
            end
///////NOt in use
//////////////////
//General states//
//////////////////
            Print:begin
                  startPrint=1;
                  if(done_print==1) begin
                        state_d=buffer_state_q;
                        print_rst=1;
                        startPrint=0;
                        //ctr_d=0;
                  end
            end
            Reset_CM:begin
                  data_reg_d=Common_mode;
                  DAC_START =1;
                  if (DAC_begin==1 && signle_done==1) begin
                        DAC_START=0;
                        state_d = IDLE;
                        ctr_d=0;
                  end
            end
      endcase
end
always @(posedge clk) begin
      if (rst) begin
```

```
state_q <= IDLE;
DAC_START_q <=0;
ADC_MeasuEN_q <=0;
DACindex_q <=0;
done_q <=0;
Measure_done_q<=0;
ctr_q<=0;
Read_mem_q<=0;
phase_out_A_buff_q<=0;
phase_out_B_buff_q<=0;
rd_fft_Mag_Dividend_q<=0;
rd_fft_Mag_Divisor_q<=0;
rd_fft_A_re_buff_q<=0;
rd_fft_A_im_buff_q<=0;
rd_fft_A_Mag_squared_q<=0;
ave_index_q<=0;
data_reg_q<=0;
data_in_A_ave_q<=0;
data_in_B_ave_q<=0;
debug_msg_q<=0;
direc_q<=0;
CV_cycle_count_q<=0;
EIS_DACcount_q<=0;
EIS_index_q<=0;
ctr_EIS_q<=0;
//ACG part
BASE_PREMUX_A_q<=0;
```

```
        BASE_AMPMUX_A_q<=0;
        BASE_AMPMUXC_A_q<=0;
        buffer_state_q<=0;
        mag_max_q<=0;
        mag_min_q<=0;
        mag_eis_acg_q<=0;
        ACG_down_temp_q<=0;
        ACG_up_temp_q<=0;
    end else begin
        state_q <= state_d;
        done_q <=done_d;
        DAC_START_q <=DAC_START_d;
        DACindex_q <=DACindex_d;
        Measure_done_q<=Measure_done_d;
        ctr_q<=ctr_d;
        Read_mem_q<=Read_mem_d;
        rd_fft_A_Mag_squared_q<=rd_fft_A_Mag_squared_d;
        phase_out_A_buff_q<=phase_out_A_buff_d;
        phase_out_B_buff_q<=phase_out_B_buff_d;
        rd_fft_Mag_Dividend_q<=rd_fft_Mag_Dividend_d;
        rd_fft_Mag_Divisor_q<=rd_fft_Mag_Divisor_d;
        rd_fft_A_re_buff_q<=rd_fft_A_re_buff_d;
        rd_fft_A_im_buff_q<=rd_fft_A_im_buff_d;
        ave_index_q<=ave_index_d;
        data_reg_q<=data_reg_d;
        data_in_A_ave_q<=data_in_A_ave_d;
        data_in_B_ave_q<=data_in_B_ave_d;
```

```
            debug_msg_q<=debug_msg_d;
            direc_q<=direc_d;
            CV_cycle_count_q<=CV_cycle_count_d;
            //EIS part
            EIS_DACcount_q<=EIS_DACcount_d;
            EIS_index_q<=EIS_index_d;
            ctr_EIS_q<=ctr_EIS_d;
            //ACG part
            BASE_PREMUX_A_q<=BASE_PREMUX_A_d;
            BASE_AMPMUX_A_q<=BASE_AMPMUX_A_d;
            BASE_AMPMUXC_A_q<=BASE_AMPMUXC_A_d;
            buffer_state_q<=buffer_state_d;
            mag_max_q<=mag_max_d;
            mag_min_q<=mag_min_d;
            mag_eis_acg_q<=mag_eis_acg_d;
            ACG_down_temp_q<=ACG_down_temp_d;
            ACG_up_temp_q<=ACG_up_temp_d;
      end
end
endmodule
```