

DISSERTATION

A BIOSENSOR SYSTEM WITH AN INTEGRATED CMOS MICROELECTRODE ARRAY  
FOR HIGH SPATIO-TEMPORAL ELECTROCHEMICAL IMAGING

Submitted by

William Tedjo

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: Thomas Chen

Stuart Tobet  
George Collins  
Jesse Wilson



Copyright by William Tedjo 2019

All Rights Reserved



## ABSTRACT

### A BIOSENSOR SYSTEM WITH AN INTEGRATED CMOS MICROELECTRODE ARRAY FOR HIGH SPATIO-TEMPORAL ELECTROCHEMICAL IMAGING

The ability to view biological events in real time has contributed significantly to research in life sciences. While optical microscopy is important to observe anatomical and morphological changes, it is equally important to capture real-time two-dimensional (2D) chemical activities that drive the bio-sample behaviors. The existing chemical sensing methods (i.e. optical photoluminescence, magnetic resonance, and scanning electrochemical), are well-established and optimized for existing ex vivo or in vitro analyses. However, such methods also present various limitations in resolution, real-time performance, and costs. Electrochemical method has been advantageous to life sciences by supporting studies and discoveries in neurotransmitter signaling and metabolic activities in biological samples. In the meantime, the integration of Microelectrode Array (MEA) and Complementary-Metal-Oxide-Semiconductor (CMOS) technology to the electrochemical method provides biosensing capabilities with high spatial and temporal resolutions. This work discusses three related subtopics in this specific order: improvements to an electrochemical imaging system with 8,192 sensing points for neurotransmitter sensing; comprehensive design processes of an electrochemical imaging system with 16,064 sensing points based on the previous system; and the application of the system for imaging oxygen concentration gradients in metabolizing bovine oocytes.

The first attempt of high spatial electrochemical imaging was based on an integrated CMOS microchip with 8,192 configurable Pt surface electrodes, on-chip potentiostat, on-chip control logic, and a microfluidic device designed to support ex vivo tissue experimentation. Using norepinephrine as a target analyte for proof of concept, the system is capable of differentiating concentrations of norepinephrine as low as 8  $\mu\text{M}$  and up to 1,024  $\mu\text{M}$  with a linear response and a spatial resolution of 25.5  $\times$  30.4  $\mu\text{m}$ . Electrochemical imaging was performed using murine adrenal tissue as a biological model and successfully showed caffeine-stimulated release of catecholamines from live slices of adrenal tissue with desired spatial and temporal resolutions. This system demonstrates the capability of an electrochemical imaging system capable of capturing changes in chemical gradients in live tissue slices.



An enhanced system was designed and implemented in a CMOS microchip based on the previous generation. The enhanced CMOS microchip has an expanded sensing area of 3.6×3.6mm containing 16,064 Pt electrodes and the associated 16,064 integrated read channels. The novel three-electrode electrochemical sensor system designed at 27.5×27.5μm pitch enables spatially dense cellular level chemical gradient imaging. The noise level of the on-chip read channels allow amperometric linear detection of neurotransmitter (norepinephrine) concentrations from 4μM to 512μM with 4.7pA/μM sensitivity (R=0.98). Electrochemical response to dissolved oxygen concentration or oxygen partial pressure (pO<sub>2</sub>) was also characterized with deoxygenated deionized water containing 10μM to 165 μM pO<sub>2</sub> with 8.21pA/μM sensitivity (R=0.89). The enhanced biosensor system also demonstrates selectivity to different target analytes using cyclic voltammetry to simultaneously detect NE and uric acid. In addition, a custom-designed indium tin oxide and Au glass electrode is integrated into the microfluidic support system to enable pH measurement, ensuring viability of bio-samples in ex vivo experiments. Electrochemical images confirm the spatiotemporal performance at four frames per second while maintaining the sensitivity to target analytes. The overall system is controlled and continuously monitored by a custom-designed user interface, which is optimized for real-time high spatiotemporal resolution chemical bioimaging.

It is well known that physiological events related to oxygen concentration gradients provide valuable information to determine the state of metabolizing biological cells. Utilizing the CMOS microchip with 16,064 Pt MEA and an improved three-electrode system configuration, the system is capable of imaging low oxygen concentration with limit of detection of 18.3μM, 0.58mg/L, or 13.8mmHg. A modified microfluidic support system allows convenient bio-sample handling and delivery to the MEA surface for sensing. In vitro oxygen imaging experiments were performed using bovine cumulus-oocytes-complexes cells with custom software algorithms to analyze its flux density and oxygen consumption rate. The imaging results are processed and presented as 2D heatmaps, representing the dissolved oxygen concentration in the immediate proximity of the cell. The 2D images and analysis of oxygen consumption provide a unique insight into the spatial and temporal dynamics of cell metabolism.



## ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to Colorado State University, the College of Engineering and the Department of Electrical & Computer Engineering, for the opportunities and stipends during my graduate study. The CMOS microchip fabrication was generously supported by the engineers from Broadcom Incorporated (Fort Collins, CO, previously Avago Technologies) and partially funded by the National Science Foundation (GDE-0841259 and GDE-1450032). This work would not have been possible without their significant contributions.

For the professors – I would like to thank Dr. Thomas Chen for continuously recognizing my potential and providing the necessary resources and training. Besides my advisor, I would like to thank the rest of my committee members: Dr. Stuart Tobet, Dr. George Collins, and Dr. Jesse Wilson; and other faculties: Dr. Charles Henry (Chemistry), Dr. Elaine Carnevale and Dr. Adam Chicco (Biomedical Sciences). I appreciate your valuable time for sharing your expertise and providing mentorships.

For the fellows – my sincere thanks for their collaborations and stimulating discussions. In a chronological order, I would like to acknowledge: Kristy Scholfield, Tucker Kern, William Wilson, Abhiram Reddy, Samuel Wright, Lang Yang, Nicholas Grant, Yusra Obeidat, Ming-Hao Cheng, Caleb Begly, Daniel Ball, and Gitesh Kulkarni (Electrical and Computer Engineering); John Wydallis and Rachel Feeny (Chemistry); Jasmine Nejad (Biomedical Engineering); August DeMann (Physics); Luke Schwerdtfeger, Chad Eitel, Parvathy Thampi, and Giovana Catandi (Biomedical Sciences).

Finally, I would like to thank my families, the Tedjos, the Palczynskis, and the Lews, for their immeasurable care and support. Finally, I would like to thank Dr. Williana Basuki for believing in me, admitting my weaknesses, and catalyzing my strengths to help me to be a better person.



## TABLE OF CONTENTS

ABSTRACT .....	ii
ACKNOWLEDGMENTS .....	iv
LIST OF TABLES .....	ix
LIST OF FIGURES.....	x
CHAPTER I: INTRODUCTION .....	1
CHAPTER II: MOTIVATIONS AND EXISTING WORKS .....	5
2.1.    Electrochemical Measurement Overview.....	5
2.2.    Other Works in Electrochemistry and MEA .....	7
2.3.    Biosensor Generation 2 with CMOS MEA Microchip.....	9
2.3.1. Biosensor System .....	10
2.3.2. Microfluidic System .....	14
2.3.3. System Performance: Noise, Limit of Detection, and Dynamic Range .....	15
2.3.4. Processing and Visualization .....	18
2.3.5. Ex Vivo Electrochemical imaging.....	21
2.4.    Limitations and Baselines for Generation 3 .....	26
CHAPTER III: GENERATION 3 SYSTEM OVERVIEW.....	28
3.1.    System Level Configuration .....	28
3.2.    CMOS Microchip with MEA.....	29
3.3.    Board Level Integration .....	31
3.4.    Software Integration .....	33
3.5.    Microfluidics Integration .....	34
CHAPTER IV: CMOS MICROCHIP WITH MICROELECTRODE ARRAY .....	35
4.1.    Top Level Arrangement .....	35
4.2.    Operational Amplifier.....	40
4.3.    Potentiostat .....	42
4.4.    Transimpedance Amplifier .....	42



4.5.	Surface Electrodes.....	47
4.6.	Array Integration.....	47
4.7.	Global Bias Circuit.....	49
4.8.	Analog Multiplexer.....	50
4.9.	6 to 64 Decoder.....	52
CHAPTER V: CMOS MICROCHIP SIMULATIONS.....		53
5.1.	Operational Amplifier.....	56
5.2.	Trans-impedance Amplifier .....	60
5.3.	Top Level Simulation.....	65
CHAPTER VI: SYSTEM INTEGRATION: HARDWARE, SOFTWARE, AND MICROFLUIDIC .....		69
6.1.	Circuit Boards: Prototypes .....	70
6.2.	Circuit Boards: Final Design .....	73
6.3.	Software Integration .....	78
6.3.1.	Real-time Data Processing and Control .....	79
6.3.2.	Post-Processing and Visualization.....	81
6.4.	Microfluidic Platform.....	84
CHAPTER VII: SYSTEM VALIDATION: ELECTRICAL AND CHEMICAL MEASUREMENTS.....		87
7.1.	Electrical Performance .....	87
7.2.	Amperometry.....	89
7.3.	Voltammetry .....	91
7.4.	Electrical and Chemical Measurement Variations .....	93
7.5.	pH Response .....	94
7.6.	High Resolution Chemical Imaging.....	94
CHAPTER VIII: APPLICATION: OXYGEN IMAGING .....		96
8.1.	Oxygen Imaging Preparation .....	96
8.1.1.	MEA Improvements for Oxygen Imaging.....	97
8.1.2.	Bovine Cumulus-Oocytes-Complexes Handling and Preparation .....	98
8.1.3.	Experiment Setups.....	99



8.2.	Oxygen Sensing Results.....	100
8.2.1.	Electrochemical Responses of Oxygen .....	100
8.2.2.	COC Basal Respiration and the Effect of WEs Oxygen Reduction .....	102
8.2.3.	Heatmap Smoothing .....	106
8.2.4.	Comparison of Healthy and Dead COC.....	106
8.2.5.	Flux Density and Consumption Rate Analysis.....	110
CHAPTER IX: CONCLUSION & FUTURE WORKS.....		112
REFERENCES.....		116
APPENDIX A: CMOS MICROELECTRODE ARRAY MICROCHIP .....		128
A.1	Cadence Detailed Schematics.....	128
A.2.1	Entire CMOS Microchip .....	128
A.2.2	Quadrant .....	129
A.2.3	Transimpedance Amplifier: Operational Amplifier .....	130
A.2.4	Transimpedance Amplifier .....	131
A.2.5	Global Bias Circuit.....	132
A.2.6	Transmission Gate (TG) .....	133
A.2.7	Voltage Buffer .....	134
A.2.8	Potentiostat .....	135
A.2	Surface MEA.....	136
A.2.1	Dimension of MEA configuration.....	136
A.2.2	Entire Surface .....	137
A.3	External Pins.....	138
A.4	Cadence SKILL Scripts.....	140
APPENDIX B: PCB SCHEMATIC & LAYOUT .....		141
B.1.	PCB – Prototype board .....	141
B.2.	PCB – version 1 .....	143
B.3.	PCB – version 2 .....	146
B.4.	PCB – version 3 .....	153



B.5.	Bill of Material – version 3 .....	164
APPENDIX C: SOFTWARE - FIRMWARE AND SCRIPT .....		167
C.1.	MCU Firmware .....	167
C.2.	MATLAB Real-Time GUI.....	175
C.2.1.	MainGUI.m .....	175
C.2.2.	AmpCalDC.m .....	192
C.2.3.	AmpPotSweep.m .....	195
C.2.4.	CV.m .....	195
C.2.5.	enableAmp.m .....	199
C.2.6.	enableCV.m .....	201
C.2.7.	vMonitor.m .....	202
C.3.	MATLAB Post-Processing GUI .....	204
C.3.1.	prePlotProcess.m.....	204
C.3.2.	PlotGUI.m.....	207
C.3.3.	heatmapPlot.m .....	212
APPENDIX D: PUBLICATIONS .....		215
D.1.	Poster Presentations.....	215
D.2.	Conference and Journal Publications .....	218
LIST OF ABBREVIATIONS.....		219



## LIST OF TABLES

Table 1. Generation 2: Comparison of electrochemical biosensor system using CMOS MEA. ....	17
Table 2. List of CMOS MEA microchip pins and the corresponding coordinates. ....	38
Table 3. Operational amplifier simulation: Conditions. ....	57
Table 4. Operational amplifier: Specifications summary.....	59
Table 5. Transimpedance amplifier simulation: Conditions. ....	61
Table 6. Transimpedance amplifier: Specifications summary. ....	65
Table 7. Simulated opamp and measured TIA specifications.....	88
Table 8. Comparison of electrochemical biosensor systems with CMOS MEA. ....	113



## LIST OF FIGURES

Figure 1. The three-electrode system. ....	5
Figure 2. A basic continuous feedback TIA circuit. ....	7
Figure 3. Generation 1: Floorplan with 21 test patterns of microelectrode [17]. ....	8
Figure 4. Generation 2: System level diagram of the biosensor system. ....	10
Figure 5. Generation 2: Hierarchy of biosensor hardware components. ....	11
Figure 6. Generation 2: Microfluidic setup and fabrication. ....	14
Figure 7. Generation 2: System performance summary. ....	16
Figure 8. Generation 2: Example of data configuration and heatmap processing. ....	19
Figure 9. Generation 2: Redox of norepinephrine, epinephrine, and dopamine. ....	22
Figure 10. Generation 2: Chemical heatmaps of ex vivo tissue experiments. ....	24
Figure 11. Generation 3: System level functional diagram. ....	28
Figure 12. Generation 3: The CMOS microchip with the alumina ceramic package. ....	30
Figure 13. Generation 3: Development of circuit boards timeline. ....	31
Figure 14. Generation 3: A typical system setup during wet experiments. ....	32
Figure 15. Generation 3: The first design of the microfluidics setup. ....	33
Figure 16. A micrograph of the CMOS microchip with the MEA. ....	36
Figure 17. Quadrant & read channel array arrangement. ....	37
Figure 18. Operational amplifier topology, a transistor level schematic diagram. ....	39
Figure 19. Operational amplifier layout arrangement. ....	40
Figure 20. Potentiostat layout arrangement. ....	41
Figure 21. TIA with T-Network feedback resistor as read channel. ....	42
Figure 22. Frequency response of opamp, noise analysis, and stability condition in TIA. ....	43
Figure 23. TIA layout arrangement. ....	45
Figure 24. Electrode array layout, comparison of Generation 2 and 3 MEA. ....	46
Figure 25. Read channel array configuration. ....	48
Figure 26. Global bias circuit layout arrangement. ....	49



Figure 27. Analog transmission gate switch with the smooth transitioning driver. ....	50
Figure 28. 6 to 64 decoder building blocks. ....	51
Figure 29. 6 to 64 decoder layout arrangement. ....	51
Figure 30. Flowchart of hierarchical schematic and layout simulations. ....	54
Figure 31. Operational amplifier and TIA testbench in Cadence schematic view. ....	55
Figure 32. Operational amplifier simulation: Setup. ....	56
Figure 33. Operational amplifier simulation: AC responses. ....	57
Figure 34. Operational amplifier simulation: DC responses. ....	58
Figure 35. Operational amplifier simulation: transient response. ....	59
Figure 36. Transimpedance amplifier simulation: Setup. ....	61
Figure 37. Transimpedance amplifier simulation: AC responses. ....	62
Figure 38. Transimpedance amplifier simulation: DC responses. ....	63
Figure 39. Transimpedance amplifier simulation: Gain variations. ....	63
Figure 40. Transimpedance amplifier simulation: Transient response. ....	64
Figure 41. Transimpedance amplifier simulation: Power dissipation. ....	64
Figure 42. CMOS Microchip simulation: Transition between read channels. ....	66
Figure 43. CMOS Microchip simulation: Setup. ....	66
Figure 44. CMOS Microchip simulation: output voltage response to quadrant enables. ....	67
Figure 45. CMOS Microchip taped-out: output voltage response. ....	68
Figure 46. Biosensor system 3D model assembly. ....	69
Figure 47. Prototype board during the first Generation 3 CMOS microchip test. ....	70
Figure 48. Prototype breakout board and PCB version 1. ....	71
Figure 49. PCB version 2: Sensor and Control boards. ....	72
Figure 50. PCB version 3: 3D thermal simulation comparison. ....	73
Figure 51. PCB version 3: Custom heatsink fabrication. ....	74
Figure 52. PCB version 3: Shield board arrangement. ....	75
Figure 53. PCB version 3: Final board-level configuration. ....	76
Figure 54. High spatial resolution data acquisition flowchart. ....	78



Figure 55. Graphical User Interface: Real-time (MainGUI.m).....	79
Figure 56. Graphical User Interface: Post-processing (PlotGUI.m).....	83
Figure 57. Microfluidic support system. ....	84
Figure 58. Assembled biosensor and microfluidic support system.....	85
Figure 59. Measured frequency response of the T-network TIA, gain and noise response. ....	87
Figure 60. Flow injection: Setup.....	89
Figure 61. Flow injection: Step function responses. ....	89
Figure 62. Amperometry response to NE and $pO_2$ . ....	90
Figure 63. Limit of Detection of NE. ....	91
Figure 64. Cyclic Voltammetry response to NE, and selectivity to NE and UA. ....	92
Figure 65. pH sensitivity response. ....	93
Figure 66. High spatiotemporal resolution imaging results of flow injection. ....	95
Figure 67. CMOS microchip MEA setup for oxygen imaging. ....	97
Figure 68. A group of COCs in a 4-well dish.....	98
Figure 69. Oxygen imaging in vitro experiment setup. ....	99
Figure 70. Electrochemical response to oxygen in an improved setup. ....	101
Figure 71. Oxygen reduction rate at three different ratios of sample and rest period. ....	102
Figure 72. The modified real-time GUI showing the sample and rest periods.....	104
Figure 73. Steps of heatmap spatial and temporal smoothing. ....	105
Figure 74. Effects of $pO_2$ imaging after microfluidic flow. ....	107
Figure 75. Oxygen gradient imaging in static media of a healthy COC. ....	109



## CHAPTER I: INTRODUCTION

The ability to view biological events in real-time contributes significantly to the understanding of critical life processes. Visualizing chemical changes at smaller dimensions and shorter timescales allows scientists to better understand the driving forces that regulate fundamental and obscure biological phenomena. Traditional optical microscopy techniques provide a means of observing the movement of small molecules in live biological samples with a respectable spatial and temporal resolution, however, with limitations. These methods are restricted to a library of inherently colored or fluorescent molecules, and those that can be selectively labeled through genetic modifications, exogenous fluorophores [1], or quantum dots [2]. In addition, these modifications add a considerable molecular weight when conjugated to small molecules such as neurotransmitters or pharmaceuticals, which can have a significant impact on behavior and function, making it difficult to visualize molecular changes in ways that replicate in vivo environments.

Electrochemical sensing for detecting specific analytes has allowed integration with other ex vivo or in vitro bioanalytical platforms (e.g. microfluidics systems and other microscopy techniques). The electrochemical methods support and enable biological discoveries that were difficult to achieve in the past. Recent applications of electrochemical sensing include studies in: metabolism activities by sensing levels of dissolved oxygen [3]–[5], glucose, lactate, and pyruvate [6]–[8] concentrations; roles of nitric oxide in physiology [9], [10]; DNA hybridization and analysis [11], [12]; and understanding complex roles of neurotransmitters in cell growth, communications, and deaths [13], [14]. Furthermore, applications of Microelectrode Arrays (MEAs) in electrochemical sensing have provided significantly enhanced spatial resolution for enabling studies of cell-to-cell signaling pathways and other complex biological phenomena. Sensor devices employing MEAs were first reported by Thomas [15] and further improved by others [16]–[18] to illustrate their capabilities of recording two-dimensional (2D) action potentials to better understand the biological signaling mechanism. The manufacturing compatibility of MEAs with CMOS technology allows tight integration of MEAs and supporting sensor circuitry within a single silicon substrate. Such compatibility provides the necessary performance needed to achieve high temporal resolution. Simultaneously, accessibility to CMOS processes offers a low-cost solution for integrating customized and complex electronic circuits with a higher signal-to-noise ratio (SNR). A biosensor system that combines



electrochemical methods and CMOS technologies would benefit the life-science community by providing real-time chemical images at high spatial and temporal resolutions.

The biosensor system development presented in this paper focuses on the common electrochemical sensing modalities (voltammetry and amperometry) and all-around advancements to the existing MEA-based electrochemical sensing systems. Various design aspects of the MEA biosensor system presented in this paper are based on our established work with electrode geometry to maximize electron transfer efficiency [19]. The system utilizes the electrochemical reduction-oxidation (redox) mechanism with a three-electrode configuration consisting of Working Electrode (WE), Reference Electrode (RE), and Auxiliary Electrode (AE). An earlier version of the biosensor CMOS MEA microchip was tested with microfluidic flow injection experiments to demonstrate its chemical imaging capability [20], [21]. Incorporating the same CMOS MEA microchip, an improved system with significantly higher SNR verified the viability of high resolution spatiotemporal electrochemical imaging in stimulated tissue *ex vivo* experiments [22], [23]. The chemical images from the prior system demonstrated its potential as a tool for studying real-time cell-cell signaling of live tissues with, however, some limitations.

This manuscript describes a development of a fully customized electrochemical biosensor system. The presented system employs a custom CMOS microchip with 16,064 Pt MEA at 27.5 $\mu$ m pitch and a compact system to support chemical imaging of live biological samples with up to four frames per second (FPS). The new system design addresses issues from the previous generation [23] at the CMOS microchip level while maximizing the integration of other features at the board level, which includes device monitoring (e.g. power consumption and temperature sensing points), control circuits for MEA multiplexing, and microfluidic system with pH monitoring. In comparison with previous design [22], [23], the new design increases the number of WEs and area coverage from 8,192 (2.0mm $\times$ 2.0mm) to 16,064 (3.6mm $\times$ 3.6mm), and its uniform geometrical arrangement of the three-electrode system helps to minimize the diffusion layer overlaps. Arrays of compact on-chip current to voltage (I-V) amplifier circuits (read channels) are individually connected to each corresponding WE without intermediate multiplexing analog switches or current buffers. The only on-chip multiplexing occurs at the voltage outputs of the read channels. This configuration was selected to ensure the continuation of electrochemical cell redox reaction and to minimize any disturbance to the noise-sensitive electrochemically active samples at the WE surface. Therefore, the new design allows



faster data acquisition scheme to scan through all 16,064 read channels. Significant temporal improvement was achieved with the generation of a 16-thousand pixel chemical image in every 0.25 seconds, while the previous generation required at least 30 seconds to generate an 8,192-pixel single-frame chemical image. At the board level, the control and support circuits were designed around a microcontroller unit (MCU). The entire system is controlled and monitored in real-time using a custom-built Graphical User Interface (GUI) on a host computer. Microfluidic flow injection experiments of neurotransmitter (norepinephrine (NE)) and deoxygenated deionized water (DI-H<sub>2</sub>O) confirmed the sensitivity performance and high spatiotemporal fidelity. The cyclic voltammetry (CV) results showed feasibility in simultaneous detections of different target analytes in all 16,064 WEs. This system not only presents significantly improved performance from the existing works but also provides an all-in-one high spatiotemporal chemical biosensing system solution for practical applications.

As one of electrochemical imaging applications, oxygen measurement at the cellular level has been of interest in branches of physiological studies. Primarily, oxygen is responsible for mitochondrial oxidative phosphorylation (OXPHOS), a vital process in generating a chemical complex (adenosine triphosphate (ATP)), the high-energy substance required in cellular growth and maintenance [24], [25]. The ability to image oxygen gradients in real-time is highly desirable to better understand the metabolic progress of non-communicable diseases (e.g. cancer, neurodegenerative, autoimmune, and cardiovascular diseases) [26], mechanism of tumor cell growth [27], [28], embryo morphogenesis [29], and development of assisted reproductive technology (ART) [7], [30], [31]. Dissolve oxygen concentration (DOC) or oxygen partial pressure (pO<sub>2</sub>) indicates the level of hypoxic condition in tumor cells. This determines its stage, prognosis, and is potentially useful for rapid therapy treatments developments [32], [33]. Meanwhile, in an oocyte's and early embryo's development, monitoring pO<sub>2</sub> provides oxygen consumption rate (OCR) due to mitochondrial OXPHOS, which is a proven predictor of a cell's viability during its maturation and its effect to various therapies in ART [34]–[37]. Access to more sensitive and higher spatial and temporal resolution pO<sub>2</sub> measurement method would benefit treatments and drugs development and overall advancement in the medical field. Sets of in vitro experiments on bovine cumulus-oocytes-complexes (COCs) were performed to produce 2D pO<sub>2</sub> images. Our results were analyzed according to the previous pO<sub>2</sub> gradient analysis simulations and methods [5], to show pO<sub>2</sub> flux density and consumption rate in the high-density



MEA. In combination with the custom-designed microfluidic support system for oocytes/embryo cells handling, this work offers unique insights into high spatial and temporal response of bovine COCs metabolism.

The remaining part of the manuscript is organized as follows: Chapter II describes the bases of this work as a high spatial and temporal resolution electrochemical sensor; previews of the previous generation of our MEA-based electrochemical sensor and its results are also discussed. Chapter III provides a brief overview of the entire biosensor system. Chapter IV discusses further details of the integrated circuit design and the MEA layout of the CMOS MEA microchip. Chapter V presents the CMOS microchip simulation results. Chapter VI describes the system-level integration, consisting of iterations of board-level prototyping, design of real-time user interfaces and data processing, and fabrication of the microfluidics support system. The validation of electrical performances with and without target chemical compounds are illustrated in Chapter VII; the results are presented both quantitatively (calibration curves) and qualitatively (2D heatmaps). Chapter VIII describes the application to the oxygen gradient measurement in bovine COC cells and analysis of the oxygen flux density and consumption rate. Finally, conclusions and suggestions for future research and applications are given in Chapter IX.



## CHAPTER II: MOTIVATIONS AND EXISTING WORKS

### 2.1. Electrochemical Measurement Overview

A common electrochemical system typically adopts a traditional three-electrode system method, employing WE, RE, and AE as shown in Figure 1. The three-electrode system offers flexibility in performing various methods of electrochemical analysis, including amperometry, chronoamperometry, CV, and differential pulse voltammetry. In an amperometric measurement, the current detection of reduction and reduction (redox) of electrochemically active compounds is activated by the constant potential difference between the WE and RE, where RE potential is continuously maintained by the potentiostat through the continuous electron supplies provided by AE. The potentiostat is responsible for reflecting the bias voltage ( $V_B$ ) by creating closed-loop feedback between the RE and AE, a path with finite impedance value.  $V_B$  is a voltage potential defined by user to accomplish the specified redox reaction of a chemical compound. Due to the redox reaction, the level of current density flows through the WE can be correlated to the concentration of the target chemical compound at the immediate proximity around the WE surface [38]. Lastly, the current is converted to voltage by a Transimpedance Amplifier (TIA) circuit for data sampling or acquisition for the following data analysis process.

The redox process causes a chemical compound to lose or gain electrons in oxidation and reduction processes, respectively. For an example, a simplified redox reaction between oxygen and water

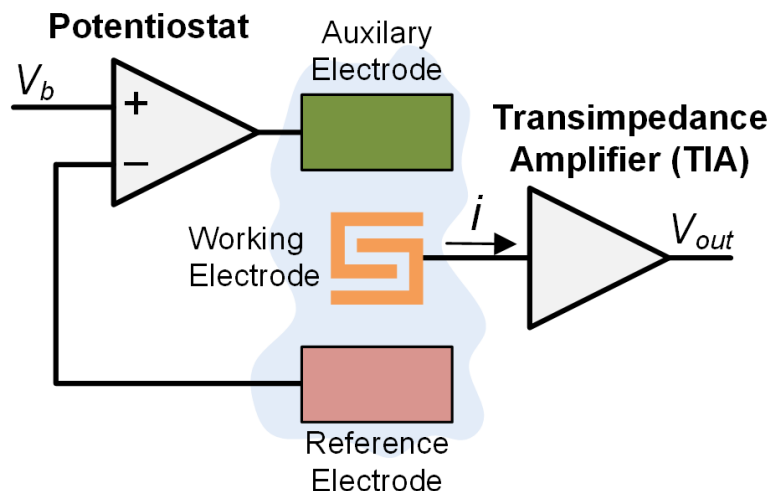


Figure 1. The three-electrode system.

*The traditional three-electrode system consists of AE, RE, WE, a potentiostat, and a read channel.*



can be summarized as  $O_2 + 4H^+ + 4e^- \rightleftharpoons 2H_2O$ . The electrical current response of a spherical microelectrode is not only defined by the redox electron transfer based on the chemical concentrations (Q), but also by its radius (r), surface area (A), diffusion coefficient of the solution (D), concentration of the reactant (c), and time (t) as described by equation (1). In this case, an MEA is affected by the  $\sqrt{\pi Dt}$  term, which is a time-dependent diffusion layer area. As time increases, the diffusion layer grows larger than the microelectrode radius (r), forming a steady-state current response (2) [39].

$$i(t) = \frac{QDAc}{r} \left[ 1 + \frac{r}{\sqrt{\pi Dt}} \right] \quad (1)$$

$$i = \frac{QDAc}{r} \quad (2)$$

The read channels are constructed in a form of current to voltage (I-V) converter. A TIA circuit, as shown in Figure 2, is the simplest and well-proven method, is used to convert the current readings defined by equation (3).

$$-V_{out} = R \times i \quad (3)$$

$$BW_{TIA} \approx 1.4(f_i) = 1.4 \sqrt{\frac{f_{GBWP}}{2\pi R_F(C_F + C_i)}} \quad (4)$$

Each read channel consists of a single-ended opamp, a resistor ( $R_F$ ), and a feedback capacitor ( $C_F$ ). The value of  $R_F$  and  $C_F$  are based on the need for high SNR and moderately low I-V conversion bandwidth (i.e. tens of kHz) for biological electrochemical imaging. The bandwidth of the TIA is defined by (4), where  $f_i$  is the intercept frequency between the opamp open-loop gain and the feedback frequency response,  $f_{GBWP}$  is the opamp gain-bandwidth product, and  $C_i$  is the total interconnects parasitic capacitance and the double-layer capacitances formed at the surface of WE. Furthermore, as defined by (5), the output rms noise consists of three noise contributors:  $e_{nR}$  is the thermal noise of  $R_F$ ,  $e_{ni}$  is the shot noise due to input bias current of the opamp, and  $e_{ne}$  is the overall output noise generated by the opamp [40].



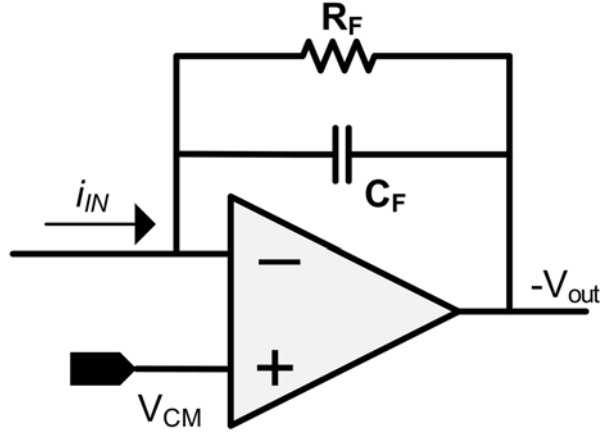


Figure 2. A basic continuous feedback TIA circuit.

$$V_n = \sqrt{\frac{e_{nR}^2 + e_{ni}^2 + e_{ne}^2}{2\pi R_F(C_F + C_i)}} = \sqrt{\frac{(4KTR_F) + (R_F i_n)^2 + e_{ne}^2}{2\pi R_F(C_F + C_i)}} \quad (5)$$

In TIA read channel design, the use of opamp with extremely low output noise ( $e_{ne}$ ) and input current noise ( $i_n$ ) would yield relatively negligible  $e_{ni}$  and  $e_{ne}$ . The  $R_F$  thermal noise ( $e_{nR}$ ) over a bandwidth defined by  $R_F(C_F + C_i)$  will dominate the output rms noise. Increasing  $R_F$  will result in a linear increase in I-V gain and a square root increase in output rms noise, leading to a higher SNR. In addition, the choices of  $R_F$  and  $C_F$  provide a mean to perform tradeoff between the required bandwidth in (4) and the output rms noise in (5).

## 2.2. Other Works in Electrochemistry and MEA

The integration of MEA fabrication and CMOS technology into biosensing applications requires integration of multi-disciplinary knowledge in electronics circuit design, analytical chemistry, and physiology. Currently, there is a spectrum of MEA biosensors with CMOS technology that focuses on electrophysiology, impedancemetry, and electrochemistry. Electrophysiology-focused MEA systems emphasize studies on action potential signal acquisition and stimulation of neuronal cells up to hundreds of hertz [41]–[44]. The addition of impedance measurements in an MEA configuration assists the determination of biological conditions of target cells or tissue samples [45]. Multimodal MEA systems integrate multiple biosensing functionalities, such as optical and temperature sensing, in addition to electrophysiology and impedance



measurements [46], [47]. The existing electrochemical MEA systems emphasize on sensing electrochemically active substances with integrated CMOS technology [23], [48], [57], [49]–[56] and without integrated CMOS technology [58]–[60]. The existing designs of MEA-based systems provide valuable insights into 2D electrochemical sensing. However, the existing systems present limitations in chemical imaging for bio-samples due to limited spatial resolution for cellular-level imaging, insufficient temporal resolutions to capture the chemical gradient actions over time, or low SNR read channels for micromolar concentration sensing.

Electrochemical methods, such as scanning electrochemical microscopy (SECM) and MEAs, have been previously explored as alternative methods for acquiring spatial and temporal chemical information from small molecules that cannot be reliably measured with existing microscopy techniques. SECM uses a microelectrode tip to scan the surface of a sample to measure the current generated by the reduction-oxidation (redox) of chemically active species, which can provide spatial chemical information, but is limited in its temporal resolution [61]. On the other hand, the MEA system approach provides a means of taking measurements from multiple electrodes simultaneously, allowing for improved spatial and temporal resolution [18], [62]. One of the first biological applications of MEAs utilized an array of thirty nickel-gold-

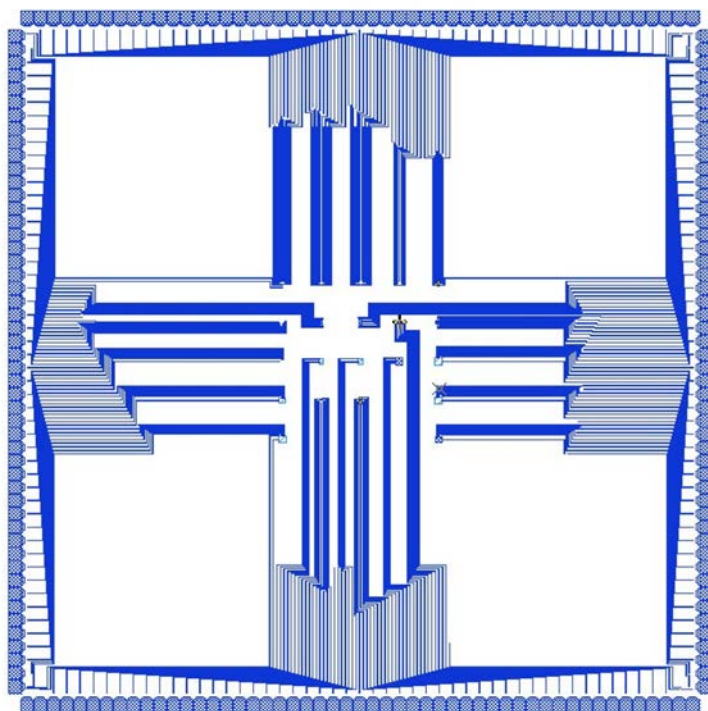


Figure 3. Generation 1: Floorplan with 21 test patterns of microelectrode [17].



platinum electrodes on a glass substrate to gather bioelectric measurements on the cellular level [63]. Over the years, the MEA system approach has substantially improved with the advancement of CMOS manufacturing processes, which have allowed for the fabrication of higher density arrays capable of gathering spatially resolved electrical readings. Most of these CMOS-fabricated microchips have been used for bioelectric potential measurements and stimulation on electrogenic cells, such as neurons [41], [44], [64] and cardiomyocytes [65]. In addition, some CMOS-based MEA systems are capable of simultaneous electrochemical, impedimetric, potentiometric, and thermal measurements [45], [46], [66]. Our research introduces an electrochemistry-based imaging device based on redox reactions rather than bioelectric potential measurements which can be employed as a complementary platform within existing microscopy systems, allowing for simultaneous capturing of electrochemical and optical information.

In prior work, our group studied the effect of microelectrode geometry on sensitivity [19]. Our initial research in microelectrode, Generation 1, concludes that the highest current density is generated from the microelectrode with the highest perimeter to the electrode surface area ratio. Our first attempt of integration of MEA to a CMOS microchip, Generation 2, established a method for interfacing MEA microchips with microfluidics [20], [21]. Preliminary live tissue electrochemical imaging results were reported along with a brief discussion of the system improvements [67]. The following section discusses the findings of the first electrochemical biosensor system employing an MEA.

### **2.3. Biosensor Generation 2 with CMOS MEA Microchip**

The objective of this system is to demonstrate that redox-based electrochemical analytical techniques can be used to image biomarker release on live tissue samples using a high-density MEA, with proof of concept in the imaging of neurotransmitter release from live adrenal tissue upon stimulation with caffeine. For this work, we chose to focus on amperometry, where the constant difference in potential between the WE and RE activates the reduction or oxidation of electrochemically active compounds. Electron transfer from the redox reaction at the surface of the WE produces a current that is linearly proportional to the analyte concentration [38], enabling the determination of catecholamine concentration from current values. This section provides a comprehensive discussion of the Generation 2 CMOS biosensor microchip design with hardware, software, and tissue handling protocols for gathering chemical



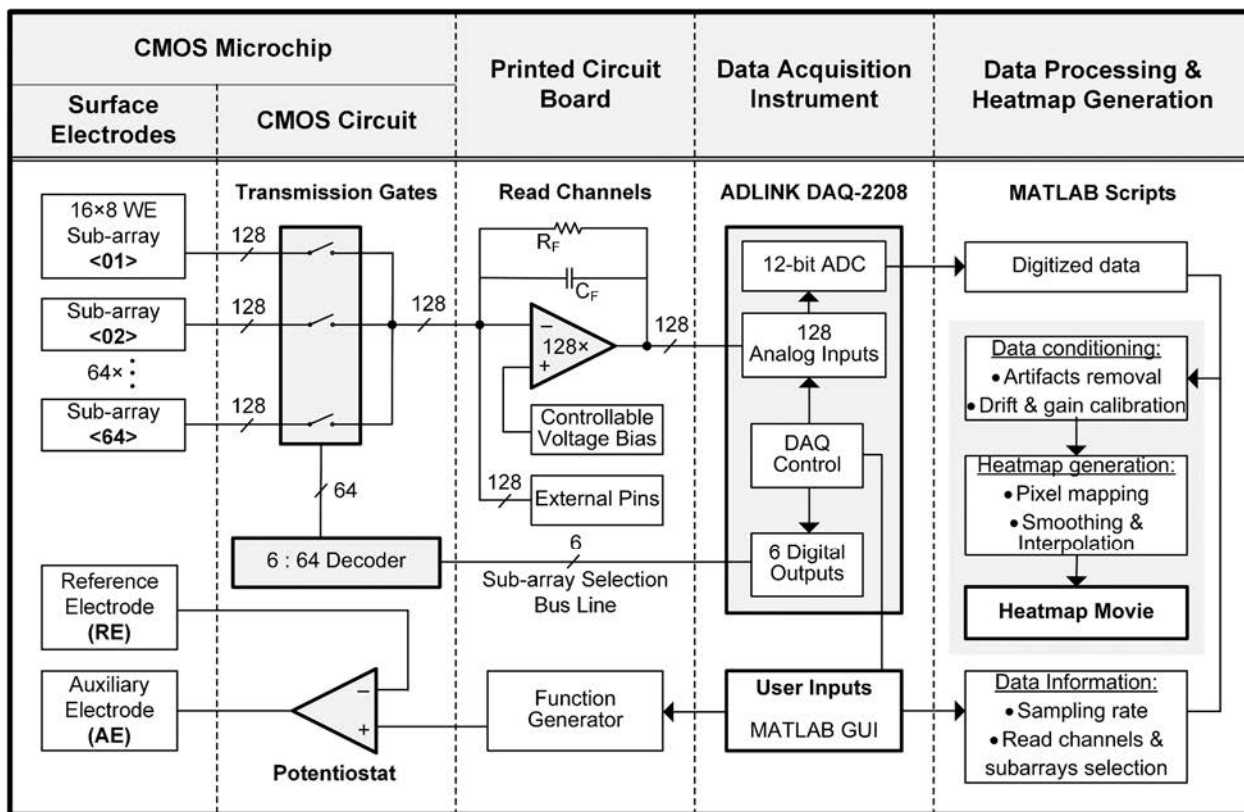


Figure 4. Generation 2: System level diagram of the biosensor system.

readings from live tissue samples with high spatiotemporal resolution and demonstrates imaging of caffeine-stimulated catecholamine release from live murine tissue.

### 2.3.1. Biosensor System

The biosensor system is comprised of hardware, software, and methods for ex vivo electrochemical imaging. Figure 4 describes the hardware and software integration, including the CMOS microchip, Printed Circuit Board (PCB), and data acquisition instrument as hardware components, and data processing methods with heatmap generation technique as software components.

The CMOS biosensor microchip functions as a miniaturized electrochemical analysis device employing a three-electrode electrochemical cell system in an MEA configuration. Fabrication of the electrode array has been previously described [20]. Each pair of WEs (Figure 5A) has a spatial pitch of  $25.5\mu\text{m} \times 30.4\mu\text{m}$  to neighboring WE pairs. The interdigitated WE configuration is utilized to maximize current density flow while maintaining high spatial resolution [19], [68]. Each subarray is a collection of 128



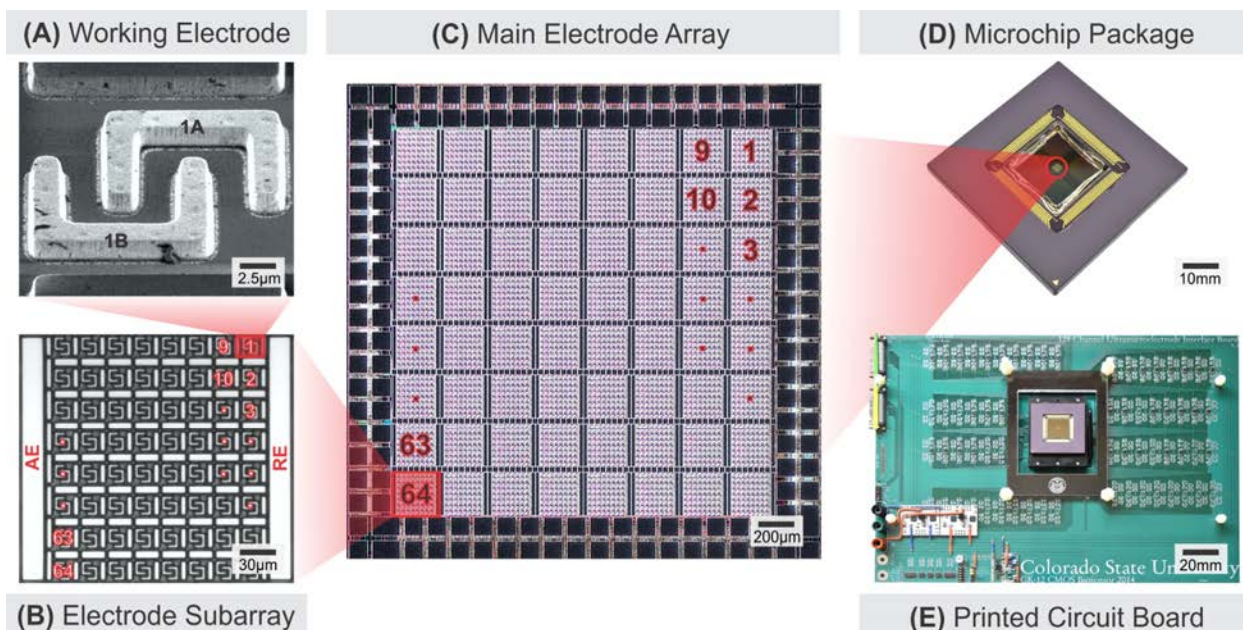


Figure 5. Generation 2: Hierarchy of biosensor hardware components.

(A) Pair of WEs in a simplified interdigitated configuration. (B) A subarray consisting of a rectangular CE and RE on either side of the 128 individual (or 64 pairs of) WEs. (C) The entire 2.0mm  $\times$  2.0mm MEA with all 64 subarrays. (D) CMOS biosensor microchip die (19mm  $\times$  19mm) implanted in a 280-pin PGA ceramic package. (E) PCB platform electrochemical imaging with sockets for the switchable microchip, read channel array, low noise DC power regulator, and interface connection to the data acquisition board.

C-shaped electrodes, or 64 pairs of electrodes, enclosed by shared rectangular RE and AE (Figure 5B). The entire array consists of 64 subarrays arranged in an 8 $\times$ 8 grid (Figure 5C), which is equal to 8,192 individual C-shaped electrodes. All surface electrodes are raised 1.5 $\mu$ m above the microchip passivation surface and composed of metal layers of Ti, Au, and Pt, with Pt as the outermost layer to maintain electrical conductivity, mechanical stiffness, and biological compatibility.

At any given point in time, the 128 electrodes in a subarray can be selected and directly connected to the off-chip connection pins. The direct connection configuration allows simultaneous readings from all 128 electrodes in one subarray and can be cycled through different subarrays at a rate and spatial pattern determined by the user through the control logic. This direct connection architecture provides flexibility in modifying the configuration of the surface electrodes and in performing other methods of detection (e.g., impedimetric and potentiometric). The surface MEA configuration is visible as a 2.0mm $\times$ 2.0mm glossy square area in the middle of the silicon die (Figure 5D).

The potentiostat circuit was designed to facilitate various methods of electrochemical analysis,



including amperometry, CV, and differential pulse voltammetry. The on-chip potentiostat employs a simple single-ended opamp unit. During the electrochemical reaction, the op-amp configuration forms a non-zero impedance path between the AE and the RE, setting the potential of the background environment to a desirable voltage. The op-amp circuit is based on a novel inverter-based topology [69], [70] and its transistor level design and specifications were previously reported [22].

The analog switches, NMOS-PMOS transmission-gate (TG) switches, were responsible for switching groups of WEs between subarrays. The selected subarray is controlled by the user via 6-bit digital signals via the data acquisition board, which are connected to an on-chip 6-bit input to 64 outputs decoder, where one of 64 outputs from the decoder selects all 128 analog switches in the subarray being selected. This customized selection of subarrays allows the user to focus on a specific area of interest in the MEA. There is one analog TG switch for each of the 8,192 WEs in the MEA, however, the digital logic control only allows for one subarray to be simultaneously connected to the off-chip read-channels. The WE surface area was enlarged to increase electrochemical current input while maintaining the same electrical background noise. Each pair of the 128 WEs was externally shorted to make 64 larger WE pairs with increased electrode surface area. Figure 5A shows two C-shaped electrodes that were shorted to form a single interdigitated WE.

The silicon-based microchip was custom designed and fabricated using 600nm CMOS technology with three metal layers for interconnects and the fourth metal layer for Pt-coated electrodes. As shown in Figure 5D, the 19.0mm×19.0mm CMOS microchip was packaged in a 280-pin ceramic pin grid array (IPKY8F, NTK Technologies). To prevent physical damage and unintentional electrical shorting during wet experiments, the wire-bonding area was sealed with a medical grade epoxy (EPO-TEK® 301, Epoxy Technology Inc.).

The custom-designed FR-4 2-layer PCB board (330mm×240mm), shown in Figure 5E, houses the 280-pin socket, 64 read channels, power voltage regulators, bias voltage generator, and input-output headers. The 280-pin socket (Textool™ Burn-In Grid ZIP 200-6319-9UN-1900, 3M™) provides a platform for the switchable custom CMOS biosensor chip. Four low-dropout (LDO) voltage regulators (TPS7A33 and TPS7A4700, Texas Instrument) generate ultra-low noise dual-rail voltage, supplying  $\pm 1.5V$  to the MEA microchip and  $\pm 2.5V$  to the on-board read channels and all other components on the PCB. Two adjustable



voltage generators deliver the necessary potential bias for the reference voltage of the on-chip potentiostat and the off-chip read channels. Finally, 68-pin very-high-density cable interconnect (VHDCI) small computer system interface (SCSI) headers provide paths to the data acquisition board for 64 read channel outputs and 6-bit digital control input for subarray selection.

The read channels are constructed in the form of a current to voltage (I-V) converter. The trans-impedance amplifier (TIA) circuits converting electrical current inputs readings generated by 128 WE. Each read channel consists of a single-ended op-amp (OPA2376, Texas Instruments), a high precision  $50\text{M}\Omega \pm 1\%$  resistor ( $R_F$ ), and a 1nF stabilization feedback capacitor ( $C_F$ ). Details of read channel design and component selection for the op-amp,  $R_F$ , and  $C_F$  has been discussed previously [22].

The data acquisition is handled by a multichannel commercial data acquisition board with a custom-built MATLAB Graphical User Interface (GUI) with support from MATLAB Data Acquisition, Signal Processing, and Statistics Toolboxes (MATLAB, MathWorks). The data acquisition board (DAQ-2208, ADLINK Technology) provides up to 96-channel simultaneous sampling with a resolution of 12-bit analog to digital conversion. In order to sample all 64 read channels through time-multiplexing, the maximum sampling rate allowed by the DAQ-2208 architecture and MATLAB toolboxes is 240 data points per second per channel. The data acquisition board also supports up to 24 bits of digital input-output channels, six of which are being used to control the 6 to 64 decoder that regulates subarray selection. The 12-bit digitized data from the data acquisition board is fed in real time to the MATLAB GUI and simultaneously stored for offline data processing and conversion to a video of a heatmap. In addition to providing real-time acquisition for 64 channels, the MATLAB GUI allows the user to manually configure subarray and read channel selection, subarray switching rate, sampling rate, and filtering options.

All tissue experiments used subarrays 1 through 64 with 0.5 second intervals between subarray switching and 240 samples per read channel per second. As data readings were gathered from the sensor, the output voltage from each read channel was displayed in real time and stored for offline processing. The stored data structure contained user input data (i.e., subarray and selected read channels), output voltage readings from each read channel, and time values for each output reading and each subarray switching event. The raw voltage data was post-processed in MATLAB for visualization.



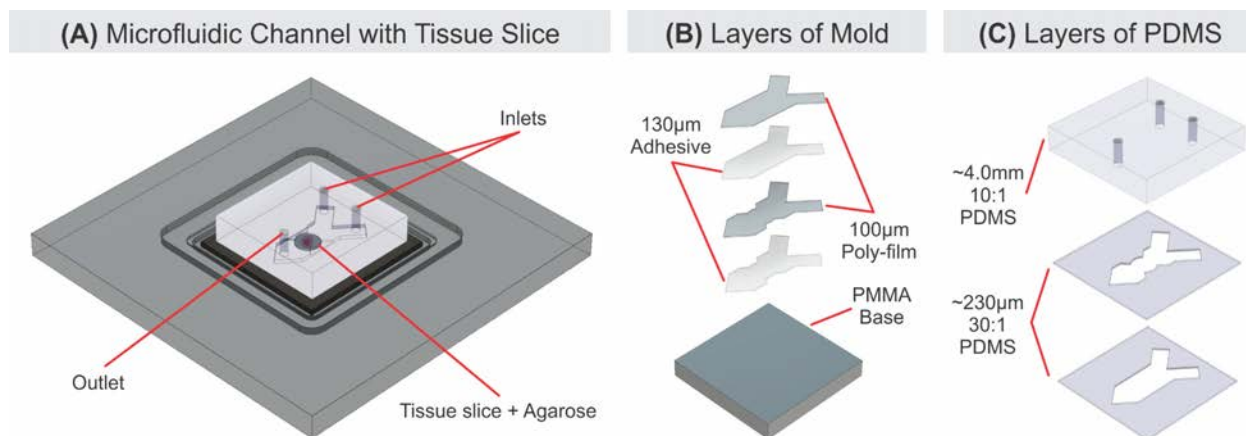


Figure 6. Generation 2: Microfluidic setup and fabrication.

(A) CMOS MEA microchip with microfluidic attachment. (B) Layers of mold for rapid PDMS microfluidic prototyping. (C) Microfluidic device showing firmness and shape of each layer.

### 2.3.2. Microfluidic System

The microfluidic system uses a multi-layer Y-shaped polydimethylsiloxane (PDMS) channel with two inlets and one outlet. The PDMS channel was developed to physically hold the tissue in place and to deliver stimulant while maintaining favorable conditions for live tissue during electrochemical imaging experiments. The main chamber of the PDMS channel, measuring 4mm wide, 6mm long, and 230µm tall, formed a sufficient space to confine a ~250µm thick, 3mm diameter tissue slice in the channel as portrayed in Figure 6A. The main chamber consisted of two different layers. The bottom layer provided a space to hold the agarose-embedded tissue slice. The upper layer was designed to hold the 3mm diameter agarose-tissue slice down in contact with the surface MEA while providing a ~230µm tall by 2mm wide channel for fluid to flow over the tissue. For limit of detection and dynamic range experiments, the Y-shaped microfluidic channel shown in Figure 6 was modified to include an additional inlet hole between the existing two for cleaning purposes. Additionally, the geometry of the lower chamber was replicated for the upper chamber, creating a straight channel with a height of ~260µm for fluid flow without tissue.

Soft lithography and laser cutting techniques were used to create the channel features. As shown in Figure 6B, a mold with multiple feature heights was created by assembling 230µm thick layers composed of pressure-sensitive adhesive (468MP-130µm, 3M™) and poly-film overhead transparency (PP2200-100µm, 3M™) on a clean polymethylmethacrylate (PMMA) plate. The PMMA base was cut to 25mm x



25mm and adhesive/transparency layers were laser cut to the desired geometries using a laser engraving system (Zing 16 Lasers, Epilog Laser). Figure 6C shows the two layers of different compositions of PDMS used in the device. The bottom layer, directly in contact with the microchip, was ~460 $\mu$ m thick and made using PDMS (Sylgard 184, Dow Corning) at a 30:1 ratio of oligomer to cross-linker, creating a more adhesive layer to better seal the device and prevent leakage at the interface with the microchip. The top layer was ~4mm thick and comprised of a 10:1 ratio of oligomer to cross-linker, creating a more rigid polymer to maintain structural integrity and prevent excessive deformation of channel features. For each layer, the PDMS was degassed, poured into the mold, and cured for 30 minutes at 80°C. A 1mm biopsy punch was used to create inlet and outlet ports, which were connected to vinyl PVC tubing (1/32" ID, 3/32" OD, Thermo Fisher Scientific) using stainless steel connectors (Loctite, Henkel Corporation).

For all sensitivity characterization and tissue experiments, the outlet tubing was directed to a waste container and each inlet's tubing was connected to a 10mL syringe (Benton, Dickinson and Company) driven by a syringe pump (NE-1000, New Era Pump Systems), allowing for control of flow rate for two types of fluids. To ensure flow stability, the PDMS device was held in place over the tissue with a custom-made compression plate laser cut from 0.125-inch thick PMMA. The plate contained holes matching the PDMS device to allow access to the inlet and outlet ports and holes for securing the plate to the microchip and imaging system board.

### **2.3.3. System Performance: Noise, Limit of Detection, and Dynamic Range**

The limit of detection and dynamic range of the electrochemical sensor were quantified among different electrodes and read channels within the system to account for CMOS process variation. These data were measured by the relative signal change between the baseline signal of the culture media and the increased signal due to the presence of catecholamines.

The biosensor system delivered a significantly improved SNR than results previously reported [20]. As shown in Figure 7A, the low-noise LDO and enhanced read channels in the existing system generate about 100 $\times$  less integrated noise ( $I_{n-RMS} = 0.046nA_{RMS}$ ) than previous system ( $I_{n-RMS} = 3.611nA_{RMS}$ ). The enhanced read channel design reduces overall flicker and white noise components, while the low-noise LDO suppresses the AC power line frequency component at 60Hz.



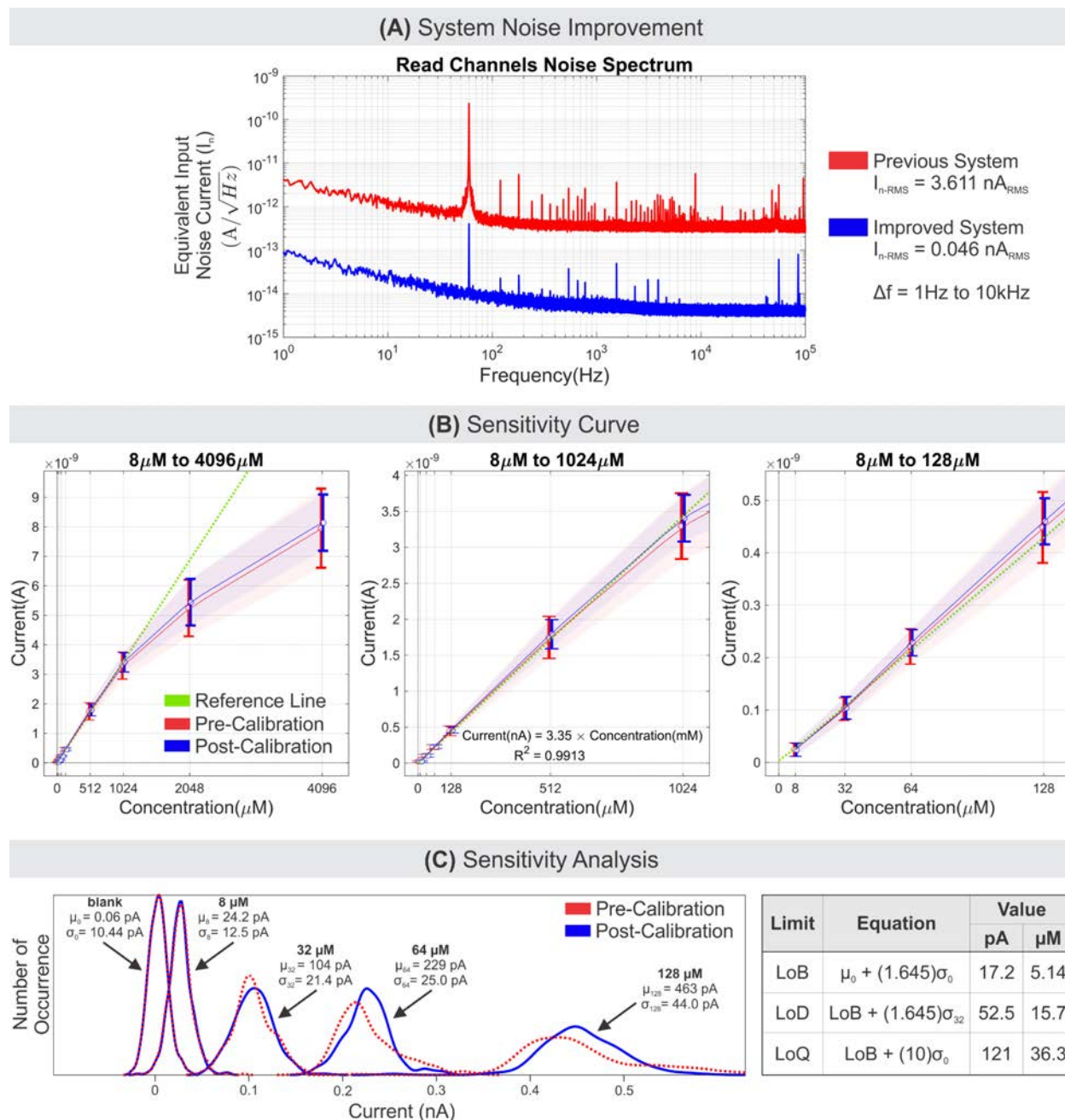


Figure 7. Generation 2: System performance summary.

(A) Measured read channel noise spectrum shows improvement over previous system. (B) Electrochemical imaging system response and sensitivity curves with one-sigma distribution band. The reference line is a fitted linear line generated from 8 $\mu$ M to 1024 $\mu$ M concentrations. (C) Probability Density Function (PDF) of 8 $\mu$ M to 128 $\mu$ M concentration readings with corresponding Limit of Blank (LoB), Limit of Detection (LoD), and Limit of Quantitation (LoQ) values.

Each measurement was performed multiple times over a set of randomly selected electrodes, subarrays, and microchips. First, DI-H<sub>2</sub>O was run through the center inlet, then switched to media flow



Table 1. Generation 2: Comparison of electrochemical biosensor system using CMOS MEA.

Parameter	Unit	[72]	[56]	[121]	[53]	This Work
CMOS Technology	$\mu\text{m}$	0.25	0.35	0.35	0.35	0.6
Power Supply	V	2.5	3.3	3.3	3.3	3.3
Die Size	mm	$5 \times 3$	$3.8 \times 3.1$	$7.5 \times 4.8$	$3.79 \times 3.79$	$19 \times 19$
Sensing Area	mm	N/A	$3.15 \times 1.9$	$3.2 \times 3.2$	$1.81 \times 1.81$	$2.2 \times 2.2$
WE Material	-	Au	Au	Pt Black	Au	Pt
WE Size	$\mu\text{m}$	$70 \times 70$ to $100 \times 100$	100 (3D bumps)	$\varnothing 5 - 50$	$20 \times 20$	$17.5 \times 15$ (interdigitated)
WE Pitch	$\mu\text{m}$	N/A	200	100	114	$25.5 \times 30.4$
Number of WEs	-	16	192	1024	256	8192
Number of Read Ch.	-	16	192	16	16	64
Bandwidth	kHz	10	1	0.1	150	10
Limit of Detection	nA	0.55	0.024	0.058	1.39	0.052
Power Dissipation	mW	N/A	36	N/A	125	120

through the first inlet, followed by media + norepinephrine flow through the second inlet. Once the signal of media + norepinephrine reached steady state, the value was recorded for three seconds, then the flow was switched back to blank medium and recorded for three seconds as the baseline. The side inlet flows were stopped, and DI-H<sub>2</sub>O was run through the center inlet to wash the surface electrode before continuing with another set of experiments. To conclude this process, the difference between the average three seconds recorded signals and baselines were calculated and recorded for each individual electrode and read channel. Using the data acquisition setup and MATLAB scripts, this procedure was automated and repeated multiple times with the same norepinephrine concentration to increase statistical accuracy and over a range of norepinephrine concentrations to generate the limit of detection and dynamic range data.

Figure 7B shows the results from three microchips with randomly selected subarrays and electrodes in three separate experiments. In this series of sensitivity experiments, 512 samples were gathered for each concentration, resulting in a total of 4096 sample points for concentrations of 8 $\mu\text{M}$ , 32 $\mu\text{M}$ , 64 $\mu\text{M}$ , 128 $\mu\text{M}$ , 512 $\mu\text{M}$ , 1024 $\mu\text{M}$ , 2048 $\mu\text{M}$ , 4096 $\mu\text{M}$ . In general, the system showed a saturated response



above 1024 $\mu$ M, a linear response from 8 $\mu$ M to 1024 $\mu$ M (defining the dynamic range), and a well-defined response at single digit  $\mu$ M concentrations. Since the data was collected based on the relative change from the baseline, the linear region (8 $\mu$ M to 1024 $\mu$ M) was fit to a regression equation without a zero intercept point, resulting in current (nA) equal to 3.35 times the concentration (mM), with a coefficient of determination,  $R^2 = 0.9913$ .

A further sensitivity investigation was performed and reported in Figure 7C. The pre-calibration (red curves) and post-calibration (blue curves) results for blank media and 8 $\mu$ M to 128 $\mu$ M concentrations are presented in the form of a Probability Density Function (PDF). The calibration algorithm effectively reduces variation and improves the distribution normality down to concentrations in the tens of  $\mu$ M. Mean ( $\mu$ ) and standard deviation ( $\sigma$ ) values are listed and used to find limits of blank, detection, and quantitation, based on the CLSI-EP17-A Standard [71]. Sensitivity of the biosensor system can be approximated by using of the device noise figure ( $I_{n-RMS} = 0.046$ nA), the sensitivity curve at lower concentrations (8 $\mu$ M to 128 $\mu$ M), and the calculated Limit of Detection (LoD) value. The equivalent chemical concentration noise is  $0.046$ nA<sub>RMS</sub> / 3.35 = 13.7 $\mu$ M<sub>RMS</sub> and the LoD is 15.7 $\mu$ M. Therefore, the read channel maximum sensitivity can be approximated to be in the range between 10 $\mu$ M to 20 $\mu$ M. Even though the result was obtained using norepinephrine as a representative catecholamine, the limit of detection results for the biosensor system should be applicable to catecholamine in general. Table 1 summarizes and compared the significant properties to other published works in electrochemistry using MEA [51], [53], [56], [72].

#### **2.3.4. Processing and Visualization**

To extract accurate reads from the sensor, the output data was post-processed in MATLAB to remove subarray switching artifacts and minimize noise and interference. Figure 8A and B show selected raw data recorded from subarrays 1 through 64 with 0.5 second intervals between subarray switching and 240 samples per read channel per second. As data readings were gathered from the sensor, the output showed spikes following each subarray switching event, after which the values settled to a steady point in a form similar to capacitor exponential decay. The form of these spikes is critical in the functionality of the electrochemical system, where a stable and continuous feedback loop incorporating the WE and AE is required to collect accurate read values.



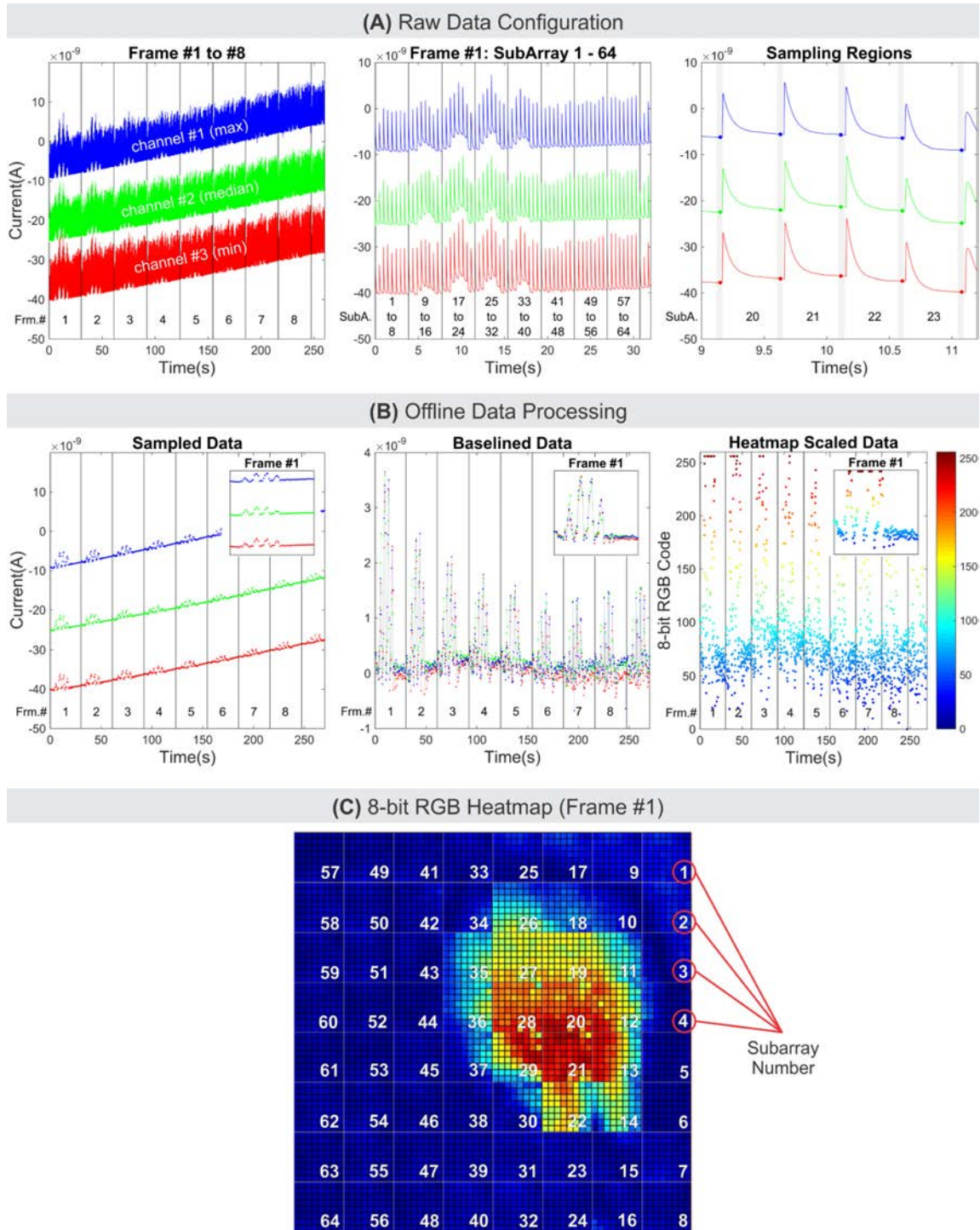


Figure 8. Generation 2: Example of data configuration and heatmap processing.

(A) Raw data from three read channels representing the minimum (red), median (green), and maximum (blue) channel values. Left: raw data for the entire video, including voltage spikes. Middle: the first frame of the video. Right: Data from 4 subarrays, with the sampling regions (94-99% of each subarray) shown as a dot before each peak. (B) Left: the sampled (settled) data values over the entire video. Middle: data after baseline removal. Right: baselined data after thresholding, color-coded to the MATLAB jet scale shown on the right axis. (C) An example heatmap resulting from mapping the data to the actual physical location in the MEA.



Figure 8A shows the unprocessed output of three selected read channels, out of 64. These channels represent the median, minimum, and maximum baseline values from the experiment, where the other channel readouts fall within this range. Most of the time-dependent drift is due to on-chip current leakage and electrode fouling of the MEA. Under a constant temperature, the typical baseline drift of the system was observed to be within 50pA/min. The experimental results in Figure 8A were selected to show how the signal processing handles an extreme case of signal drift (2.8nA/min) over 270 seconds of ex vivo imaging.

During electrochemical imaging experiments, a rough estimation of the final heatmap can be made by observing patterns in the real-time voltage values displayed on the MATLAB GUI before any further data processing. The raw data shown in Figure 8A correlates with the heatmap in Figure 8C. The location of the tissue can be observed in the raw data as bumps, shown in subarrays 9 to 40, when compared to the flat regions seen in subarrays 1 to 8 and 41 to 64. In some cases, the magnitudes and trends of these bumps can also give a rough approximation of the pattern and shaped of the object being detected. To better visualize this data, a video of a heatmap was generated through the following offline data processing steps:

1. Data Sampling – To accurately determine the sample at which the read channels switch to the next subarray, the locations of the voltage spikes were determined by finding the maximum values of the difference between each of the voltage signal readings or taking the derivative of the data. The settled data values were then determined by taking the mean of the data while removing the outliers from a selected interval between peaks. The selected interval is 94% to 99% of the distance between peaks, portrayed as a thin shaded area before each peak in Figure 8A. The resulting values are plotted in Figure 8B.
2. Baseline Removal – The baseline was determined for each channel by taking the minimum values from each frame corresponding to regions of the array that were not under the tissue slice. These values were then fit using polynomial regression with a configurable order polynomial. The expected baseline values from the polynomial fit equations were then subtracted from the sample data, resulting in the baseline-corrected data shown in Figure 8B. In this data, a 6th order polynomial provided sufficient baseline cancellation without adversely affecting the electrochemical signals.



3. Read Channel Calibration – This calibration minimizes gain variation between the 64 read channels due to feedback resistance variation, proximity of the WE to the CE [19], and the variation in electrode interconnections as previously discussed. Each calibration weight value was extracted from the scaled difference between read channel gain to the average of all channel gains. Finally, the weight value from each read channel was used to normalize output response, forcing the gain value closer to the gain average of all read channels.
4. Visualization – Minimum and maximum threshold values were set to remove extreme data values before mapping to MATLAB's 8-bit 'jet' color scale, represented by the numbers 0 (dark blue) to 255 (dark red). In this data, values below  $-0.3\text{nA}$  or above  $+2.0\text{nA}$  were translated to the lowest and highest values of the color map scale, 0 and 255, respectively. Fig. 5B provides the color-coded scatterplot for each sampled signal. The color-scaled data values were then mapped to the electrode location within the corresponding subarray as shown in Figure 8C.
5. Spatiotemporal Smoothing – To generate a continuous video over time, the data from each subarray was interpolated between each frame, creating a full frame of updated values for each new subarray reading to fill abrupt changes from one frame to the next. Outlying data values were removed and blended with the surrounding pixels through spatial averaging to prevent hotspots. The spatiotemporal smoothing algorithm improves heatmap visualization; although slightly altering the accuracy of the heatmap relative to the actual data.

The sources of noise and interference in the system include differences in read channel voltage gain, electrode surface morphology and size variation, and possible electrode biofouling. Various sources of inherent noise were removed using MATLAB in the post-processing calibration procedures. Variability in individual electrode readings due to slight differences in surface area or structure from the manufacturing process led to small amounts of variability between electrodes. These signal fluctuations were minimized in the final smoothing step in heatmap visualization.

### **2.3.5. Ex Vivo Electrochemical imaging**

The system was enclosed in an environmental chamber with Olympus BX51/61QI microscope, as previously described [22], to imitate the internal conditions of the body by limiting light disturbances and



maintaining a temperature of 37°C. The microchip, imaging system board, and microfluidic device were placed under an optical microscope (BX61WI, Olympus) with a DSLR recording camera (D5500, Nikon), allowing for simultaneous optical and electrochemical imaging. Rubber-sealed access holes in the environmental chamber provided cable routes for connecting the power and signal sources to the imaging system board and for connecting the tubing from the inlet ports to the two syringe pumps outside of the chamber, allowing for instantaneous manual flow adjustments during the experiment.

Ex vivo experiments were carried out 24 hours after tissue collection, allowing the tissue to acclimate after stress induced by slicing. The tissue was then placed directly on the electrode array, after which the PDMS microfluidic device was positioned over the slice and secured with the compression plate setup. The inlet and outlet ports were connected to the microfluidic device and the syringe pump for media was immediately turned on to provide a constant flow of media over the tissue slice. All media used in the microfluidic system was Neurobasal® medium with 1.3% penicillin streptomycin, 5% B-27 Supplement, and 5% HEPES. Caffeine solutions used for stimulation were 55µM in media. Media and caffeine solutions were maintained at 37°C at a pH of 7.1-7.4 and were administered at a flow rate of 20µL/min. The MATLAB GUI was used to gather baseline signal readings from the adrenal slice for three minutes under media flow, after which the media flow was switched to caffeine. Once caffeine reached the tissue slice, approximately two

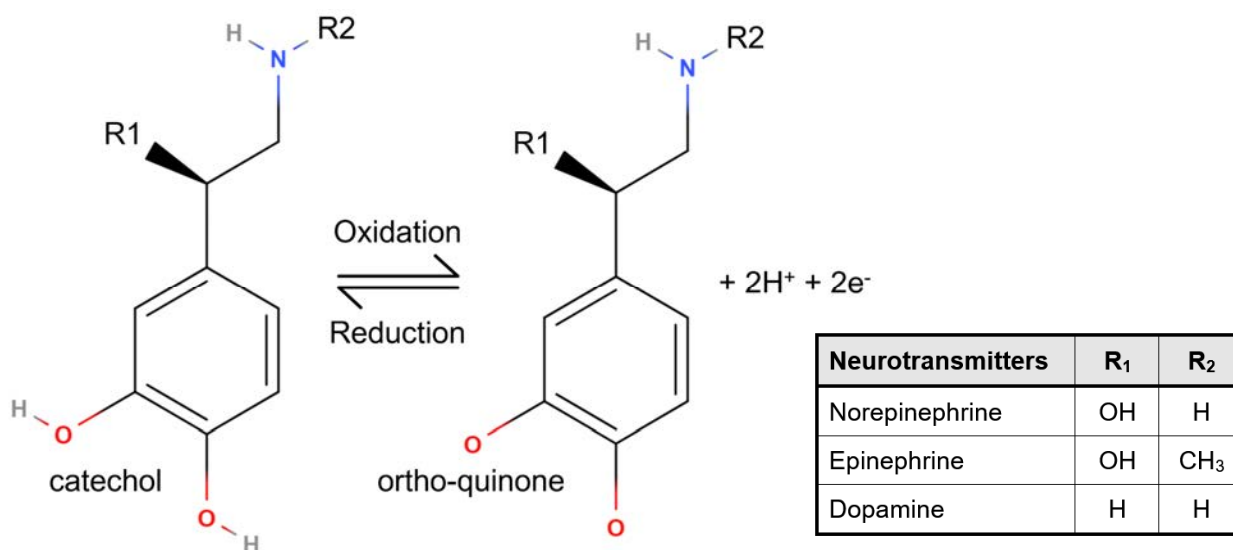


Figure 9. Generation 2: Redox of norepinephrine, epinephrine, and dopamine.



minutes after the switching point, signal recordings were taken for an additional 10 to 15 minutes. At the end of each experiment, the system was disassembled, and the microfluidic system and microchip were flushed with DI-H<sub>2</sub>O. For the amperometric detection of catecholamines, the class of neurotransmitters that includes epinephrine, norepinephrine, and dopamine, we used an oxidation potential of +0.6V (vs. Pt) [20]. To measure the system performance in sensing neurotransmitters, media (Neurobasal®-A Medium, Gibco™) and a neurotransmitter ((±)-norepinephrine (+)-bitartrate salt, Sigma-Aldrich) solution dissolved in media were used as the blank sample and target analyte sample, respectively.

In this experimental setup, we induce the release of primarily norepinephrine and epinephrine from the adrenal medulla, though small amounts of dopamine may also be present, as dopamine is a precursor for both of these molecules [73]. Since the catechol functional group that participates in the redox reaction is present in all three of these molecules and the applied potential of 0.6V is sufficient to activate oxidation of the hydroxyl groups, all three of these molecules are detected in our system. Though this system does not differentiate between these molecules, the total catecholamine concentration released from the tissue was calibrated from norepinephrine titrations representative of catecholamine oxidation. Detailed information about these redox reactions is shown in Figure 9 [74], [75].

It is a well-understood limitation of amperometric detection of target analytes that it detects all electrochemically active species in which the redox reaction can be activated at or below the applied potential. Therefore, knowledge of the species present in the system is important in amperometric detection and it will allow for more accurate determination of the molecule being detected. Specificity can be improved by using CV at each electrode, allowing for the detection of which electrochemically active species are present in the sample and tailoring for specificity. However, the use of CV for high-density MEAs will severely limit systems' temporal resolution.

Murine adrenal tissue offers an excellent biological model for demonstrating catecholamine release with spatial (e.g. gradients) and temporal components. Adrenal tissue has been well-studied [76]–[78] and releases high amounts of catecholamines (epinephrine, norepinephrine, and dopamine) in response to specific stimuli. The catecholamine release solely originates from the central medulla, without release from the surrounding cortex after stimulation with caffeine [79]–[81]. In this case, the electrochemical imaging



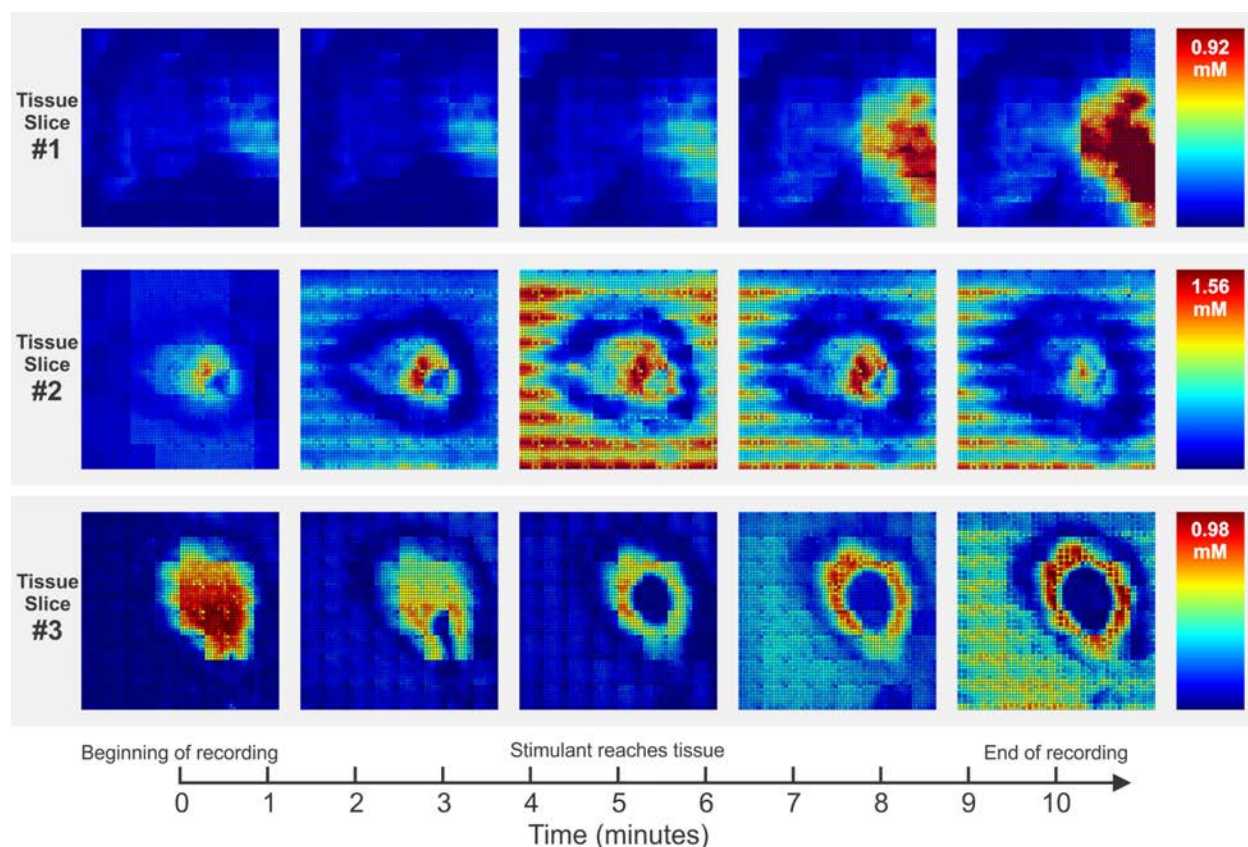


Figure 10. Generation 2: Chemical heatmaps of ex vivo tissue experiments.

*Three different murine adrenal tissue slices before and after caffeine stimulation. The scale bar, generated from the sensitivity curve, indicates catecholamine concentrations (up to 1.5 mM) specific to each experiment and tissue slice.*

device detected the amperometric current from catecholamine release without differentiating among epinephrine, norepinephrine, and dopamine.

Interfacing the tissue with the MEA required careful and rapid transfer of the tissue from the incubation petri dish to the electrode array using small spatulas, after which the PDMS microfluidic device was placed over the tissue and media flow injection was immediately initiated. Careful and swift tissue handling was critical in maintaining functional tissue and minimizing damage, as additional stress from excessive exposure to light, change in temperature and humidity, exposure to non-physiological oxygen conditions, lack of media, and physical pressure all had damaging effects on the tissue. In experiments with unscheduled prolonged handling times, tissue was discarded, and other tissue slices were used to avoid bio-related reliability issues.



All experiments employed an identical microfluidic setup and comparable stimulant delivery timing that reached the MEA around five minutes after the beginning of signal acquisition. Live tissue electrochemical imaging yielded various chemical gradient results and patterns; however, they each depict a general trend showing localized release and clear correlation with the location of the tissue slice on the MEA. Figure 10 illustrates time-lapsed screenshots of videos of three independent experiments (different murine adrenal glands) representing common results from ex vivo electrochemical imaging experiments employing the biosensor system. The results rely on relative measurement by calculating the signal of interest as the difference from the initial baseline, such that the heatmap indicates whether the readings are higher (more red) or lower (more blue) with respect to the baseline value calculated in Data Processing - Baseline Removal steps.

The experimental results from tissue #1 were presented previously [22] and used as a reference to show the expected results from this type of electrochemical imaging experiment. Tissue #1 results show a well-defined chemical gradient pattern, enlargement, and localized increase in catecholamine concentration originating at the medulla towards the cortex after caffeine stimulation around  $t = 5\text{min}$ .

Tissue #2 results show common occurrences of inconsistent tissue activity (i.e., decreasing signal after caffeine stimulation) and electrochemical signal readings outside tissue area. First, the number of catecholamine-releasing chromaffin cells in this slice may have been low, resulting in spatially concentrated but brief catecholamine release patterns. Besides concerns in tissue quality, biofouling of the MEA could have also been a contributing factor in unreliable signal output. Biofouling is mostly caused by organic deposition, hydrogen adsorption, and oxidation of the Pt electrode surface [82], [83]. Despite electrode treatments, biofouling still indicates some impairment of electron transfer functionality at the electrode and uneven readings from the MEA configuration. Second, non-electrochemically active areas, such as the adrenal cortex, create a high-impedance path that can disturb the baseline value calculation in a tissue experiment. In other words, the redox reactions occurring under the adrenal cortex area were dynamically lower than in flowing media surrounding the tissue slice. Tissue #2 results represent the population of experiments with complications in tissue quality, electrode biofouling, and electrode-tissue interfacing.

Tissue #3 results presented other challenges, with an initial spike of catecholamines concentrated in the medulla and accumulation of air bubbles. First, the high concentration readings at the beginning of



the recorded phase correlated with when the biosensor device was initially turned on and the electrochemical reactions were initiated. The high current readings in the first frame ( $t = 0$ ) suggest that the cells in the tissue released and accumulated a significant number of electrochemically active compounds at the surface of the MEA, resulting in rapid changes in electron flux once the recording was initially started. Moreover, excessive mechanical movement and stress induced during tissue placement on the MEA may have undesirably stimulated the chromaffin cells. Second, at  $t = 2$  to 3 minutes, air bubbles were observed entering between the MEA and tissue slice, where they resided for the rest of the experiment. These types of air bubbles may be trapped during the process of placing the tissue slice over the MEA or generated as a result of electrolysis at the MEA surface. Tissue #3 results represent the population of experiments with high initial readings and the presence of air bubbles.

Results from these experiments would not be sufficient to make biological inferences due to the challenges discussed above. Nevertheless, these results indicate there are well-defined spatial boundaries between the medulla, the cortex, and the area outside the tissue, demonstrating the potential of ex vivo high spatial resolution electrochemical imaging of tissue samples.

#### **2.4. Limitations and Baselines for Generation 3**

The Generation 2 Biosensor system with CMOS MEA microchip provides a system-level solution for obtaining chemical images with high temporal and spatial resolution using a dense micro-electrode array. The system consists of custom-designed CMOS microchip with a Pt-coated MEA, supporting PCB with low noise read channels and the supporting software with GUI for real-time monitoring capabilities. The integration of the key components in the system provides enhanced performance to allow for reliable sensing of micro-molar range catecholamine concentrations with  $25.5\mu\text{m} \times 30.4\mu\text{m}$  spatial resolution. In addition to demonstrating the capability of the high-resolution electrochemical imaging system, this study also provides necessary procedures for analysis of live tissue with further considerations in sample preparation and handling for interfacing with the biosensor device with environmental control.

The tissue electrochemical images reveal respectable preliminary results, however, with some limitations in generating consistent chemical gradient images over multiple ex vivo experiments. Overall, the performance of the previous generations of our CMOS MEA microchip and its MEA integration suffered



from three major limitations. First, the extended signal settling time is required during the multiplexing process of connecting read channels to the WE array in the MEA. The multiplexing process, though only a few seconds long, disturbs the continuity of the electrochemical cell and lowers the temporal resolution. Second, each WE accumulates random current leakage up to 100 nA due to direct connections to an array of analog CMOS switches (transmission gates) as part of the multiplexing scheme. This leakage current heavily impacts the read channel's overall SNR performance because the electrochemical signal is observed to be within pA to few nA from a single WE. Third, the arrangement of the three-electrode system is not optimal for dense MEA configuration. The sparsely placed RE and AE blocks impose irregular readings across the WE array, creating the proximity effect of high reading values closer to AE blocks within a sub-array. This artifact is shown as repeating stripes or block patterns in the previously published chemical image results [20], [23]. Also, unbounded and tightly packed WEs generate heavily overlapping diffusion layers, which is not desirable when obtaining high signal-to-background current ratio in scanning voltammetry methods [84], [85].



## CHAPTER III: GENERATION 3 SYSTEM OVERVIEW

### 3.1. System Level Configuration

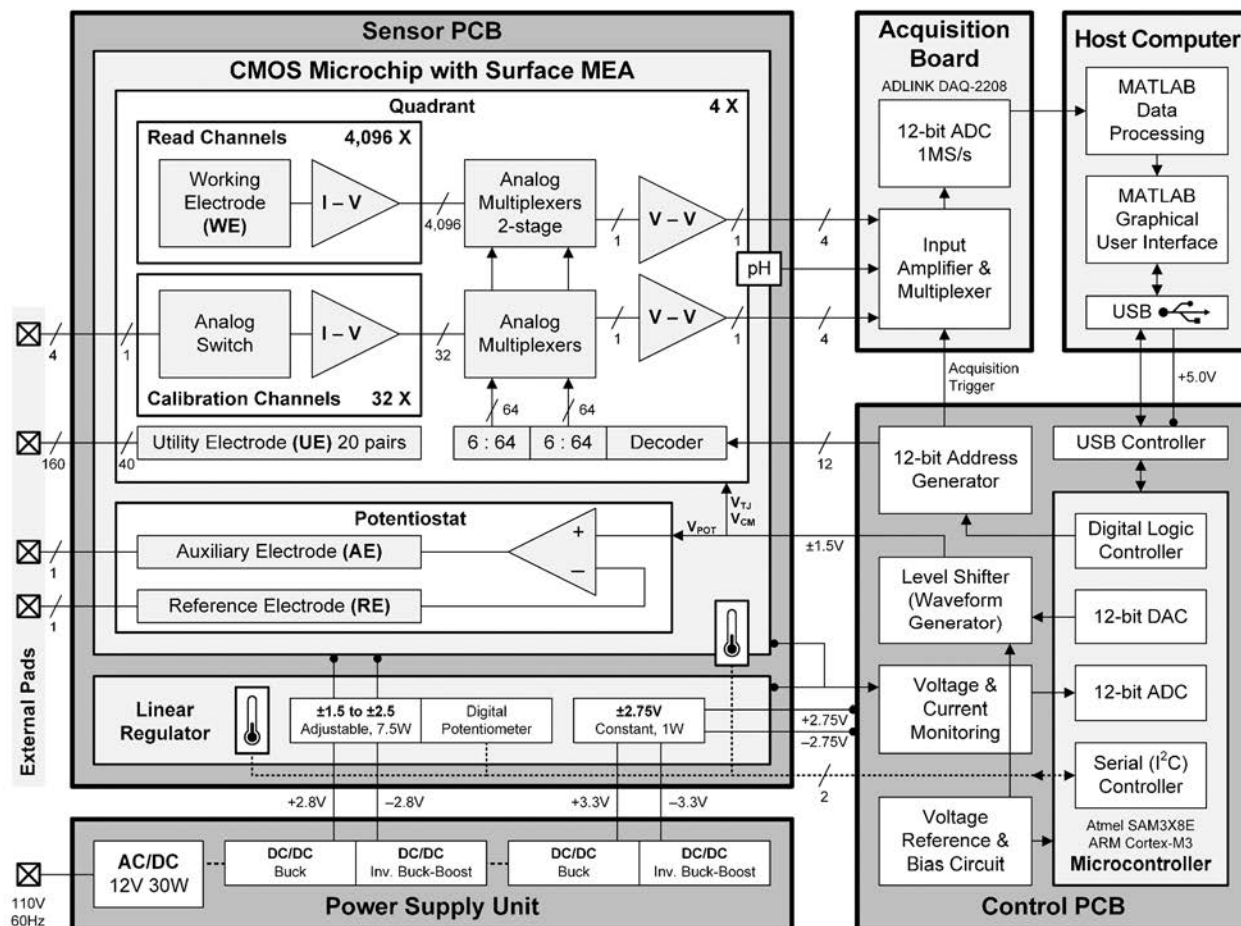


Figure 11. Generation 3: System level functional diagram.

The Generation 3 electrochemical biosensor system comprises of custom Printed Circuit Boards (PCBs), a Power Supply Unit (PSU), a Data Acquisition (DAQ) board, and a host computer running a custom-built software for control and displaying results as shown in Figure 11. This functional diagram is the final design of the system. One of the custom PCBs is the Control PCB (C-PCB) consists of a Microcontroller Unit (MCU), analog and digital signal generators, voltage monitoring circuits, and a USB interface for communication with the host computer. The other custom PCBs is the Sensor PCB (S-PCB). It holds the CMOS MEA microchip and voltage linear regulators to supply low-noise power to the CMOS MEA microchip and the C-PCB. The DAQ board, ADLINK DAQ-2208 (ADLINK, New Taipei, Taiwan),



converts analog to digital signals from the S-PCB to be processed and stored in the host computer. The control and display software and its graphical user interface (GUI) on the host computer is written in MATLAB. The GUI reports biosensing processed results, shows the information obtained from the C-PCB, and lets users perform real-time system control. All power needs are supplied by the dual-rail PSU, which converts standard 110V 60Hz power line input to multiple dual-rail DC potentials.

### **3.2. CMOS Microchip with MEA**

The CMOS MEA microchip, as the core of the system, holds surface MEA and read channels array, where the electrochemical electron transfer and current to voltage conversion occur, respectively. The on-chip read channel outputs are multiplexed and buffered through multiplexers and voltage buffer. To assist WEs array performing three-electrode electrochemical analysis, a single-amplifier on-chip potentiostat is employed to set the surface MEA potentials from the RE and provide feedback current at AE.

The silicon-based microchip was custom designed and fabricated using 0.6 $\mu$ m CMOS technology with three metal layers for interconnects and the fourth metal layer for Pt-coated electrodes. As shown in Figure 12, the 19.0mm $\times$ 19.0mm CMOS microchip was packaged in a 280-pin ceramic pin grid array (IPKY8F, NTK Technologies). To prevent physical damage and unintentional electrical shorting during wet experiments, the wire-bonding area was sealed with a medical-grade epoxy (EPO-TEK® 301, Epoxy Technology Inc.). The CMOS MEA microchip ceramic package fits to the 280-pin socket (Textool™ Burn-In Grid ZIP 200-6319-9UN-1900, 3M™), a zero-insertion-force (ZIF) socket that provides convenient of swapping multiple CMOS MEA microchip to the same system.

The custom CMOS MEA microchip was designed using Cadence® Virtuoso® platform utilizing the AMOS10A models, a 0.6 $\mu$ m CMOS process technology with 4-Metal layers, Broadcom® Incorporated, formerly Avago Technologies (Fort Collins, CO.). Cadence® Virtuoso® Analog Design Environment (ADE) was used for the circuit design and various simulations (e.g. DC, AC, transient, noise, and Monte-Carlo analysis) of the custom IC. All schematics simulation was performed to incorporate variations of temperature, power supply, and the process provided in AMOS10A model.



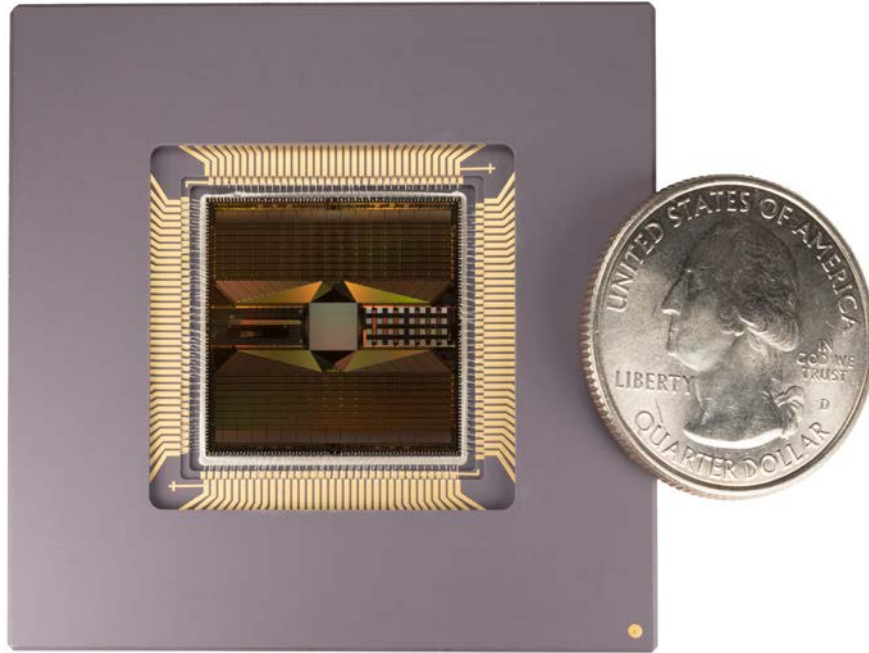


Figure 12. Generation 3: The CMOS microchip with the alumina ceramic package.

Utilizing the Cadence® Virtuoso® Layout Suite, the properly sized CMOS transistors polygons (N-Well, P-Island, N-Island, Poly, Contact), were laid out to create custom analog arrangements in a bottom-top hierarchical fashion (device, cells, block, then chip level). The layout process follows the general good practices for reducing transistor variations, increases yields, and manufacturability defined by the Design Rules published by Broadcom® Incorporated. Some layout techniques, such as: matching of transistor pairs to minimize offset in opamps, integrating dummy-cells to minimize cornering effect, splitting transistor for compact design, placing guard rings to increase isolation and minimize risk of latch-up, and applying multiple vias/contacts to ensure connectivity were carefully studied and applied to the layout design process [86]–[88]. Any repetitive works in the Cadence® Virtuoso® platform were automated and scripted in SKILL® language.

Chapter IV and V discuss the CMOS MEA microchip circuit design and layout processes and their corresponding simulation results, respectively. Details of the CMOS MEA microchip schematic, layout, and the SKILL® scripts are included in Appendix A.



### 3.3. Board Level Integration





PCB DEVELOPMENT		GEN.3 CMOS BIOSENSOR MICROELECTRODE ARRAY	
 <p>50mm</p> <p>Original Prototype PCB assisted by Nicholas Grant</p>	<b>Prototype</b> Jan 2017 to May 2017	<b>First power-up of post fabricated CMOS microchip.</b> <ul style="list-style-type: none"><li>• Assembled breadboard components and bench-top test equipment.</li><li>• Verified a test read channel for basic functionalities.</li><li>• Tested power dissipation and temperature at various voltage supply.</li><li>• Established digital control and analog reading using Analog Discovery.</li></ul>	
 <p>50mm</p>	<b>Prototype + PCB v.1</b> Jun 2017 to Dec 2017	<b>High-speed acquisition (1MS/s) of 16-thousand read channel outputs.</b> <ul style="list-style-type: none"><li>• Replaced breadboard components with PCB v.1</li><li>• Utilized low performance MCU (ATmega328P) for control.</li><li>• Included dynamic voltage regulator and temperature sensor.</li><li>• Employed external data acquisition board (DAQ2208) for analog read-out.</li></ul>	
 <p>20mm</p>	<b>PCB v.2</b> Jan 2018 to Oct 2018	<b>Integration attempt of all function into a compact device.</b> <ul style="list-style-type: none"><li>• Reconstructed PCB v.1 to a compact system with a custom power supply.</li><li>• Utilized high performance MCU (ATSAM3X8E) for all control.</li><li>• Established serial interface to PC for real-time monitoring and control.</li><li>• Integrated digital &amp; analog circuit to support electrochemistry functions.</li></ul>	
 <p>20mm</p>	<b>PCB v.3</b> Nov 2018 to May 2019	<b>Improvement of reliability, robustness, and sensing sensitivity.</b> <ul style="list-style-type: none"><li>• Refined power-up/down sequence to extend CMOS microchip lifetime.</li><li>• Enhanced thermal performance; redesigned voltage regulation.</li><li>• Fabricated custom aluminum heatsink with temperature sensor.</li><li>• Improved noise performance and added shields for the sensitive circuits.</li></ul>	

Figure 13. Generation 3: Development of circuit boards timeline.

The board-level integration was an effort of four different iterations of PCB design and assembly as shown in Figure 13. The first custom circuit board (prototype) was designed to provide the basic functionality in the form of a break-out board. It provides direct connections to the 280-pin pinouts for power lines, digital control, output signal, and other peripheral pins. The following iteration uses the prototype board as a platform for the PCB version 1 to integrate low-noise voltage regulators; the regulators are voltage controllable by an MCU. Reconditioning of the digital control signal, from single-rail to dual-rail were also achieved. The setup in version 1 offers the first fully dual-rail configurations. Board redesign and significant improvements were completed on the PCB version 2. The second version strategically placed



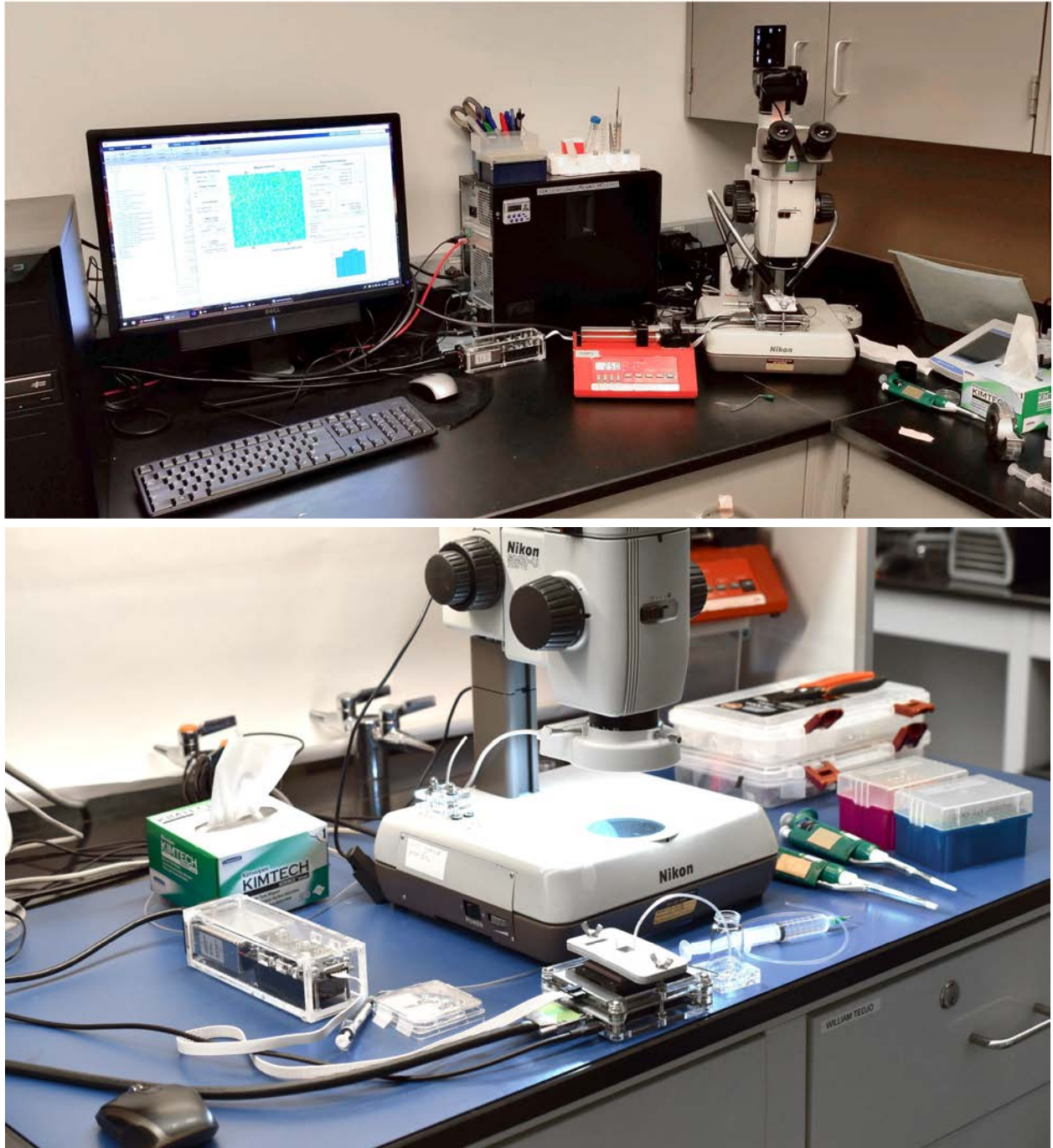


Figure 14. Generation 3: A typical system setup during wet experiments.

and separated the analog and digital circuitries. The compact 10cm×10cm dimension allows convenient of placement in a stereomicroscope. Higher performance MCU was employed to fully control and monitor the system activities in real-time. Finally, the third version extends the system's compatibility to in vitro or ex vivo experiments, where highly regulated temperature control is required. This includes significant thermal



performance improvements by redesigned the voltage regulator scheme and adding a shield board for better thermal regulations. Chapter VI discusses each PCB design iteration and its improvements in details. PCBs schematics and layout diagrams of each version are also attached in Appendix B.

### 3.4. Software Integration

The data acquisition is handled by a multichannel commercial data acquisition board with a custom-built MATLAB Graphical User Interface (GUI) with support from MATLAB Data Acquisition, Signal Processing, and Statistics Toolboxes (MATLAB, MathWorks). The DAQ board provides up to 96-channel simultaneous sampling with a resolution of the 12-bit analog to digital conversion. Any real-time and post-processing are performed in MATLAB using algorithms specifically developed for this system. Figure 14

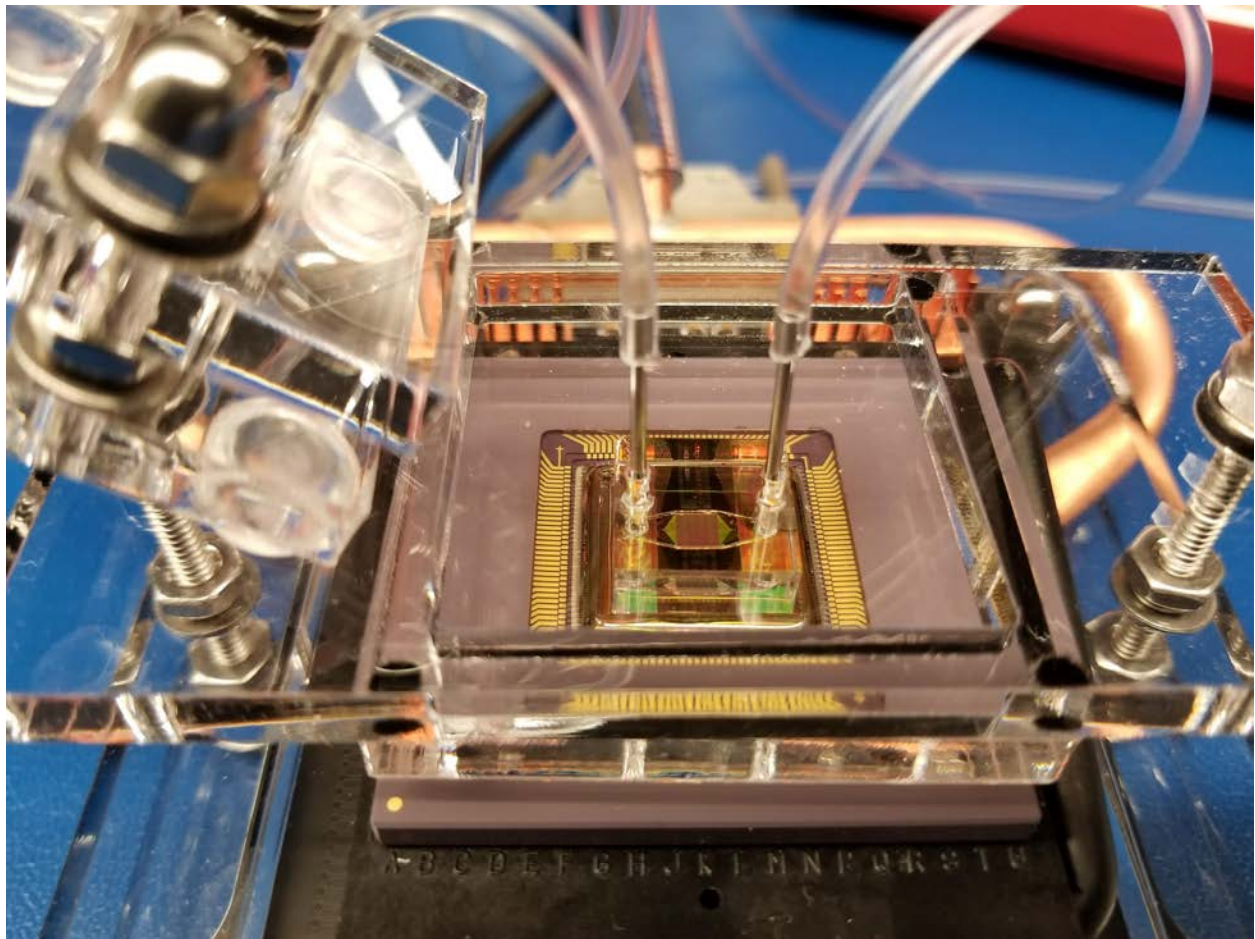


Figure 15. Generation 3: The first design of the microfluidics setup.



illustrates a typical system setup running real-time electrochemical imaging. Chapter VI discusses the software and algorithm integrations in detail. The corresponding scripts are attached in Appendix C.

### **3.5. Microfluidics Integration**

Microfluidics system offers a suitable method to perform wet experiments, flow injection for calibrating sensor, cell delivery, or even providing a proper environment for ex vivo experiments. Despite the extra steps in microfluidics device fabrication, the fully controlled fluidic flow provides predictable and repeatable responses in comparison to manual injection by pipette. Figure 15 illustrates one of the first attempt of integrating a microfluidic system to the CMOS microchip. Chapter VI discusses the microfluidics setup and fabrications in details.



## CHAPTER IV: CMOS MICROCHIP WITH MICROELECTRODE ARRAY

### 4.1. Top Level Arrangement

The CMOS MEA microchip holds the surface MEA and read channel array, where the electrochemical electron transfer and current to voltage (I-V) conversion occur, respectively. The on-chip read channel outputs are multiplexed and buffered to the external I/O pads. To assist the WE array performing three-electrode electrochemical analysis, an on-chip potentiostat is employed to set the surface MEA potentials from the RE and provide feedback current at AE.

The CMOS MEA microchip was custom designed on a standard 0.6 $\mu$ m and 3-metal layer CMOS process with an additional metal layer as the surface MEA. The choice of the 0.6 $\mu$ m CMOS process is mainly due to the availability of its post-platinization process for the MEA and adequate performance required for the target sensor applications. The post-platinization process for the MEA was described in [19], [23]. At its surface, the CMOS MEA microchip holds 128 $\times$ 128 or 16,064 WEs, 80 pairs of general-purpose interdigitated UEs, and one shared interleaved RE and AE. Each pair of UE takes an area of four WEs, hence, 16,064 total WEs instead of 16,384.

The total MEA area covers 3.6mm $\times$ 3.6mm. The array of WEs and read channels are compartmentalized into quadrants and into a group of columns with all 16,064 WE directly connected to 16,064 read channels. Electrode selection can be customized to form a desirable 2D pattern through the analog multiplexers and digital control logic. As depicted in Figure 16, the quadrants are mirrored, vertically and horizontally, for the benefit of matching and modularity of floorplan layout. In an outwards order: MEA, metal routing bus, read channel array, two-stage multiplexing, and control logic are in the direction towards each quadrant corner. In a quadrant, the read channel array is further arranged to 64 vertical columns, each containing 64 read channel cells. This specific floorplan was chosen to maximize the MEA density at the center and distribute the read channel outward, accommodating chemical imaging of up to 3.6mm $\times$ 3.6mm size live tissues with cellular-level resolution. This similar topology was used in high-density MEA integration [23], [45], [48], [50], while the in-pixel circuits topology limits the MEA density [53], [54], [56], [57].



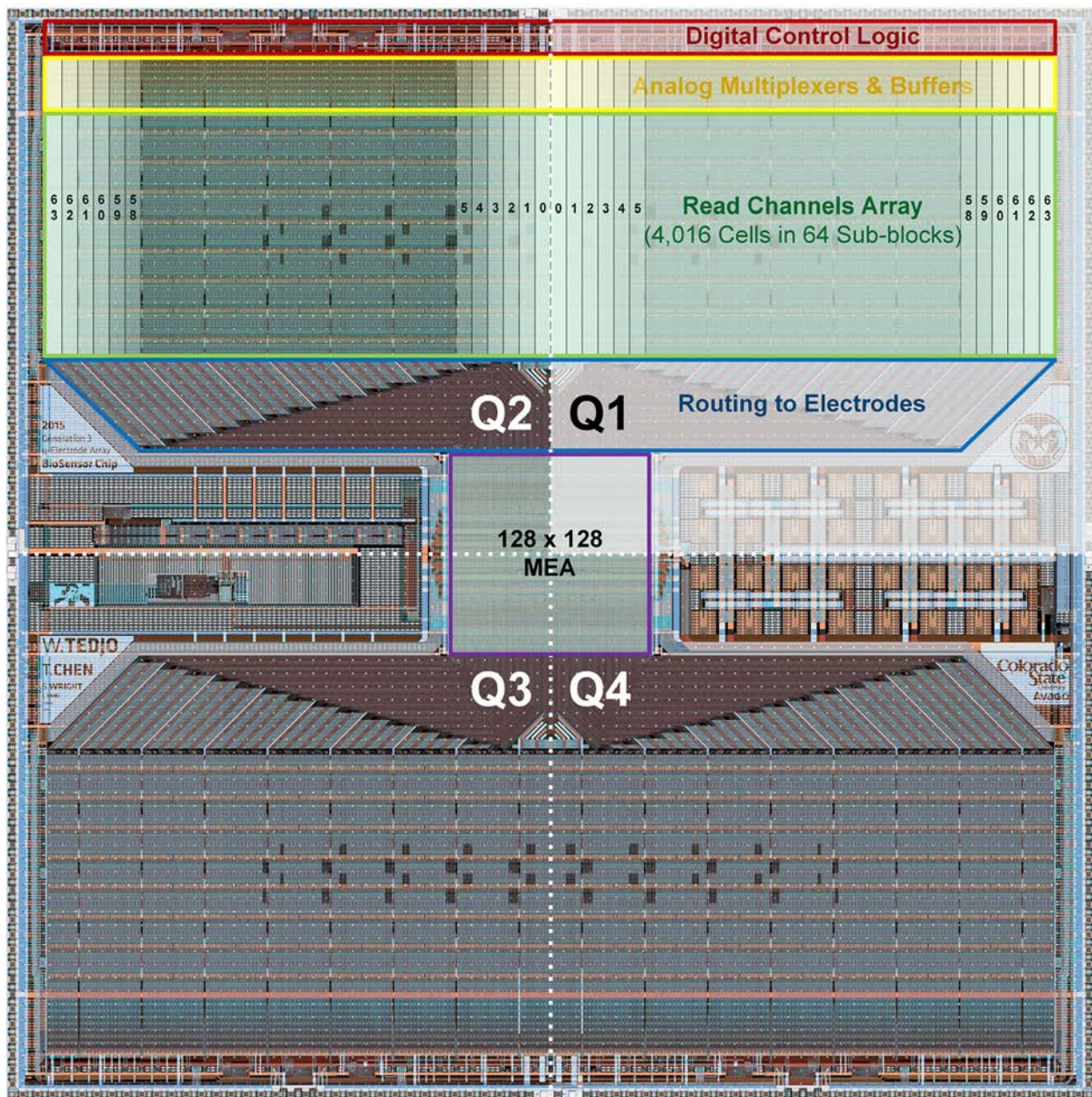


Figure 16. A micrograph of the CMOS microchip with the MEA.

The CMOS MEA microchip contains a total of 1.2 million CMOS transistors, optimized to operate within  $\pm 1.5\text{V}$  and  $\pm 2.5\text{V}$  supply rails. The choice for power supply voltage is determined by the microchip's surface MEA temperature and the upper limit of read channel dynamic range (the expected maximum input current ( $I_{IN}$ )). For most biosensing application, the desirable surface MEA temperature is  $37 \pm 0.5^\circ\text{C}$ , which corresponds to a voltage supply of around  $\pm 1.65\text{V}$ , depending on the ambient temperature. To increase system flexibility for different sensing applications, the overall power dissipation can be reduced by power



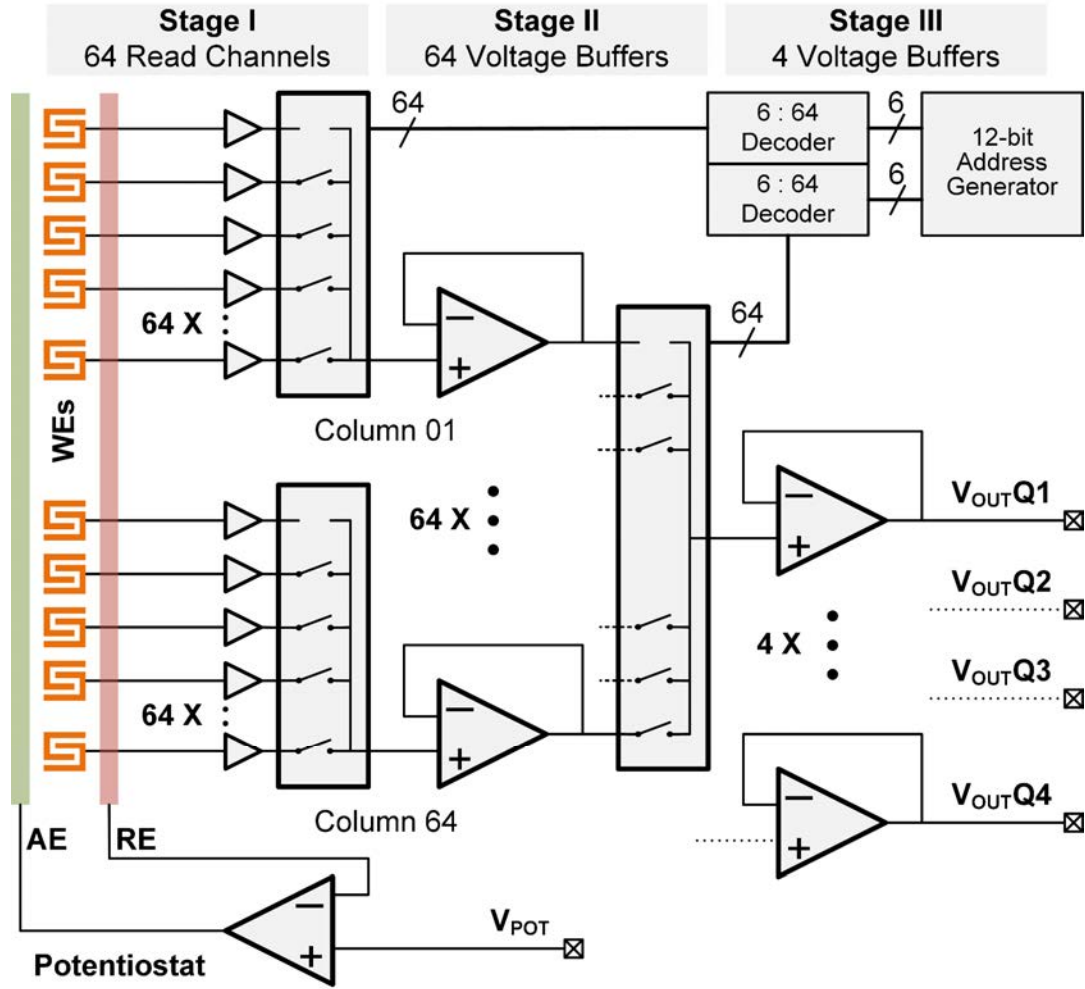


Figure 17. Quadrant & read channel array arrangement.

*Schematic diagram of the read channel array and stages of analog signal multiplexing driven by a 12-bit address generator and decoded for a fully customizable two-dimensional pattern.*

gating the read channels in each independently selected quadrant. Power dissipation of the entire CMOS MEA microchip is measured at 957mW (59.6 $\mu$ W/channel) for  $\pm 1.65$ V when four quadrants are turned on, and as low as 159mW (39.6 $\mu$ W/channel) for  $\pm 1.5$ V when only one quadrant is turned on.

In each quadrant, an array of 4,016 read channels is divided into 64 column groups. This column grouping can be visualized as thin columns shown in Figure 16 and is considered as the first stage of the multiplexing structure to get the WE current signals out to the pins. As shown in Figure 17, each multiplexer contains 64 analog transmission gates (TG) with approximately 1.0k $\Omega$  on-resistance. The TGs are controlled by a 64-wide logic bus from 6-to-64 logic decoder. The second stage of the multiplexing structure

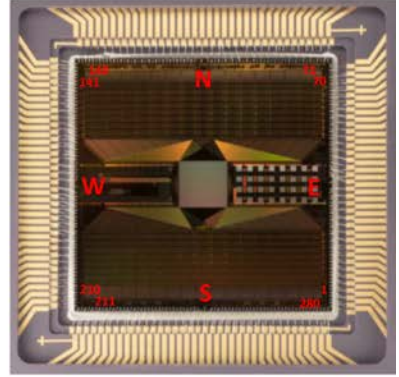


Table 2. List of CMOS MEA microchip pins and the corresponding coordinates.

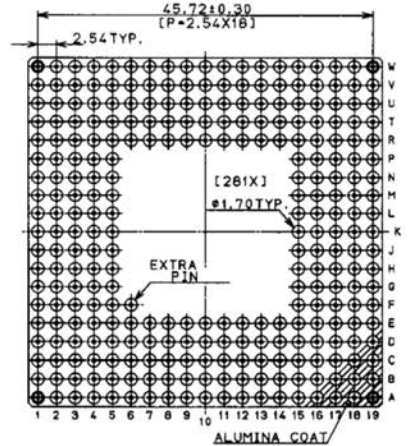
EAST			NORTH			WEST			SOUTH		
Pin	Name	Coordinate	Pin	Name	Coordinate	Pin	Name	Coordinate	Pin	Name	Coordinate
1	Sti_A<74>	E5	71	Cal_I_In<0>	Q5	141	Sti_A<34>	Q15	211	Cal_I_In<2>	F15
2	Sti_B<74>	C3	72	Sti_B<13>	S3	142	Sti_B<34>	S17	212	Sti_B<53>	C17
3	Sti_A<75>	D4	73	Sti_A<13>	R4	143	Sti_A<35>	R16	213	Sti_A<53>	D16
4	Sti_B<75>	B2	74	Sti_B<12>	T2	144	Sti_B<35>	T18	214	Sti_B<52>	B18
5	Sti_A<76>	F5	75	Sti_A<12>	Q6	145	Sti_A<36>	P15	215	Sti_A<52>	F14
6	Sti_B<76>	B1	76	Sti_B<11>	U2	146	Sti_B<36>	T19	216	Sti_B<51>	A18
7	AvddP	A1	77	Sti_A<11>	U1	147	AvddP	U19	217	Sti_A<51>	A19
8	vddN	C2	78	Sti_B<10>	T3	148	vddN	S18	218	Sti_B<50>	B17
9	Sti_A<77>	F4	79	Sti_A<10>	R5	149	Sti_A<37>	Q16	219	Sti_A<50>	D15
10	Sti_B<77>	C1	80	Sti_B<9>	U3	150	Sti_B<37>	S19	220	Sti_B<49>	A17
11	Sti_A<78>	G5	81	Sti_A<9>	Q7	151	Sti_A<38>	N15	221	Sti_A<49>	F13
12	Sti_B<78>	D2	82	Sti_B<8>	T4	152	Sti_B<38>	R18	222	Sti_B<48>	B16
13	AvddP	O3	83	Sti_A<8>	S4	153	AvddP	R17	223	Sti_A<48>	C16
14	vddN	D1	84	Sti_B<7>	U4	154	vddN	R19	224	Sti_B<47>	A16
15	Sti_A<79>	F6	85	Sti_A<7>	R6	155	Sti_A<39>	P16	225	Sti_A<47>	D14
16	Sti_B<79>	E2	86	Sti_B<6>	T5	156	Sti_B<39>	Q18	226	Sti_B<46>	B15
17	AvddP	E3	87	Sti_A<6>	S5	157	AvddP	Q17	227	Sti_A<46>	C15
18	vddN	F1	88	Cal_Vout<0>	U5	158	vddN	Q19	228	Cal_Vout<2>	A15
19	Sq_B_Vout	H5	89	DvddP	Q8	159	Sq_St<0>	M15	229	DvddP	E12
20	Sq_A_Vout	F2	90	vddN	T6	160	Sq_St<1>	P18	230	vddN	B14
21	AvddP	G4	91	vddN	R7	161	AvddP	N16	231	vddN	D13
22	vddN	F1	92	Sti_B<5>	U6	162	vddN	P19	232	Sti_B<45>	A14
23	Sq_elt_en	F3	93	Sti_A<5>	S6	163	CapSense_S<1>	P17	233	Sti_A<45>	C14
24	CapSense_en	G2	94	Sti_B<4>	T7	164	CapSense_S<2>	N18	234	Sti_B<44>	B13
25	Cal_en	H4	95	Sti_A<4>	R8	165	CapSense_Pad<0>	M16	235	Sti_A<44>	D12
26	Pstat_en	G1	96	Sti_B<3>	U7	166	CapSense_Pad<1>	N19	236	Sti_B<43>	A13
27	Pstat_Vin	J5	97	Sti_A<3>	Q9	167	CapSense_Pad<2>	L15	237	Sti_A<43>	E11
28	RE	H2	98	Sti_B<2>	T8	168	CapSense_Pad<3>	M18	238	Sti_B<42>	B12
29	CE	G3	99	Sti_A<2>	S7	169	CapSense_Pad<4>	N17	239	Sti_A<42>	C13
30	Quad_sel<1>	H1	100	Sti_B<1>	U8	170	CapSense_Pad<5>	M19	240	Sti_B<41>	A12
31	Quad_sel<0>	H3	101	Sti_A<1>	S8	171	CapSense_Pad<6>	M17	241	Sti_A<41>	C12
32	Imp_elt_en	J2	102	Sti_B<0>	T9	172	CapSense_Pad<7>	L18	242	Sti_B<40>	B11
33	Imp_out_B	J4	103	Sti_A<0>	R9	173	CapSense_Pad<8>	L16	243	Sti_A<40>	D11
34	Imp_out_A	J1	104	AvddP	U9	174	CapSense_Pad<9>	L19	244	AvddP	A11
35	AvddP	J3	105	vddN	S9	175	CapSense_vddP	L17	245	vddN	C11
36	vddN	K2	106	vddN	T10	176	AvddP	K18	246	vddN	B10
37	addr<11>	K5	107	AvddP	Q10	177	vddN	K15	247	AvddP	E10
38	addr<10>	K1	108	Sti_A<20>	U10	178	CapSense_Pad<10>	K19	248	Sti_A<60>	A10
39	addr<9>	K4	109	Sti_B<20>	R10	179	CapSense_Pad<11>	K16	249	Sti_B<60>	D10
40	addr<8>	L1	110	Sti_A<21>	U11	180	CapSense_Pad<12>	J19	250	Sti_A<61>	A9
41	addr<7>	K3	111	Sti_B<21>	S10	181	CapSense_Pad<13>	K17	251	Sti_B<61>	C10
42	addr<6>	L2	112	Sti_A<22>	T11	182	CapSense_Pad<14>	J18	252	Sti_A<62>	B9
43	addr<5>	L3	113	Sti_B<22>	S11	183	CapSense_Pad<15>	J17	253	Sti_B<62>	C9
44	addr<4>	M1	114	Sti_A<23>	U12	184	CapSense_Pad<16>	H19	254	Sti_A<63>	A8
45	addr<3>	L5	115	Sti_B<23>	Q11	185	CapSense_Pad<17>	J15	255	Sti_B<63>	E9
46	addr<2>	M2	116	Sti_A<24>	T12	186	CapSense_Pad<18>	H18	256	Sti_A<64>	B8
47	addr<1>	L4	117	Sti_B<24>	R11	187	CapSense_Pad<19>	J16	257	Sti_B<64>	D9
48	addr<0>	N1	118	Sti_A<25>	U13	188	CapSense_Update	G19	258	Sti_A<65>	A7
49	vddN	M3	119	Sti_B<25>	S12	189	vddN	H17	259	Sti_B<65>	C8
50	AvddP	N2	120	Vout<1>	T13	190	AvddP	G18	260	Vout<3>	B7
51	VDC_Cal	M4	121	vddN	R12	191	Sq_St<2>	H16	261	vddN	D8
52	TIA_CM	P1	122	DvddP	U14	192	Sq_St<3>	F19	262	DvddP	A6
53	vddN	M5	123	Cal_Vout<1>	Q12	193	vddN	H15	263	Cal_Vout<3>	E8
54	AvddP	P2	124	Sti_A<26>	T14	194	AvddP	F18	264	Sti_A<66>	B6
55	Sti_B<19>	N3	125	Sti_B<26>	S13	195	Sti_B<59>	G17	265	Sti_B<66>	C7
56	Sti_A<19>	Q1	126	Sti_A<27>	U15	196	Sti_A<59>	G19	266	Sti_A<67>	A5
57	vddN	N4	127	Sti_B<27>	R13	197	vddN	G16	267	Sti_B<67>	D7
58	AvddP	R1	128	Sti_A<28>	U16	198	AvddP	D19	268	Sti_A<68>	A4
59	Sti_B<18>	P3	129	Sti_B<28>	S14	199	Sti_B<58>	F17	269	Sti_B<68>	C6
60	Sti_A<18>	Q2	130	Sti_A<29>	T15	200	Sti_A<58>	E18	270	Sti_A<69>	B5
61	Sti_B<17>	N5	131	Sti_B<29>	Q13	201	Sti_B<57>	G15	271	Sti_B<69>	E7
62	Sti_A<17>	S1	132	Sti_A<30>	U17	202	Sti_A<57>	C19	272	Sti_A<70>	A3
63	vddN	P4	133	Sti_B<30>	R14	203	vddN	F16	273	Sti_B<70>	D6
64	AvddP	R2	134	Sti_A<31>	T16	204	AvddP	D18	274	Sti_A<71>	B4
65	Sti_B<16>	Q3	135	Sti_B<31>	S15	205	Sti_B<56>	F17	275	Sti_B<71>	C5
66	Sti_A<16>	T1	136	Sti_A<32>	U18	206	Sti_A<56>	B19	276	Sti_A<72>	A2
67	Sti_B<15>	Q4	137	Sti_B<32>	R15	207	Sti_B<55>	E16	277	Sti_B<72>	D5
68	Sti_A<15>	R3	138	Sti_A<33>	S16	208	Sti_A<55>	D17	278	Sti_A<73>	C4
69	Sti_B<14>	P5	139	Sti_B<33>	Q14	209	Sti_B<54>	F15	279	Sti_B<73>	E6
70	Sti_A<14>	S2	140	Cal_I_In<1>	T17	210	Sti_A<54>	C18	280	Cal_I_In<3>	B3

- Legends
- Positive power
  - Negative power
  - Other functions
  - Enable pins
  - Potential 8 bias
  - Addresses
  - Calibration
  - Digital power
  - Main Outputs

Top View of Silicon Die in 280-pin Package



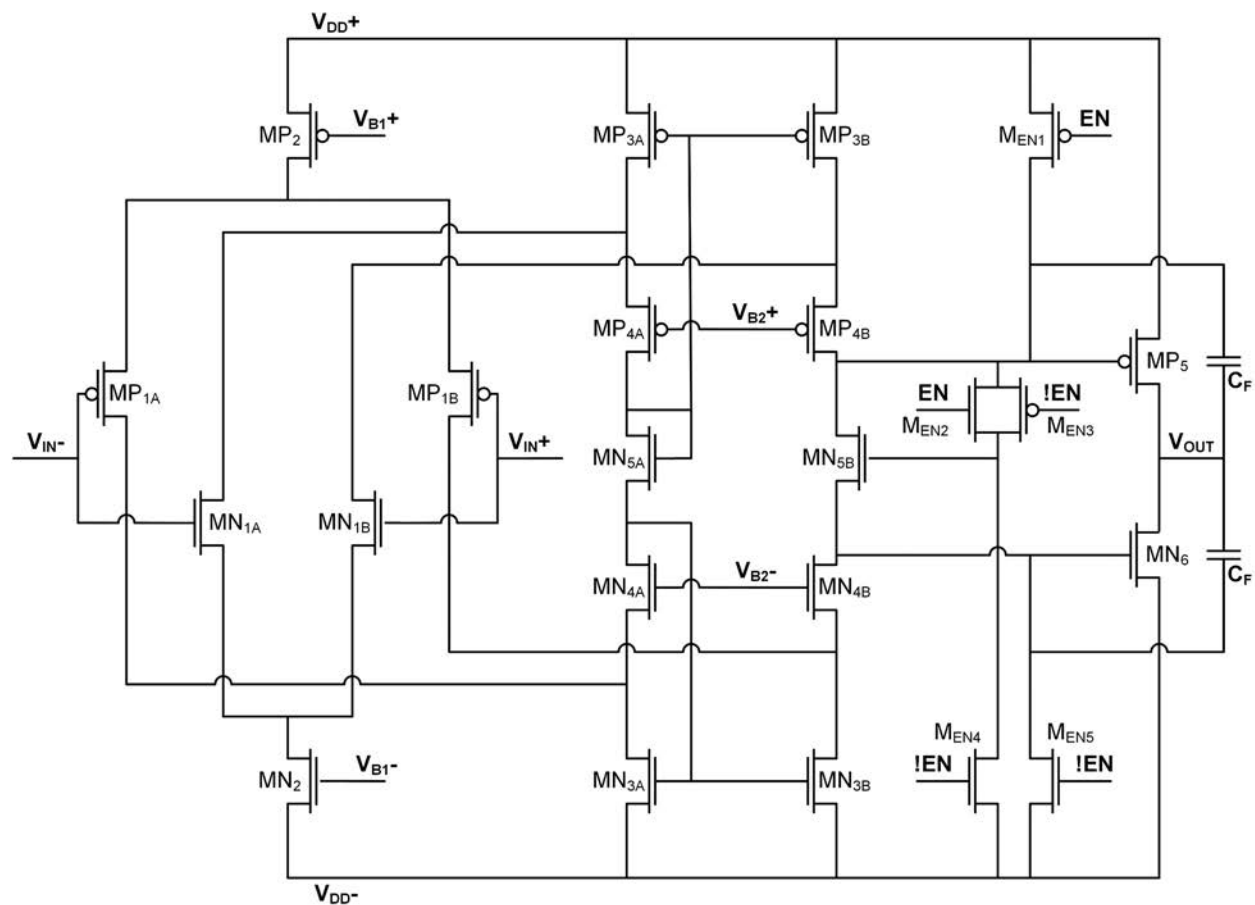
Bottom View of 280-pin Package



uses the identical multiplexers to further reduce the selection from 64 column groups to one output. At any given time, only one from 4,016 read channel in each quadrant is selected and its output voltage is buffered to the output pin assigned to the corresponding quadrant, labeled as  $V_{OUTQx}$  (where,  $x = 1,2,3,4$ ). This configuration allows the selected signal from the TIA at the first stage to propagate to the quadrant output within  $1\mu s$ , allowing scanning time of all 4,016 read channels of a quadrant in 4.1ms. A custom address cycles can be set to select any number of read channels less than 4,016, in any pattern, providing the



flexibility to select a specific area in the MEA and to increase the frame update rate based on the number of selected read channels. A complete Cadence schematic diagram can be found in Appendix A-1.





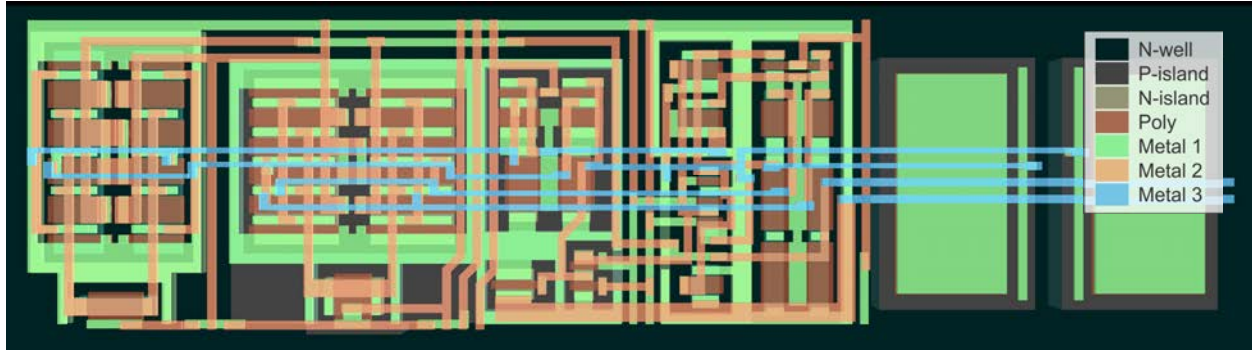


Figure 19. Operational amplifier layout arrangement.

#### 4.2. Operational Amplifier

A complementary input NMOS-PMOS folded cascoded opamp is employed as the basic building block of the analog circuit in the CMOS MEA microchip. The TIA, voltage buffer, potentiostat use the same topology with modified input/output stages to complement the input or output driving strength and voltage compliance. The opamp topology main considerations include high open-loop gain, kHz bandwidth, and low noise while minimizing silicon area and power dissipation. As depicted in Figure 18, a complementary NMOS-PMOS input stage ( $MN_1$  and  $MP_1$ ) and a push-pull output stage ( $MN_6$  and  $MP_5$ ) are utilized to allow maximum voltage swing range both at inputs and output terminals of the opamp. The folded cascode topology ( $MN_{3-4}$  and  $MP_{3-4}$ ) generates a high impedance gain stage, boosting the open-loop gain, and equipped with high swing voltage bias generated by  $MN_5$ .  $VB_{1+}$  and  $VB_{1-}$  bias the differential input-stage tail transistors ( $MN_2$  and  $MP_2$ ) and  $VB_{2+}$  and  $VB_{2-}$  bias the active load of the high-swing flooded cascode. These bias points are driven by on-chip independent bias circuits and can be connected to  $V_{DD+}$  for  $VB_{1+}$  and  $VB_{2+}$ , and to  $V_{DD-}$  for  $VB_{1-}$  and  $VB_{2-}$ , to set  $MP/N_2$  and  $MP/N_4$  to a cutoff region, stopping any current flow on the corresponding branches. The power gating can be further ensured by including enable transistors, ( $M_{EN}$ ), which their gates are merged and controlled by a single enable pin. These enable transistors would set the output stage transistor to cutoff region when the disable signal is activated. All circuit in the CMOS microchip MEA (i.e. TIA read channels, potentiostat, and voltage buffers) use the same topology, however, with slight output stage driving strength improvements to accommodate greater capacitive and/ or resistive load.

The opamp layout was custom-fitted to a  $134.4\mu\text{m} \times 35.7\mu\text{m}$  silicon area. As shown in Figure 19, the opamp layout was designed to be modular, can be easily attached side by side with other opamps to form an array. The layout is geometrically resembles the schematic shown in Figure 18, with the most left



part is the NMOS input stage followed by PMOS input stage, the middle part is the folded-cascoded, and the output stage with two blocks of stabilization capacitor ( $C_F$ ) on the very right side. Any connections towards the edge of the opamp on Metal 1 to 3, are the inputs and outputs; and shared enables, biases, and power connections. Metal 1 mostly forms the substrate connections to the N-Island and P-Island strips around the transistors, which is also behaves as a guard ring, isolating the enclosed transistors from the substrate noise. Other Metal 1 connections are also used for joining the split and matched transistors, such as the input transistors, MP/N<sub>1</sub>. Metal 2 connects within different functional parts of the opamp (e.g. bias tail to the input stage, or input stage to the folded-cascoded), and to the outside of the opamp for voltage bias purpose (e.g. VB<sub>1</sub>, VB<sub>2</sub>, and EN). These connections are shown as vertical tracks around the center part of the opamp layout as illustrated in Figure 19. Metal 3 responsible for connecting some inter-opamp connections (e.g. the  $C_F$  capacitor to the necessary branches) and mainly for providing V<sub>IN</sub> and V<sub>out</sub> connections to and from the opamp, which is shown as horizontal tracks in Figure 19. These vertical and horizontal tracks carrying biases and enables signal are shared over the entire chip with other 16-thousands read channels.

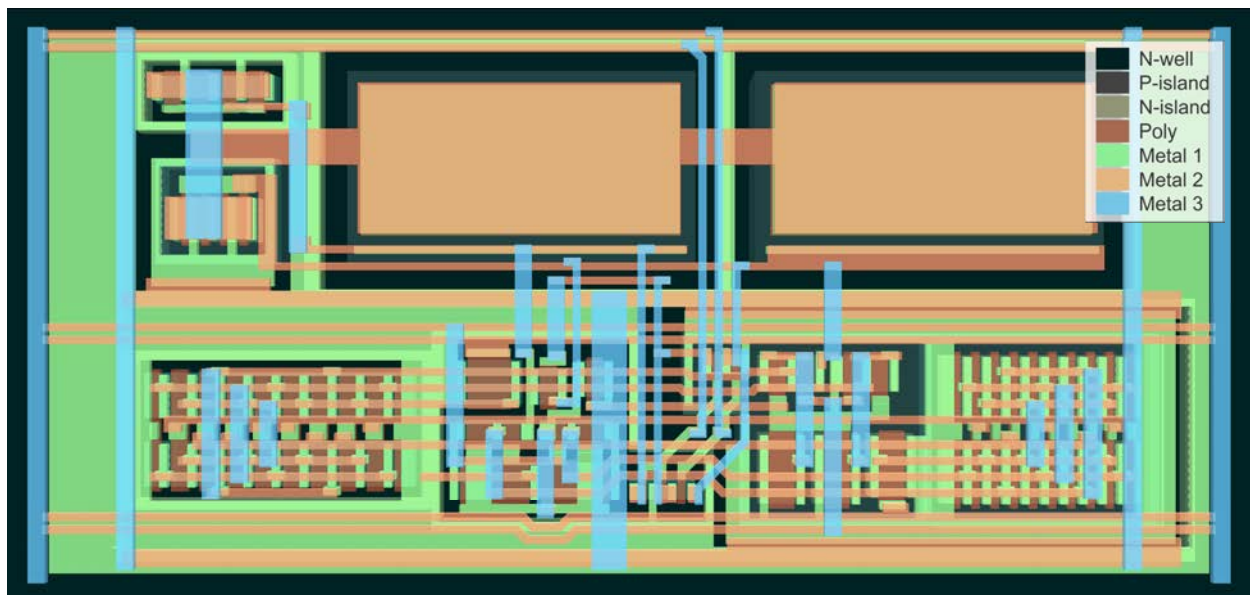


Figure 20. Potentiostat layout arrangement.



### 4.3. Potentiostat

The on-chip potentiostat circuit utilizes a single-opamp configuration with the same topology as the opamp of the TIA, however, with higher output driving strength. In the CV mode, the output stage was designed to handle up to 5kV/s scan rate while driving millifarads capacitive load at the AE, the worst-case estimation of  $C_{DL}$  forms at the surface of AE and RE. The potentiostat requires not only high current drive strength, but also better response to a wide range of capacitive load. The output would drive an entire AE area in the MEA with significant capacitive load fluctuations during electrochemical reactions. Therefore, as depicted in Figure 20, the large size of 2pF stability feedback capacitor  $C_F$  was selected to ensure high stability. In a case of limited current strength or voltage compliance, the on-chip potentiostat can be fully disabled to allow the external potentiostat with higher compliance voltage to take over the connection to RE and AE. A detailed schematic of the potentiostat transistor sizing can be found in Appendix A.

### 4.4. Transimpedance Amplifier

Input current to output voltage conversion is performed by a continuous Trans-Impedance Amplifier (TIA). The opamp topology is based on a folded cascode structure with complementary NMOS-PMOS input stage and a push-pull output stage. This configuration was chosen to maximize input and output rail-to-rail

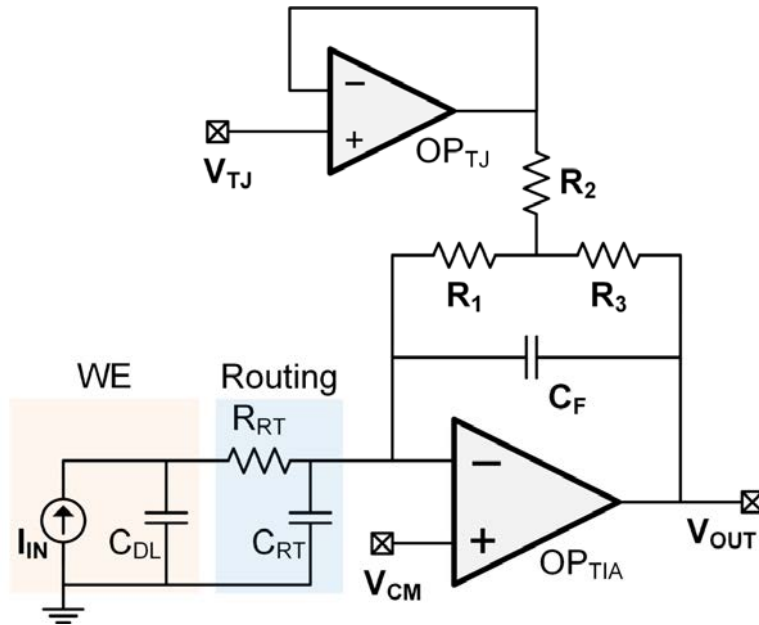


Figure 21. TIA with T-Network feedback resistor as read channel.



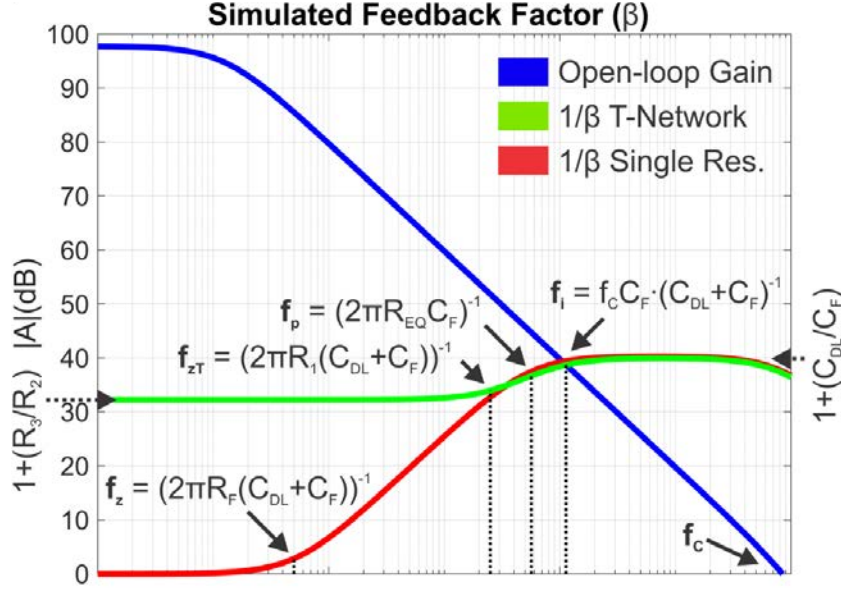


Figure 22. Frequency response of opamp, noise analysis, and stability condition in TIA.

swing. A T-network feedback resistor configuration was selected to accommodate the area constraints involved in fitting 16,064 read channels and the large transimpedance gain.

As depicted in Figure 21, the OP<sub>TIA</sub> with R<sub>1-3</sub> forms the T-network TIA. OP<sub>TJ</sub> in unity-gain configuration provides voltage buffering and isolation from an off-chip external bias source (V<sub>TJ</sub>). The double-layer capacitance formation at WE surface is approximated as C<sub>DL</sub>, while the interconnect routing parasitic from WE to TIA input is modeled as R<sub>RT</sub> and C<sub>RT</sub>. C<sub>RT</sub> was extracted to be ~1pF and C<sub>DL</sub> was approximated to be 100pF. The C<sub>DL</sub> approximation uses the worst-case scenario capacitance coefficient (250μF/cm<sup>2</sup>) based on a previous study [89] and the WE surface area (427.5μm<sup>2</sup>). Therefore, the capacitance at the TIA input node is expected to be dominated by C<sub>DL</sub>. A feedback capacitor (C<sub>F</sub>) limits the TIA and its noise bandwidth and provides compensation to ensure transient response stability. The equivalent resistance is defined by Eq. (6), resulting in output potential shown in Eq. (7). Assuming constant 0V and low-noise voltage sources are connected to V<sub>CM</sub> and V<sub>TJ</sub>, the I-V conversion is simplified to (8).

$$R_{EQ} = R1 + R3 + \left(\frac{R1 \cdot R3}{R2}\right) \approx R1 \cdot \left(1 + \frac{R3}{R2}\right) \quad (6)$$

$$V_{OUT} = -I_{IN} \cdot R_{EQ} + \left(1 + \frac{R3}{R2}\right) \cdot V_{CM} - \left(\frac{R3}{R2}\right) \cdot V_{TJ} \quad (7)$$

$$V_{OUT} = -I_{IN} \cdot R1 \cdot \left(1 + \frac{R3}{R2}\right) \quad (8)$$



The Common Mode Voltage ( $V_{CM}$ ) and T-Network Feedback Voltage ( $V_{TJ}$ ) are globally shared and their voltage potentials can be customized off-chip.  $V_{CM}$  defines the potential of WEs and should be set at constant 0V for electrochemical reading, or pulsed for electrode cleaning [21], [83].  $V_{TJ}$  should be finely adjusted to regulate the  $V_{OUT}$  DC offset near 0V, the middle voltage potential between the power supply rails.

The T-network noise performance is defined by the product of noise sources (i.e. resistor thermal noise and opamp noise) and the inversed feedback factor or noise gain ( $1/\beta$ ). Equations (9) and (10) describe the noise gain of a traditional single resistor ( $R_F$ ) TIA and a T-network TIA, respectively. Figure 22 illustrates the  $OP_{TIA}$  open-loop gain associated with the noise gain for both TIA cases with equal resistive feedback values ( $R_F = R_{EQ}$ ). At DC, the T-network configuration demonstrates noise gain magnitude of  $(1 + R_3/R_2)$  in comparison to the unity-gain in the single resistor TIA, showing that the T-network configuration amplifies the low-frequency noise of  $OP_{TJ}$  and  $OP_{TIA}$  by  $(1 + R_3/R_2)$ . While keeping the same  $R_{EQ}$ , reducing  $(1 + R_3/R_2)$  and increasing  $R_1$  would push the T-network zero ( $f_{zT}$ ) approaching the single resistor zero ( $f_z$ ) at a lower frequency.

$$\frac{1}{\beta(s)} = \frac{f_z(s)}{f_{pf}(s)} = \frac{1 + R_F(C_{DL} + C_F)(s)}{1 + R_F C_F(s)} \quad (9)$$

$$\frac{1}{\beta_T(s)} = \frac{f_{zT}(s)}{f_{pf}(s)} = \left(1 + \frac{R_3}{R_2}\right) \left(\frac{1 + R_1(C_{DL} + C_F)(s)}{1 + R_C C_F(s)}\right) \quad (10)$$

$$V_n = \left(1 + \frac{R_3}{R_2}\right) \sqrt{(e_{nR_1})^2 + e_{ne}^2 + e_{nTJ}^2 + e_{ni}^2} \quad (11)$$

The total rms output noise ( $V_n$ ) in a T-network TIA can be summarized to Eq. (11), where  $e_{nR_1}$  is the thermal noise of  $R_1$ ,  $e_{ne}$  and  $e_{nTJ}$  are the opamp noise of  $OP_{TJ}$  and  $OP_{TIA}$ , respectively, and  $e_{ni}$  is the noise generated by the opamp input bias current, a negligible value in the CMOS process. All noise components are amplified by  $(1 + R_3/R_2)$ . Considering the area and power dissipation constraints and noise gain behavior of TIA using T-network, the  $1/f$  noise of  $OP_{TIA}$  and  $OP_{TJ}$  are expected to dominate  $V_n$ . Therefore, minimizing  $1/f$  noise of the opamp input stage and careful balancing between  $R_1$  and  $R_3/R_2$  were the highest priority for optimizing the noise performance of the read channel.



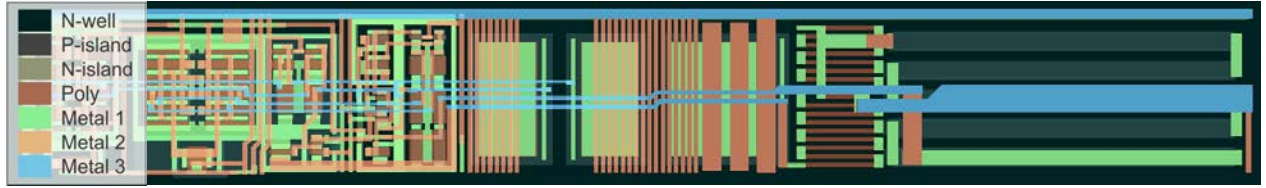


Figure 23. TIA layout arrangement.

Stability performance of a TIA is defined at higher frequency, specifically, at the intersection between the noise gain and the  $OP_{TIA}$  open-loop gain at  $f_i$ . A stable TIA requires  $f_p \leq f_i$ , when the noise gain is at 0db/decade rate when intersecting the -20dB/decade open-loop gain. The 20db/decade difference at the intersect point indicates a 45° TIA phase margin. An illustration in Figure 22 shows that larger  $C_{DL}$  pushes  $f_{zT}$  and  $f_i$  to a lower frequency and raises the  $(1+C_{DL}/C_F)$  plateau while leaving  $f_p$  unchanged. An extremely large  $C_{DL}$  could push  $f_i$  lower than  $f_p$ , causing noise gain to intersect the  $OP_{TIA}$  open-loop gain at +20db/decade, generating unstable condition. Therefore, a proper approximation of  $C_{DL}$  is needed to optimize the TIA stability while minimizing  $C_F$ .

Based on previous results of electrochemical ex vivo experiments [23], the equivalent feedback resistance ( $R_{EQ}$ ) of the TIA was set to  $R_{EQ} = 20M\Omega$  ( $R_1 = 480k\Omega$ ,  $R_2 = 0.6k\Omega$ , and  $R_3 = 24k\Omega$ ).  $R_{1-3}$  values were selected to accommodate the upper limits of the input current dynamic range to tens of nanoamperes without sacrificing sensitivity due to the noise gain factor  $(1+ R_3/R_2)$ .  $C_F$  is set at 1pF to maintain stable transient response with the worst-case approximated  $C_{DL}$  value of 100pF.

The layout of the TIA is an extension of the previously mentioned opamp layout in Figure 19 with additional  $R_1$  to  $R_3$ , and  $C_F$ . As shown in Figure 23, the largest area is allocated for  $R_1$ , positioned on the most right, to reduce  $R_{EQ}$  across the read channel array where its absolute value is important. In contrary, the absolute values of  $R_2$  and  $R_3$  are less of a concern, since only the ratio defines  $R_{EQ}$ . Therefore,  $R_2$  and  $R_3$ , located to the left of  $R_1$ , are split to multiple fingers and one-dimensionally matched. The 2pF  $C_F$  shares the same substrate area as the opamp feedback capacitor to minimize the layout area. The layout design is modular and can be mirrored, flipped, and combined to form a group of 2×2 TIA, while sharing their biases and enable signal tracks.

To ensure proper voltage bias point with the flexibility of DC offset adjustment,  $V_{TJ}$  is externally controlled off-chip and buffered by on-chip buffer ( $OP_{TJ}$ ) as shown in Figure 21. In an extreme case, assume



that the input current  $I_{IN} = -100\text{nA}$  with TIA equivalent gain of  $20\text{M}\Omega$  and  $V_{TJ} = V_{CM} = 0\text{V}$ , the output voltage approximately yields  $2.0\text{V}$ . To provide adequate current sourcing, the output stage of  $OP_{TJ}$  must be able to source  $2\text{V} / (R_1 || R_2 + R_3)$  which is approximated as  $2\text{V} / 24.6\text{k}\Omega = 81.3\mu\text{A}$ . To reduce silicon area usage, a group of 64 read channels shares one  $OP_{TJ}$  buffer, therefore, requiring  $OP_{TJ}$  buffer output stage with minimum current driving strength of  $64 * 81.3\mu\text{A} = 5.2\text{mA}$ . The topology used for  $OP_{TJ}$  buffer is similar to the opamp described in the previous section, however, with much larger NMOS PMOS push-pull output stage current drive.

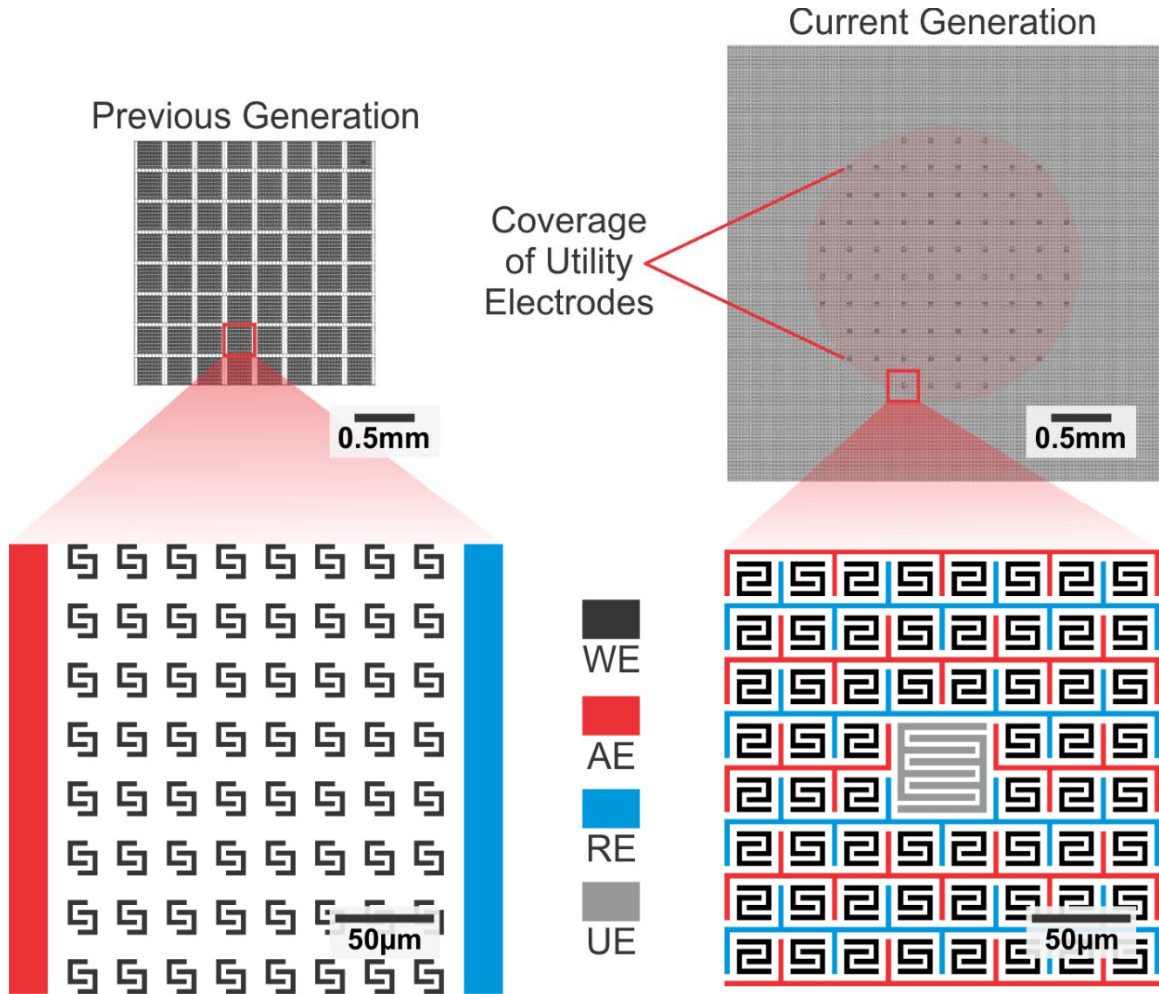


Figure 24. Electrode array layout, comparison of Generation 2 and 3 MEA.

*A segment of MEA showing 8x8 WE array comparing the geometrical modification over previous generation microchip's MEA configuration. The current generation configuration shows a more compact WE array with alternating AE and RE in comparison with the previous generation electrode configuration with two large blocks of AE and RE confining a sparse array of WEs. One pair of UE, a four-finger interdigitated electrode, with four times the size of WE, is also shown in this segment of the current generation of MEA. There are 80 UEs pair evenly distributed in the center area of the MEA.*



#### 4.5. Surface Electrodes

As shown and compared in Figure 24, the improved surface MEA configuration used the same fabrication process based on our previous designs [19], [23]. Each pair of C-shaped electrodes are shorted at the lower metal layer to form a single WE to cover a  $17.5\mu\text{m}\times 17.5\mu\text{m}$  cross-section area. The WE array is configured in uniform horizontal and vertical pitch of  $27.5/27.5\mu\text{m}$ , in comparison to  $25.5/30.4\mu\text{m}$  pitch on the previous work. The WE surface area, which is the main factor in determining the amount of steady-state current flow in microelectrode [38], is also improved by 37% from  $317.5\mu\text{m}^2$  to  $435\mu\text{m}^2$ . Each WE are routed from the middle area of MEA towards the corner of each quadrant to the input of read channel using combinations of all metal layer with metal width ranging from  $0.5\mu\text{m}$  to  $0.7\mu\text{m}$ . The thin metal routing over millimeters length was simulated to be approximately  $500\Omega$  to  $2,000\Omega$  parasitic resistance and  $0.5\text{pF}$  to  $2.5\text{pF}$  parasitic capacitance for various combinations of electrode-to-read channel distance in the chip.

The improved RE and AE use a  $2.5\mu\text{m}$  wide metal line which encloses and interleaves within the WE array. The interleaved RE and AE geometry create enclosed areas around each WE to reduce the effect of overlapping diffusion area within WE and to improve the localization of redox cycle in the highly-dense WE array [50], [53], [90]. The RE and AE are shared among all WEs and can be connected to the on-chip or external potentiostat. Disabling the on-chip potentiostat would allow a flexibility direct connection to the surface RE and AE for external potentiostat connection, combined to form RE or AE, or left inactive. In addition, 80 pairs of interdigitated UEs are distributed around the center area of the MEA array. With a minor spatial compromise of four WEs per UE, each UE provides direct access from the surface MEA to the external I/O pads to allow future integration of biosensing features, such as impedance sensing and electrical stimulation.

#### 4.6. Array Integration

Figure 25 depicts the silicon layout of the read channel array. As reflected by the read channel schematic in Figure 21,  $R_1$  ( $480\text{k}\Omega$ ) uses a high sheet resistivity but a low-precision N-type material, whereas  $R_2$  ( $0.6\text{k}\Omega$ ) and  $R_3$  ( $24\text{k}\Omega$ ) use a high-precision but low sheet resistivity polysilicon material. This arrangement creates approximately  $20\text{M}\Omega$  equivalent feedback resistance with  $15\times$  smaller layout footprint than an equivalent  $20\text{M}\Omega$  low-precision N-type resistor.  $R_3$  and  $R_2$  were matched to minimize the  $R_3/R_2$







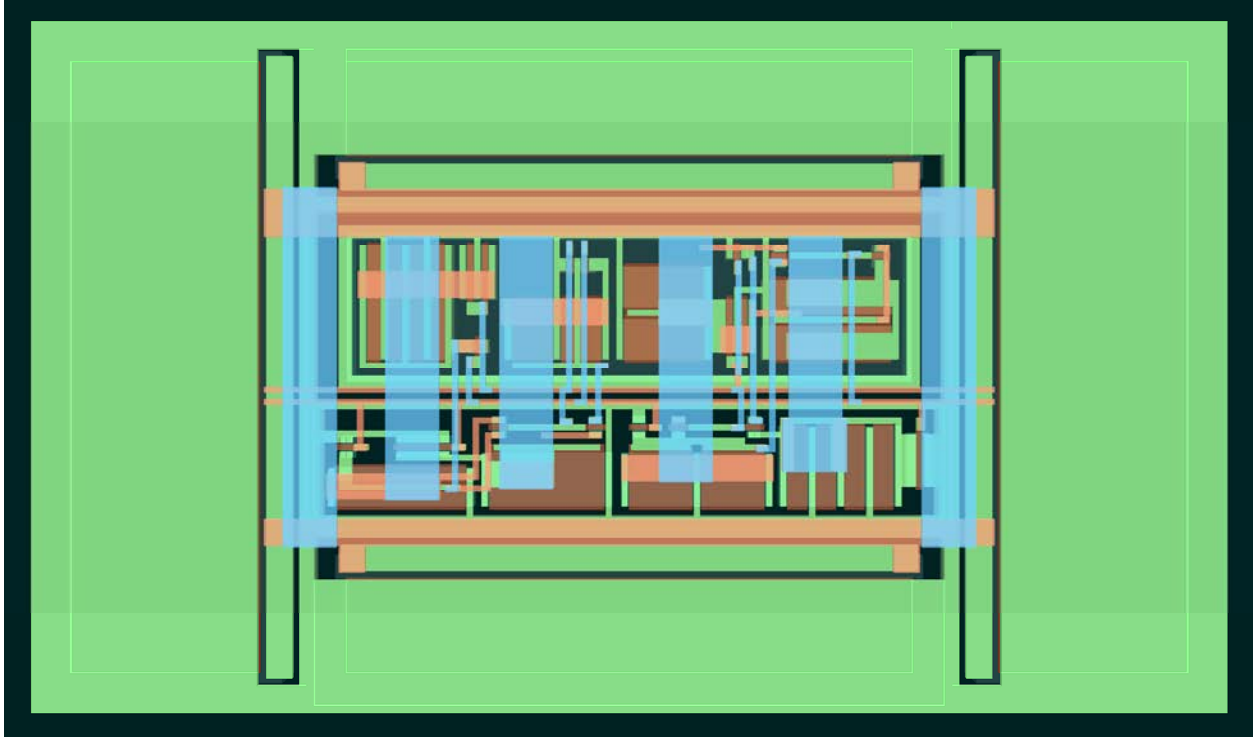


Figure 26. Global bias circuit layout arrangement.

Each read channel circuit is closely associated with neighboring read channels with the shared silicon substrate and metal routing patterns vertically and horizontally. As illustrated in Figure 25, input current and output voltages are routed vertically (Metal 3), whereas voltage biases (e.g.  $V_{CM}$  and  $V_{TJ}$ ) and power lines are shared horizontally (Metal 2). This metal routing pattern is optimized to accommodate modularity between read channels and the limitations of the three metal layers available in this CMOS process.

#### 4.7. Global Bias Circuit

The global bias circuit responsible to provide a constant DC bias to all opamp in a quadrant, which includes 4 thousand TIA read channels and analog buffers at State II and III. In reference to Figure 18, the bias circuit generates  $VB_1$  and  $VB_2$ , for both the positive and negative polarity. The bias circuit takes the quadrant enable input to control  $VB_1$  and  $VB_2$ , shorting the positive biases to  $V_{DD+}$  the negative biases to  $V_{DD-}$ , disabling all PMOS and NMOS in the opamp of a quadrant as previously described in Section 4.3. As



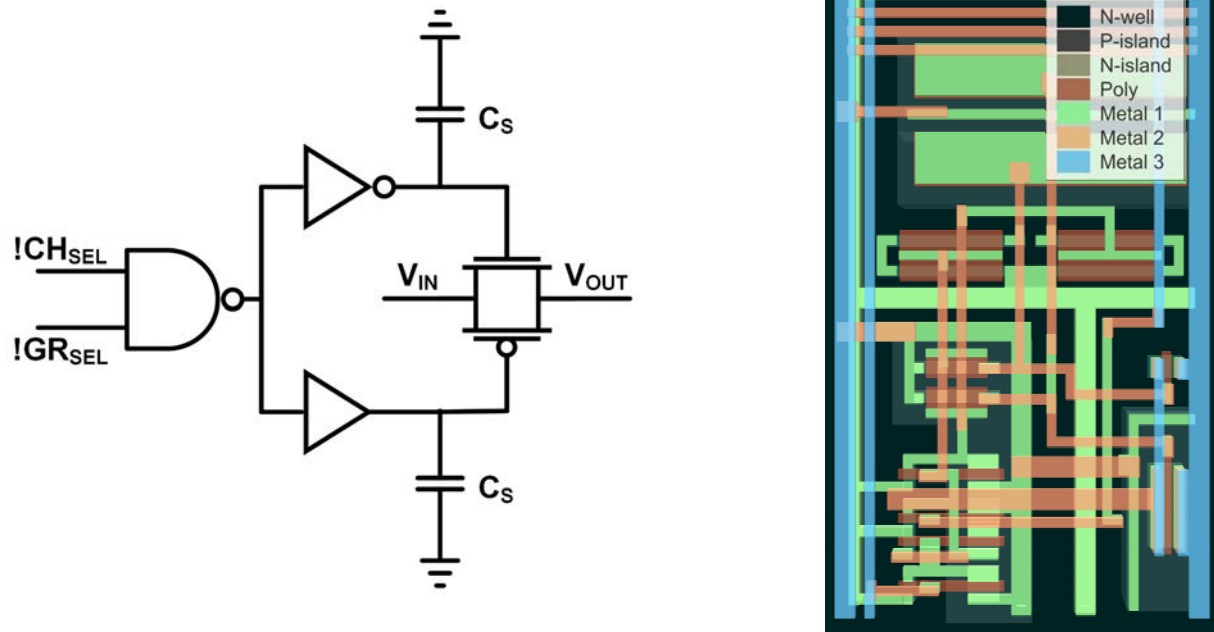


Figure 27. Analog transmission gate switch with the smooth transitioning driver.

shown in Figure 26, the transistor sizes were maximized and made square ( $W/L \sim 1$ ) to minimize eth current variation, thus, the voltage variations due to CMOS fabrication inconsistencies.

#### 4.8. Analog Multiplexer

The array of multiplexers and its corresponding decoders are utilized to translate the 12-bit address input to select a single voltage output from a read channel to be reflected as a signal output from a quadrant. The multiplexer array is in a simple complementary NMOS-PMOS TG switch with delay capacitors attached to the driver. The complementary design is to ensure a symmetrical response to both positive and negative rail, with minimum on resistance variation over a wide range of input/output voltages. The TG was sized to have approximately  $1k\Omega$  on resistance, a good balance between area and resistance for nA current flow. As shown in Figure 27, the driver of TG, pairs of inverter and buffer, are loaded with 125fF capacitors  $C_S$  to reduce transition response down to  $\sim 50ns$  from  $\sim 1ns$ . A sharp transition would create kickback noise to the read channel output and increase the chance of having the switching current noise reflected at WE. A slower transition also provides margins for clock skew variations of the signal generated by the decoder (the channel and group select signals), which would reduce the risk of drive fights between two transitioning



read channels. Referring to Figure 17, the channel select ( $CH_{SEL}$ ) receives a signal from the Stage I decoder, while Group select ( $GR_{SEL}$ ) receives a signal from the Stage II decoder. The layout arrangement of the TG and its drivers follow the same width as an opamp or a TIA, hence it can be placed in parallel with the corresponding read channel on the array as depicted in Figure 16.

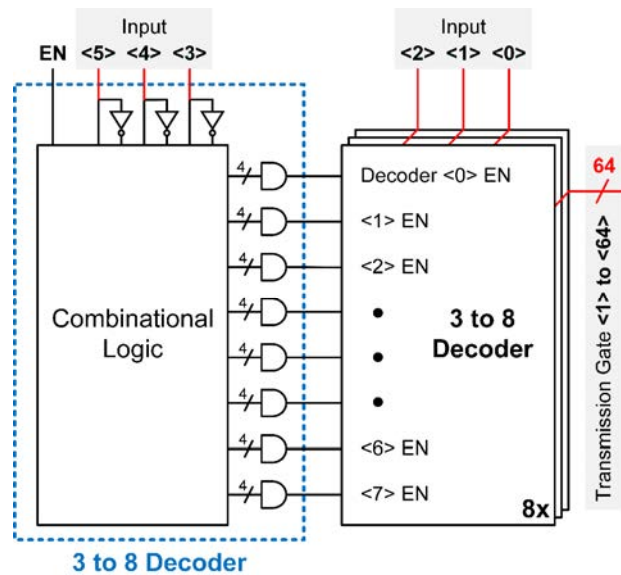


Figure 28. 6 to 64 decoder building blocks.

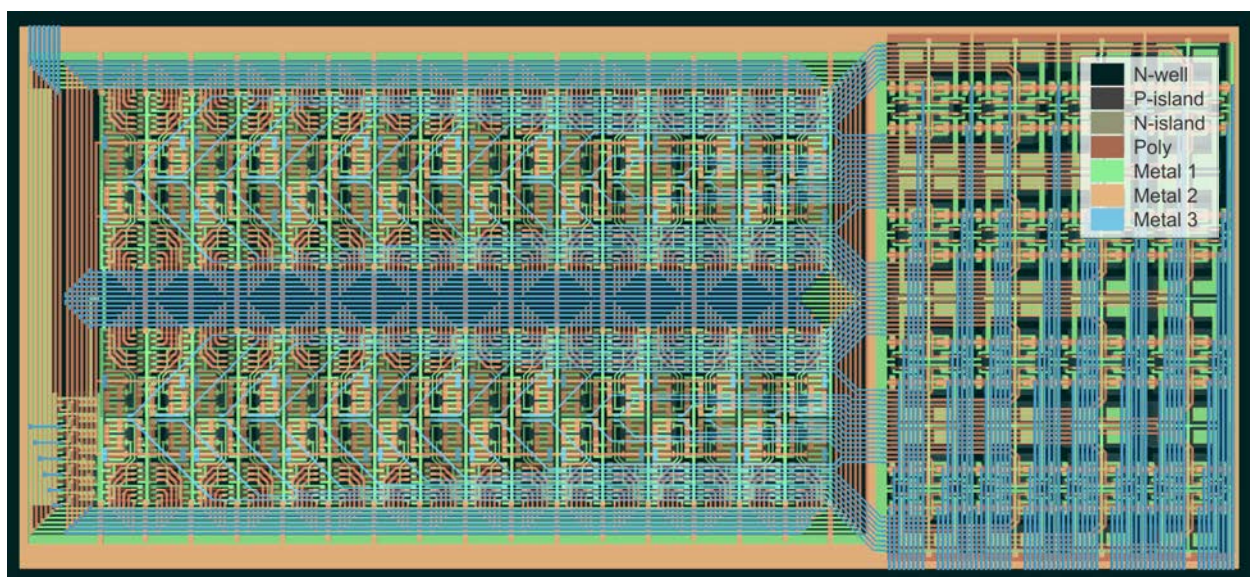


Figure 29. 6 to 64 decoder layout arrangement.



#### **4.9. 6 to 64 Decoder**

The decoder takes 6-bit inputs and generates a select signal on one of its 64 outputs. As illustrated in Figure 28 diagram, the 6 to 64 decoder consists of two-stage of 3 to 8 decoders with the last stage uses the MSB part (bit 0 to 2) of the input and the first stage uses the LSB (bit 3 to 5). Each 3 to 8 decoder consist of combinational logics utilizing up to 4-input NAND gates. As seen in Figure 17, for each quadrant, two of the 6 to 64 decoders work in conjunction to control the selection of 4,016 read channel to one output. The rest of 80 addresses are not used due to the replacement by UEs. The layout configuration of 6 to 64 decoder is shown in Figure 29.



## CHAPTER V: CMOS MICROCHIP SIMULATIONS

All schematic design and extracted layout simulations were performed using Cadence® Virtuoso®. Schematic and Layout using the provided A10 model of the 0.6 $\mu$ m CMOS process. The circuit simulations were run at different variation corners, extreme temperatures, a wide range of voltage supplies, and three process variations. Process variation incorporates gate oxide thickness ( $T_{ox}$ ), random doping fluctuations (RDF), and device geometry due to lithography inconsistencies. These variations mostly define the threshold voltage of individual transistors, defining the general DC and AC performances (gain, bandwidth, and noise) in an opamp. Nevertheless, there are other factors in designing an amplifier to ensure good integration to a large array configuration, where circuits performance discrepancies become more apparent in a large 19.0mm $\times$ 19.0mm die area due to inconsistencies during CMOS fabrication processes.

First, suppressing mismatch within paired transistors play an important role in circuits in general sense. In an opamp, transistor mismatch defines input offset that ultimately would be reflected at the output of the amplifier circuit. The input differential pairs, as the major source of the offset, should be sized properly to minimize the mismatch effect while keeping practical total transistor area. In the case of integrating 16 thousand read channels, the transistor gate area should be minimized, while the offset should be kept small enough that a post-processing calibration algorithm is feasible.

Second, the ability to simulate current leakage at extremely low current is crucial because the level of electrochemical current signal in microelectrode is expected to be within nano to single pico-amperes. Unknown level of current leakage would promote a risk in railing voltage output in a highly resistive TIA as previously occurred in the Generation 2 CMOS microchip. However, the AMOS10A does not provide an adequate current leakage model. Hence, one should perform a proper design practice to minimize this leakage current; for an example, avoids the high current path of a CMOS source or drain to be directly connected to the input of TIA, and provides enough margin for the leakage current variation across the 16 thousand read channel.

This chapter discusses and presents the simulation results based on iterations of modeling cycle between schematic and layout views as illustrated in Figure 30. The flowchart represents a workflow starting from a cell level of a single TIA read channel to the block level consisting of a group of 64 read channels



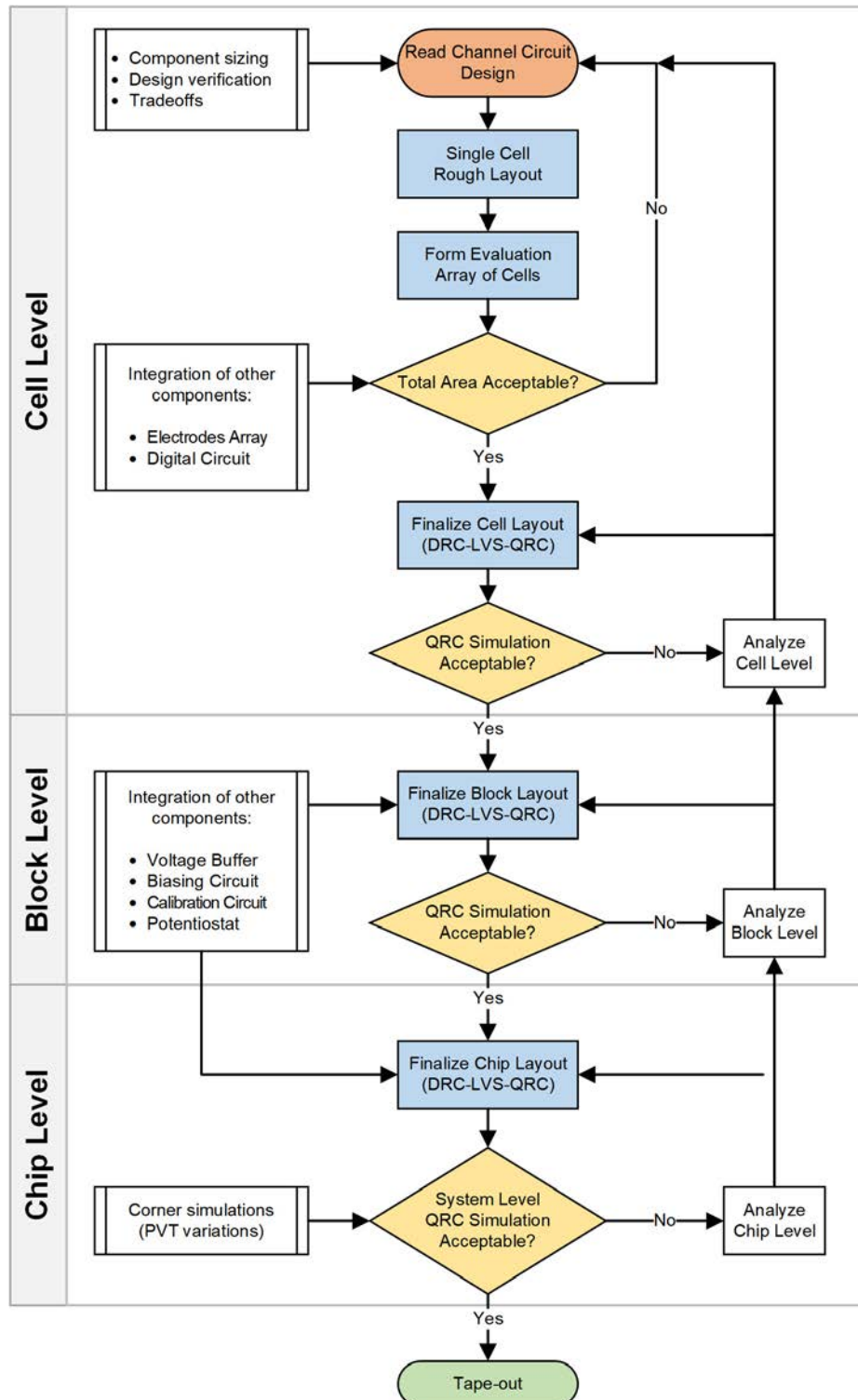


Figure 30. Flowchart of hierarchical schematic and layout simulations.

with the multiplexer and voltage buffer on Stage II, then the array level at a quadrant of the microchip shown at Stage III in Figure 17. This workflow shows a strong dependency from a single read channel's area to



the entire process of the schematic and layout process. A high performing opamp alone (e.g. high-open loop gain and bandwidth, and low noise and DC offset) would not be enough to meet the end goal of integrating all 16 thousand read channel. Thus, the physical area must be minimized, and each transistors W and L have to be fine-tuned to achieve this end goal. Moreover, as shown previously in Figure 25, the routings of bias voltages and input/output from the read channels array create additional challenges due to the limited number of three metal layers available to the process.

To represent the best approximation, any capacitive and resistive load values are sized based on the combination of constants calculation provided by the Design Rules, and values generated from the RC parasitic extraction. DC, AC, transient, noise, and Monte-Carlo (MC) analyses were performed to verify the

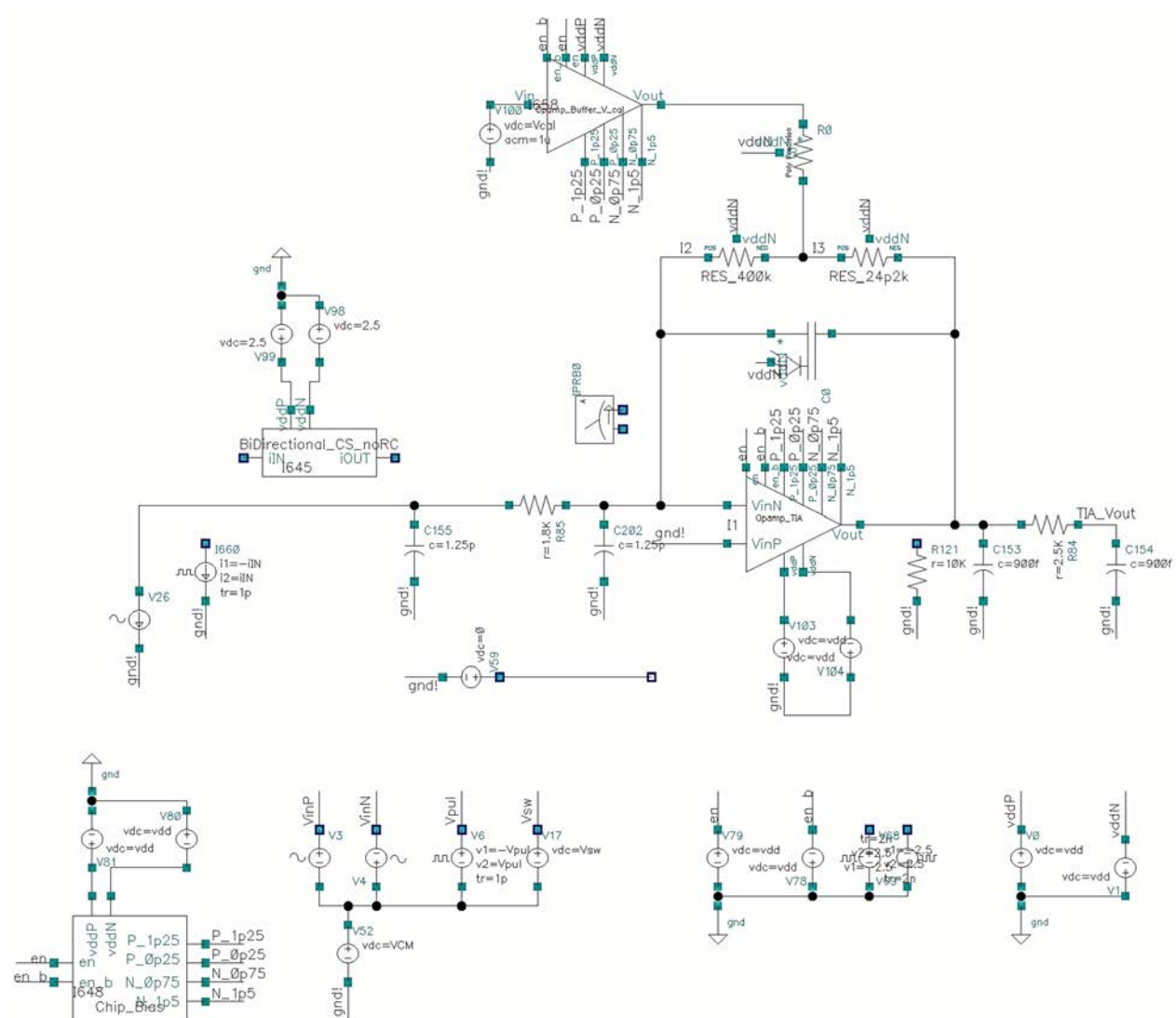


Figure 31. Operational amplifier and TIA testbench in Cadence schematic view.



basic functionality of TIA with an addition of the worst-case approximation of RC parasitic load due to lengthy routings. All results presented are generated from simulations with extracted layout TIA. The simulations were performed in a testbench that accommodates realistic modeling to thoroughly test the circuits for any possible flaws. Figure 31 illustrates the actual testbench in Cadence schematic view for the opamp and TIA. It includes the actual bias circuit, opamp buffers, non-ideal current source for testing the TIA, ideal voltage and current sources, and all other passive components.

### 5.1. Operational Amplifier

As the basic building block, an opamp must be designed with high linearity (high open-loop gain) across a wide range of input and output voltage; however, with minimal input offset, total gate area, and noise to accommodate picoamperes I-V conversion of the TIA. This section presents all basic simulations results showing the opamp performance being used in the TIA.

Figure 32 illustrates the opamp test setup and Table 3 summarizes three simulation condition, representing a typical and two extreme PVT variation conditions. The RC values attached to the opamp output are based on the approximation of maximum parasitic RC string extracted from the interconnect metal between the opamp to the next available voltage buffer in Stage II. The 10k $\Omega$  parallel resistor is to model the resistance value that the output of the opamp would directly drive, the R3 of the T-junction in the TIA. The 10k $\Omega$  load models the worst-case scenario.

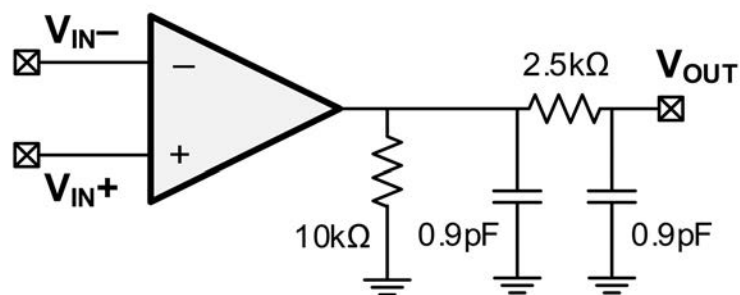


Figure 32. Operational amplifier simulation: Setup.



Table 3. Operational amplifier simulation: Conditions.

Corner	VDD(V)	Temp(°C)	Process
<b>SLOW</b>	±1.5	75	SS
<b>TYP</b>	±2.0	37	TT
<b>FAST</b>	±2.5	0	FF

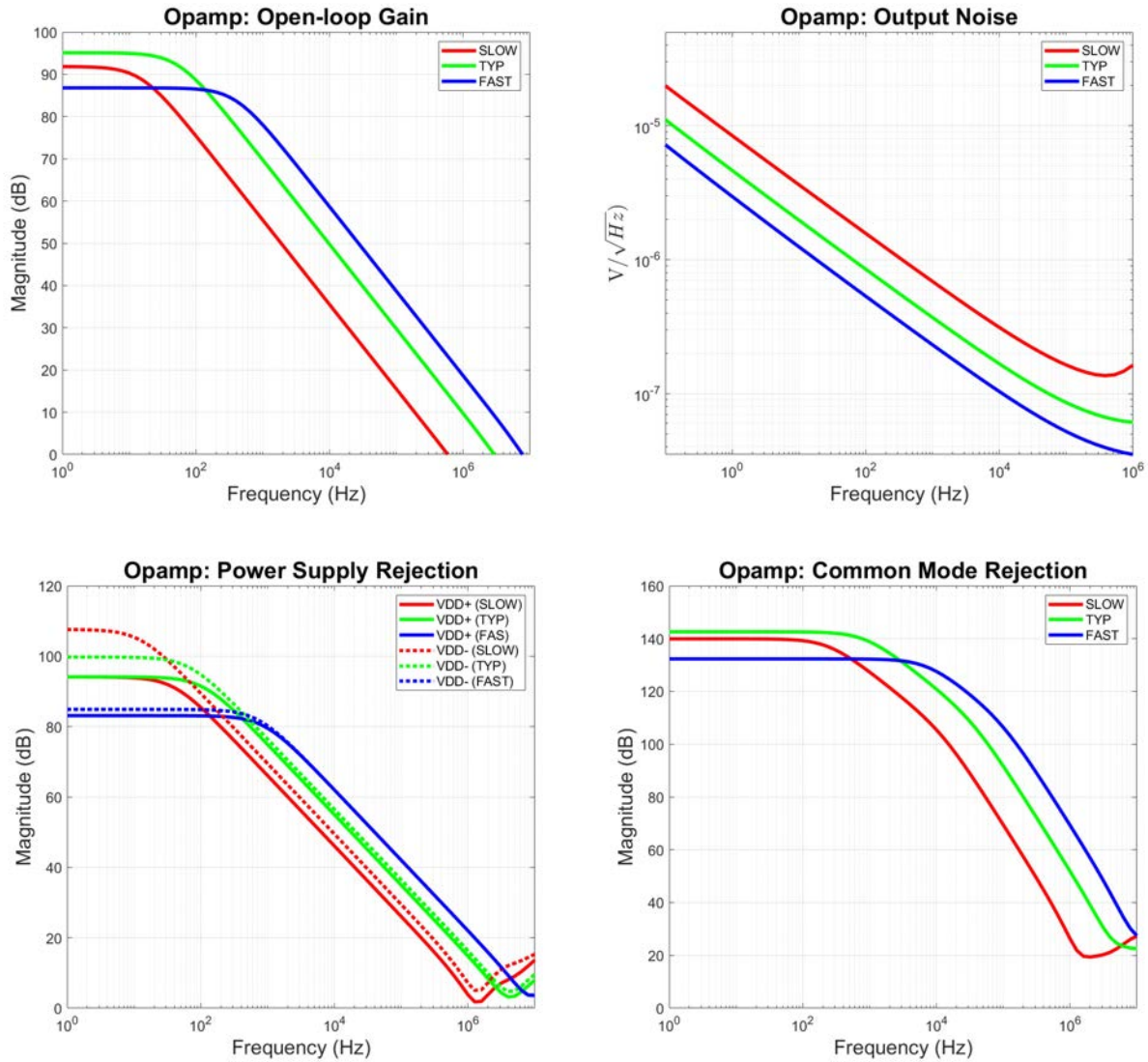


Figure 33. Operational amplifier simulation: AC responses.

As plotted in Figure 33, the opamp open loop-gain and GBWP minimum value were set to be around 90dB and 1Mhz, respectively. The output noise corresponds to under 1 $\mu$ V/ $\sqrt{Hz}$  at 1kHz, the



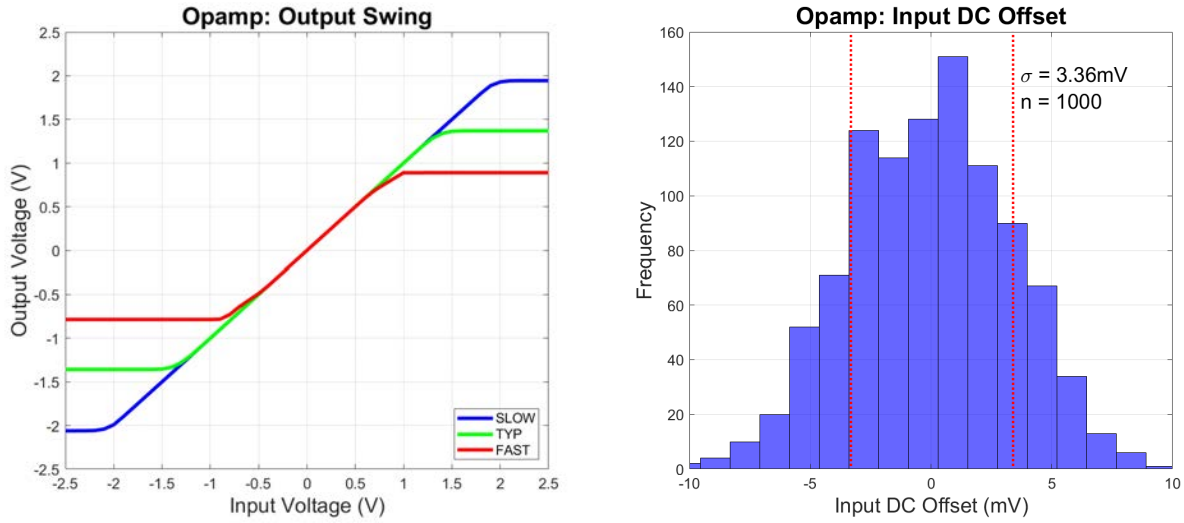


Figure 34. Operational amplifier simulation: DC responses.

expected maximum bandwidth of the TIA. The power supply rejection figure shows the capability of an opamp to attenuate the power supply rails noise reflected to the output, which is well above 80dB for all corner cases. Common mode rejection indicates the differential inputs performance of the opamp of suppressing a common signal shows up at the inputs without affecting the output, measured at 130db and above.

As plotted in Figure 34. The DC simulation results indicate approximately symmetrical voltage drop of 0.6V to positive and negative rails at the worst-case scenario with a high current load of 10kΩ. A Monte-Carlo analysis with 1000 iteration was performed to measure the DC input offset of the opamp. The  $\sigma$  value indicates 3.36mV, an acceptable range for electrochemical sensing purpose, where voltage shifts in the range of tens of millivolts are common. The DC simulations were performed with the opamp set to a unity gain configuration.

To ensure overall performance and stability of the opamp, a transient simulation was performed in a unity gain configuration over an input step function with 1ps rise time. The slew rate shows how the opamp response to the step function without introducing instability at all corner cases. However, the Slow corner indicates a noticeable performance reduction, such as the bandwidth and stability, in comparison to the other two cases. This indicates that high temperature in combination with low power supply voltage may introduce some inaccuracies.



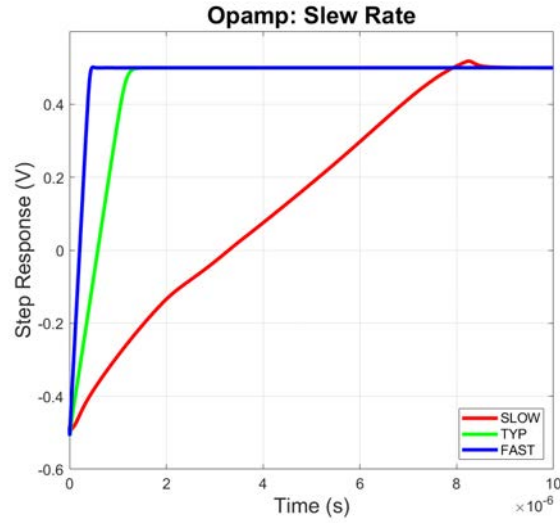


Figure 35. Operational amplifier simulation: transient response.

**Error! Reference source not found.** summarizes the opamp performance for all three cases of corners. This opamp topology was designed to accommodate the block level TIA, voltage buffers at all stages, and the potentiostat, however, with a slight adjustment of the input and output stages to accommodate various capacitive and resistive load.

Table 4. Operational amplifier: Specifications summary.

Parameter	Condition	SLOW	TYP	FAST	Units
Offset Voltage	G = 1		3.36		mV- $\sigma$
Open-Loop Gain		91.84	95.09	86.8	dB
<b>Frequency Response</b>					
Phase Margin		70.56	57.78	46.15	$^{\circ}$
Bandwidth (-3dB)		15.22	53.91	390.5	Hz
Unity Gain Bandwidth		0.592	2.932	7.748	MHz
PSRR+	DC	94.11	94.12	83.11	dB
PSRR-	DC	107.6	99.75	84.88	dB
CMRR	DC	139.9	142.6	132.3	dB
<b>Noise</b>					
Output Noise	100Hz	1.579	0.854	0.537	$\mu\text{V}/\sqrt{\text{Hz}}$



	1kHz	0.692	0.372	0.233	$\mu\text{V}/\sqrt{\text{Hz}}$
Noise Density	0.1Hz – 10Hz	54.40	29.76	18.95	$\mu\text{V}_{\text{RMS}}$
<b>Transient Response</b>					
Slew Rate	$V_{\text{in}}=1\text{V}_{\text{pp}}$	0.123	0.847	2.382	$\text{V}/\mu\text{s}$
Total Harmonic Distortion	$V_{\text{in}}=0.1\text{V}_{\text{pp}}$ 1kHz	1.39	0.17	0.39	%
<b>Output</b>					
Output Swing +	Range to $V_{\text{DD+}}$	0.611	0.631	0.556	V
Output Swing -	Range to $V_{\text{DD-}}$	0.715	0.642	0.440	V
Short Circuit Current		0.150	0.286	0.565	mA
<b>Power Supply</b>					
Quiescent Current		4.416	33.06	116.9	$\mu\text{A}$

## 5.2. Trans-impedance Amplifier

Simulations of the TIA encompass AC, DC, transient, and Monte-Carlo to measure possible gain variations. The simulation setup depicted in Figure 36, uses the same RC output as the opamp simulation counterpart, except that the TIA placed the actual T-junction feedback resistors and capacitor. The selection values of  $R_1$  to  $R_3$  are previously discussed in Chapter IV. The parasitic approximations value was derived from extracted layout of the longest metal path between a WE and a read channel located at the most corner of each quadrant to accurately simulate input parasitic capacitance and resistance due to a long and thin metal routing. Any shorter path results in less parasitic resistance and capacitance, creating higher stability and noise performance [40]. The input current is generated by a non-ideal current mirror, which its input is being driven by a known ideal current source as depicted previously in Figure 31.  $V_{\text{CM}}$  and  $V_{\text{TJ}}$  were set to 0V using an ideal voltage source and a voltage buffer, respectively.

The opamp simulations indicate a well-performing amplifier across different extreme corners. Assuming a well-controlled power supply voltage and temperature, the only source of inconsistency from using the exact same opamp would be due to the N-WELL ( $R_1$ ) resistor and the statistical variances of the



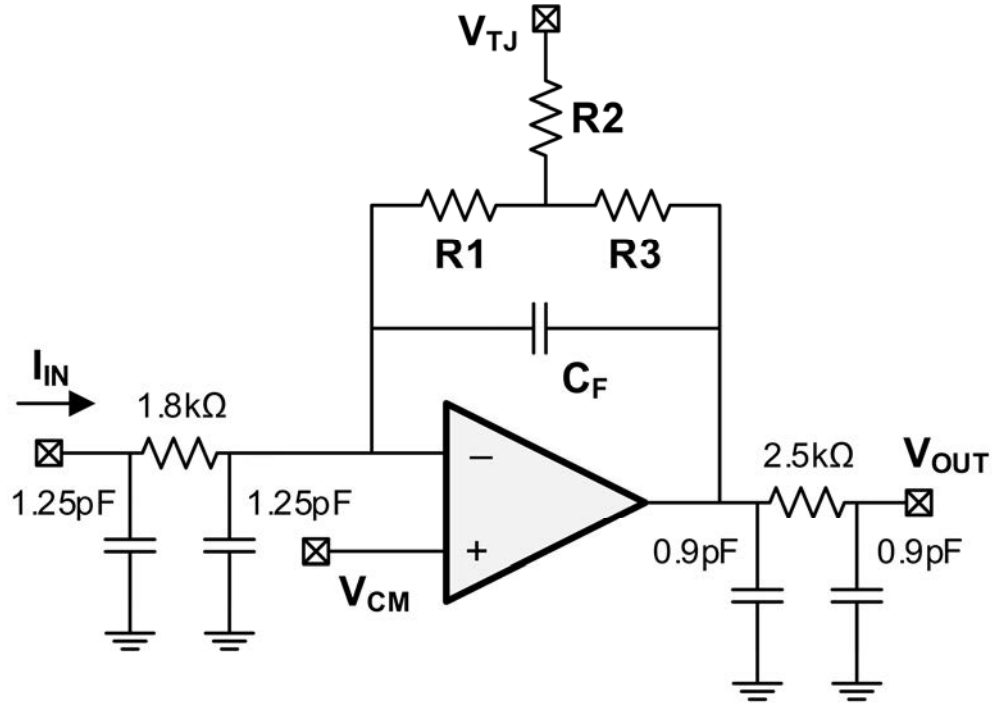


Figure 36. Transimpedance amplifier simulation: Setup

Table 5. Transimpedance amplifier simulation: Conditions.

Corner	VDD (V)	Temp (°C)	Process
<b>SLOW</b>	±2.0	37	SS
<b>TYP</b>	±2.0	37	TT
<b>FAST</b>	±2.0	37	FF

poly resistor in  $R_2$  and  $R_3$  due to the process variations. Table 5 summarizes the corner cases for the following simulations.

Figure 37 summarizes the AC performance of TIA. The bandwidth is constant around 6kHz for all cases, while the gain varies across different corner cases. The model confirms that resistance of  $R_1$  varies due to the doping concentration incorporated in the process variation, resulting in gain variation of up to 6MΩ around the typical 20MΩ. This process variation is usually in the form of a gradient across the silicon wafer during the fabrication, without any abrupt differences. Therefore, the actual variation of the read channel gain across the read channel array within one microchip would be smaller. The gain variation within



the same corner was simulated at  $0.11\text{M}\Omega\text{-}\sigma$ . Any remained gain various would be taken care of in the data post-processing phase. The noise density is simulated to be under  $100\text{pA}$  across all corner, which is a respectable value considering a tight power supply and area specifications. The power supply rejection shows a minimum rejection of  $20\text{dB}$ , which is not a desirable value. The low power supply rejection value in TIA simulation indicates strong coupling to the power supply. The N-WELL substrate bias is connected to the negative rail (VDD-), representing a direct injection of noise from R1 to the output. Meanwhile, the positive VDD+ power supply rejection is at the range of acceptable value. Hence, a low noise power supply, especially for the negative rail (VDD-), is crucial to ensure minimal additional noise from outside sources.

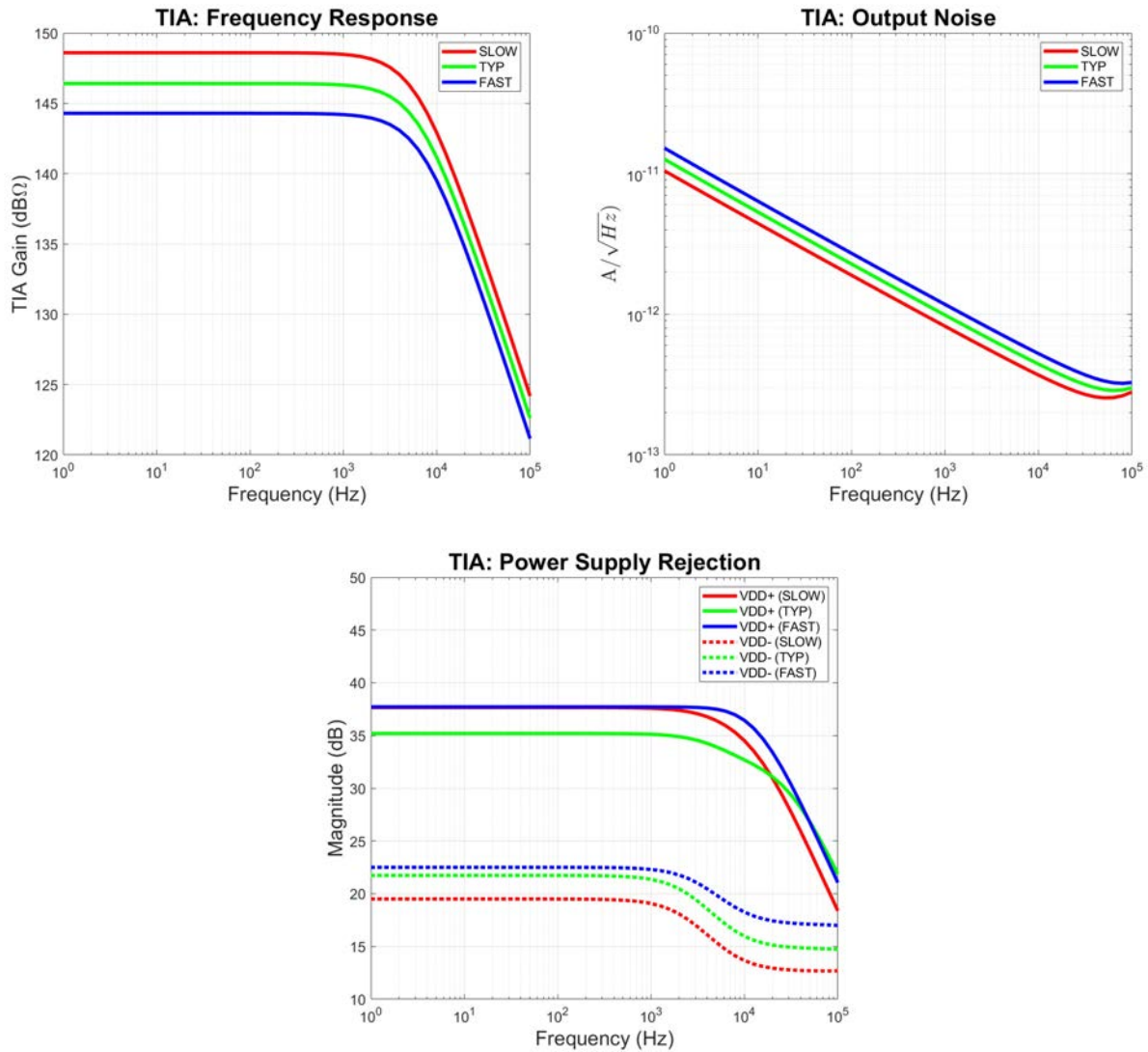


Figure 37. Transimpedance amplifier simulation: AC responses.



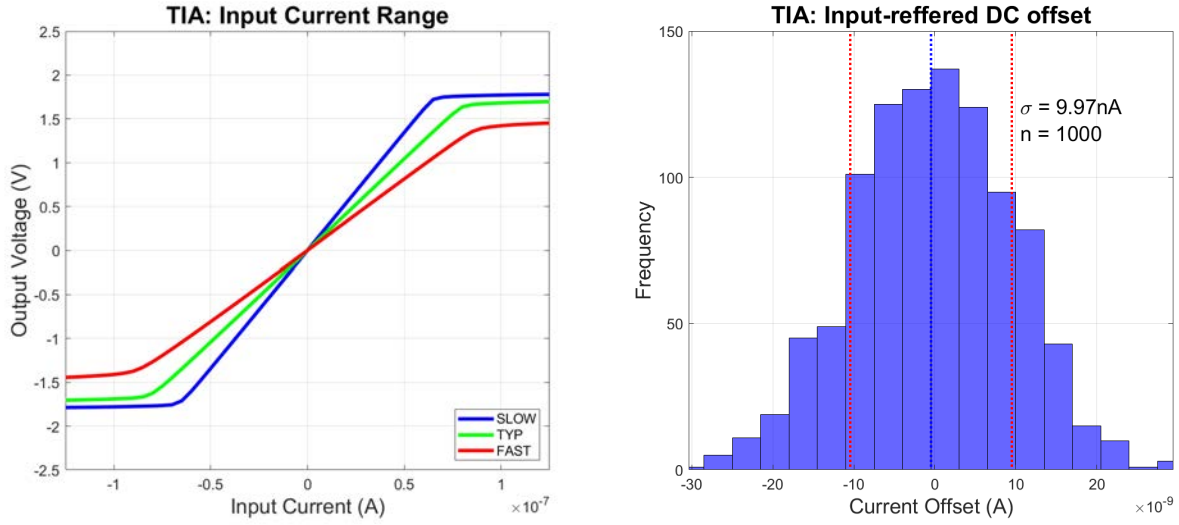


Figure 38. Transimpedance amplifier simulation: DC responses.

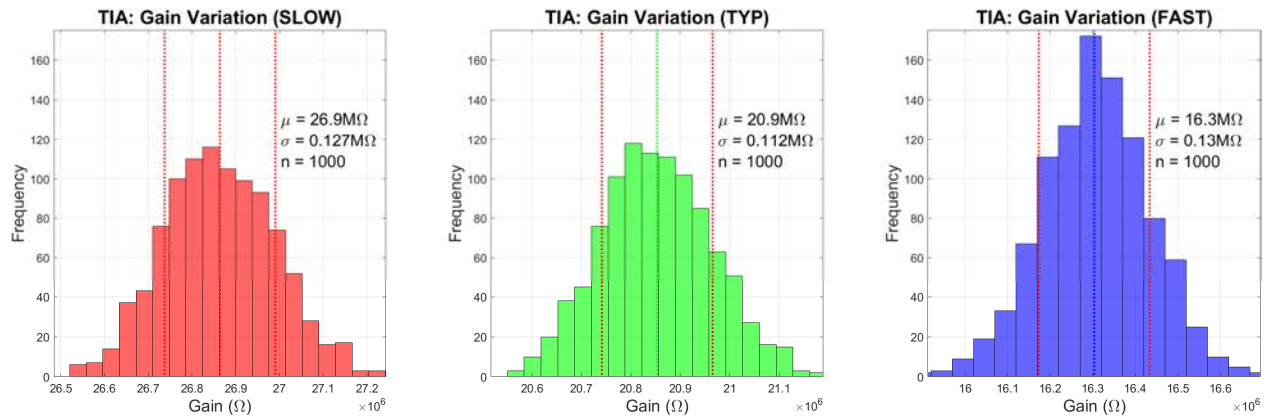


Figure 39. Transimpedance amplifier simulation: Gain variations.

The DC simulations were performed with swept current input to show the dynamic range of the TIA. It shows 0.3V voltage drop to both rails for slow and typical, and 0.5V for the fast. The slope in Figure 38 and the histogram distribution of Figure 39 show the gain variation due to the process corner. In a realistic condition, the reach channel array in a microchip would have a DC offset of  $\sigma = 10\text{nA}$ , gain variation of  $\sigma = 0.13\text{M}\Omega$ , and voltage output compliance of approximately 0.3V from the power supply rail. The DC offset variation can be captured for each read channel, stored, and used to calibrate this constant DC offset error, while the gain variation must be captured by performing a standard electrochemical procedure with at least two-point calibrations.



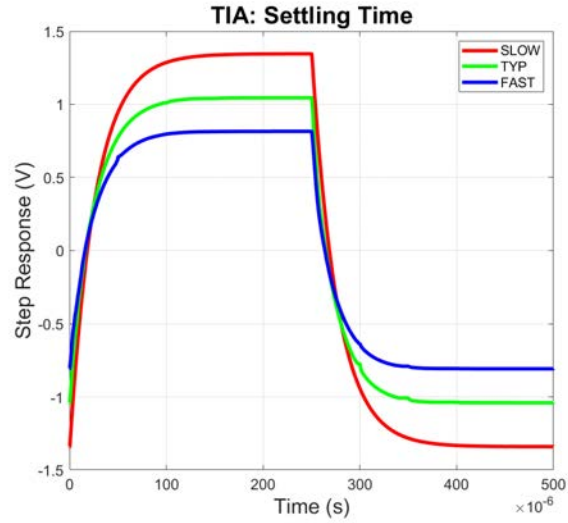


Figure 40. Transimpedance amplifier simulation: Transient response.

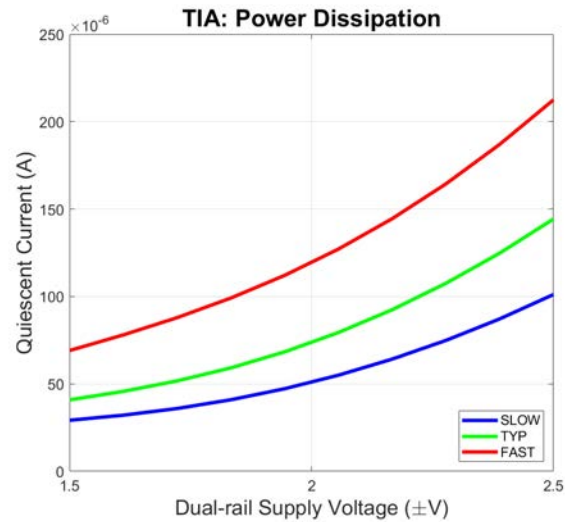


Figure 41. Transimpedance amplifier simulation: Power dissipation.

A transition response was captured for each corner case to show the TIA output response to a square-wave input. As shown in Figure 40, The transition captures the kilohertz bandwidth with no stability issue, in the form of a critically damped signal. The gain variation is also consistent with previous AC and DC simulations.

Power dissipation plays an important role in each single read channel. Figure 41 shows an exponential relationship between the power supply voltage and the power dissipation with Fast process as the lowest-performing corner. Based on the simulation, the practical range of power supply voltage should



be kept between  $\pm 1.5\text{V}$  to  $\pm 2.0\text{V}$  to optimize the performance while keeping the heat dissipation low due to quiescent current draw on the read channels. Table 6 summarizes the TIA performance for all three cases of corners.

Table 6. Transimpedance amplifier: Specifications summary.

Parameter	Condition	SLOW	TYP	FAST	Units
Transimpedance Gain		26.9	20.9	16.3	$\text{M}\Omega$
Gain Variation		0.127	0.112	0.13	$\text{M}\Omega\text{-}\sigma$
Input-referred Offset	$I_{in} = 0\text{A}$		9.97		$\text{nA}\text{-}\sigma$
Current Input Max		66.63	82.42	91.46	$\text{nA}$
Current Input Min		-66.8	-82.14	-90.9	$\text{nA}$
<b>Transient Response</b>					
Settling Time	$I_{in} = 50\text{nA}$	102.7	94.26	89.96	$\mu\text{s}$
<b>Frequency Response</b>					
Bandwidth	-3dB	6.36	6.24	6.16	$\text{kHz}$
PSRR+	DC	37.6	35.2	37.7	$\text{dB}$
PSRR-	DC	19.5	21.7	22.5	$\text{dB}$
<b>Noise</b>					
Input-referred Noise	1Hz	10.49	12.68	15.21	$\text{pA}/\sqrt{\text{Hz}}$
	100Hz	1.896	2.28	2.717	$\text{pA}/\sqrt{\text{Hz}}$
	10kHz	0.370	0.442	0.525	$\text{pA}/\sqrt{\text{Hz}}$
Noise Density	1Hz – 10kHz	66.1	79.4	94.54	$\text{pA}_{\text{RMS}}$
<b>Power Supply</b>					
Quiescent Current	$V_{DD} = \pm 2.0\text{V}$	51.6	72.8	124.5	$\mu\text{A}$

### 5.3. Top Level Simulation

At the top-level, entire microchip level simulations were performed with all circuit block, MEA, and read channels array routed and extracted for parasitic RC. The extracted view was placed into a component



block and each input is properly biased and driven by a voltage or current sources, while the outputs were given realistic resistive or capacitive loads. The full test bench setup in Cadence is shown in Figure 42.

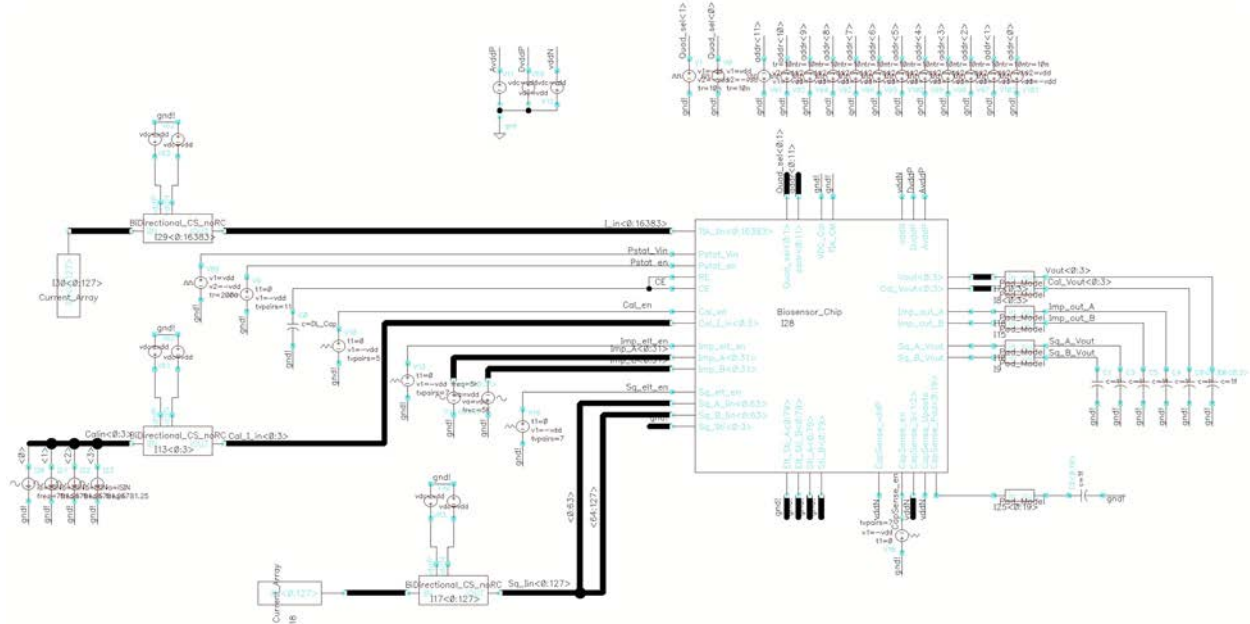


Figure 43. CMOS Microchip simulation: Setup.

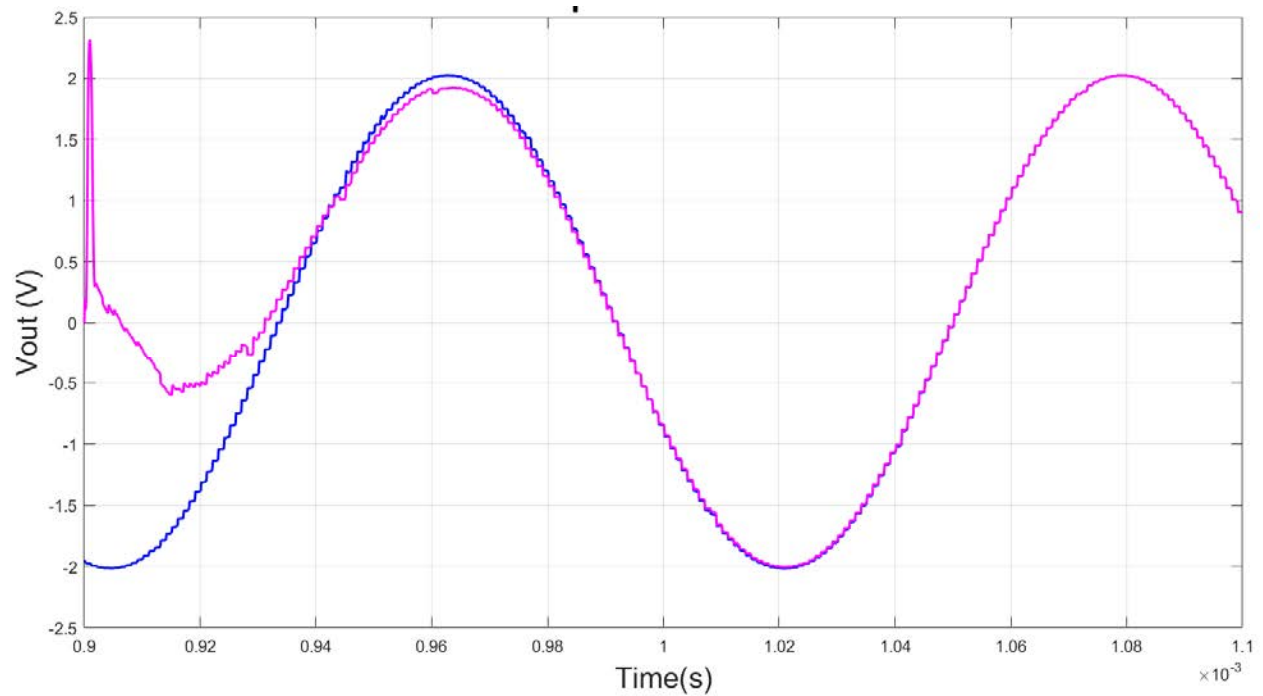


Figure 42. CMOS Microchip simulation: Transition between read channels



The simulation setup connects each of the 16 thousand read channel input to 16 thousand non-ideal current sources. The point of contact of the current source is at the WE surface, indicated by the pin layers in the extracted layout, therefore, the simulation process resembles an actual current flow generated by an electrochemical reaction. In this specific simulation, every WE are connected to a sine-wave current input with equal phase. An external clock generator drives the on-chip decoder at 1MHz to sample the voltage output from each read channel and reflect it to the quadrant buffer at 1 $\mu$ s window. The expected

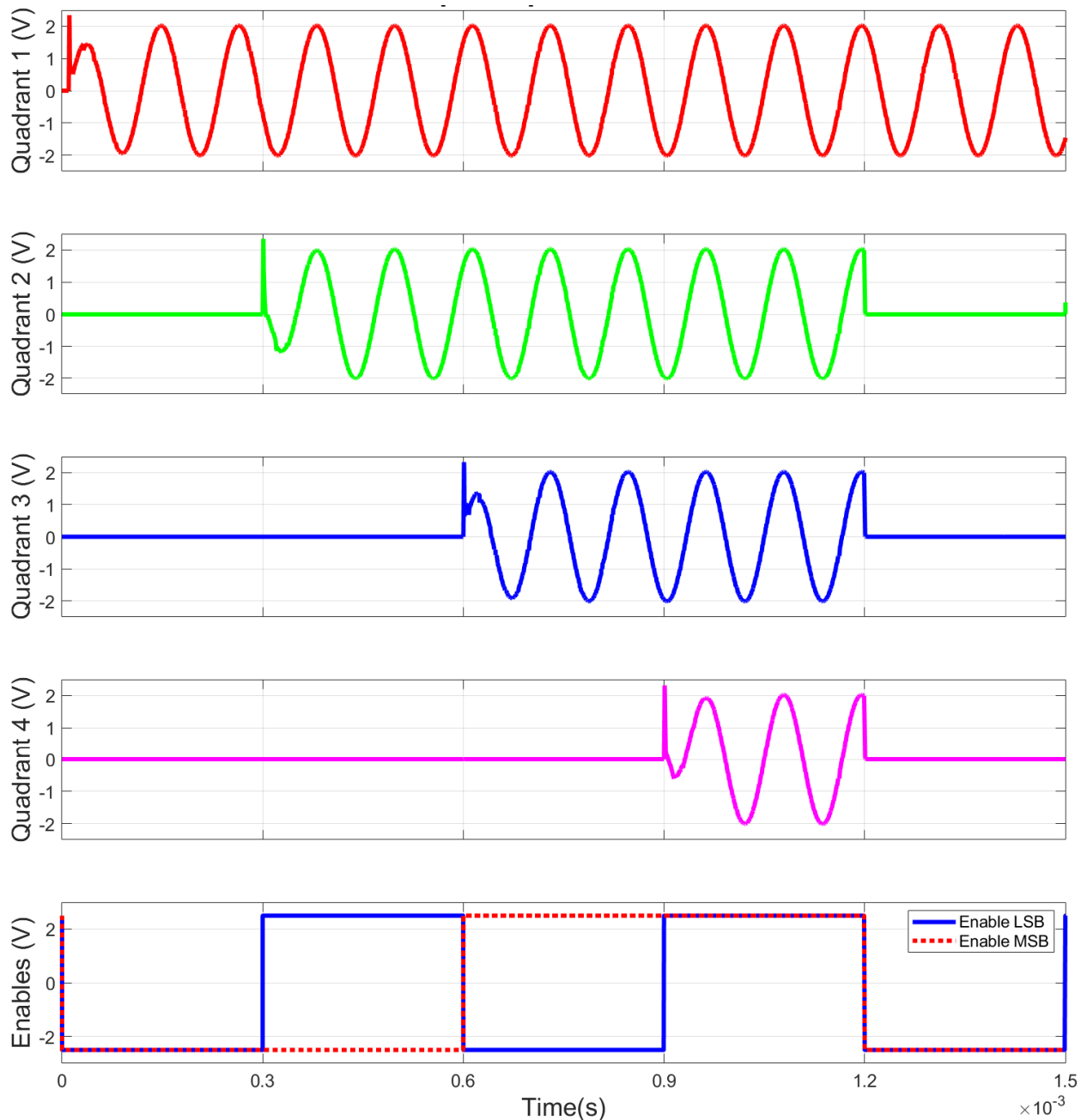


Figure 44. CMOS Microchip simulation: output voltage response to quadrant enables.



output should also resemble the sinewave current, however, with sampling points every  $1\mu\text{s}$  in a form of monotonic steps. As shown in Figure 42 the blue curve represents a quadrant that initially turned on and continuously sampling the input, while the magenta curve represents a quadrant that is recently turned-on, at  $t=0.9\text{ms}$ . This result shows how the CMOS microchip would response to quadrant enables during the start-up period, which is expected to be stable after  $1\text{ms}$ . The response of each voltage output in a quadrant to enables is shown in better details in Figure 44. The simulation setup increasingly turns-on one quadrant every  $3\text{ms}$  and back to one active quadrant at  $t = 1.2\text{ms}$ , with quadrant one being turned-on continuously. The response captures both starting-up and turning-off transitions for each quadrant. This top-level simulation confirmed basic functionalities of the read channel and the multiplexing actions in each quadrant on a parasitic extracted model. Therefore, the fabricated CMOS microchip is expected to resemble this performance, however, with the expected DC offset simulated in the previous section.

In a comparison, Figure 45 presents the fabricated CMOS microchip quadrant output (yellow curve) in a test setup with a sine wave current (teal curve), resembling the simulation output from Figure 44. As expected, the output also contains the effect of the DC offset. The calibration process will be discussed in the Software Integration section.

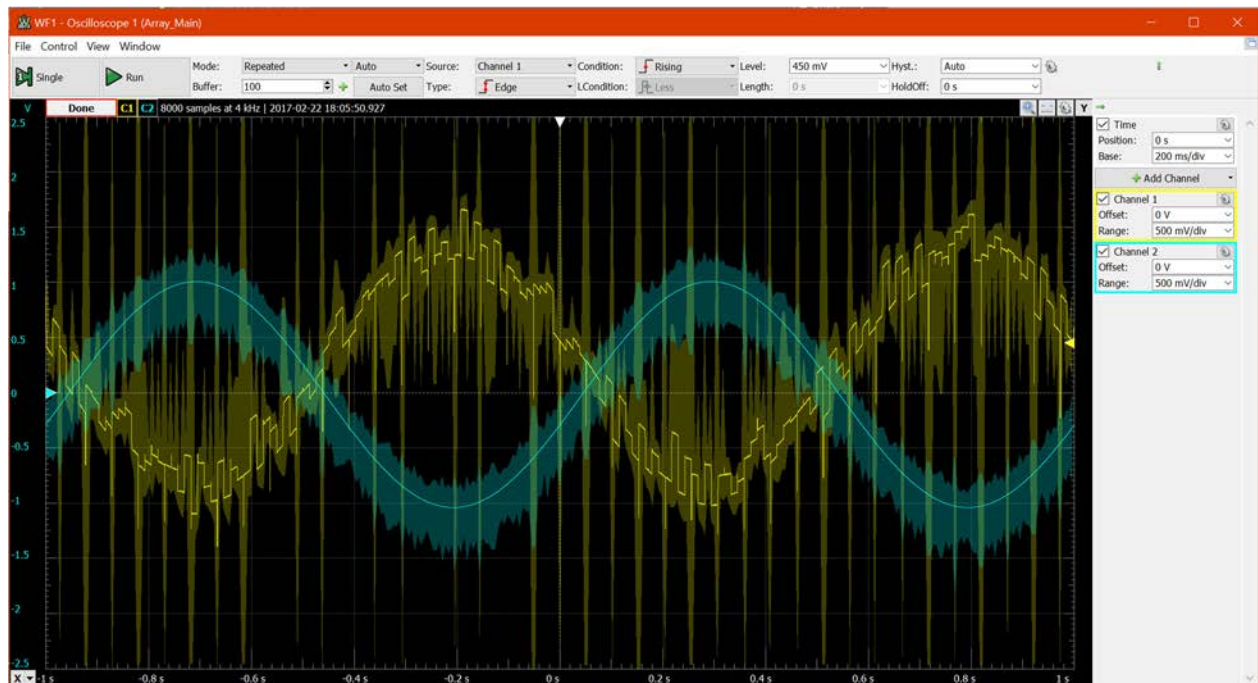


Figure 45. CMOS Microchip taped-out: output voltage response.



## CHAPTER VI: SYSTEM INTEGRATION: HARDWARE, SOFTWARE, AND MICROFLUIDIC

In the final design (Figure 46), the electrochemical biosensor system is comprised of the custom CMOS MEA microchip, three custom Printed Circuit Boards (PCBs), a Power Supply Unit (PSU), a commercial Data Acquisition (DAQ), and a host computer running a custom-built software for control and displaying results. A custom Control PCB (C-PCB) consists of a Microcontroller Unit (MCU), analog and digital signal generators, voltage monitoring circuits, and a USB interface for communication with the host computer. The Sensor PCB (S-PCB) holds the CMOS MEA microchip socket and voltage linear regulators

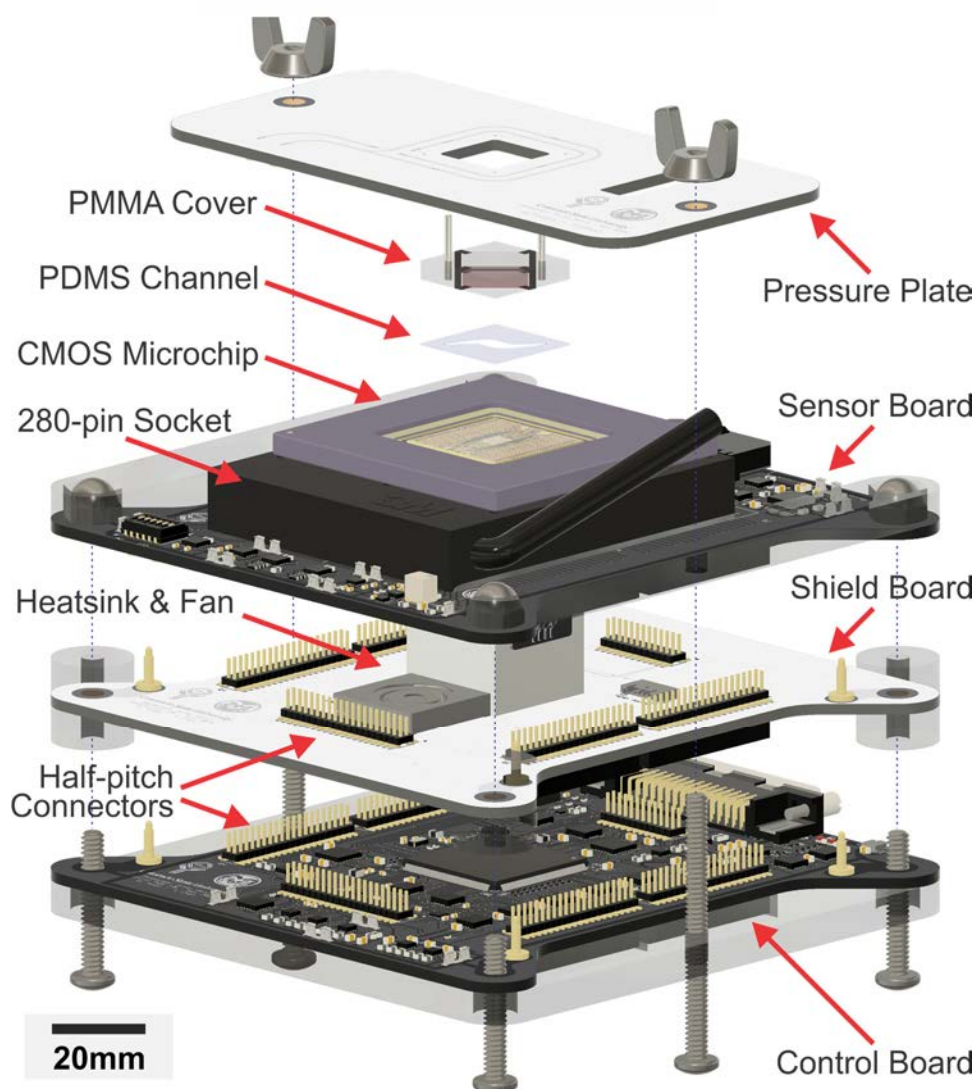


Figure 46. Biosensor system 3D model assembly.

*The biosensor system with the CMOS microchip, support PCB, and microfluidic system arrangement.*



to supply controllable low-noise power to the CMOS MEA microchip and the C-PCB. A Shield PCB (Sh-PCB) with a custom heatsink provides excess thermal dissipation for the CMOS MEA microchip and protects analog signals in S-PCB from electromagnetic interference (EMI) from digital control signals in C-PCB. A DAQ board, ADLINK DAQ-2208 (ADLINK, New Taipei, Taiwan), converts analog voltage readings from the CMOS MEA microchip to 12-bit digital signals to be processed and stored in the host computer. The control and display software and its GUI on the host computer is written in MATLAB. The GUI reports processed biosensing results, shows the information obtained from the C-PCB, and lets users perform real-time system control. All power needs are supplied by a custom dual-rail PSU, which converts standard 110V 60Hz power line input to multiple dual-rail DC potentials. The board-level development comprises of four major iterations as shown in Figure 13 timeline. The following sections in this chapter discuss the improvements of each iteration.

### 6.1. Circuit Boards: Prototypes

The biosensor first prototype was in a form of breakout board, directly connecting all 280 pins for a proper individual connection. A breadboard was attached for supporting custom test of various functions

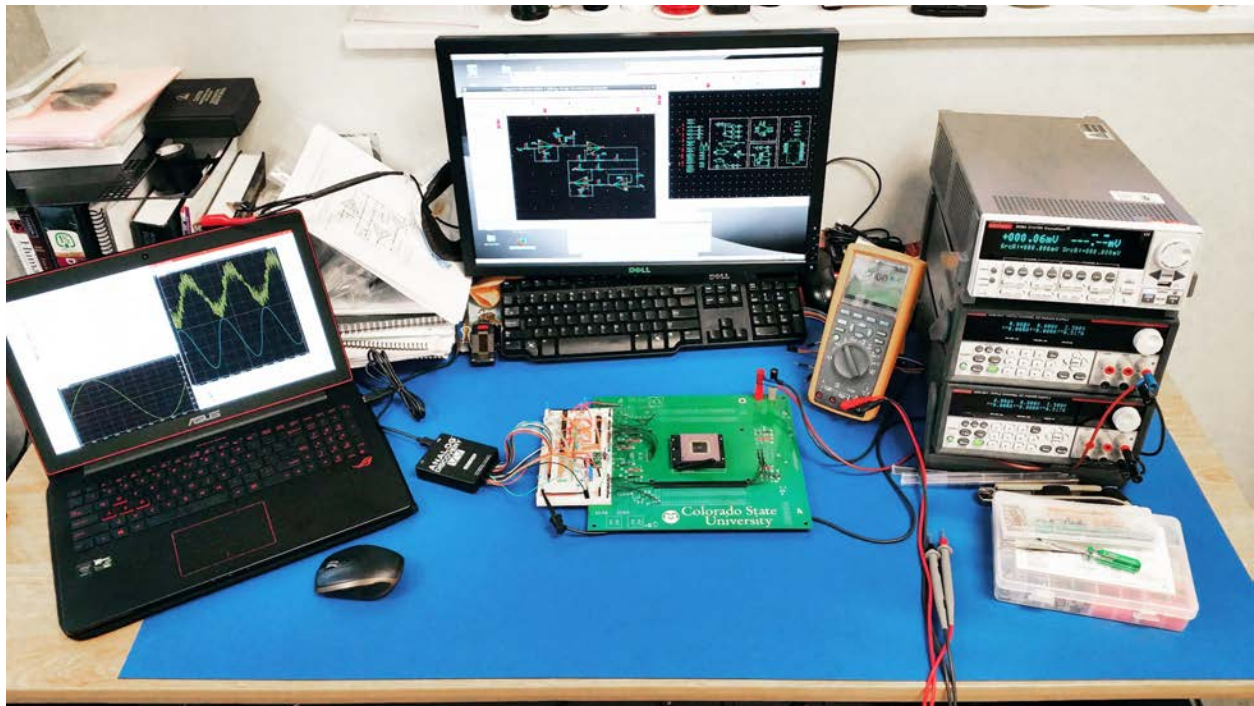


Figure 47. Prototype board during the first Generation 3 CMOS microchip test.



using DIP package components. Analog Discovery was utilized for the convenient of simultaneous generating custom analog, digital, and trigger signals from various type of initial tests. External top-bench power supply supplied the required voltage as shown in Figure 47. This rather simple setup was used for the fundamental tests, such as test on-chip potentiostat for basic opamp function, read channel for basic I-V conversion, and the general power dissipation measurements. Any test for precision measurement (e.g. noise and linearity of I-V conversion) were performed with high-performance benchtop instruments.

The next PCB iteration was a 50mm×100mm PCB attachment to the prototype board. As shown in Figure 48, there are two breadboards attached to the prototype breakout board. One for holding the PCB version 1 and a disassembled and attached Analog Discovery, minimizing the risk of electrical interruptions during tests. The other breadboard was used to attach a breakout converter for the 68-pin connection for the commercial DAQ board (ADLINK-DAQ2208). This setup also adopted the first temperature sensing to automatically control the CMOS microchip surface temperature. The temperature sensor is a square PCB with an opening attached on top of the CMOS microchip. Most support and control circuits were integrated

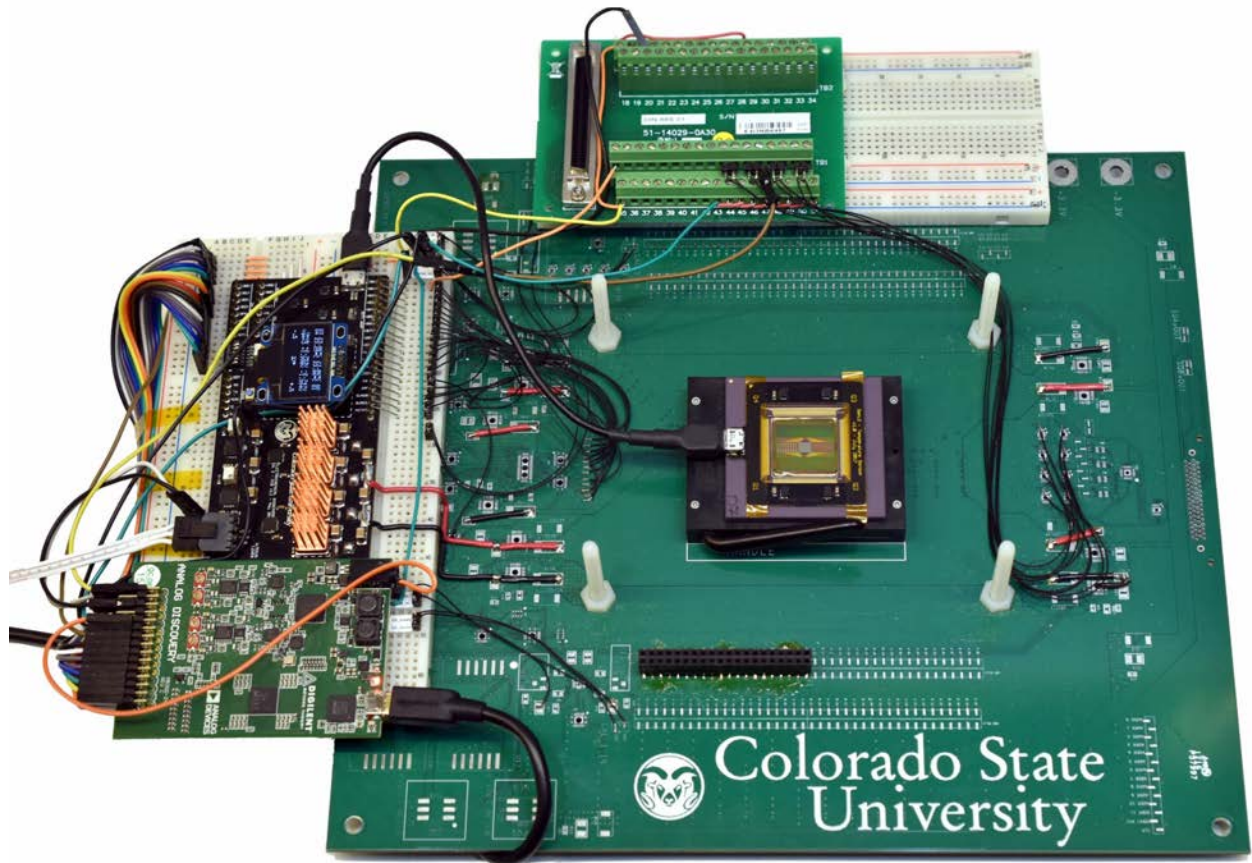


Figure 48. Prototype breakout board and PCB version 1.



into the version 1 board, such as the low noise voltage regulator and an MCU (ATmega328P) for automatic control of the temperature feedback loop. Lastly, this setup also allowed the first high-speed analog readout using the external DAQ boards for testing the multiplexing and sampling action of the CMOS microchip.

The function integrations became increasingly complex towards the end of PCB version 1. The PCB version 2, as shown in Figure 49, comprises of two separates boards, Sensor and Control boards. The major modifications were made to compartmentalize analog and digital to two different boards,

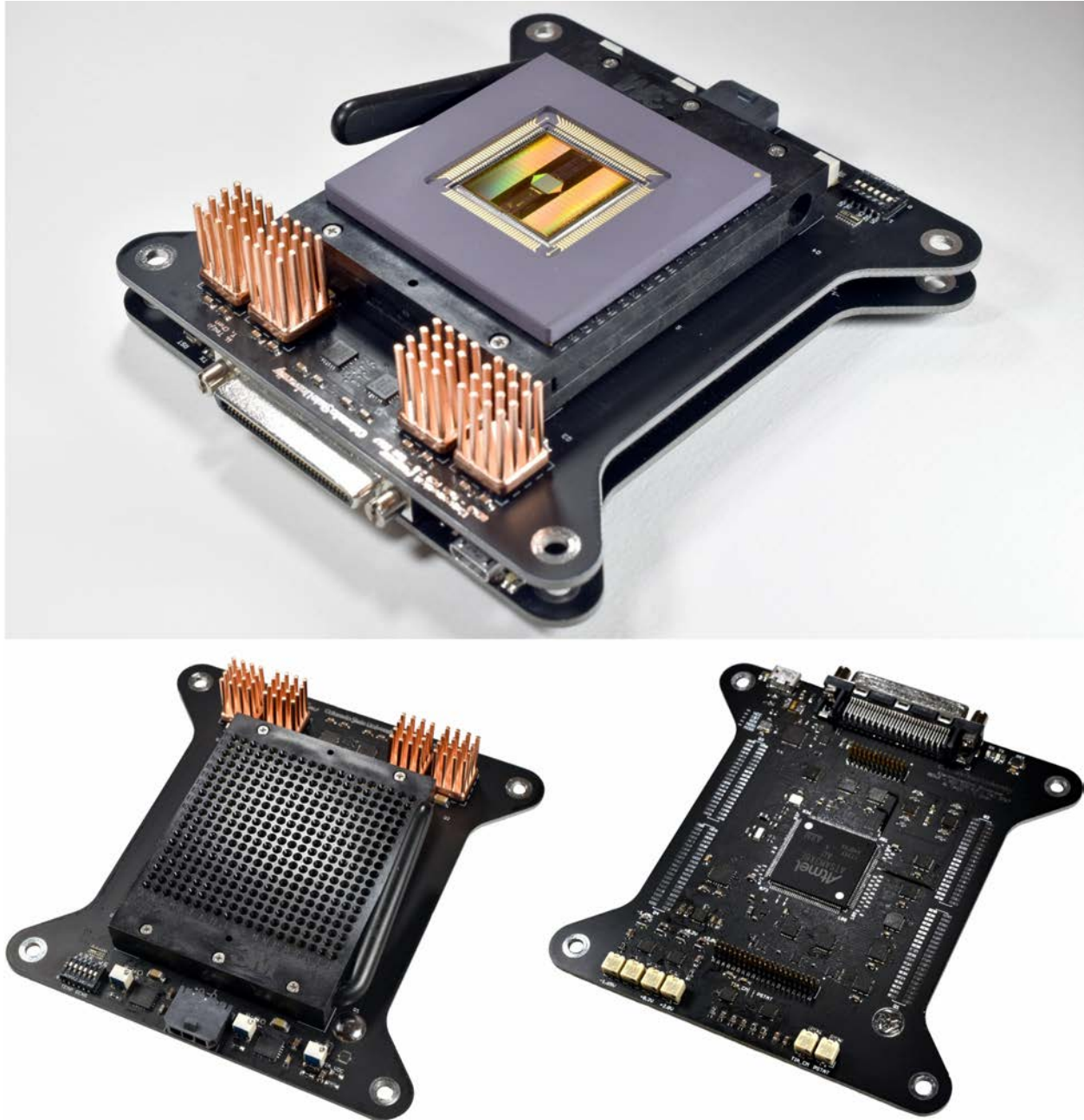


Figure 49. PCB version 2: Sensor and Control boards.



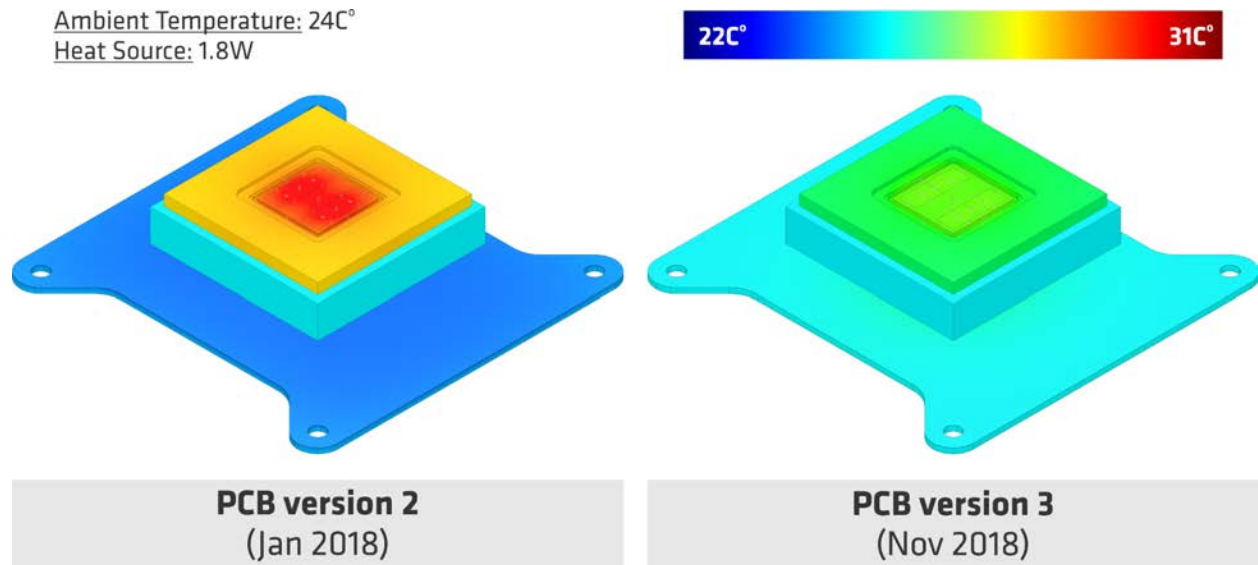


Figure 50. PCB version 3: 3D thermal simulation comparison.

separating and minimizing cross-talks between the two. The Sensor board holds the extremely low noise LDO for both positive and negative rail to drive the CMOS microchip; another set of dual-rail voltage regulator for supplying other peripheral circuit in both Sensor and Control boards; voltages buffers and filters to accommodate low-noise voltage bias requirements for CMOS microchip; and the 280-pin sockets for convenient access swapping the CMOS microchip. The Control board holds a high-performance 32-bit MCU (ATSAM3X8E) for generating controllable clock for driving the 12-bit address; various voltage sensing circuits for reading the real-time value of CMOS microchip power supply voltage and current; and the circuit to generate potentiostat signals. All monitor and control signals are continuously transmitted and received from a control PC through a standard USB protocol. The analog read-out scanning was improved to allow up to four FPS scanning rate. However, thermal dissipation of the CMOS microchips and the supporting LDO dominates and stays within the PCB stack arrangement, causing continuously increasing temperature of the entire system. Details of the PCB schematic and layout of each version is included in Appendix B.

## 6.2. Circuit Boards: Final Design

The board-level final design improvements include a major improvement in active thermal control, which were addition of the Shield board with custom heatsink and a mini blower; and redesigned of the custom power supply to incorporate higher efficiency DC/DC voltage converters and to move most of the



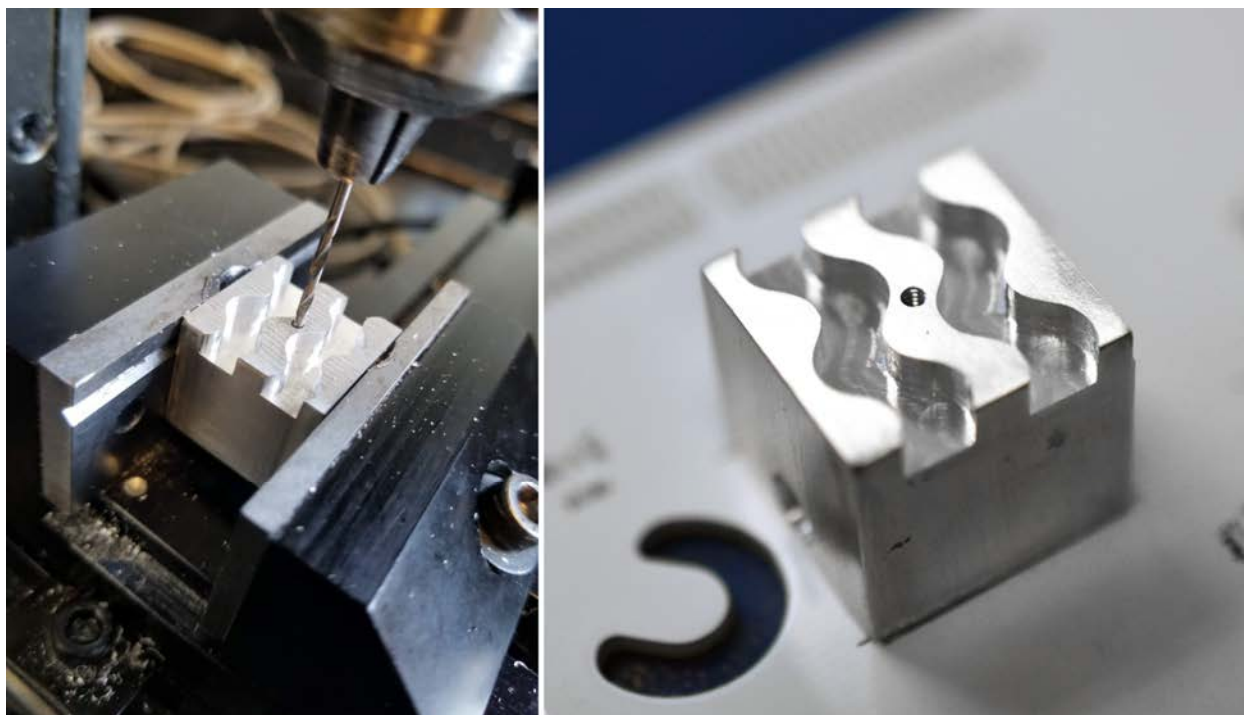


Figure 51. PCB version 3: Custom heatsink fabrication.

thermal dissipation from the low noise voltage regulators in the Sensor board. Figure 50 shows the three-dimensional (3D) thermal simulations of the CMOS microchip due to the improvement mentioned above. PCB version 2 arrangement shows a significant thermal accumulated in the silicon die area, while PCB version 3 distributes the concentrated heat towards the Shields board, eliminating excessive thermal accumulation in the CMOS microchip. To achieve this thermal performance, a custom 20mm×20mm×18mm milled heatsink was manufactured to allow an airflow generated by a mini blower (). The shield board arrangement, as shown in Figure 51, illustrates how the wavy part of the heatsink is attached to the Shield board and a blower fan creates an active airflow through it. Meanwhile, the flat part of the heatsink is exposed and protruded through the Sensor board and the 280-Socket, providing a direct contact to the heat concentrated spot in the CMOS microchip (the bottom side of the ceramic package). To accommodate the heatsink overhang, the Sensor board was modified, and the 280-pin socket center area was manually milled. The fabricated Shield board and the final PCB assembly is portrayed in Figure 52A.

Accurate temperature control in biosensing is crucial to maintain the sample physiological functions. The Sh-PCB is equipped with a custom milled heatsink and a 17mm mini blower fan as shown in Figure 52A. The heatsink creates a direct thermal contact (layered with thermally conductive pad/paste)



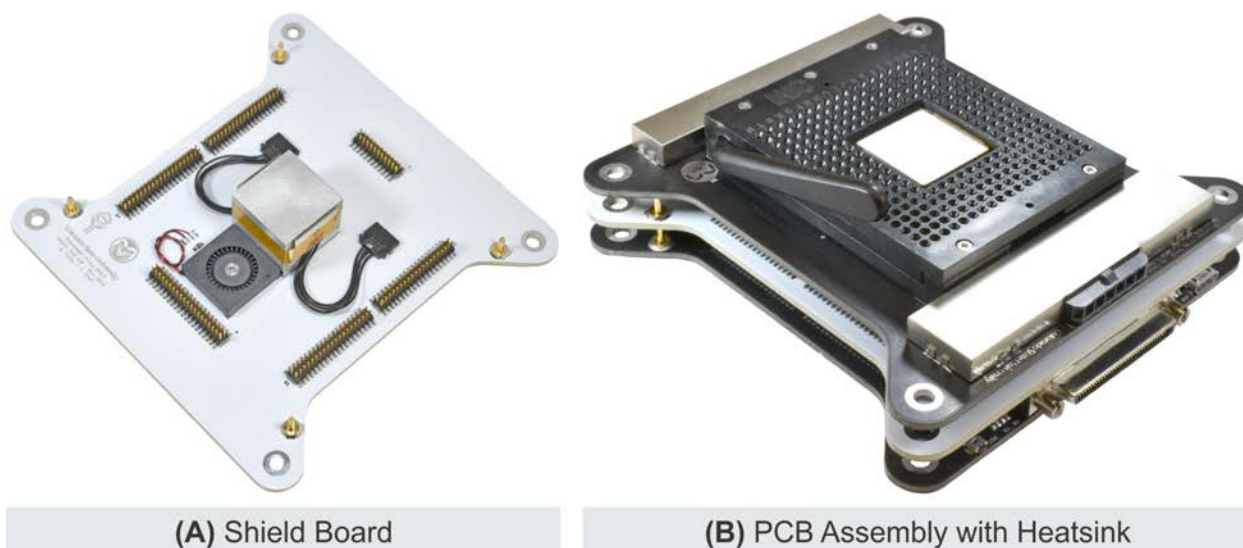


Figure 52. PCB version 3: Shield board arrangement.

with the bottom side of the CMOS MEA microchip ceramic package. Sh-PCB holds I<sup>2</sup>C serial connections for two precision temperature sensors located on the heatsink. The temperature reading can be used to establish feedback control to the CMOS MEA microchip supply voltage to achieve desired power dissipation and specific temperatures at the MEA surface. The feedback is controlled by the MCU and its target temperature is set up by the user through the GUI. To ensure absolute thermal accuracy at the MEA surface, the thermal sensor readings were calibrated based on direct measurements of the surface MEA using a high accuracy ( $\pm 0.05^{\circ}\text{C}$ ) thermometer, Traceable (Fisher Scientific, Hampton, USA).

The final version, version 3, depicted in Figure 53, was built based on a dual-rail power supply to provide both positive and negative side of signal readings with a reference to common ground (0V). The PSU supplies two dual-rail of  $\pm 2.8\text{V}$  for supplying on-board controllable voltage regulators that supply the CMOS MEA microchip and  $\pm 3.3\text{V}$  for supporting other peripheral circuits. The PSU converts a 110V 60Hz AC to a single 12V DC, then converts the single 12V DC to dual-rail voltages using high-efficiency commercial DC/DC buck and inverting buck-boost switching converters for positive and negative power lines, respectively. The PSU utilizes four integrated DC/DC switching power modules LMZM33602 (Texas Instruments, Dallas, USA) that can be configured as a standard step-down DC/DC converter or as an inverting DC/DC converter. As shown in Figure 53A and D, the PSU assembly contains AC entry module with fuse and passive filter, AC/DC voltage converter with 12V constant output, and four DC/DC switching power modules.



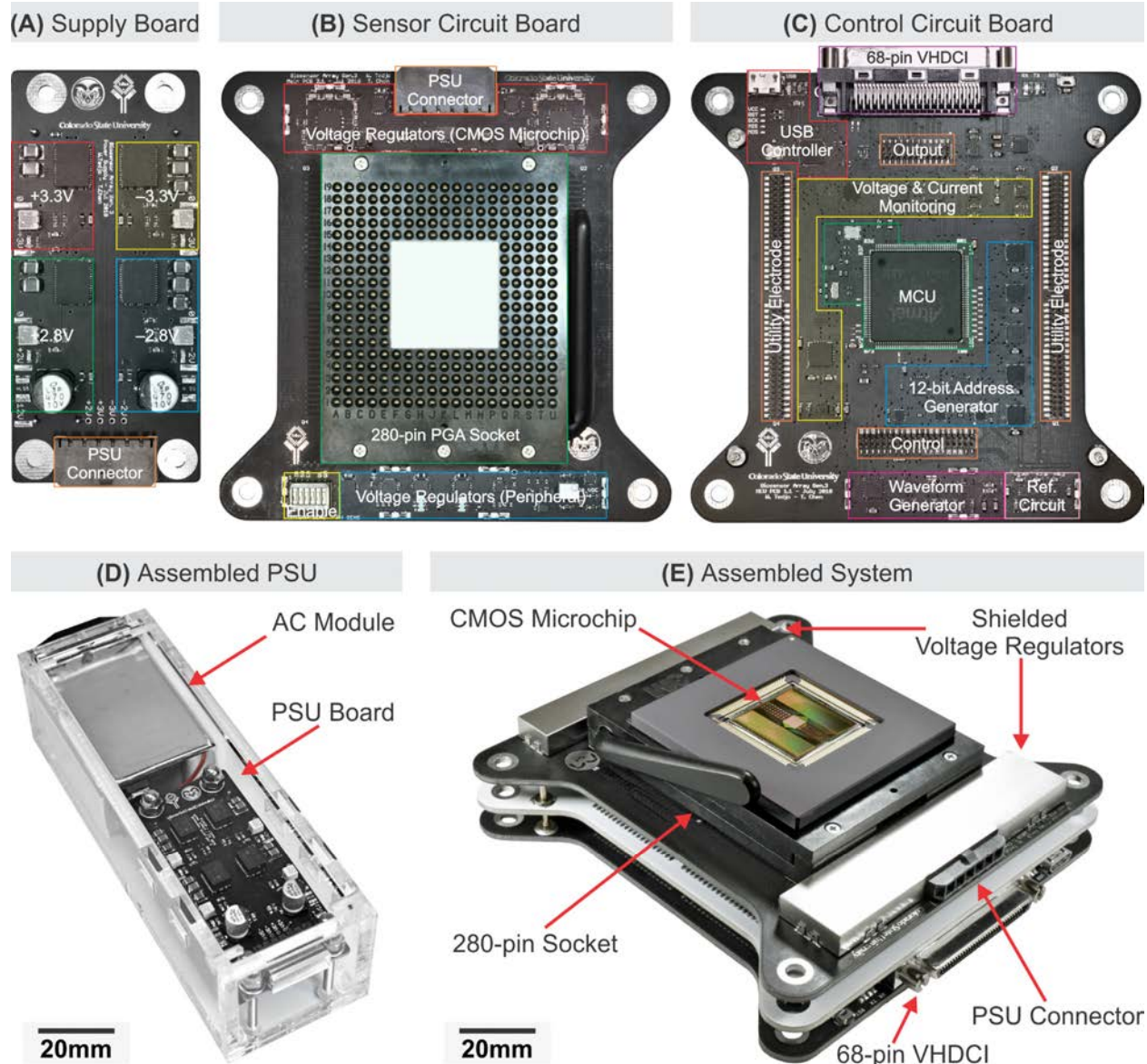


Figure 53. PCB version 3: Final board-level configuration.

(A) Power supply unit circuit board with four DC/DC regulators. (B) Sensor circuit board holds the CMOS MEA microchip and the low noise voltage regulator. (C) Control circuit board with the MCU and other peripheral circuits. (d) Assembled Power supply unit. (E) Assembled system with shielded analog components and installed the CMOS MEA microchip.

The custom CMOS MEA microchip is housed in a 280 pin-grid array (PGA) alumina ceramic package. The package sits on a zero-insertion-force socket which offers convenience for swapping CMOS MEA microchips in and out of the system. As shown in Figure 53B, the socket is centered in the S-PCB and surrounded by the support circuits for the CMOS MEA microchip. The top side holds two power regulating opamp each for adjustable positive and negative potential ( $\pm 1.5V$  to  $\pm 2.5V$ ) for powering the



CMOS MEA microchip. Their potential can be set dynamically by a digital potentiometer where its value is determined by a digital signal sent from the C-PCB. The bottom side holds switches for enabling on-chip circuits, voltage references, constant voltage regulator ( $\pm 2.75\text{V}$ ) for peripheral circuits, and connectors for temperature and pH sensor. The pH sensing circuit employs the instrumentation amplifier INA333 (Texas Instruments, Dallas, USA), which is in the same form of a previously discussed work [91].

As shown in Figure 53B and depicted in a schematic in Figure 11, the MCU chip Atmel SAM3X8E (Microchip, Chandler, USA) generates digital addresses for read channel selection, sets potentiostat voltage, records PCBs and the CMOS biosensor microchip temperature, and monitors current and voltages (the complete MCU firmware is included in Appendix C-1). All information is processed and sent via a USB controller to the host computer. Pulse width modulator (PWM) generates accurate digital clocks to drive digital counters that generate a total of 12 digital bits address. The digital-to-analog converter (DAC) generates specified waveforms for driving the on-chip potentiostat. The analog-to-digital converter (ADC) receives multiplexed signals from various monitoring readings, such as the power supply voltages, current flows in the power lines, and voltage potentials of RE and AE. The I2C serial line provides communication to the digital potentiometer for CMOS MEA microchip voltage supply adjustment, and communication to the temperature chip sensors. These control functions can be modified, and their responses can be monitored in real-time by users in the GUI. All output signals from the CMOS MEA microchip, pH sensor reading, and potentiostat are routed to the external DAQ for accurate analog acquisition through a 68-pin Very-High-Density Cable Interconnect (VHDCI) connector.

The signal connections from and to the S-PCB and C-PCB are made through half-pitch (1.27mm/0.05") headers at four corners of the board, with 160 side connectors dedicated for the 80 pairs of UEs, top side for output voltages from all quadrants, and bottom sides for digital control signals as shown in Figure 53C. All sensitive electrical components (i.e. power supply regulators and voltage reference circuits) on both S-PCB and C-PCB is shielded to reduce EMI and to minimize accidental damage during wet laboratory experiments. This stacking configuration of headers forms rigid electrical and mechanical connections and allows direct vertical add-ons of other custom PCB for additional future expansions as depicted in Figure 53E and Figure 46.



### 6.3. Software Integration

The data interface between the electrochemical imaging system and the host computer is through the DAQ as illustrated in Figure 11. Each analog output signal is effectively sampled at a rate of 100kS/s by the DAQ. Each read channel is given 64 $\mu$ s for its output signal to settle, resulting in approximately six sample points for each read channel before the control logic in the electrochemical imaging system selects the next read channel. Once the output signals from all the read channels are sampled, the dataset is transferred to the host computer and stored in a 16-bit signed integer format. Promptly, a trigger command for starting the next frame acquisition occurs, in parallel with the start of data processing of the acquired frame. During data processing, the acquired data samples from each read channel are averaged over the six samples and stored as a single value representing a pixel in an MEA, stored, and plotted as a heatmap

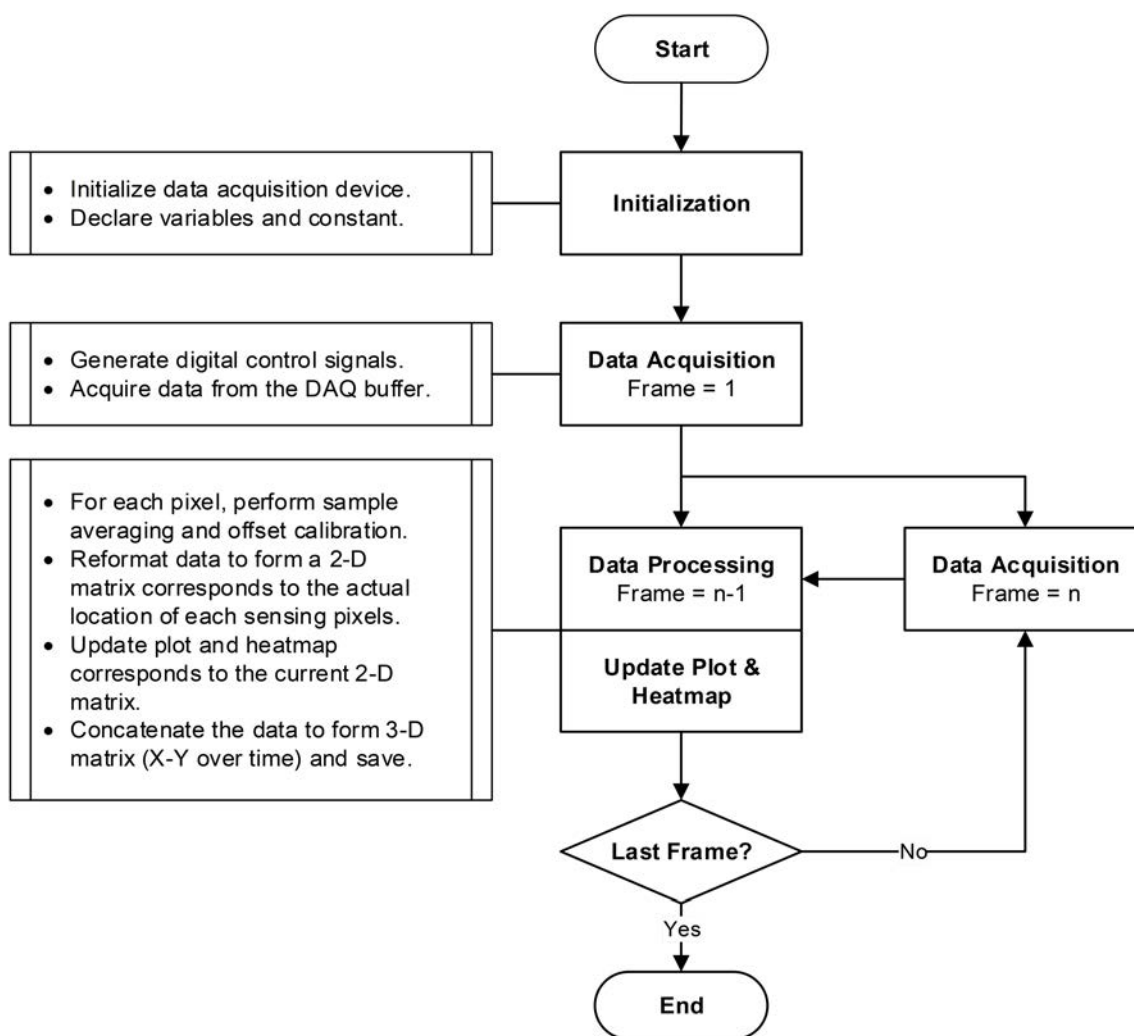


Figure 54. High spatial resolution data acquisition flowchart.



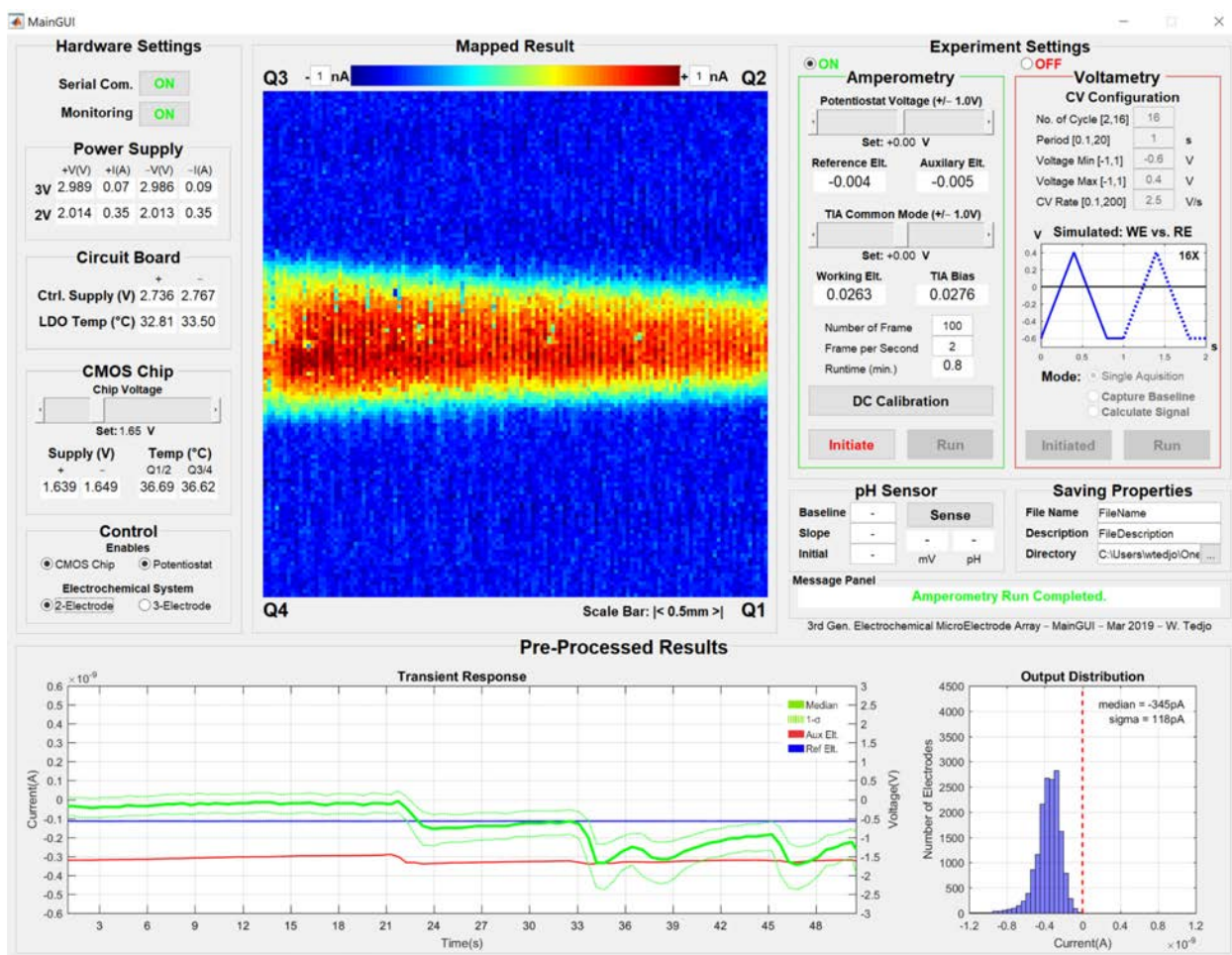


Figure 55. Graphical User Interface: Real-time (MainGUI.m).

of 16-thousand pixel in real-time. A detailed flowchart of the real-time acquisition, processing, and visualization is illustrated in Figure 54. The following subsections discuss the real-time GUI features and baseline calibration algorithm, and the gain calibration algorithm and the post-processing GUI for convenient data processing and visualization.

### 6.3.1. Real-time Data Processing and Control

A custom GUI was created on a host computer using MATLAB with a direct serial interface for communication with the C-PCB. The GUI provides comprehensive control and monitoring over ongoing events and environments of the system. There are two types of analytical chemistry methods supported by the system's GUI: amperometry and CV. In the amperometry mode, user can set the activation voltage, the number of frames to acquire, and the frame update rate up to four FPS. Electrochemical gradient



measurement frames are recorded and shown in real-time as a 16-thousand pixel heatmap. In the voltammetry mode, user can set minimum and maximum potentiostat voltage, scan period, scan rate, and the number of cycles. Voltammetry results were recorded from 64 selected and uniformly distributed WEs across the MEA. Figure 55 shows the real-time heatmap during a flow injection experiment using the amperometry mode to capture 100 frames with a rate of two FPS. During the real-time measurement, the GUI presents the transient responses of the data average and RE and AE voltage over the specified duration, and instantaneous histograms, showing the data distribution corresponds to the present heatmap.

During an imaging experiment, the baseline calibration is manually performed at the beginning to remove the residual DC offset error due to random leakage at each WEs as described previously. The DC calibration should be performed when the system is at a steady-state and stable environmental and electrical condition, and within every imaging experiment run. The process of the spatial baseline calibration follows this algorithm step: First, Electrode Array (A) configuration in x-y coordinates is defined by:

$$\mathbf{A}_{xy} = \begin{bmatrix} a_{11} & \cdots & a_{1y} \\ \vdots & \ddots & \vdots \\ a_{x1} & \cdots & a_{xy} \end{bmatrix} \quad (12)$$

$$1 \leq x \leq 128, \quad 1 \leq y \leq 128$$

Electrode array *Data Raw (DR)* over the *time (t)* is defined by:

$$\mathbf{DR}_{xyt} = \begin{bmatrix} dr_{11t} & \cdots & dr_{1yt} \\ \vdots & \ddots & \vdots \\ dr_{x1t} & \cdots & dr_{xyt} \end{bmatrix} \quad (13)$$

$$1 \leq t \leq \# \text{ of frame in an experiment}$$

Baseline Calibration (BC) data is captured at the beginning of experiment (dry electrode array) over specified number of frames (f), and defined as:



$$\begin{bmatrix} bc_{11} & \cdots & bc_{1y} \\ \vdots & \ddots & \vdots \\ bc_{x1} & \cdots & bc_{xy} \end{bmatrix} = \mathbf{BC}_{xy} = \frac{1}{nbase} \sum_{f=1}^{nbase} \begin{bmatrix} bc_{11f} & \cdots & bc_{1yf} \\ \vdots & \ddots & \vdots \\ bc_{x1f} & \cdots & bc_{xyf} \end{bmatrix} \quad (14)$$

$f = \# \text{ of frame for baseline calculation (nbase)}$

Calibration of baseline; generates *Data Baselined (DB)* output:

$$\begin{bmatrix} bc_{11t} & \cdots & bc_{1yt} \\ \vdots & \ddots & \vdots \\ bc_{x1t} & \cdots & bc_{xyt} \end{bmatrix} = \mathbf{DB}_{xyt} = \sum_{t=1}^{stop} (DR_{xyt} - BC_{xy}) \quad (15)$$

Therefore,

$$\mathbf{DB}_{xyt} = \sum_{t=1}^{stop} \left( \begin{bmatrix} dr_{11t} & \cdots & dr_{1yt} \\ \vdots & \ddots & \vdots \\ dr_{x1t} & \cdots & dr_{xyt} \end{bmatrix} - \frac{1}{nbase} \sum_{f=1}^{nbase} \begin{bmatrix} bc_{11f} & \cdots & bc_{1yf} \\ \vdots & \ddots & \vdots \\ bc_{x1f} & \cdots & bc_{xyf} \end{bmatrix} \right) \quad (16)$$

$1 \leq x \leq 128, \quad 1 \leq y \leq 128, \quad t = \# \text{ of frame in an experiment until (stop)}$

This step concludes the baseline calibration process. DB is shown in real-time in the GUI, while BC, is saved together with the other data and information for further data analysis processes.

### 6.3.2. Post-Processing and Visualization

Gain calibration due to variation in WE responses and the read channel gain variation is performed in the post-processing of calibration data generations. This gain calibration is also possible for future integration in every electrochemical imaging run. Gain Calibration (GC) data is captured at a certain duration of exposure to calibration chemicals at different concentration. The following steps describe the spatial gain calibration algorithm. First, calculate the mean of a concentration ( $\mu C$ ) for each frame for a specified duration from *calStart* to *calEnd*.

$$\begin{bmatrix} \mu C_{calStart} \\ \vdots \\ \mu C_{calEnd} \end{bmatrix} = \mu C_g = \frac{1}{x \cdot y} \sum_{x=1}^{128} \sum_{y=1}^{128} \left( \frac{1}{calEnd - calStart} \sum_{g=calStart}^{calEnd} DB_{xyg} \right) \quad (17)$$



$g = \text{frame for gain calculation (calStart to calEnd)}$

Use  $(\mu C)$  to calculate deviation of gain/ratio value for each electrode over a duration from  $calStart$  to  $calEnd$ . The average of deviation of gain/ratio value from each electrode stored as Gain Calibration (GC). Recall, Data Baselined (DB) output is the baseline calibrated data over the time:

$$DB_{xyt} = \begin{bmatrix} db_{11t} & \cdots & db_{1yt} \\ \vdots & \ddots & \vdots \\ db_{x1t} & \cdots & db_{xyt} \end{bmatrix} \quad (18)$$

Therefore, Single Gain Calibration (SGC) in x-y array for a concentration:

$$\begin{bmatrix} sgC_{11} & \cdots & sgC_{1y} \\ \vdots & \ddots & \vdots \\ sgC_{x1} & \cdots & sgC_{xy} \end{bmatrix} = \mathbf{SGC}_{xy} = \frac{1}{calEnd - calStart} \sum_{g=calStart}^{calEnd} \left( \frac{DB_{xyg}}{\mu C_g} \right) \quad (19)$$

At last, take average of all concentration ( $nconc$ ) data to generate a x-y array Gain Calibration (GC) data:

$$\begin{bmatrix} gC_{11} & \cdots & gC_{1y} \\ \vdots & \ddots & \vdots \\ gC_{x1} & \cdots & gC_{xy} \end{bmatrix} = \mathbf{GC}_{xy} = \frac{1}{nconc} \sum_{c=1}^{nconc} (SGC_{xyc}) \quad (20)$$

$c = \# \text{ of concentration for gain calculation (nconc)}$

Generate *Data Gain Calibrated (DGC)* output:

$$DGC_{xyt} = \sum_{t=1}^{stop} \left( \sum_{x=1}^{128} \sum_{y=1}^{128} \frac{db_{xyt}}{gC_{xy}} \right) \quad (21)$$

This concludes the gain calibration steps.



A generated dataset from an electrochemical imaging experiment is a form of an unprocessed data over a defined period, voltages readings (e.g. RE, AE, CMOS microchip power supply), current readings, and temperature. A custom GUI was created to support easy data processing of an electrochemical imaging experiment. The GUI support real-time scale adjustment of the heatmap, instantaneous plotting of a selected frame, and spatial selection for defining reference and signal over a specific target location. Figure 56 depicts the post-processing GUI, showing smoothed data heatmap, in corresponds to an optical image that was previously recorded during an electrochemical imaging experiment. The scale settings help the user to show current scaling representation (the blue to red scale) in order to properly analyze the heatmap spatially. While the time slider bar offers quick transitions between different time points. Spatial analysis features are also available to manual select a specific location in the MEA. In this specific example, the

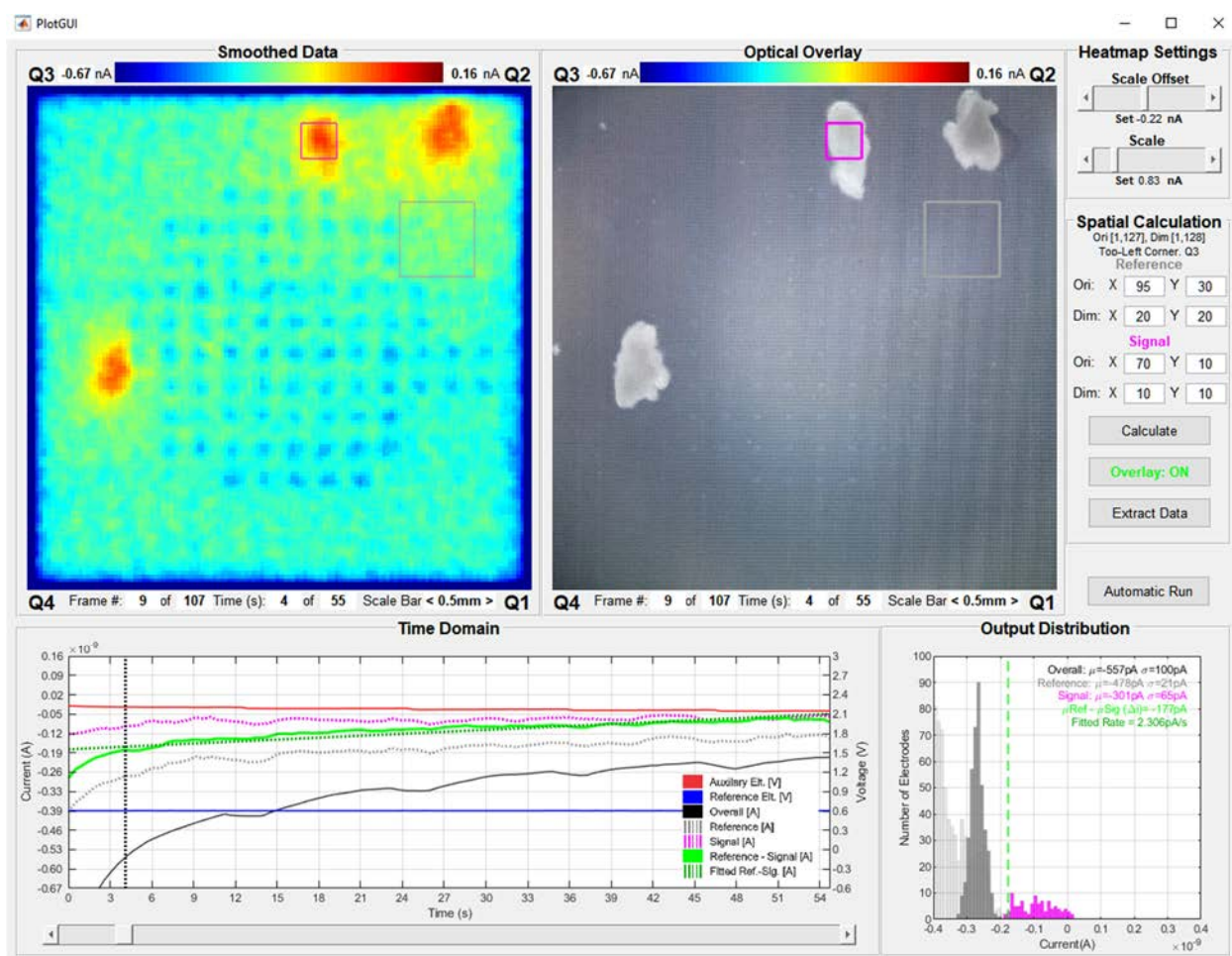


Figure 56. Graphical User Interface: Post-processing (PlotGUI.m).



magenta and grey boxes were selected to represent signal over a bio-sample and the reference as the background, respectively. The complete MATLAB code for the real-time GUI is presented in Appendix C.

#### 6.4. Microfluidic Platform

Bio-sample handling and delivery to sensing devices also presents its own challenges. Microfluidics system offers a convenient method to safely and reliably handling oocytes/embryo while allowing integrations of sensing electrodes [92], [93]. The microfluidic system presented in the paper requires minimum resources to fabricate, offers rapid prototyping turnover time, and provide the flexibility of various

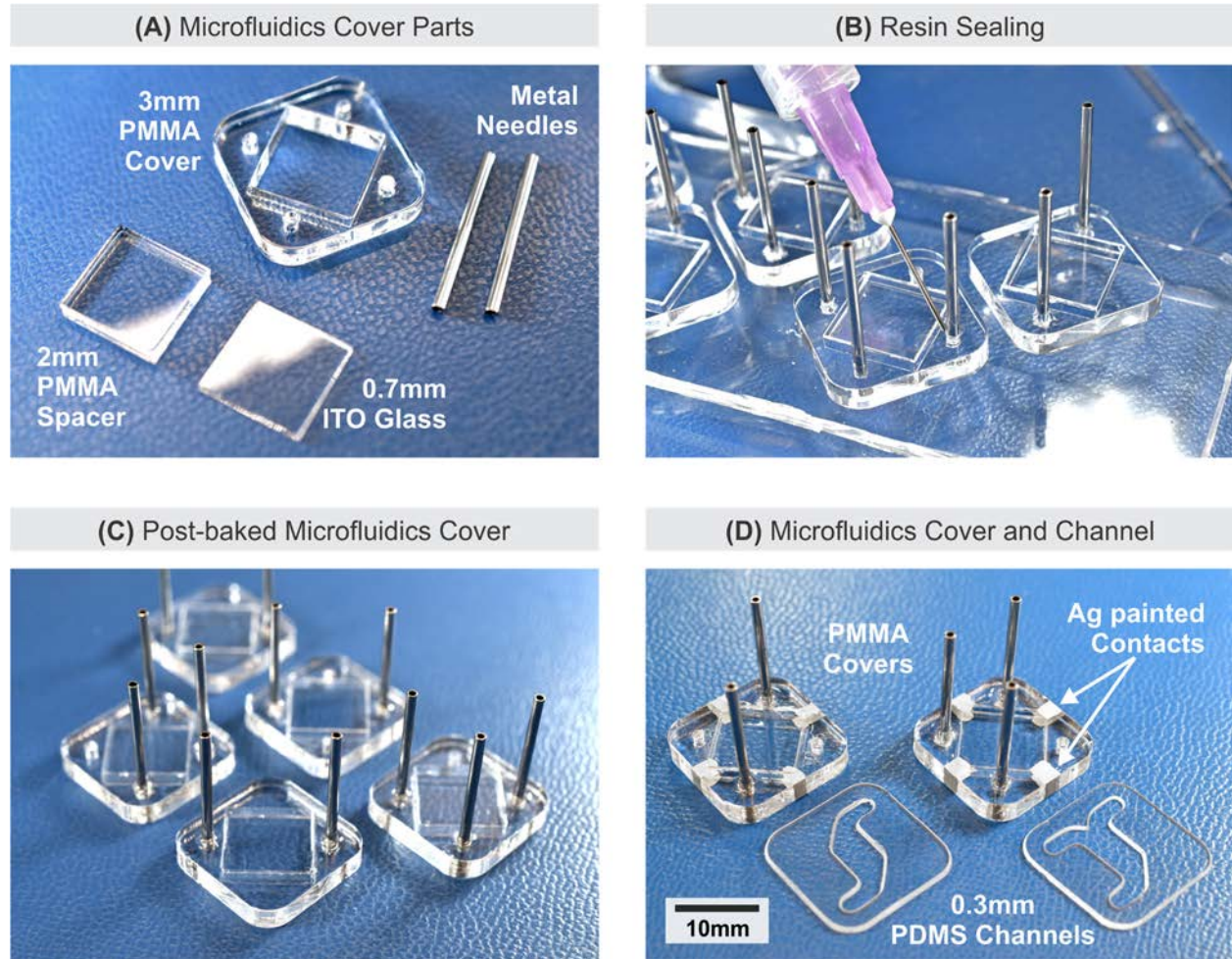


Figure 57. Microfluidic support system.

(A) Unassembled parts of the microfluidic cover obtained and fabricated from low-cost off-the-shelf components and a laser cutter. (B) Assembly process by depositing medical grade resin to seal the microfluidic cover. (C) Cured microfluidic covers. (D) Two types of assembled microfluidic PPMA cover and PDMS channel.



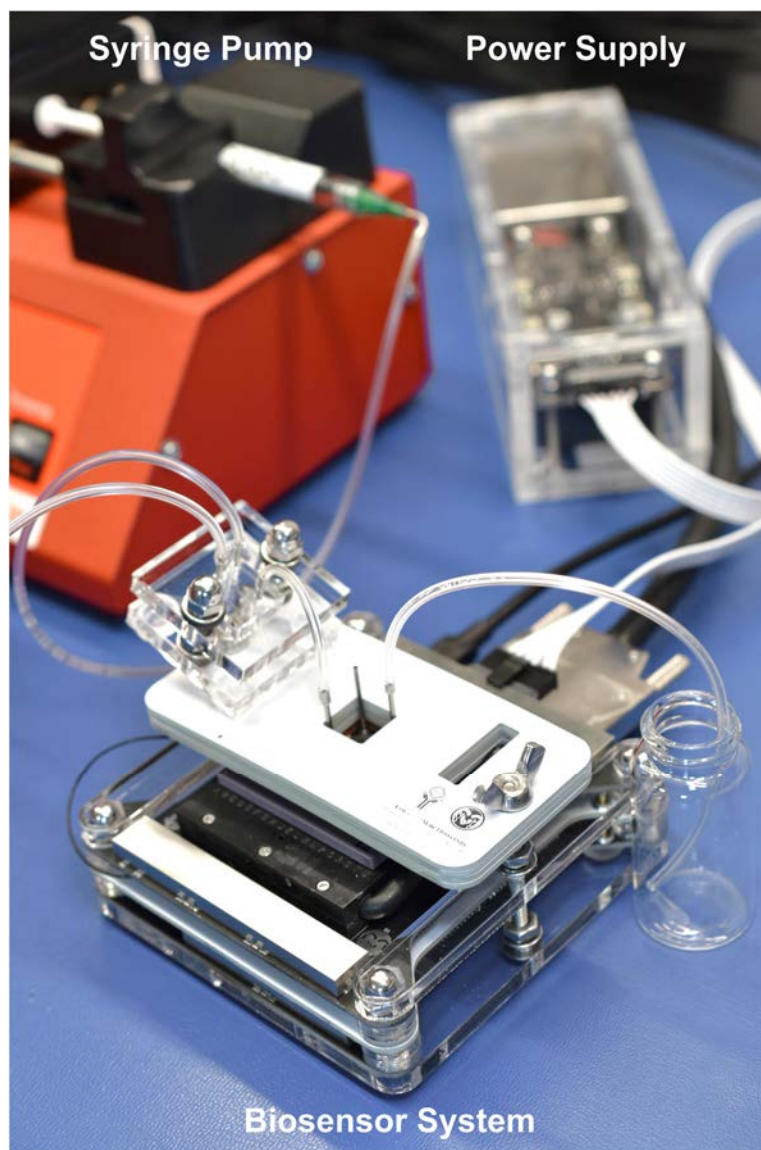


Figure 58. Assembled biosensor and microfluidic support system.

configuration. The fabrication process requires only low-cost off-the-shelf components and a laser cutter typically available in bioengineering laboratories.

First, a 3mm thick PMMA sheet is cut to form an 18mm×18mm frame with 1.0mm circular hole openings for inlet/outlet metal needles, and a 10mm×10mm placeholder for the ITO-coated glass. The inlet/outlet metal needles were extracted from 0.5" 18-ga blunt dispense needles (Part# 98399, Loctite®). The 120nm ITO-coated glass (ITO-101007-15C, MTI Corp.) is 0.7mm-thick and in an unmodified state, which provides electrical conductivity as AE and transparency for optical imaging. The ITO surface is placed facing down in the middle of the PMMA frame and a 10mm×10mm of 2mm-thick PMMA is placed



immediately above it to fill the excess of the square hole in PMMA frame. The unassembled parts for the microfluidics cover are depicted in Figure 57A. The microfluidics cover parts are carefully assembled, held together using medical grade-epoxy (EPO-TEK® 301, Epoxy Technology Inc.), and placed in a 3mm-thick 1:40 ratio PDMS substrate. The PDMS substrate is then positioned in a flat surface panel (e.g. a thick PPMA sheet or unused PCB) for resin curing process as shown in Figure 57B. This setup is to ensure the flatness of PMMA cover while the sticky 1:40 PDMS substrate safely covers the ITO-coated surface from excess pre-cured resin. Figure 57C shows variants of PMMA covers with two and three inlet/outlet needles, providing options for delivering up to two different fluids for in vitro experiment. Finally, as shown in Figure 57D, Ag conductive paint (Part# 05001-AB, SPI® Supplies) is applied to form electrical contact from the ITO surface on the bottom side to PMMA cover top side where the Pressure plate makes the electrical contact to be connected to the Sensor board.

Fabricating the PDMS channel membrane requires molding of 1:10 PDMS to a custom-made mold with a 300µm thick gap. The mold is a stack two PMMAs with a thin spacer at any thickness in between. This spacer could be transparency plastic, rubber membrane, or any other laser cuttable materials. To ensure the edge quality of the laser cut PDMS membrane, the 300µm PDMS membrane is suspended between two thick PMMA sheets and cut to desirable patterns (one-inlet one-outlet and two-inlet one-outlet.) as shown in Figure 57D. The 300µm channel height forms a channel that holds 3.9uL fluid capacity above the 3.6mm×3.6mm MEA, while the entire microfluidics inlet/outlet needles and channel volume holds approximately 50uL.

The microfluidic support system, and the pressure plate to hold and create a tightly sealed microfluidic channel. As shown in Figure 58, the microfluidic setup allows controlled fluid injection using a syringe pump (NE-1000, New Era Pump System Inc.) for delivering and retrieving bio-samples to and from the MEA location on the CMOS microchip.



## CHAPTER VII: SYSTEM VALIDATION: ELECTRICAL AND CHEMICAL MEASUREMENTS

The overall system, especially the CMOS MEA microchip, underwent a stringent initial verification phase to determine its electrical performance, such as its performance in the areas of the limit of detection, sensitivity, selectivity, and spatiotemporal resolutions. Experiment results discussed in this section were performed at a constant power supply potential at  $\pm 1.65\text{V}$  to keep the CMOS MEA microchip temperature constant within  $37 \pm 0.5^\circ\text{C}$ . Throughout the experiments, the chemical imaging system was operated at a 15.625kHz read channel scanning rate. With all 4,016 read channels available in each quadrant, the frame update rate is approximately four FPS. The on-chip potentiostat and three-electrode system were used to perform both amperometry and CV to generate the analytical chemistry results. The surface MEA is in an unmodified state, which is bare Pt electrodes.

### 7.1. Electrical Performance

As described in Chapter IV, the read channel is in a form of continuous TIA with T-network feedback resistance. The requirements for the read channel are high I-V conversion gain, an adequate frequency response for CV, and low input-referred noise for better sensitivity. The tradeoff between noise and

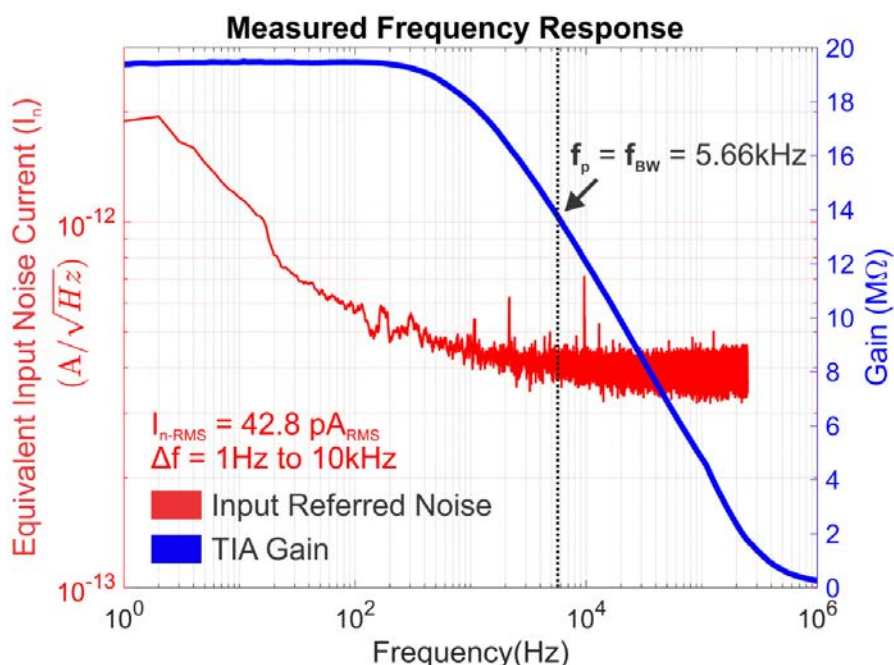


Figure 59. Measured frequency response of the T-network TIA, gain and noise response.



Table 7. Simulated opamp and measured TIA specifications.

Opamp - Simulated				TIA - Measured			
Parameters	Condition	Val.	Unit	Parameters	Condition	Val.	Unit
<b>Ao</b>	DC	97.7	dB	Gain		19.6	MΩ
<b>Offset</b>	G = 1	3.34	mV-σ		variation	0.48	MΩ-σ
<b>PM</b>		51.9	°	Offset	I <sub>IN</sub> = 0A	5.81	nA-σ
<b>BW</b>	-3dB	12.6	Hz	BW	-3dB	5.66	kHz
<b>GBWP</b>		844	kHz				
<b>Output Noise</b>	1Hz	1.66	μV/√Hz	<b>Input Noise</b>	1Hz	1.89	pA/√Hz
	100Hz	0.29	μV/√Hz		100Hz	0.57	pA/√Hz
	10kHz	0.06	μV/√Hz		10kHz	0.44	pA/√Hz
	1Hz-10kHz	10.4	μV <sub>RMS</sub>		1Hz-10kHz	42.8	pA <sub>RMS</sub>
<b>Output Range</b>	to V <sub>DD</sub> +	0.32	V	<b>Output Range</b>	Max	72.3	nA
	to V <sub>DD</sub> -	0.34	V		Min	-74	nA
<b>Power</b>	G = 1	55.0	μA	Power	I <sub>IN</sub> = 0A	59.6	μA

\* Opamp simulation: VDD = ±1.65V, T = 37.0°C, R<sub>L</sub> = 25kΩ, C<sub>L</sub> = 2pF.

\* TIA Measurement: VDD = ±1.65V, T = 37.0°C, C<sub>DL</sub> = 100pF, C<sub>L</sub> = 2pF.

bandwidth of the read channel is achieved by fine-tuning the feedback capacitor, C<sub>F</sub>. Figure 59 shows the TIA gain, bandwidth, and integrated noise responses measurement of 19.6MΩ, 5.66kHz, and 42.8pA<sub>RMS</sub>, respectively. The bandwidth of the read channel is suitable to capture the chemical molecule diffusivity action in fluid, approximated between 0.1 to 10μm<sup>2</sup>/ms, within the 27.5μm spatial resolution. The integrated noise frequency range of 1Hz to 10kHz was defined to incorporate low frequency noise close to DC and approximately up to twice of the TIA bandwidth. Table 7 summarizes the opamp and TIA read channel specifications with according to the real electrochemical experiment specifications of normal biological temperature at 37.0°C and C<sub>DL</sub> = 100pF.



## 7.2. Amperometry

A flow injection system with two syringe pumps was set up for determining sensitivity to NE and  $pO_2$ . Figure 60 illustrates the two pumps setup for the flow injection experiments which their results are presented in this section. The differential values between control (DI- $H_2O$ ) and sample (concentrations of NE) were recorded. The flow injection setup allows sudden flow change between the control and sample to produce ranges of step-responses for different NE concentrations as depicted in Figure 62 (error bars signify  $1-\sigma$ ). An activation voltage of +0.6V was used in the neurotransmitter amperometric experiments.

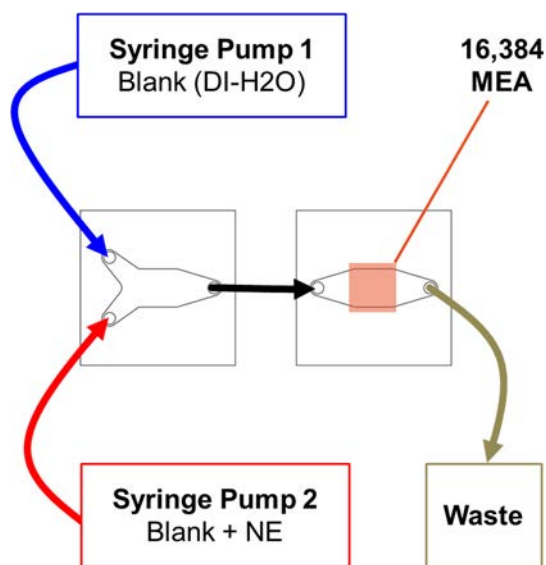


Figure 60. Flow injection: Setup.

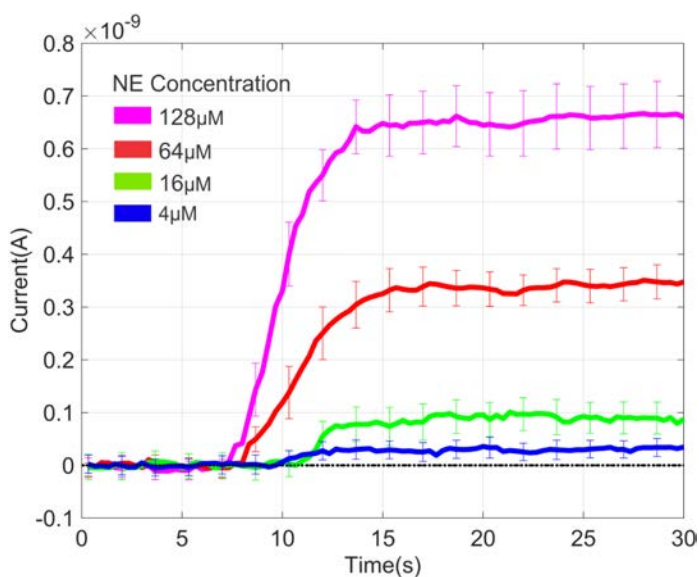


Figure 61. Flow injection: Step function responses.



The sensitivity response data in Figure 62A and B were generated from differential values between blank readings average at  $t = 0\text{ s} - 5\text{ s}$ , and signal readings at  $t = 15\text{ s} - 20\text{ s}$  shown by Figure 61. The result summarized the dynamic range and sensitivity at the micromolar concentration range. This was used to establish the range of detection and to show the system's limitation in differentiating low concentrations of analytes down to  $4\mu\text{M}$  and the linear dynamic range is limited below  $512\mu\text{M}$ . This limitation may be influenced by several factors that limit the redox current flow between AE and WEs (e.g. the low ratio of AE to WE surface area, and limited voltage compliance and current driving strength of the on-chip potentiostat).

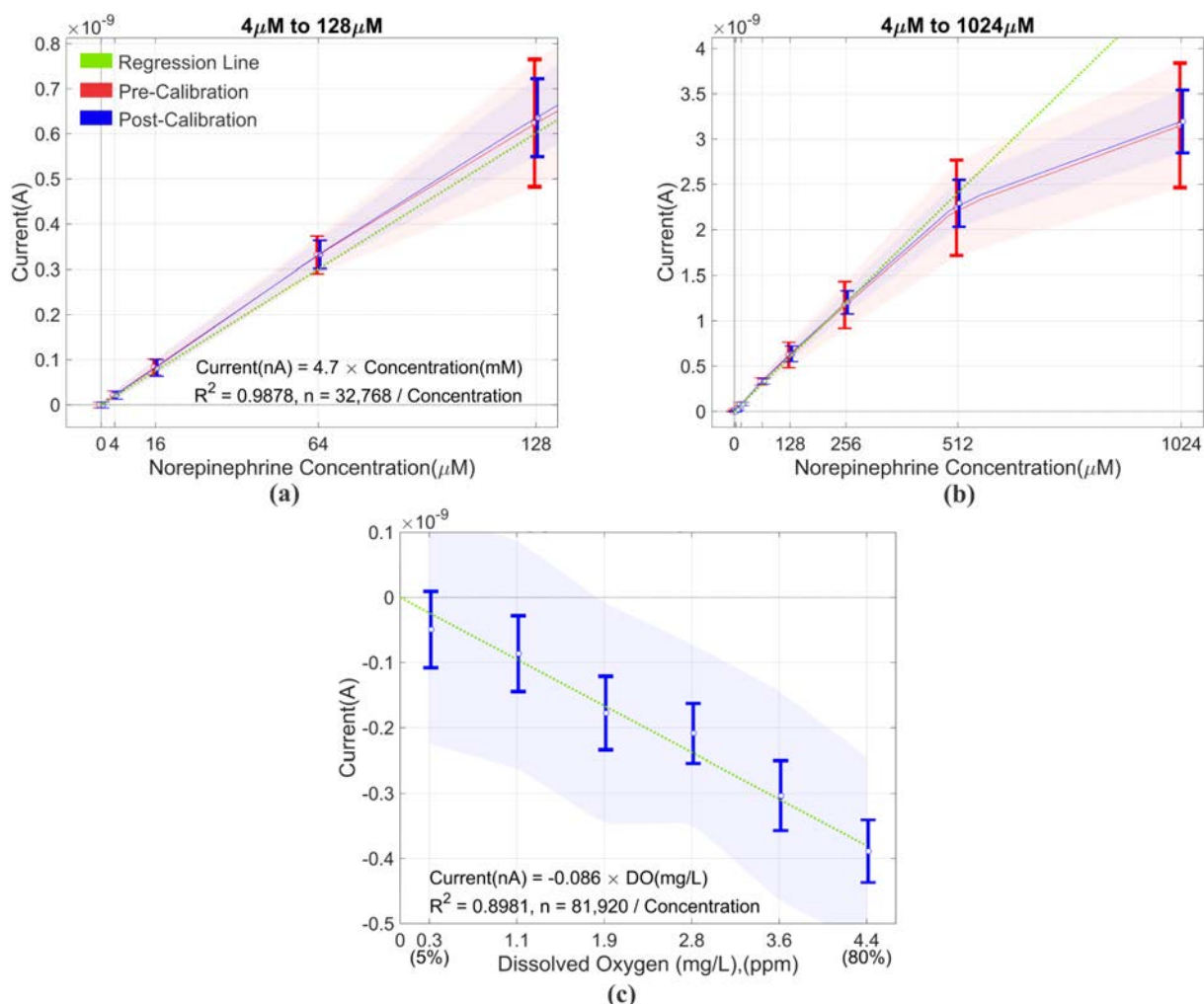


Figure 62. Amperometry response to NE and  $\text{pO}_2$ .

(A) Sensitivity response generated from the step-response experiments, showing calibration curve and dynamic range of the system for NE concentration from  $4\mu\text{M}$  to  $128\mu\text{M}$ . (B)  $4\mu\text{M}$  to  $1,024\mu\text{M}$ . Error bars signify the  $1-\sigma$  values of 16-thousand read channels with  $n = 32,768$  for each concentration. (C) Response to  $\text{pO}_2$  with  $n = 81,920$  per concentration. Error bars and the shaded area signify  $1-\sigma$  and  $3-\sigma$  values, respectively.



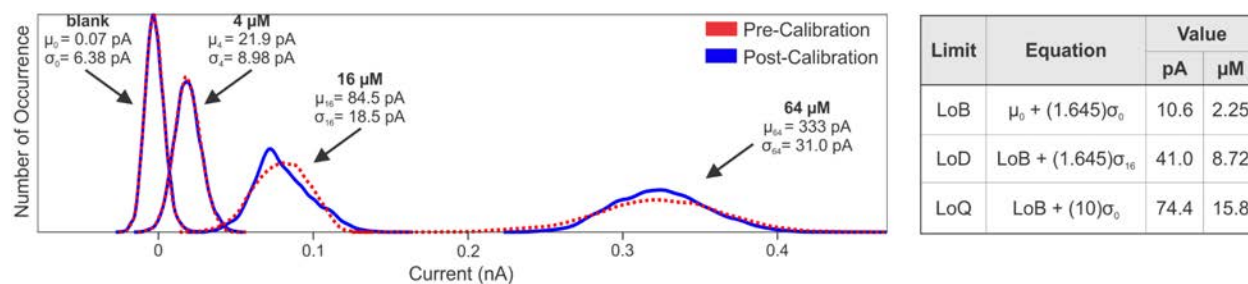


Figure 63. Limit of Detection of NE.

Improvements to the dynamic range could be made by substituting the on-chip potentiostat with a higher voltage compliance external on-board potentiostat and utilizing the ITO glass electrode to increase AE to WE area ratio. The sensitivity result shows a higher WE current of 4.7nA/mM in comparison with the previous generation MEA of 3.35nA/mM [23], confirming that larger WE surface area increases steady-state current.

Figure 63C depicts the system response to pO<sub>2</sub> concentrations between 5% and 80%. Similar experiment method to sense NE was used, except the activation voltage was set to -0.7V to ensure complete oxygen reduction as previously described in [7], [60]. The system sensitivity for pO<sub>2</sub> is approximated at 86pA/ppm or 4.7pA/% pO<sub>2</sub> at 37°C and 5,003 feet (1,525m) altitude.

To fully appreciate the sensitivity limit of the system, calculations of blank (LoB), detection (LoD), and quantitation (LoQ), were also performed based on the Clinical and Laboratory Standards Institute (CLSI) EP17 guidelines [94]. Based on the NE chemical experiment, the LoD is calculated to be 41.0pA, an improvement over the previous system (52.5pA) [22], [23]. This indicates that the system can reproduce chemical images at similar or better fidelity. The Probability Density Function (PDF) of a blank sample, and NE at 4μM, 16μM, and 64μM using amperometry with n=32,768 per concentration are plotted and shown in Figure 63.

### 7.3. Voltammetry

Simultaneous detection of multiple target analytes has increasingly received more attention in biosensing. Recent works showed simultaneous detection of multiple analytes with Differential Pulse Voltammetry (DPV) [95]–[97]. Figure 64A, shows CV responses of NE at different concentrations with different scan rates. Each curve represents simultaneous recording from 64 (16 per quadrant) WEs



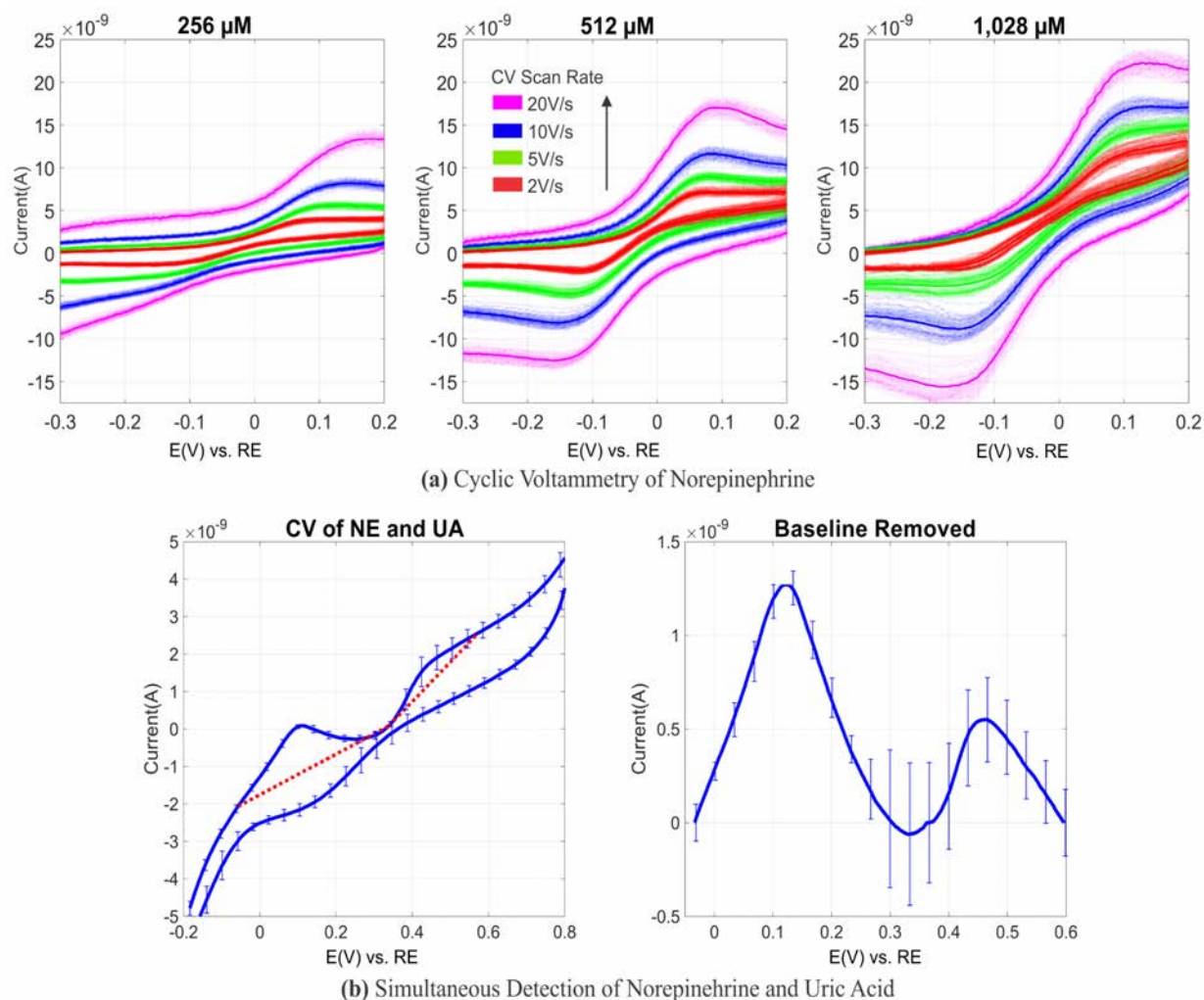


Figure 64. Cyclic Voltammetry response to NE, and selectivity to NE and UA.

(A) CV response of various sweeping rate (2V/s to 20V/s) for three NE concentrations, 256 $\mu$ M, 512 $\mu$ M, and 1,028 $\mu$ M across 64 selected WEs. (B) Simultaneous detection of multiple analytes, mixture of sample containing 0.5mM NE and 0.5mM UA across 64 selected WEs. Each error bar indicates 1- $\sigma$  values.

uniformly distributed across the entire MEA area. Each read channel output is presented in opaque lines and their average is plotted as a solid line. The results are consistent with other reported results [98], [99] that showed the trend of increasing current peaks with higher analyte concentrations and the effect of scan rate [38]. Figure 64B shows the CV results of 64 selected WEs for a 0.5mM NE and 0.5mM UA solution at 2V/s scan rate. A simultaneous detection result is indicated by the first peak at 0.12V for NE and second peak at 0.47V for UA.



#### 7.4. Electrical and Chemical Measurement Variations

The gain and baseline offset variations across the 16,064 read channels are expected due to combinations of random variations related to the circuit topology, CMOS fabrication, and the transducer. These variations include TIA feedback resistance variation ( $R_{EQ-\sigma}$ ), input offset of  $OP_{TJ}$  and  $OP_{TIA}$  ( $V_{OS-TJ}$  and  $V_{OS-TIA}$ ), WE geometry or surface area variation, and the WE surface condition. The interconnect parasitic resistance ( $R_{RT}$ ) was approximated and simulated to vary from 0.5 k $\Omega$  to 1.8k $\Omega$  due to various distances from the read channel to its WE, which causes the potential at the WE to shift from  $V_{CM}$  based on the value of  $I_{IN}$ . However, an extreme case scenario of  $I_{IN} = 100\text{nA}$  and  $R_{RT} = 1.8\text{k}\Omega$  yields a potential shift of 0.18mV, a negligible value in an electrochemical process. In summary, in an agreement to Eq. (2), the  $I_{IN}$  conversion gain variation is defined by the  $R_{EQ}$  with the addition of variations in WE geometry and surface condition. Meanwhile, the baseline variation is defined by the product of  $(1 + R_3/R_2)$  and  $V_{TJ}$  and  $V_{CM}$ , which is directly affected by the input offsets of  $OP_{TJ}$  and  $OP_{TIA}$ , respectively.

The  $R_{EQ-\sigma}$  variation is 0.11M $\Omega$ - $\sigma$  in simulation and 0.48M $\Omega$ - $\sigma$  in measurement across the 128 calibration channels. To capture the overall gain variation, chemical calibration experiments were performed to incorporate the effect of WE geometry and surface condition variations. The overall gain variation was measured to be 4.27M $\Omega$ - $\sigma$ . The results suggest that the gain variation is dominated by the WE variations as the transducer. Therefore, chemical calibration should be performed periodically to

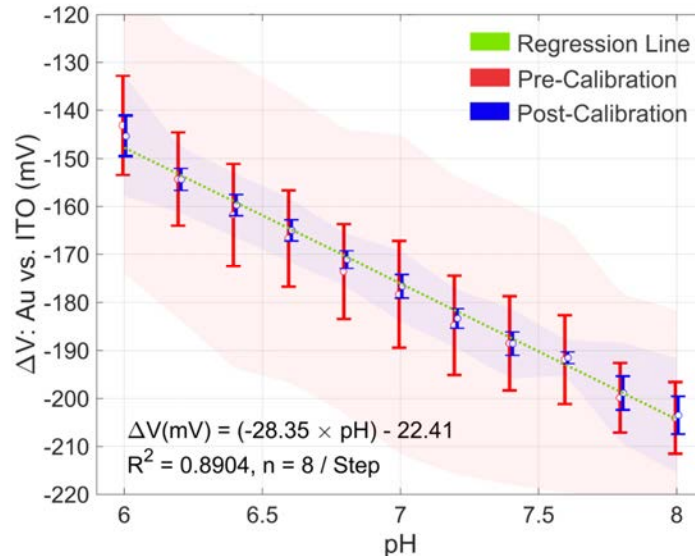


Figure 65. pH sensitivity response.

*Response to pH between 6.0 and 8.0 using the ITO and Au glass electrode. Error bars and the shaded area signify 1- $\sigma$  and 3- $\sigma$  values, respectively.*



account for the WE surface degradation or fouling. Due to the  $(1 + R_3/R_2)$  amplification to the 3.34mV- $\sigma$  opamp input offset, the measured baseline variation is  $V_{OUT-\sigma} = 118\text{mV}$  or equal to the input-referred variation of  $I_{IN-\sigma} = 5.94\text{nA}$ . Gain and baseline offset variations were measured at the TIA  $V_{OUT}$  and defined as coefficients for each 16,064 read channel. With regulated temperature and power supply voltage, these values are mainly static and can be individually measured, stored, and used to calibrate each pixel during the data post-processing.

### 7.5. pH Response

Due to the potential fluctuation of pH during electrochemical experiments, it is imperative to monitor pH in real-time to ensure predictability of the bio-sample behavior [100], [101]. Using the glass electrode described in Chapter VI, the ITO and Au electrodes offer linear pH response of -28.35mV/pH for a range of 6.0pH to 8.0pH measured at 37°C as shown in Figure 65.

### 7.6. High Resolution Chemical Imaging

In ex vivo experiments with live samples, it is highly desirable to capture releases of target analytes in a high spatiotemporal resolution. As depicted in Figure 66, screenshots of the electrochemical imaging heatmaps were generated from flow injection experiments. The experiments were performed with the support of three-inlet microfluidic channel where two outer inlets were connected to two syringes containing control (DI-H<sub>2</sub>O), set at initial flow rate of 50 $\mu\text{L}/\text{min}$ , and the middle inlet was connected to a syringe containing samples (i.e. NE and degassed 40%-oxygen DI-H<sub>2</sub>O). The heatmap results were post-processed with DC offset adjustment, gain calibration, frame-to-frame smoothing, and simple spatial mean filtering [102] to show the gradient actions. However, excessive spatial mean filtering would decrease the spatial resolution below the initial 27.5 $\mu\text{m}$  pitch. For results presented in Fig. 10, the three-pixel spatial mean filtering was used.

The first imaging was performed by flowing 100 $\mu\text{M}$  NE at a 10 $\mu\text{L}/\text{m}$  between two streams of 50 $\mu\text{L}/\text{m}$  DI-H<sub>2</sub>O. At  $t = 2\text{s}$ , an abrupt change of outer DI-H<sub>2</sub>O flow rate to 90 $\mu\text{L}/\text{m}$  and 10 $\mu\text{L}/\text{m}$  was introduced to shifts of the constant NE flow to the bottom corner. The imaging results are shown in Figure 66A. The second imaging experiment was performed by introducing rapid increase of NE flow rate from 0 $\mu\text{L}/\text{m}$  to



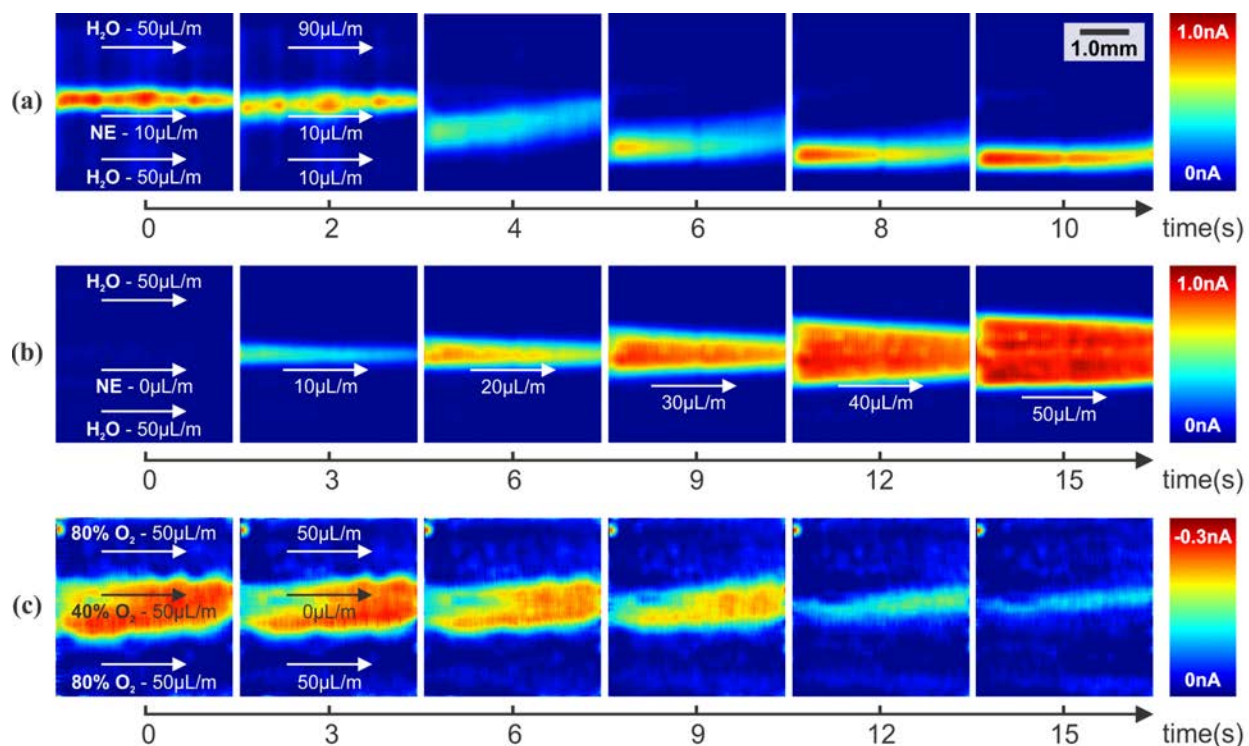


Figure 66. High spatiotemporal resolution imaging results of flow injection.

(A) and (B) Different pattern of flow injection of 100 μM Norepinephrine. (C) Flow injection of DI-H<sub>2</sub>O at different oxygen concentrations of 80% and 40%.

50 μL/m within 15s, creating a wider NE flow pattern in the middle with the increase in NE flow rate. The results are shown in Figure 66B. Finally, as presented in Figure 66C, the imaging for pO<sub>2</sub> concentration was started with 50 μL/m flow rates of the 80% and 40% pO<sub>2</sub>. A sudden stop of the 40% pO<sub>2</sub> concentration was introduced at t = 2s and followed by the middle flow signal decaying within 10s. The resulting heatmaps with hundreds of pA gradients not only show sensitivity performance but also demonstrate the spatiotemporal resolution of the system.



## CHAPTER VIII: APPLICATION: OXYGEN IMAGING

### 8.1. Oxygen Imaging Preparation

Developments of oxygen sensing methods for microenvironment are generally tailored to specific applications. Desirable features for a  $pO_2$  imaging system include high spatial and temporal resolution, low limit of detection, non-invasiveness to bio-samples, sensor repeatability and reusability, and compatibilities to other types of sensing (e.g. temperature, pH, chemical, impedance). The existing methods of oxygen sensing include electrochemical methods (amperometric and potentiometric), optical methods (fluorescence and phosphorescence), and magnetic and paramagnetic resonance imaging methods (MRI and EPRI). Both MRI and EPRI methods offer 3D non-invasive in vivo sensing while requiring contrast agents and complex magnetic resonance instrumentations to perform [103]–[106]. The optical method has been well established and commercialized over the years for in vivo and in vitro tissue and cell culture imaging [107]. Design and modifications of the luminescence probes and reagents lead to various kind of sensing in combination with  $pO_2$  sensing, each with its own specific applications, advantages, and disadvantages [108], [109]. The latest development of optical  $pO_2$  imaging enables scientists to perform micrometer spatial resolution intracellular imaging [110], in vivo imaging with 2Hz frame rate with 600 $\mu$ m-deep probe penetration [111], and 3D cell cultures imaging [112]. Despite its benefits, the optical method possesses some complications that must be minimized during live sample experiments, such as phototoxicity, photobleaching, and photostability [113].

The electrochemical method for  $pO_2$  sensing of biological samples has been developed [114], [115] and the Clark-type sensor's capability has also been demonstrated [4], [7], [8] in one-dimensional data points. Capturing the  $pO_2$  gradients has also been attempted with a limited number of microelectrodes in a circular configuration [5]. In a recent development, SECM has been advanced to realize nanometer spatial resolution and picomolar sensitivity; however, with very limited temporal resolution [116]–[119]. The 2D electrochemical method using solid-state CMOS microchip with MEA has been developed for chemical imaging without any attempt in measuring  $pO_2$  [23], [45], [52], [53], [66], [120], [121]. Finally, the first attempt of  $pO_2$  imaging using a CMOS-based sensor was presented [122]. However, the device was only



characterized in non-biological flow injection experiments at 250 $\mu\text{m}$  spatial resolution and 400 measurement points.

An application in oxygen sensing allows the biosensor device to demonstrate the high spatial and temporal resolution in measuring  $\text{pO}_2$  gradients that have not been possible in the past. These preliminary oxygen gradient maps offer a unique insight not only for the life-sciences disciplines, but also in the engineering of the instrumentation development, specifically for the microenvironment scale oxygen imaging. The final version of the system was used in the application, however, with a minor change in the RE and RE configuration. Using the previously discussed microfluidic setup, sets of in vitro experiments on bovine cumulus-oocytes-complexes (COCs) were performed to produce 2D  $\text{pO}_2$  images. Oxygen imaging results were analyzed according to the previous  $\text{pO}_2$  gradient analysis simulations and methods [5], to show  $\text{pO}_2$  flux density and consumption rate in the high-density MEA.

#### 8.1.1. MEA Improvements for Oxygen Imaging

The CMOS microchip MEA surface was in the original state, unmodified bare Pt. As previously mentioned, each WEs are in the form of a pair of C-shaped electrode covering a 17.5 $\mu\text{m}$ ×17.5 $\mu\text{m}$  area, with 27.5 $\mu\text{m}$  pitch adjacent to surrounding WEs in X and Y directions. Figure 67 illustrates the 2.5 $\mu\text{m}$  wide tracks encloses individual WEs are connected to RE, where one of them was initially the AE (Figure 24). While, the AE is the ITO-coated glass electrode embedded in the microfluidic PMMA cover, which was initially used for sensing pH. This configuration offers improved spatial resolution by ensuring localized

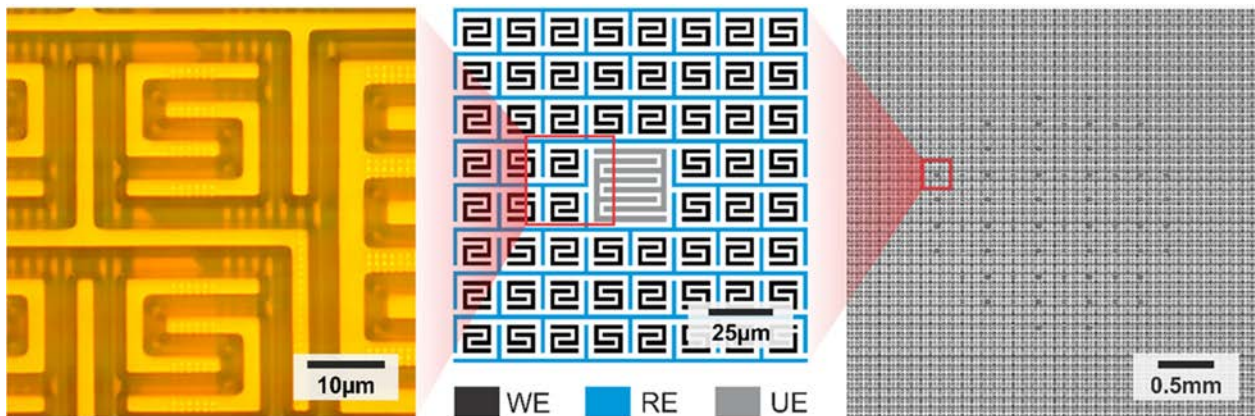


Figure 67. CMOS microchip MEA setup for oxygen imaging.



redox cycle occurring in the area above each WE without any interference from nearby WE, while increasing the sensitivity to  $pO_2$  by approximately 3 $\times$ .

### 8.1.2. Bovine Cumulus-Oocytes-Complexes Handling and Preparation

Bovine ovaries were obtained within 30 minutes of post-mortem from a local slaughterhouse. After collection, ovaries were rinsed with room temperature saline solution and transported in the same solution in an insulated container to the Equine Reproduction Laboratory within one hour. At the laboratory, ovaries were rinsed three times with sterile saline and held in the same solution at room temperature. COCs were aspirated from follicles ranging between 3 to 8 mm in diameter with a 6 mL syringe and an 18-ga sterile needle, identified with a stereomicroscope at 7 $\times$  to 10 $\times$  magnification, picked up using a micropipette, and moved to media, a mixture of G-MOPS™ (Vitrolife, Englewood, CO) handling media and 0.4% bovine serum albumin (BSA) (Sigma, St Louis, MO). The COCs were moved to a 1.5mL microtube filled with media and kept in an insulated container in an incubator (38.5°C) until imaging experiment. COC was selected for

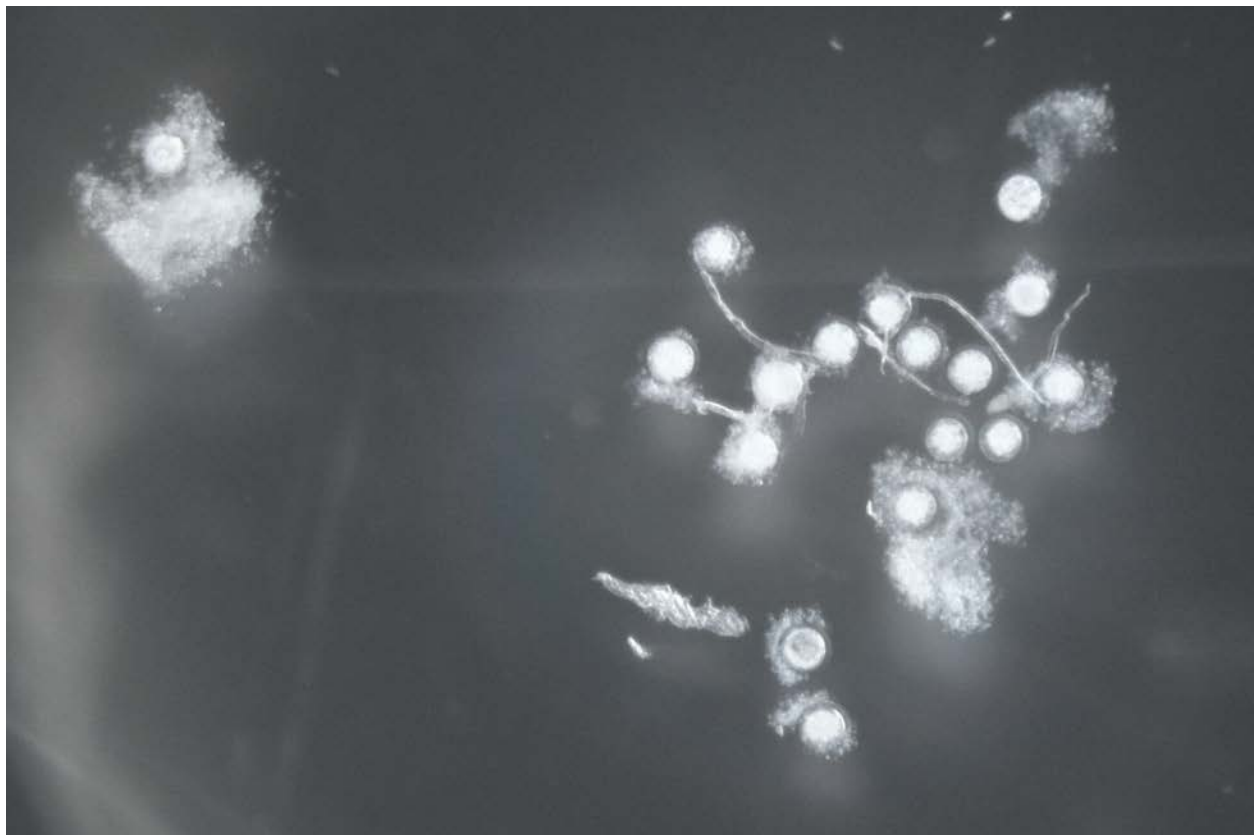


Figure 68. A group of COCs in a 4-well dish.



the assays based on the homogeneity of cytoplasm and compact cumulus cells, as previously described [123]. All pO<sub>2</sub> imaging experiment was performed within six hours from the COC extraction process from the ovaries.

### 8.1.3. Experiment Setups

Oocytes in vitro experiments require a regulated temperature of 38.5°C [124]. The system was calibrated to the ambient room temperature to provide 38.0±0.5°C at the MEA surface. To reach the MEA surface temperature to a stable point, the CMOS microchip is powered-up for the pre-heat phase for around

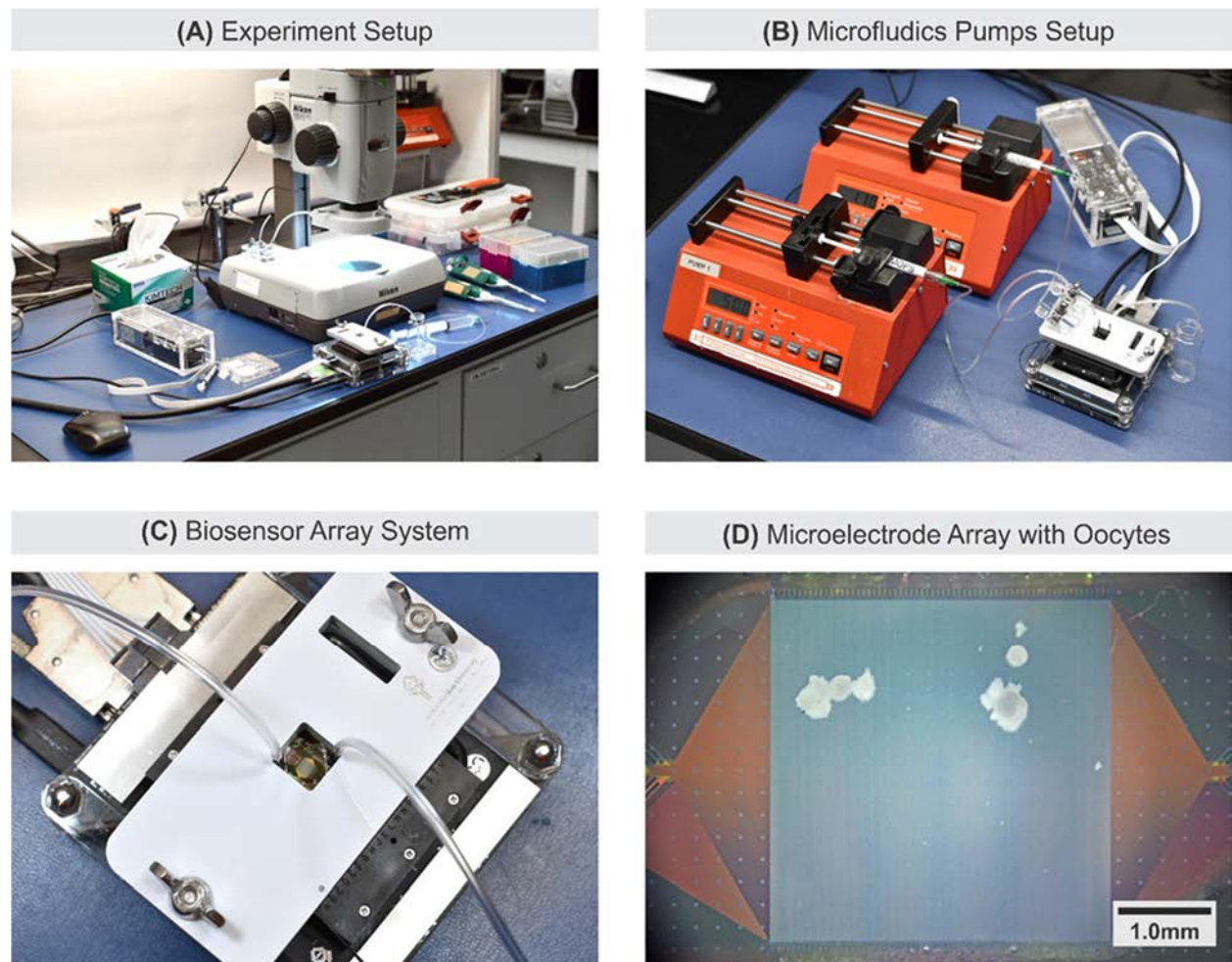


Figure 69. Oxygen imaging in vitro experiment setup.

(A) Typical In vitro experiment setup employing a stereomicroscope. (B) A setup employing two syringe pumps for simultaneous delivery of two types of solutions. (C) A close-up system setup showing the clear microfluidic PMMA cover with integrated ITO-coated glass, and an opening in the pressure plate allowing optical imaging of the MEA area. (D) An image from the stereomicroscope depicting loaded COCs to the MEA surface.



30 minutes before the first experimental run. Figure 69A shows a typical arrangement of the  $pO_2$  imaging experiment. Figure 69B shows another setup variation with two inlets and one outlet microfluidic support system, and each inlet is being driven by a syringe pump. With the two-inlet configuration, the user can configure different chemical solution delivery at various flow rates within one experiment. During each  $pO_2$  imaging experiments, COCs are moved to 4-well dishes (Part # 179830, Nunc™) filled with the media as shown in Figure 68. The storing dish is covered with the accompanying dish lid to allow gas exchange, placed in temperature-controlled warming plate, and left for approximately 15 minutes to allow the media  $pO_2$  to reach the atmospheric saturation point. A 1mL syringe with 1/32" inside diameter tubing (ND 100-65, Tygon®) is used to extract the COCs from storing dish to be injected into the initially dry microfluidic inlet. To retrieve the COCs, withdrawal of the syringe is performed using the syringe pump or manually. Figure 69C shows a top view of the system during the experiment. The transparent ITO AE and opening on the pressure plate provides an opening for the stereomicroscope to optically capture the entire MEA surface. Figure 69D depicts a view of COC positioned over the MEA area, ready for  $pO_2$  imaging.

## **8.2. Oxygen Sensing Results**

### **8.2.1. Electrochemical Responses of Oxygen**

To imitate low  $pO_2$ , media was degassed in a vacuum chamber and sparged with nitrogen to reach  $9.3\mu M$   $pO_2$ , approximately 1% of the maximum 20% oxygen concentration in ambient. The oxygen-depleted media then was gassed with oxygen-rich air to reach higher  $pO_2$ . Once the media reaches desirable  $pO_2$ , a small volume sample was quickly stored to a syringe to conserve the  $pO_2$  level. A commercial oxygen meter (Oakton DO6+, Cole-Parmer) was used to confirm seven media  $pO_2$  samples ranging from  $9.3\mu M$  to  $165.6\mu M$ , correspond to approximately 1% to 20% of ambient oxygen concentrations. The flow injection using two-inlet and one-outlet microfluidic channels were employed to switch between two different  $pO_2$  samples. The experiments were performed at  $37.0 \pm 0.5^\circ C$ , 5,003 feet (1,525m) altitude, and 632 mmHg atmospheric pressure.



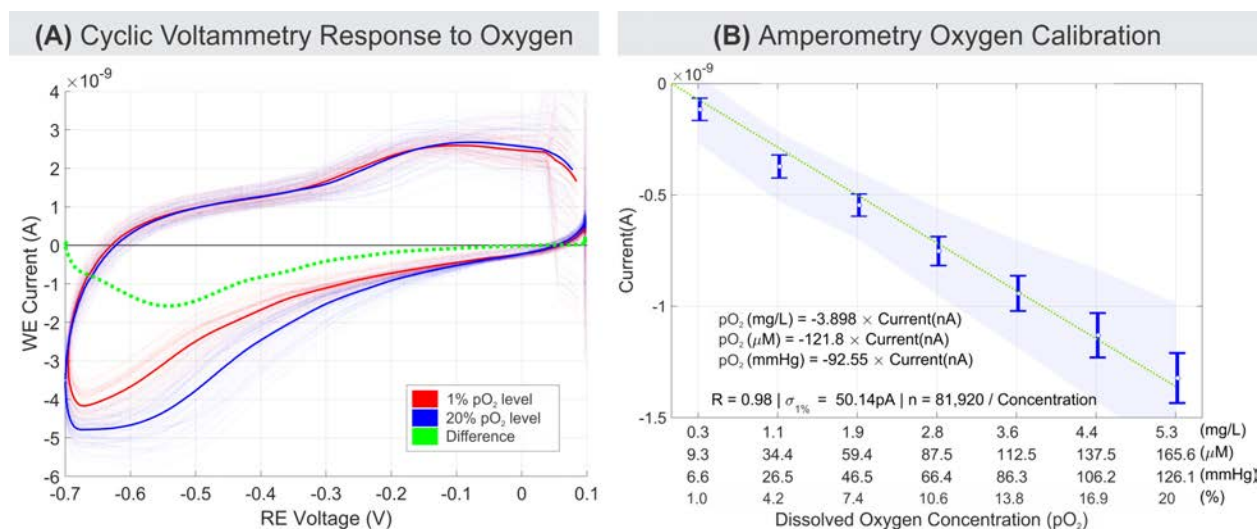


Figure 70. Electrochemical response to oxygen in an improved setup.

(A) Cyclic voltammetry comparison between media with 1% to 20% pO<sub>2</sub> at 10V/s scan rate over 64 selected WEs. All 64 channel is plotted as overlapping opaque curves and their mean is plotted as solid curves. (B) Amperometry sensitivity response of oxygen in media for 1% to 20% pO<sub>2</sub>. The error bar and the shaded area signify 1- $\sigma$  and 3- $\sigma$  values, respectively.

In three-electrode configuration, the electrical current is proportional to the amount of electrochemical reactions that occur at the surface of WE [38]. Assuming the direct four-electron reduction, four electrons are consumed to reduce each oxygen molecule into a water molecule [125]. Therefore, most electrical current generated by the reduction process can be represented as oxygen reduction. Previous publications confirm that the common reduction potential of oxygen occurs when the potential of WE vs RE is around -0.5V to -0.6V [5], [126], [127]. Our CV experiment confirmed oxygen reduction occurs around -0.55V as depicted in Figure 70A. These CV results were generated from flow injection experiment with media samples with 9.3 $\mu$ M and 165.6 $\mu$ M pO<sub>2</sub>. Figure 70B shows the sensitivity curve showing all seven pO<sub>2</sub> media solutions. The curve shows a good linear response ( $R = 0.98$ ) across all 16-thousands electrode with 121.8 $\mu$ M/nA sensitivity. The LoD can be approximated using the 3 $\sigma$  value of the 9.3 $\mu$ M pO<sub>2</sub> readings, which is 150.4pA, 18.3 $\mu$ M, 0.58mg/L, or 13.8mmHg. All amperometry pO<sub>2</sub> imaging result presented in this paper used -0.6V reduction voltage, accounting for variations of reduction voltage shifts due to the variability of the WEs geometry and conditions.



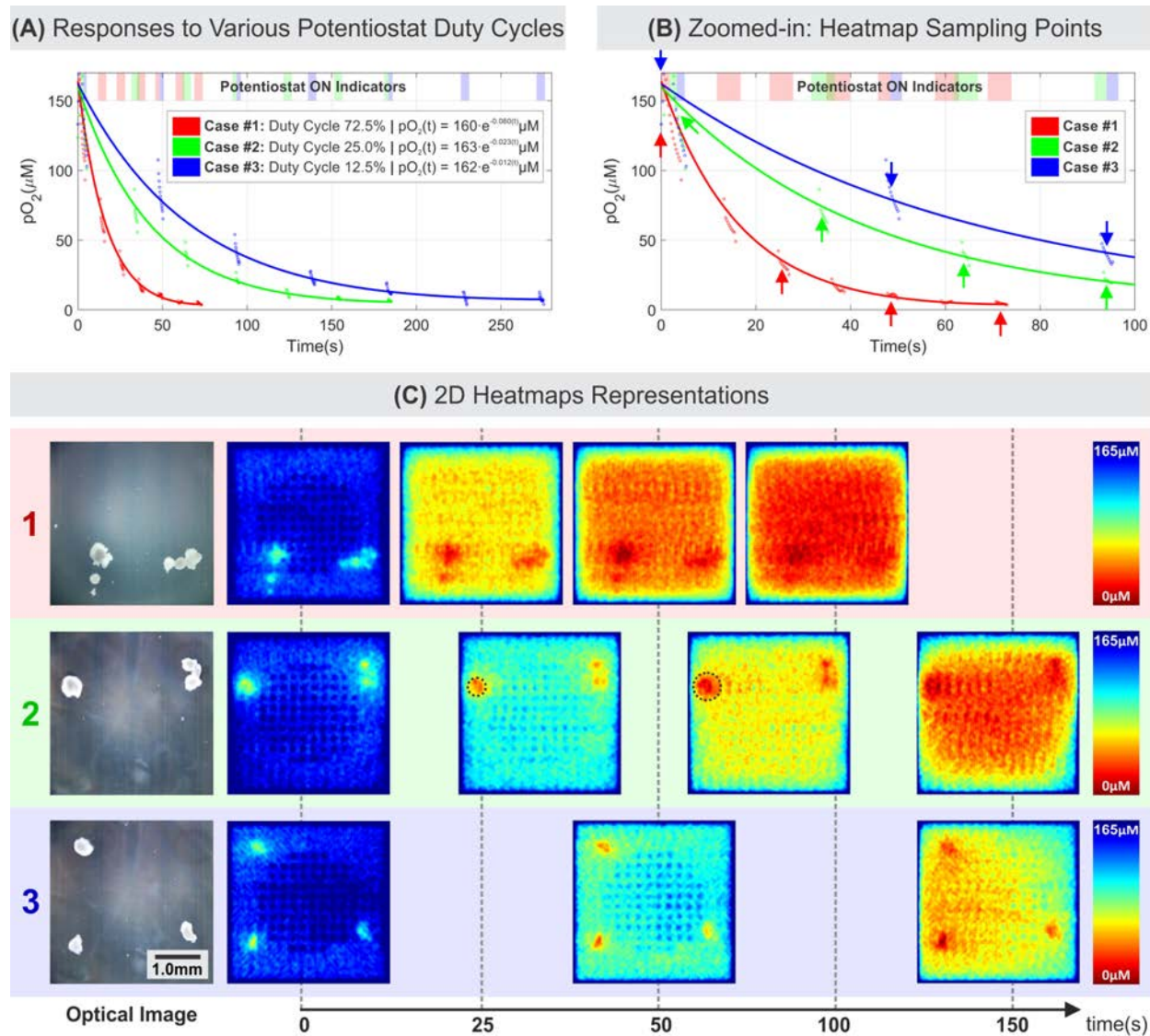


Figure 71. Oxygen reduction rate at three different ratios of sample and rest period.

(A) and (B) The oxygen reduction rate of the sensor at three different ratios of sample and rest period. (C) The corresponding 2D heatmap of three different cases of the *in vitro* experiment. Each heatmap corresponds to a 2D heatmap frame selected at a time point as pointed by the arrows in (B).

### 8.2.2. COC Basal Respiration and the Effect of WEs Oxygen Reduction

Three experiments were carried out using different potentiostat duty cycles to better understand oxygen reduction dynamics of the MEA enclosed in the microfluidic chamber. The potentiostat duty cycle refers to the duration ratio when the potentiostat is turned on ( $V_{RE} = -0.6V$ ) or when the electrochemical



reduction of occurs. Meanwhile, the potentiostat suppresses all electrochemical reaction when it is turned off ( $V_{RE} = 0V$ ).

The As plotted in Figure 71A and B, each case of a given potentiostat duty cycle consists of seven sampling segments indicated by the shaded area when the potentiostat is turned on. Each segment is a group of up to 10 data points, and each point is the average readings of all 16-thousand WEs signals at one point in time. The saturation point, where the oxygen is mostly reduced, occurs around 75s, 135s, and 275s, for cases #1 to #3, respectively. The fastest rate of oxygen reduction due to active WEs was the case #1 (72.5% duty cycle), while the slowest oxygen reduction was the case #3 (12.5% duty cycle). These reduction rate differences indicate that the WEs oxygen reduction only occurs during the potentiostat on periods ( $V_{RE} = -0.6V$ ). However, the amount of the reduced oxygen is similar for cases #1 to #3. Exponential models were fitted to each case and the amount of charge measured on each WE can approximated to be 8.37nC, 7.27nC, and 7.02nC for cases #1 to #3, respectively. The values are within a sensible range to the theoretical amount of charge required by the four-electron reduction of 165 $\mu$ M oxygen in the 3.9 $\mu$ L chamber within a single WE, which is approximated to be 15.5nC. The calculation uses microfluidic channel cross section volume of 3.9 $\mu$ L and  $pO_2$  at local atmospheric pressure of 165 $\mu$ M = 165 $\mu$ mol/L. The available  $pO_2$  in the microfluidic chamber is 165 $\mu$ mol/L  $\times$  3.9 $\mu$ L = 0.6435 $\cdot 10^{-9}$ mol, where 1 mol = 6.022 $\cdot 10^{23}$ molecules and  $1e^- = 1.602\cdot 10^{-19}C$  or  $1C = 6.2415\cdot 10^{18}C$  negatively charge electron ( $e^-$ ). Therefore, the number of oxygen molecules is 0.6435 $\cdot 10^{-9}$ mol  $\times$  6.022 $\cdot 10^{23}$ molecules = 3.875 $\cdot 10^{14}$ molecules. Assuming the reduction of oxygen is based on four-electron process, where  $O_2 + 4H^+ + 4e^- \rightarrow 2H_2O$ , the total charge consumed (in coulomb) as a result of oxygen reduction is integral for the exponential models:  $(-0.7\cdot 10^{-9}\cdot e^{-0.06(t)})nA$  for  $t = 0$  to 75s,  $(-0.7\cdot 10^{-9}\cdot e^{-0.23(t)})nA$  for  $t = 0$  to 135s,  $(-0.7\cdot 10^{-9}\cdot e^{-0.012(t)})nA$  for  $t = 0$  to 275s, for case #1 to #3, respectively.

The 2D heatmap representation of these three cases are mapped and shown in Figure 71C, where each heatmap corresponds to the data pointed by arrows shown in Figure 71B. The heatmaps revealed that the data were generated from three in vitro experiments involving similar combined total area of the COCs. The level of oxygen reduction is homogenous across the MEA surface, except where the COC and UEs are located, and at the MEA edges. The electrically inactive 80 UEs in the MEA central area create



spots where less oxygen reduction occurs. Thus, it reflected as higher  $pO_2$  readings approximately covers  $4 \times 4$  electrodes ( $110 \times 110 \mu m$ ) area around each UEs. A similar pattern also exists along the edge of the MEA. Media with abundant  $pO_2$  continuously diffuses in from the area outside MEA, where no oxygen reduction occur.

The sequences of the oxygen heatmaps in Figure 71C indicates the capability of the biosensor system in mapping 2D  $pO_2$ , the degree of oxygen reduction by the MEA, and the COC's respiration dynamics over time. In summary, first, the potentiostat should not be turned on continuously for an extended period given the  $3.9 \mu L$  volume above the MEA as indicated by case #1. One can extend the potentiostat duty cycle by increasing the microfluidic PDMS channel thickness, thus increasing the chamber volume. However, a larger volume may reduce the measurement sensitivity. Second, when an appropriate potentiostat duty cycle was used (e.g. case #2), the COC respiration over time is captured as the oxygen

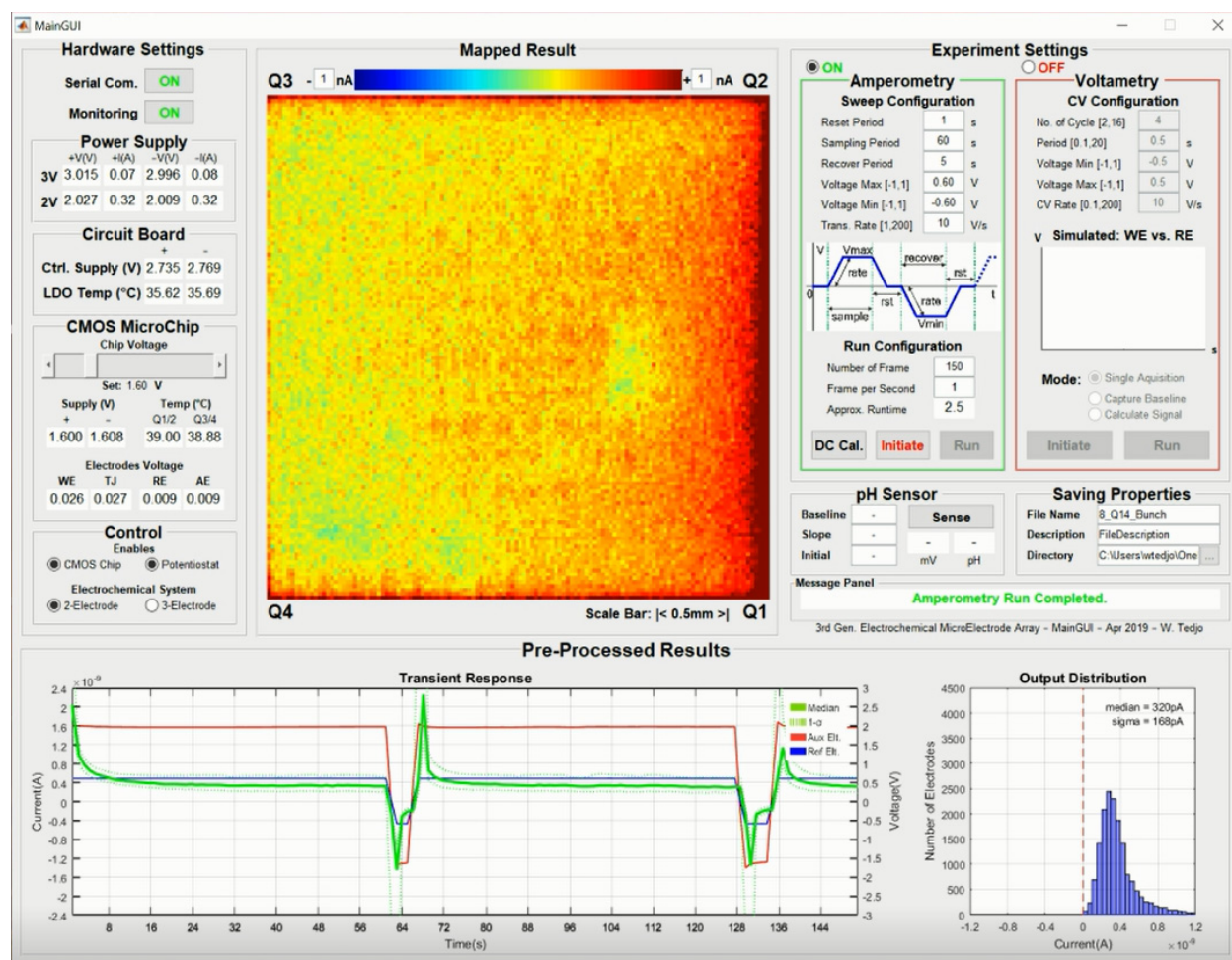


Figure 72. The modified real-time GUI showing the sample and rest periods.



reduction around the COC progressively expands outward over time as shown by dotted black circles in the case #2 in Figure 71C. At last, the potentiostat duty cycle should not be too small that it loses the temporal resolution to capture respiration dynamics as shown by case #3. Therefore, to reliably measure OCR of target cells during a desired period, the potentiostat duty cycle needs to be adjusted on a case-by-case basis to fully explore temporal characteristics of target cells.

Figure 72 shows, a screenshot of the custom GUI during an actual oxygen reading of bovine COCs. The GUI provides the user with various kind of control, monitoring, and selections of electrochemical

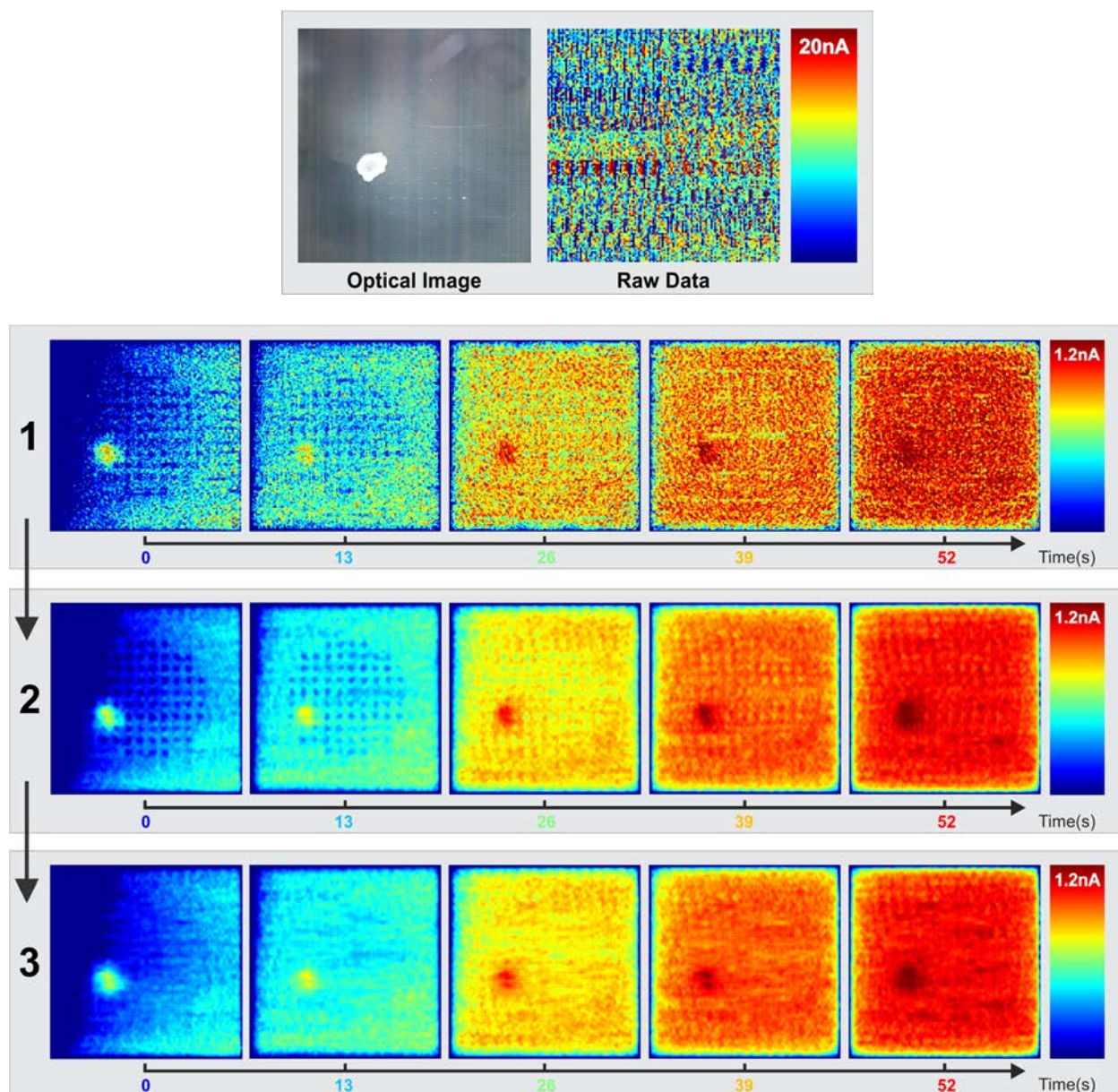


Figure 73. Steps of heatmap spatial and temporal smoothing.



analysis methods, amperometry or voltammetry. In this specific experiment, the GUI shows real-time heatmap representation, transient response, and instantaneous histogram output at 1Hz update rate for a duration of 2.5 minutes. The amperometry mode were set to have 60s sampling ( $V_{RE} = -0.6V$ ), 1s reset ( $V_{RE} = 0V$ ), and 5s recover ( $V_{RE} = 0.6V$ ).

### **8.2.3. Heatmap Smoothing**

To increase accuracy in COC oxygen consumption analysis, correction and smoothing step was performed to remove the UEs artifacts. The following section demonstrates steps of spatial data processing on amperometry experiment dataset that was used to improve the data quality before further data analysis should be performed. As shown in Figure 73, the raw data or the uncalibrated DC offset result shows no correlation to the optical image. Step 1 shows the results after performing the baseline offset calibration of the raw data as previously discussed in Chapter VI. The baseline offset is due to the electrical circuit variations discussed in Chapter IV, which are constant signature values from each 16 thousand read channels with no noticeable drift over tens on minutes and tens of °C temperature variations. A COC and UEs patterns become clearly visible in this step. Step 2 performs spatial and temporal smoothing. The spatial smoothing is in a form of simple vertical and horizontal moving averaging with a 3-pixel window, and the temporal smoothing is exclusively individual to each pixel, over time. Step 3 removes the artifact due to UEs. The pixels at and surrounding UEs are detected and replaced by the pixels' values in nearby proximity.

### **8.2.4. Comparison of Healthy and Dead COC**

The microfluidic support system offers an ability to restore the  $pO_2$  to a saturated condition after each imaging experiments. This section discusses attempts to maintain oxygen-rich media flow during imaging in healthy and dead COC. The dead cells were previously exposed to media with 1% Triton™X-100 (Sigma, St Louis, MO) for 15 minutes, rinsed, and placed back to the oxygen-rich media. For both cases depicted in Figure 74, the experiments were started with an injection of cells through the microfluidic inlet. Once the cells were positioned on the MEA surface, as visualized in Figure 74A(i) and B(i), a continuous stream of 40 $\mu$ L/min of fresh media was held constant using a syringe pump. Any fluidic stream



rate above 40 $\mu$ L/min would move the cells out of the MEA area. Results presented in Figure 74 focus on the pO<sub>2</sub> readings during the transition from 40 $\mu$ L/min to 0 $\mu$ L/min microfluidic flow, indicated as t = 0s.

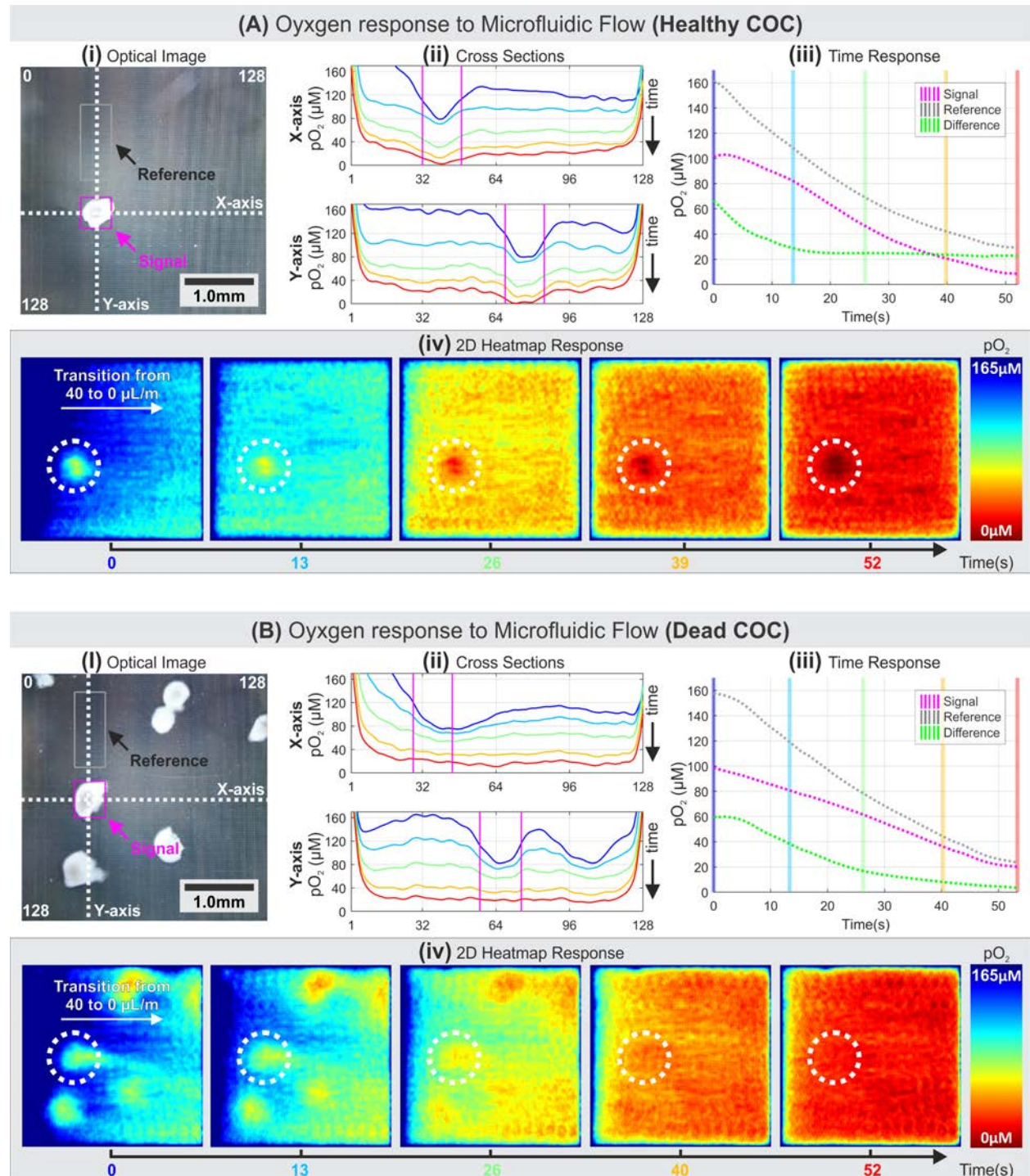


Figure 74. Effects of pO<sub>2</sub> imaging after microfluidic flow.

pO<sub>2</sub> imaging after a transition from 40 $\mu$ L/min to 0  $\mu$ L/min microfluidic flow with (A) healthy a COC and (B) dead COCs. The white dotted circles compare the pO<sub>2</sub> imaging response between the healthy and dead COC.



To better analyze the COC signal, a cross-section in X and Y-axes of a select COC are plotted over time (Figure 74A(ii) and B(ii)), with the blue to red curves signify progression over time. Using the optical image as a reference,  $pO_2$  activities can be seen occurring at and around the cells. Therefore, further analysis can be done by focusing on electrodes located beneath the cells. A magenta box outline is drawn around a COC in question and their average readings are being used to show signal measurement values over time, while the grey box is a selected group of electrodes that represent background reduction current or reference measurement value. The plot of signal and reference curves over time is shown in Figure 74A(iii) and B(iii) with each vertical line, color-coded from blue to red, corresponds to each curve on the X and Y-axis cross-section plot. Finally, a 2D heatmap representation of each corresponding timestamp is shown accordingly Figure 74A(iv) and B(iv).

The results in Figure 74 can be summarized in three-fold: first, in the healthy COC  $pO_2$  imaging, the measurement data from the cross-section in both X and Y-axes (Figure 74A(ii)) show that COC had decreasing oxygen concentration towards its center. It agrees to a previous statement that the surrounding bovine cumulus cells are known to actively utilize glucose and exhibit little oxidative metabolism, whereas the oocytes at the center actively utilize OXPHOS and require only limited glucose metabolism [128]. Second, the experiments show distinct responses of  $pO_2$  between healthy and dead COCs as depicted in Figure 74A(iii) and B(iii), respectively. In the healthy COC case, the  $pO_2$  signal reading was initially high then decreasing at a similar rate as the background oxygen reduction, while maintaining a constant difference of  $\sim 20\mu M$   $pO_2$  for  $t > 15s$ . The  $\sim 20\mu M$  difference between the signal and the reference suggests metabolic respiration by the healthy COC. Meanwhile, the dead COC case shows a trend where the signal approaching the reference within 52s, indicating the absence of active oxygen consumption. Third, as depicted in Figure 74A(iv) and B(iv), the 2D heatmaps show gradients of oxygen-rich media that were initially injected from the left side of the MEA and stopped at  $t = 0s$ . In an agreement to the first point, the signal at the healthy COC shows a visible oxygen consumption in contrast with the surrounding area over the entire period of the experiment. Another observation can be made from Figure 74B(iv) that the dead COCs physically disturbed the initial dispersion of the oxygen-rich media after the microfluidic flow was stopped at  $t = 0s$ , which was reflected as  $pO_2$  readings resembling the exact location of the COCs. However, the signal of dead COCs appears to diminish as it approaches  $t = 52s$ . Therefore, to obtain accurate



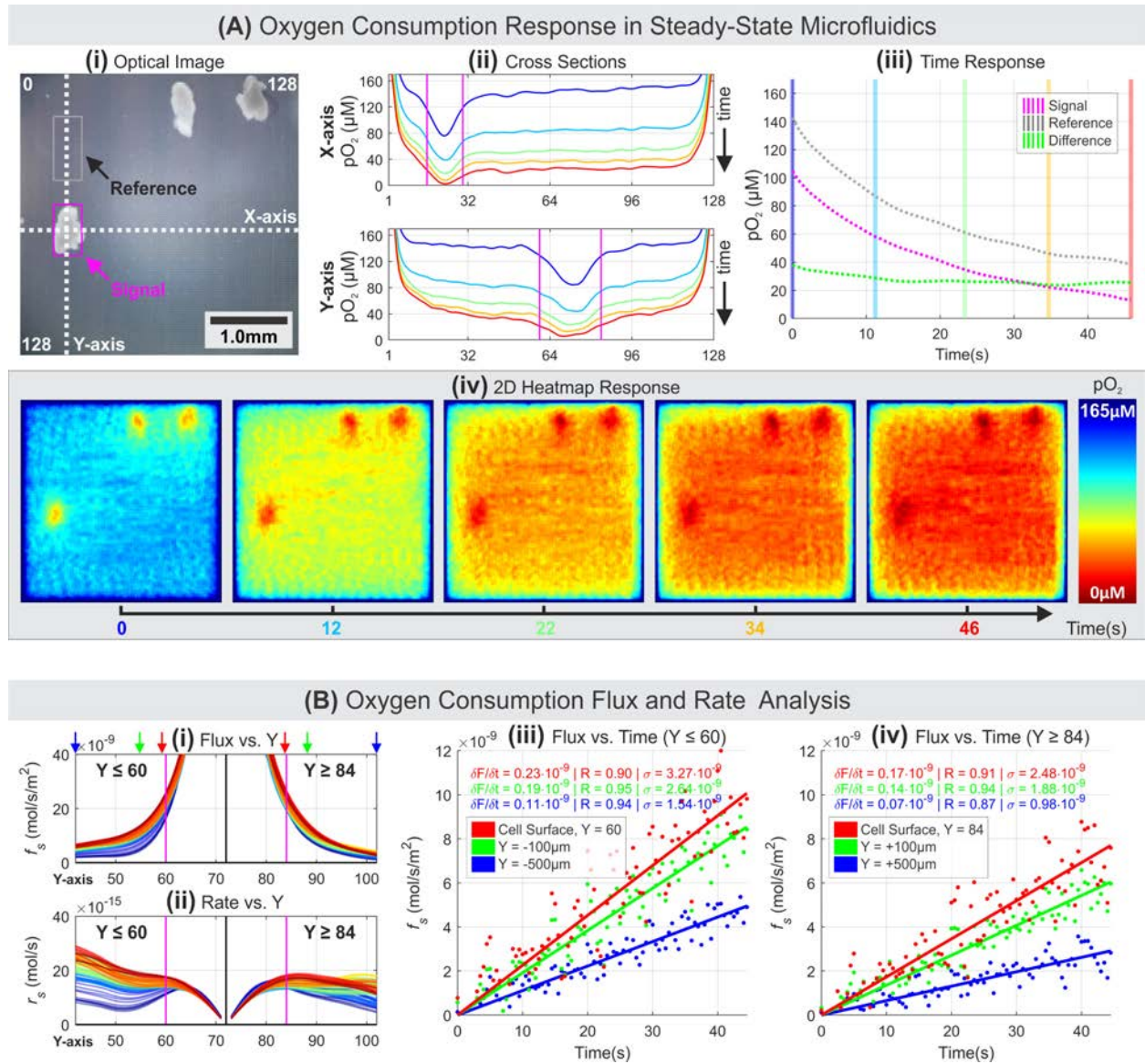


Figure 75. Oxygen gradient imaging in static media of a healthy COC.

(A) Oxygen gradient imaging in static media of a healthy COC. (B) Analysis of oxygen consumption flux and rate based on the Y-axis cross-section at multiple distances over a period. Flux scatter plots in (iii) and (iv) corresponds to Y locations at the surface of COC, 100 $\mu$ m, and 500 $\mu$ m away to either side of the COC, with each Y locations pointed by arrows in (i).

measurement data, the pO<sub>2</sub> imaging and analysis should be performed when the microfluidic is at static media (zero fluidic flow) or when the effect of the microfluidic flow or physical block is already minimized (i.e.  $t > 25$ s).



### 8.2.5. Flux Density and Consumption Rate Analysis

OCR is essential in determining the viability of oocytes. A set of healthy COCs in static media were used to calculate oxygen flux and consumption rate. Figure 75A illustrates an expected trend of oxygen consumption of healthy COCs, comparable to pO<sub>2</sub> responses in Figure 71 and Figure 74A. The flux and rate of oxygen consumption are based on the selected Y-axis cross-section shown in Figure 75A(ii). We use the spherical diffusion theory that was previously developed and described for analyzing oocyte/embryo cells [5]. Flux density (mol/s/m<sup>2</sup>) of pO<sub>2</sub> can be described as:

$$f_s = D\Delta C/x \quad (22)$$

where  $D$  (m<sup>2</sup>/s) is the diffusion coefficient of oxygen in water, approximated at  $2.1 \cdot 10^{-9}$  m<sup>2</sup>/s,  $x$  (m) is the distance or radius of a spherical sample, and  $\Delta C$  (mol/m<sup>3</sup>) is the concentration difference between the concentration at  $x$  and the bulk concentration (saturated pO<sub>2</sub>), which can be derived from the electrical current reading from each individual pixel and the calibration curve. The bulk concentration is calculated from the bottom 5% quantile electrical current values from the reference box at  $t = 0$ s, representing the highest pO<sub>2</sub> during an experiment. Hence, following the spherical diffusion theory, the oxygen consumption rate (mol/s) by a half-spherical sample can be described as:

$$r_s = S \cdot f_s = (2\pi x^2) \cdot D\Delta C/x = 2\pi x D\Delta C \quad (23)$$

where  $S$  is the surface area of a half-spherical sample at a distance  $x$ .

In this specific experiment result, the center of a COC was selected from the Y-axis cross-section at  $Y = 72$ , signifies the 72<sup>nd</sup> electrodes from the top-left origin coordinate. Where the cell's surface boundary, marked by magenta colored lines in Figure 75A(i) and (ii), was selected at  $Y = 60$  and  $Y = 84$  for either sides. Equations (1) and (2) were used to generate the flux and rate plots up to 18 electrodes (500μm) away from the cell's surface, or 30 electrodes (825μm) away from the cell's center at  $Y = 72$ . Figure 75B(i) and (ii) show oxygen flux and consumption rate over 500μm, to either side away from the cell's surface.



The curves in Figure 75B(i) and (ii) are color-coded to show different timestamp, the blue curve for  $t = 0\text{s}$  and red for  $t = 46\text{s}$ , with each curve correspond to a single frame (90 frames in total).

The flux density is approximately  $20\text{nmol/s/m}^2$  at the cell's surface and exponentially decays as it gets farther away from the cell; however, the flux curves increase as a function of time. This trend indicates that the  $\text{pO}_2$  decreases over time within the cell's surface boundary and the lack of oxygen causes the surrounding oxygen to diffuse in, affecting the surrounding  $\text{pO}_2$ . To better illustrate the flux change over time, six intercepts at different Y-axis coordinate, cell surface (red),  $100\mu\text{m}$  from the cell surface (green), and  $500\mu\text{m}$  from the cell's surface (blue) to the left and right sides of the cell are plotted and shown in 8B(iii) and (iv) respectively. Each dot represents the intersect value of the flux curve at the specific Y coordinate. The responses are normalized to 0 at  $t = 0\text{s}$  and linearly fitted to show strong linear correlations ( $R = 87$  and above) for all six cases. The plot expresses the expected trend of increasing flux over time, with the strongest flux growth occurs at the surface of the COC. Finally, the oxygen rate consumption curves at the cell's surface show  $\sim 12\text{fmol/s}$  at  $t = 0\text{s}$  and increases to  $\sim 18\text{fmol/s}$  at  $t = 46\text{s}$ . The high readings towards  $Y \leq 60$  are possibly due to the effect of oxygen consumption by two COCs located around  $Y = 20$ . The measured OCR results are comparable to other previous works using various stages of oocytes and embryos, where their OCRs were measured between 1 to  $30\text{fmol/s}$  [129]–[131].



## CHAPTER IX: CONCLUSION & FUTURE WORKS

A high spatiotemporal resolution electrochemical biosensor system with CMOS microchip employing a novel Pt MEA is presented. Flow injection chemical experiments show that each of the 16,024 WEs in the system can differentiate micromolar concentration of NE with 4.7pA/ $\mu$ M response and -86pA/mg/L for pO<sub>2</sub>. The system's electrochemical performance is further confirmed by the voltammetry performance at 2V/s to 20V/s scanning rate of 0.25mM to 1mM NE concentrations and simultaneous detection of NE and UA. The custom microfluidic device supports pH sensing with linear response sensitivity of -28.35mV/pH. Finally, the electrochemical sensing results of dynamic flow injections are mapped to 2D images captured at four FPS.

Table 8 summarizes the key features of the device and compares them with other existing MEA systems performing electrochemical sensing. Desirable features from a chemical imaging device are high pixel counts, high spatial and temporal resolution, high sensitivity to target analytes, integration of the electronics and electrodes, and integration of other supporting systems (e.g. temperature control, pH sensor, bio-sample handlings). An existing work with the highest spatial resolution was presented by Dragas [45] focused on the electrophysiology performance and shows minimal electrochemical sensing performance with no chemical imaging results. Another work with the highest temporal resolution presented by White [52] successfully performed imaging of neurotransmitter, however, with high a concentration of 10mM Dopamine (DA) and on a 1,024-pixel array. Their results show 0.5nA current response to 10mM DA, while our system shows similar current response for 0.1mM NE. DA and NE are in the same neurotransmitter family with identical electrochemical process of two electrons oxidation; thus, our WE geometry configuration has approximately 100 $\times$  sensitivity to the target analyte in comparison to the results in [52]. This paper presents a system with the highest number of pixels that provides the most comprehensive chemical experiments and verifies the chemical imaging results using multiple target analytes (i.e. NE, UA, and pO<sub>2</sub>) at micromolar concentrations. Furthermore, our system also provides real-time pH monitoring, real-time temperature control, and an improved microfluidic system based on the previous system used in ex vivo experiments [23].



Table 8. Comparison of electrochemical biosensor systems with CMOS MEA.

Para- meters	CMOS		Working Electrode					Read Channel				Imaging Performance				On-Chip	
	Tech	Die Size	Sense Area	Count	Material	Size	Pitch	Count	BW	Input Noise	Power per Ch.	Imaging Result	Frame Rate	Conc. (Analyte)	Pot.	RE/ AE	
	µm	mm	mm	-	-	µm	µm	-	kHz	pA <sub>RMS</sub>	µW	-	FPS	Molar	-	-	
Ref.																	
	[45]	0.18	12×8.9	4.5×2.4	59,760	Pt Black	3×7.5	13.5	28	10	120	178	No	-	200µ (DA)	No	No
	[48]	0.5	-	1.2×1.2	1,024	Au	7×7	37	1	-	-	73.5	No	-	0.1 (KFe)	Yes	No
	[49]	0.5	3×3	0.4×0.2	100	Ti/Pt	15×15	40×20	100	0.1	0.15	-	Yes	200	0.35µ (DA)	Yes	No
	[57]	0.18	10.4×10.4	5.0×5.0	400	Au/Pt/Ti	40	250	400	-	1	1.57	Yes	8	10 (H <sub>2</sub> O <sub>2</sub> )	Yes	No
	[50]	0.6	7.5×7.5	0.6×0.6	256	Au/Cr	10×10	35	1	-	3	430	Yes	1.4	0.1 (KFe)	No	Yes
[51]	0.35	7.5×4.8	3.2×3.2	1,024	Pt Black	25	100	32	1	58.5	-	Yes	90	30m(H <sub>2</sub> O <sub>2</sub> )	Yes	Yes	
[23]	0.6	19×19	2.0×2.0	8,192	Pt	17.5×15	25×30	64	10	52	1,875	Yes	0.03	8µ (NE)	Yes	Yes	
[52]	0.35	5×5	1.0×1.0	1,024	Au	15×15	30	1,024	4.4	0.47	12.2	Yes	10,000	10m (DA)	No	No	
[53]	0.35	3.8×3.8	1.8×1.8	256	Au	20×20	114	16	150	1,040	2,681	Yes	-	100µ (Fe)	Yes	Yes	
[54]	0.18	3×1.4	1.2×0.8	50	Al	60×60	160×120	50	-	4,680	4,900	No	2	-	No	Yes	
[55]	0.18	5×5	3.2×3.2	4,096	Au	45×45	50	4,096	26	0.6	23	No	-	125n (FF)	No	No	
[56]	0.35	3.8×3.1	3.2×2.4	192	Au	100	200	96	1	24	188	No	-	2m (DA)	Yes	Yes	
This work <sup>a</sup>	0.6	19×19	3.6×3.6	16,064	Pt	17.5×17.5	27.5	16,064	5.66	42.8	59.6	Yes	4	4µ (NE)	Yes	Yes	
This work <sup>b</sup>				4,016				4,016	3.56	59.7	39.6						

<sup>a</sup> Vsupply = ±1.65V, Surface Temp. ≈ 37.5°C at 24°C ambient. Four active quadrants. Optimal temperature for ex vivo experiments.

<sup>b</sup> Vsupply = ±1.50V, Surface Temp. ≈ 25.3°C at 24°C ambient. One active quadrant. Minimum supply potential with no significant performance reduction. Analytes: DA: dopamine, KFe: Potassium ferricyanide (K3Fe(CN)6), NE: Norepinephrine, Fe: ferrocene (Fe(C5H5)2), FF: Ferro/Ferricyanide.



The system presented in this paper offers all-around chemical imaging capability prepared for performing biological-significant studies. Applications of such system include the expansion of existing works in single-cell viability in Assisted Reproduction Technology (ART) [5], [30], rapid cytometry [66], [132], and cellular physiology related to degenerative diseases and cancer developments [24], [133].

As the system core, the versatility of the CMOS MEA microchip allows extensive feature expansions and improvements at the board and system level. First, a custom data acquisition design, similar to previously reported work [51], [52], would allow the system to perform higher temporal resolution chemical imaging of all 16,064 read channels up to 244FPS from the existing 4FPS. Second, the 80 pairs of general-purpose UE are available for incorporating additional functionality such as localized electrical stimulations and impedance sensing. Finally, the bare Pt MEA provides opportunities for further improvement in sensitivity and selectivity with enhanced surface modifications, such as gas permeable coated electrodes for oxygen detection [3], [4], enzyme-based coating for glucose and lactate sensing [6]–[8], or polymer-based coating for improved simultaneous detection of various neurotransmitters [95]–[97].

Among many potential applications, the biosensor system was used to demonstrate its capabilities in imaging the COCs oxygen respiration. A custom microfluidic setup was optimized for handling COCs above 50 $\mu$ m in diameter. Multiple in vitro experiments using bovine COC validate strong correlations between optical image and the 2D generated heatmap based on the pO<sub>2</sub> gradients. The implications of constant electrochemical reduction within a small microfluidic chamber was presented, and possible remedies to the effects were discussed. Finally, flux density and rate of the oxygen consumption of a COC was analyzed in 27.5 $\mu$ m spatial resolution and 4Hz temporal resolution. The high spatial and temporal resolution images of oxygen consumption provide information of the cell's basal respiration that has not been previously generated by other existing methods.

Future studies will aim to understand and address the current hardware, software, chemical, and biological integration challenges. Further considerations in the chemistry of the electrode-tissue interface are essential, such as the effects of diffusion in live three-dimensional tissue in the enclosed microfluidic system, as well as surface coatings and non-destructive cleaning protocols to minimize biofouling. Biological considerations of tissue viability, biocompatibility, and sustainable environmental control are also critical to increase the consistency and duration of experiments. The system would benefit from electrode



surface modifications to optimize selectivity to oxygen [4], [7], [8], to sense of other metabolizing agents (glucose and lactose) [36], [37] and to enable simultaneous neurotransmitter detections [95], [134]. In combinations with advanced microfluidic devices [92], [93], the improvements would enable the biosensor system to monitor a broader metabolic panel with high sensitivity and selectivity at high spatial and temporal resolution. These improvements could significantly benefit rapid cell screening in ART or long-term cell culturing in non-communicable diseases studies.



## REFERENCES

- [1] S. Shashkova and M. C. Leake, "Single-molecule fluorescence microscopy review: shedding new light on old problems," *Biosci. Rep.*, vol. 0, no. July, p. BSR20170031, 2017.
- [2] K. D. Wegner and N. Hildebrandt, "Quantum dots: bright and versatile in vitro and in vivo fluorescence imaging biosensors," *Chem. Soc. Rev.*, vol. 44, no. 14, pp. 4792–4834, 2015.
- [3] J. Park, Y. K. Pak, and J. J. Pak, "A microfabricated reservoir-type oxygen sensor for measuring the real-time cellular oxygen consumption rate at various conditions," *Sensors Actuators, B Chem.*, vol. 147, no. 1, pp. 263–269, 2010.
- [4] C. C. Wu, H. N. Luk, Y. T. T. Lin, and C. Y. Yuan, "A Clark-type oxygen chip for in situ estimation of the respiratory activity of adhering cells," *Talanta*, vol. 81, no. 1–2, pp. 228–234, 2010.
- [5] K. Hiramoto *et al.*, "Development of Oxygen Consumption Analysis with an on-Chip Electrochemical Device and Simulation," *Anal. Chem.*, vol. 89, no. 19, pp. 10303–10310, 2017.
- [6] A. F. Revzin, K. Sirkar, A. Simonian, and M. V. Pishko, "Glucose, lactate, and pyruvate biosensor arrays based on redox polymer/oxidoreductase nanocomposite thin-films deposited on photolithographically patterned gold microelectrodes," *Sensors Actuators, B Chem.*, vol. 81, no. 2–3, pp. 359–368, 2002.
- [7] Y. M. Obeidat, A. J. Evans, W. Tedjo, A. J. Chicco, E. Carnevale, and T. W. Chen, "Monitoring oocyte/embryo respiration using electrochemical-based oxygen sensors," *Sensors Actuators B Chem.*, vol. 276, no. February, pp. 72–81, Dec. 2018.
- [8] Y. Obeidat *et al.*, "A multi-sensor system for measuring bovine embryo metabolism," *Biosens. Bioelectron.*, vol. 126, no. September, pp. 615–623, Feb. 2019.
- [9] F. Bedioui and N. Villeneuve, "Electrochemical nitric oxide sensors for biological samples—Principle, selected examples and applications," *Electroanalysis*, vol. 15, no. 1, pp. 5–18, 2003.
- [10] B. J. Privett, J. H. Shin, and M. H. Schoenfisch, "Electrochemical nitric oxide sensors for physiological measurements," *Chem. Soc. Rev.*, vol. 39, no. 6, p. 1925, 2010.
- [11] H. M. Jafari, K. Abdelhalim, L. Soleymani, E. H. Sargent, S. O. Kelley, and R. Genov, "Nanostructured CMOS wireless ultra-wideband label-free PCR-free DNA analysis SoC," *IEEE J. Solid-State Circuits*,



- vol. 49, no. 5, pp. 1223–1241, 2014.
- [12] D. Jambrec, K. Lammers, T. Bobrowski, S. Pöller, W. Schuhmann, and A. Ruff, “Amperometric Detection of dsDNA Using an Acridine-Orange-Modified Glucose Oxidase,” *Chempluschem*, vol. 82, no. 11, pp. 1311–1314, 2017.
  - [13] E. S. Bucher and R. M. Wightman, “Electrochemical Analysis of Neurotransmitters,” *Annu. Rev. Anal. Chem.*, vol. 8, no. 1, pp. 239–261, Jul. 2015.
  - [14] B. Si and E. Song, “Recent Advances in the Detection of Neurotransmitters,” *Chemosensors*, vol. 6, no. 1, p. 1, 2018.
  - [15] C. A. Thomas, P. A. Springer, G. E. Loeb, Y. Berwald-Netter, and L. M. Okun, “A miniature microelectrode array to monitor the bioelectric activity of cultured cells,” *Exp. Cell Res.*, vol. 74, no. 1, pp. 61–66, Sep. 1972.
  - [16] A. Stett *et al.*, “Biological application of microelectrode arrays in drug discovery and basic research,” *Anal. Bioanal. Chem.*, vol. 377, no. 3, pp. 486–495, 2003.
  - [17] M. Fejtl, A. Stett, W. Nisch, K. H. Boven, and A. Möller, “On micro-electrode array revival: Its development, sophistication of recording, and stimulation,” *Adv. Netw. Electrophysiol. Using Multi-Electrode Arrays*, pp. 24–37, 2006.
  - [18] M. E. J. Obien, K. Deligkaris, T. Bullmann, D. J. Bakkum, and U. Frey, “Revealing neuronal function through microelectrode array recordings,” *Front. Neurosci.*, vol. 9, no. JAN, p. 423, 2015.
  - [19] W. Pettine, M. Jibson, T. Chen, S. Tobet, P. Nikkel, and C. S. Henry, “Characterization of novel microelectrode geometries for detection of neurotransmitters,” *IEEE Sens. J.*, vol. 12, no. 5, pp. 1187–1192, 2012.
  - [20] J. B. Wydallis *et al.*, “Spatiotemporal norepinephrine mapping using a high-density CMOS microelectrode array,” *Lab Chip*, vol. 15, no. 20, pp. 4075–4082, 2015.
  - [21] R. M. Feeny, J. B. Wydallis, T. Chen, S. Tobet, M. M. Reynolds, and C. S. Henry, “Analysis of Nitric Oxide from Chemical Donors Using CMOS Platinum Microelectrodes,” *Electroanalysis*, vol. 27, no. 5, pp. 1104–1109, 2015.
  - [22] W. Tedjo *et al.*, “Design of an integrated microelectrode array system for high spatiotemporal resolution chemical imaging,” in *2016 IEEE Biomedical Circuits and Systems Conference (BioCAS)*,



2016, pp. 46–49.

- [23] W. Tedjo *et al.*, “Electrochemical biosensor system using a CMOS microelectrode array provides high spatially and temporally resolved images,” *Biosens. Bioelectron.*, vol. 114, pp. 78–88, Aug. 2018.
- [24] G. Solaini, A. Baracca, G. Lenaz, and G. Sgarbi, “Hypoxia and mitochondrial oxidative metabolism,” *Biochim. Biophys. Acta - Bioenerg.*, vol. 1797, no. 6–7, pp. 1171–1177, 2010.
- [25] L. D. Osellame, T. S. Blacker, and M. R. Duchon, “Cellular and molecular mechanisms of mitochondrial function,” *Best Pract. Res. Clin. Endocrinol. Metab.*, vol. 26, no. 6, pp. 711–723, 2012.
- [26] V. Mirabello, F. Cortezon-Tamarit, and S. I. Pascu, “Corrigendum: Oxygen Sensing, Hypoxia Tracing and in Vivo Imaging with Functional Metalloprobes for the Early Detection of Non-communicable Diseases,” *Front. Chem.*, vol. 6, no. February, 2018.
- [27] K. L. Eales, K. E. R. Hollinshead, and D. A. Tennant, “Hypoxia and metabolic adaptation of cancer cells,” *Oncogenesis*, vol. 5, no. 1, pp. e190–e190, 2016.
- [28] R. G. Jones and C. B. Thompson, “Tumor suppressors and cell metabolism,” no. 514, pp. 1–13, 2009.
- [29] S. L. Dunwoodie, “The Role of Hypoxia in Development of the Mammalian Embryo,” *Dev. Cell*, vol. 17, no. 6, pp. 755–773, 2009.
- [30] L. Scott, J. Berntsen, D. Davies, J. Gundersen, J. Hill, and N. Ramsing, “Human oocyte respiration-rate measurement - Potential to improve oocyte and embryo selection?,” *Reprod. Biomed. Online*, vol. 17, no. 4, pp. 461–469, 2008.
- [31] I. Donnay, “Metabolic Markers of Embryo Viability,” in *Assessment of Mammalian Embryo Quality: Invasive and non-invasive techniques*, A. Van Soom and M. Boerjan, Eds. Dordrecht: Springer Netherlands, 2002, pp. 57–94.
- [32] P. Vaupel, M. Höckel, and A. Mayer, “Detection and Characterization of Tumor Hypoxia Using pO<sub>2</sub> Histography,” *Antioxid. Redox Signal.*, vol. 9, no. 8, pp. 1221–1236, 2007.
- [33] M. Nordsmark *et al.*, “Prognostic value of tumor oxygenation in 397 head and neck tumors after primary radiation therapy. An international multi-center study,” *Radiother. Oncol.*, vol. 77, no. 1, pp. 18–24, 2005.
- [34] A. Tejera, J. Herrero, M. J. De Los Santos, N. Garrido, N. Ramsing, and M. Meseguer, “Oxygen



- consumption is a quality marker for human oocyte competence conditioned by ovarian stimulation regimens," *Fertil. Steril.*, vol. 96, no. 3, 2011.
- [35] L. D. M. Ottosen, J. Hindkjær, S. Lindenberg, and H. J. Ingerslev, "Murine pre-embryo oxygen consumption and developmental competence," *J. Assist. Reprod. Genet.*, vol. 24, no. 8, pp. 359–365, 2007.
  - [36] Y. Obeidat, M. H. Cheng, G. Catandi, E. Carnevale, A. J. Chicco, and T. W. Chen, "Design of a multi-sensor platform for integrating extracellular acidification rate with multi-metabolite flux measurement for small biological samples," *Biosens. Bioelectron.*, vol. 133, no. February, pp. 39–47, 2019.
  - [37] Y. Obeidat *et al.*, "A multi-sensor system for measuring bovine embryo metabolism," *Biosens. Bioelectron.*, vol. 126, no. November, pp. 615–623, Feb. 2019.
  - [38] A. J. Bard and L. R. Faulkner, *Electrochemical Methods: Fundamentals and Applications*. 2000.
  - [39] F.-G. Bănică, *Chemical Sensors and Biosensors*. Chichester, UK: John Wiley & Sons, Ltd, 2012.
  - [40] J. Graeme, *Photodiode Amplifiers: OP AMP Solutions*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1996.
  - [41] B. Eversmann *et al.*, "A 128 × 128 CMOS Biosensor Array for Extracellular Recording of Neural Activity," *IEEE J. Solid-State Circuits*, vol. 38, no. 12, pp. 2306–2317, 2003.
  - [42] D. Tsai, D. Sawyer, A. Bradd, R. Yuste, and K. L. Shepard, "A very large-scale microelectrode array for cellular-resolution electrophysiology," *Nat. Commun.*, vol. 8, no. 1, 2017.
  - [43] M. Ballini *et al.*, "A 1024-channel CMOS microelectrode array with 26,400 electrodes for recording and stimulation of electrogenic cells in vitro," *IEEE J. Solid-State Circuits*, vol. 49, no. 11, pp. 2705–2719, 2014.
  - [44] L. Berdondini *et al.*, "Active pixel sensor array for high spatio-temporal resolution electrophysiological recordings from single cell to large scale neuronal networks," *Lab Chip*, vol. 9, no. 18, pp. 2644–2651, 2009.
  - [45] J. Dragas *et al.*, "In Vitro Multi-Functional Microelectrode Array Featuring 59760 Electrodes, 2048 Electrophysiology Channels, Stimulation, Impedance Measurement, and Neurotransmitter Detection Channels," *IEEE J. Solid-State Circuits*, vol. 52, no. 6, pp. 1576–1590, 2017.
  - [46] T. Chi *et al.*, "A Multi-Modality CMOS Sensor Array for Cell-Based Assay and Drug Screening," *IEEE*



- Trans. Biomed. Circuits Syst.*, vol. 9, no. 6, pp. 801–814, 2015.
- [47] J. S. Park *et al.*, “1024-Pixel CMOS Multimodality Joint Cellular Sensor/Stimulator Array for Real-Time Holistic Cellular Characterization and Cell-Based Drug Screening,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 12, no. 1, pp. 80–94, 2018.
  - [48] S. Hwang, C. N. LaFratta, V. Agarwal, Xin Yu, D. R. Walt, and S. Sonkusale, “CMOS Microelectrode Array for Electrochemical Lab-on-a-Chip Applications,” *IEEE Sens. J.*, vol. 9, no. 6, pp. 609–615, Jun. 2009.
  - [49] B. N. Kim, A. D. Herbst, S. J. Kim, B. A. Minch, and M. Lindau, “Parallel recording of neurotransmitters release from chromaffin cells using a 10x10 CMOS IC potentiostat array with on-chip working electrodes,” *Biosens. Bioelectron.*, vol. 41, no. 1, pp. 736–744, 2013.
  - [50] T. Kuno, K. Niitsu, and K. Nakazato, “Amperometric electrochemical sensor array for on-chip simultaneous imaging,” *Jpn. J. Appl. Phys.*, vol. 53, no. 4 SPEC. ISSUE, pp. 1–7, 2014.
  - [51] J. Rothe, O. Frey, A. Stettler, Y. Chen, and A. Hierlemann, “Fully integrated CMOS microsystem for electrochemical measurements on  $32 \times 32$  working electrodes at 90 frames per second,” *Anal. Chem.*, vol. 86, no. 13, pp. 6425–6432, 2014.
  - [52] K. A. White, G. Mulberry, and B. N. Kim, “Rapid 1024-pixel Electrochemical Imaging at 10,000 Frames per Second using Monolithic CMOS Sensor and Multifunctional Data Acquisition System,” *IEEE Sens. J.*, vol. 18, no. 13, pp. 1–1, 2018.
  - [53] C. Giagkoulouits *et al.*, “A 16 x 16 CMOS Amperometric Microelectrode Array for Simultaneous Electrochemical Measurements,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 1, pp. 1–11, 2018.
  - [54] A. Hassibi and T. H. Lee, “A programmable 0.18- $\mu\text{m}$  CMOS electrochemical sensor microarray for biomolecular detection,” *IEEE Sens. J.*, vol. 6, no. 6, pp. 1380–1388, 2006.
  - [55] A. C. Sun, S. Member, E. Alvarez-fontecilla, A. G. Venkatesh, E. Aronoff-spencer, and D. A. Hall, “High-Density Redox Amplified Coulostatic Discharge-Based Biosensor Array,” pp. 1–11, 2018.
  - [56] M. H. Nazari, H. Mazhab-Jafari, L. Leng, A. Guenther, and R. Genov, “CMOS neurotransmitter microarray: 96-channel integrated potentiostat with on-die microsensors,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 7, no. 3, pp. 338–348, 2013.
  - [57] K. Y. Inoue *et al.*, “LSI-based amperometric sensor for bio-imaging and multi-point biosensing,” *Lab*



- Chip*, vol. 12, no. 18, pp. 3481–3490, 2012.
- [58] K. Ino *et al.*, “Densified electrochemical sensors based on local redox cycling between vertically separated electrodes in substrate generation/chip collection and extended feedback modes,” *Anal. Chem.*, vol. 86, no. 8, pp. 4016–4023, 2014.
  - [59] A. Kara *et al.*, “Electrochemical imaging for microfluidics: A full-system approach,” *Lab Chip*, vol. 16, no. 6, pp. 1081–1087, 2016.
  - [60] N. Kasai, A. Shimada, T. Nyberg, and K. Torimitsu, “An electrochemical sensor array and its application to real-time brain slice imaging,” *Electron. Commun. Japan*, vol. 92, no. 9, pp. 217–221, 2009.
  - [61] D. Polcari, P. Dauphin-Ducharme, and J. Mauzeroll, “Scanning Electrochemical Microscopy: A Comprehensive Review of Experimental Parameters from 1989 to 2015,” *Chem. Rev.*, vol. 116, no. 22, pp. 13234–13278, Nov. 2016.
  - [62] R. J. Forster, “Microelectrodes: New Dimensions in Electrochemistry,” *Chem. Soc. Rev.*, no. M, pp. 289–297, 1994.
  - [63] L. . Thomas, C.A. Springer, P.A., Loeb, G.E., Berwald- Netter, Y., Okun, “A miniature microelectrode array to monitor the bioelectric activity of cultured cells,” *Exp. Cell Res.*, vol. 74, no. 1, pp. 61–66, Sep. 1972.
  - [64] B. A. Hierlemann, U. Frey, S. Hafizovic, and F. Heer, “Growing Cells Atop Microelectronic Chips,” *Proc. IEEE*, vol. 99, no. 2, 2011.
  - [65] F. Heer *et al.*, “Single-chip microelectronic system to interface with living cells,” *Biosens. Bioelectron.*, vol. 22, no. 11, pp. 2546–2553, 2007.
  - [66] K. Niitsu, S. Ota, K. Gamo, H. Kondo, M. Hori, and K. Nakazato, “Development of Microelectrode Arrays Using Electroless Plating for CMOS-Based Direct Counting of Bacterial and HeLa Cells,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 9, no. 5, pp. 607–619, 2015.
  - [67] W. Tedjo *et al.*, “Design of an integrated microelectrode array system for high spatiotemporal resolution chemical imaging,” in *Proceedings - 2016 IEEE Biomedical Circuits and Systems Conference, BioCAS 2016*, 2017.
  - [68] A. J. Bard, *Electrogenerated Chemiluminescence*, no. Cl. Elsevier B.V., 2004.



- [69] W. Wilson, T. Chen, and R. Selby, "A current-starved inverter-based differential amplifier design for ultra-low power applications," *2013 IEEE 4th Lat. Am. Symp. Circuits Syst. LASCAS 2013 - Conf. Proc.*, 2013.
- [70] M. Figueiredo, E. Santin, and J. Goes, "Biased CMOS Amplifier with High Efficiency," *Analysis*, pp. 2828–2831, 2010.
- [71] Clinical and Laboratory Standards Institute, *Protocols for Determination of Limits of Detection and Limits of Quantitation ; Approved Guideline.CLSI Document EP17-A*, vol. 24, no. 34. 2004.
- [72] P. M. Levine, P. Gong, R. Levicky, and K. L. Shepard, "Active CMOS sensor array for electrochemical biomolecular detection," *IEEE J. Solid-State Circuits*, vol. 43, no. 8, pp. 1859–1871, 2008.
- [73] D. L. Robinson, A. Hermans, A. T. Seipel, and R. M. Wightman, "Monitoring rapid chemical communication in the brain," *Chem. Rev.*, vol. 108, no. 7, pp. 2554–2584, 2008.
- [74] K. Pihel, T. J. Schroeder, and R. M. Wightman, "Rapid and Selective Cyclic Voltammetric Measurements of Epinephrine and Norepinephrine as a Method To Measure Secretion from Single Bovine Adrenal Medullary Cells," *Anal. Chem.*, vol. 66, no. 24, pp. 4532–4537, 1994.
- [75] George J. Siegel, *Basic Neurochemistry: Molecular, Cellular, and Medical Aspects*. 2006.
- [76] A. O. Martin, M. N. Mathieu, C. Chevillard, and N. C. Guérineau, "Gap junctions mediate electrical signaling and ensuing cytosolic Ca<sup>2+</sup> increases between chromaffin cells in adrenal slices: A role in catecholamine release.," *J. Neurosci.*, vol. 21, no. 15, pp. 5397–405, Aug. 2001.
- [77] H. Xu *et al.*, "Striatal dopamine release in a schizophrenia mouse model measured by electrochemical amperometry in vivo," *Analyst*, vol. 140, no. 11, pp. 3840–3845, 2015.
- [78] J. Petrovic, P. L. Walsh, K. T. Thornley, C. E. Miller, and R. M. Wightman, "Real-Time Monitoring of Chemical Transmission in Slices of the Murine Adrenal Gland," vol. 151, no. April, pp. 1773–1783, 2010.
- [79] D. C. Montgomery, *Design and Analysis of Experiments*, 8th ed. Wiley, 2012.
- [80] A. M. Poisner, "Caffeine-Induced Catecholamine Secretion: Similarity to Caffeine-Induced Muscle Contraction," *Proc. Soc. Exp. Biol. Med.*, vol. 142, no. 1, pp. 103–105, 1973.
- [81] S. Spector, J. Tarver, and B. Berkowitz, "Effects of Drugs and Physiological Factors in the Disposition of Catecholamines in Blood Vessels," *Pharmacol. Rev.*, vol. 24, no. 2, pp. 191 LP – 202, Jun. 1972.



- [82] W. G. French and T. Kuwana, "Lifetime of Activated Platinum Surface," *J. Phys. Chem.*, vol. 68, no. 6, pp. 1279–1284, Jun. 1964.
- [83] D. P. Manica, Y. Mitsumori, and A. G. Ewing, "Characterization of electrode fouling and surface regeneration for a platinum electrode on an electrophoresis microchip," *Anal. Chem.*, vol. 75, no. 17, pp. 4572–4577, 2003.
- [84] T. J. Davies *et al.*, "The linear sweep voltammetry of random arrays of microdisc electrodes: Fitting of experimental data," *J. Electroanal. Chem.*, vol. 592, no. 2, pp. 126–130, Jul. 2006.
- [85] T. J. Davies and R. G. Compton, "The cyclic and linear sweep voltammetry of regular and random arrays of microdisc electrodes: Theory," *J. Electroanal. Chem.*, vol. 585, no. 1, pp. 63–82, 2005.
- [86] Ray Alan Hastings, *The Art of Analog Layout*, 2nd ed. Upper Saddle River, NJ: Pearson/Prentice Hall, 2006.
- [87] C. Saint and J. Saint, *IC mask design : essential layout techniques*. New York SE: McGraw-Hill, 2002.
- [88] L. Han, V. Perez, and M. Cayanes, "CMOS Transistor Layout KungFu," *Lee Eng Han*, 2005.
- [89] L. Pilon and H. N. Wang, "Accurate Simulations of Electric Double Layer Capacitance of Ultramicroelectrodes," *J. Phys. Chem. C*, vol. 115, no. 33, pp. 16711–16719, 2011.
- [90] J. Hasegawa, S. Uno, and K. Nakazato, "Amperometric electrochemical sensor array for On-chip simultaneous imaging: Circuit and microelectrode design considerations," *Jpn. J. Appl. Phys.*, vol. 50, no. 4 PART 2, 2011.
- [91] L. Te Yin, J. C. Chou, W. Y. Chung, T. P. Sun, and S. K. Hsiung, "Separate structure extended gate H<sup>+</sup>-ion sensitive field effect transistor on a glass substrate," *Sensors Actuators, B Chem.*, vol. 71, no. 1–2, pp. 106–111, 2000.
- [92] G. D. Smith and S. Takayama, "Application of microfluidic technologies to human assisted reproduction," *Mol. Hum. Reprod.*, vol. 23, no. 4, pp. 257–268, 2017.
- [93] S. Le Gac and V. Nordhoff, "Microfluidics for mammalian embryo culture and selection: Where do we stand now?," *Mol. Hum. Reprod.*, vol. 23, no. 4, pp. 213–226, 2017.
- [94] D. A. Armbruster and T. Pry, "Limit of blank, limit of detection and limit of quantitation," *Clin. Biochem. Rev.*, vol. 29, no. Suppl 1, pp. S49–S52, Aug. 2008.
- [95] A. A. Ensafi, M. Taei, and T. Khayamian, "A differential pulse voltammetric method for simultaneous



- determination of ascorbic acid, dopamine, and uric acid using poly (3-(5-chloro-2-hydroxyphenylazo)-4,5-dihydroxynaphthalene-2,7-disulfonic acid) film modified glassy carbon electrode," *J. Electroanal. Chem.*, vol. 633, no. 1, pp. 212–220, 2009.
- [96] H. Sun *et al.*, "Gold nanoparticle-decorated MoS<sub>2</sub> nanosheets for simultaneous detection of ascorbic acid, dopamine and uric acid," *RSC Adv.*, vol. 4, no. 52, p. 27625, 2014.
- [97] S. Ben Aoun, "Nanostructured carbon electrode modified with N-doped graphene quantum dots–chitosan nanocomposite: a sensitive electrochemical dopamine sensor," *R. Soc. Open Sci.*, vol. 4, no. 11, p. 171199, Nov. 2017.
- [98] N. G. Mphuthi, A. S. Adekunle, and E. E. Ebenso, "Electrocatalytic oxidation of Epinephrine and Norepinephrine at metal oxide doped phthalocyanine / MWCNT composite sensor," *Nat. Publ. Gr.*, no. May, pp. 1–20, 2016.
- [99] K. Pihel, T. J. Schroeder, and R. M. Wightman, "Rapid and Selective Cyclic Voltammetric Measurements of Epinephrine and Norepinephrine as a Method To Measure Secretion from Single Bovine Adrenal Medullary Cells," vol. 66, no. 24, pp. 4532–4537, 1994.
- [100] A. Salis and M. Monduzzi, "Not only pH. Specific buffer effects in biological systems," *Curr. Opin. Colloid Interface Sci.*, vol. 23, pp. 1–9, 2016.
- [101] S. R. S. Bibby, D. A. Jones, R. M. Ripley, and J. P. G. Urban, "Metabolism of the Intervertebral Disc : Effects of Low Levels of Oxygen , Glucose , and pH on Rates of Energy Metabolism of Bovine Nucleus Pulposus Cells," vol. 30, no. 5, pp. 487–496, 2005.
- [102] E. R. Davies, "Chapter 3 - Basic Image Filtering Operations," E. R. B. T.-C. and M. V. (Fourth E. Davies, Ed. Boston: Academic Press, 2012, pp. 38–81.
- [103] M. Elas *et al.*, "Electron paramagnetic resonance oxygen images correlate spatially and quantitatively with Oxylite oxygen measurements.," *Clin. Cancer Res.*, vol. 12, no. 14 Pt 1, pp. 4209–4217, 2006.
- [104] M. Elas *et al.*, "Quantitative tumor oxymetric images from 4D electron paramagnetic resonance imaging (EPRI): Methodology and comparison with blood oxygen level-dependent (BOLD) MRI," *Magn. Reson. Med.*, vol. 49, no. 4, pp. 682–691, 2003.
- [105] M. Kotecha, B. Epel, S. Ravindran, D. Dorcemus, S. Nukavarapu, and H. Halpern, "Noninvasive Absolute Electron Paramagnetic Resonance Oxygen Imaging for the Assessment of Tissue Graft



- Oxygenation," *Tissue Eng. Part C Methods*, vol. 24, no. 1, pp. 14–19, 2017.
- [106] J. P. B. O'Connor, S. P. Robinson, and J. C. Waterton, "Imaging tumour hypoxia with oxygen-enhanced MRI and BOLD MRI," *Br. J. Radiol.*, vol. 92, no. 1096, p. 20180642, 2018.
- [107] O. S. Wolfbeis, "Luminescent sensing and imaging of oxygen: Fierce competition to the Clark electrode," *BioEssays*, vol. 37, no. 8, pp. 921–928, 2015.
- [108] D. B. Papkovsky and R. I. Dmitriev, "Imaging of oxygen and hypoxia in cell and tissue samples," *Cell. Mol. Life Sci.*, vol. 75, no. 16, pp. 2963–2980, 2018.
- [109] T. Yoshihara, Y. Hirakawa, M. Hosaka, M. Nangaku, and S. Tobita, "Oxygen imaging of living cells and tissues using luminescent molecular probes," *J. Photochem. Photobiol. C Photochem. Rev.*, vol. 30, pp. 71–95, 2017.
- [110] H. Kurokawa *et al.*, "High resolution imaging of intracellular oxygen concentration by phosphorescence lifetime," *Sci. Rep.*, vol. 5, pp. 1–3, 2015.
- [111] T. V. Esipova, M. J. P. Barrett, E. Erlebach, A. E. Masunov, B. Weber, and S. A. Vinogradov, "Oxyphor 2P: A High-Performance Probe for Deep-Tissue Longitudinal Oxygen Imaging," *Cell Metab.*, vol. 29, no. 3, pp. 736–744.e7, 2019.
- [112] S. C. Lesher-Pérez *et al.*, "Dispersible oxygen microsensors map oxygen gradients in three-dimensional cell cultures," *Biomater. Sci.*, vol. 5, no. 10, pp. 2106–2113, 2017.
- [113] J. Icha, M. Weber, J. C. Waters, and C. Norden, "Phototoxicity in live fluorescence microscopy, and how to avoid it," *BioEssays*, vol. 39, no. 8, pp. 1–15, 2017.
- [114] Y. Date *et al.*, "Monitoring oxygen consumption of single mouse embryos using an integrated electrochemical microdevice," *Biosens. Bioelectron.*, vol. 30, no. 1, pp. 100–106, 2011.
- [115] E. Vanhove *et al.*, "Development of electrochemical microsensors for the monitoring of mitochondrial activities," in *2013 Transducers & Eurosensors XXVII: The 17th International Conference on Solid-State Sensors, Actuators and Microsystems*, 2013, no. June, pp. 1135–1138.
- [116] M. A. Edwards, S. Martin, A. L. Whitworth, J. V. Macpherson, and P. R. Unwin, "Scanning electrochemical microscopy: Principles and applications to biophysical systems," *Physiol. Meas.*, vol. 27, no. 12, 2006.
- [117] R. Lazenby and R. White, "Advances and Perspectives in Chemical Imaging in Cellular Environments



- Using Electrochemical Methods,” *Chemosensors*, vol. 6, no. 2, p. 24, 2018.
- [118] T. E. Lin, S. Rapino, H. H. Girault, and A. Lesch, “Electrochemical imaging of cells and tissues,” *Chem. Sci.*, vol. 9, no. 20, pp. 4546–4554, 2018.
- [119] S. Amemiya, A. J. Bard, F.-R. F. Fan, M. V. Mirkin, and P. R. Unwin, “Scanning Electrochemical Microscopy,” *Annu. Rev. Anal. Chem.*, vol. 1, no. 1, pp. 95–131, Jul. 2008.
- [120] A. C. Sun, E. Alvarez-Fontecilla, A. G. Venkatesh, E. Aronoff-Spencer, and D. A. Hall, “High-Density Redox Amplified Coulostatic Discharge-Based Biosensor Array,” *IEEE J. Solid-State Circuits*, vol. 53, no. 7, pp. 2054–2064, 2018.
- [121] J. Rothe, O. Frey, A. Stettler, Y. Chen, and A. Hierlemann, “Fully Integrated CMOS Microsystem for Electrochemical Measurements on  $32 \times 32$  Working Electrodes at 90 Frames Per Second,” *Anal. Chem.*, vol. 86, no. 13, pp. 6425–6432, Jul. 2014.
- [122] K. Y. Inoue *et al.*, “Advanced LSI-based amperometric sensor array with light-shielding structure for effective removal of photocurrent and mode selectable function for individual operation of 400 electrodes,” *Lab Chip*, vol. 15, no. 3, pp. 848–856, 2015.
- [123] S. E. Olson and G. E. Seidel, “Reduced oxygen tension and EDTA improve bovine zygote development in a chemically defined medium,” *J. Anim. Sci.*, vol. 78, no. 1, pp. 152–157, 2000.
- [124] J. F. De La Torre-Sanchez, K. Preis, and G. E. Seidel, “Metabolic regulation of in-vitro-produced bovine embryos. I. Effects of metabolic regulators at different glucose concentrations with embryos produced by semen from different bulls,” *Reprod. Fertil. Dev.*, vol. 18, no. 5, p. 585, 2006.
- [125] F. Si *et al.*, “Electrochemical Oxygen Reduction Reaction,” in *Rotating Electrode Methods and Oxygen Reduction Electrocatalysts*, Elsevier, 2014, pp. 133–170.
- [126] H. Suzuki, A. Sugama, and N. Kojima, “Micromachined Clark oxygen electrode,” *Sensors Actuators B. Chem.*, vol. 10, no. 2, pp. 91–98, 1993.
- [127] C. C. Wu *et al.*, “Microfluidic chip integrated with amperometric detector array for in situ estimating oxygen consumption characteristics of single bovine embryos,” *Sensors Actuators, B Chem.*, vol. 125, no. 2, pp. 680–687, 2007.
- [128] J. Thompson, M. Lane, and R. Gilchrist, “Metabolism of the bovine cumulus-oocyte complex and influence on subsequent developmental competence,” *Reprod. Domest. Ruminants*, vol. 6, no. 1, pp.



179–190, 2011.

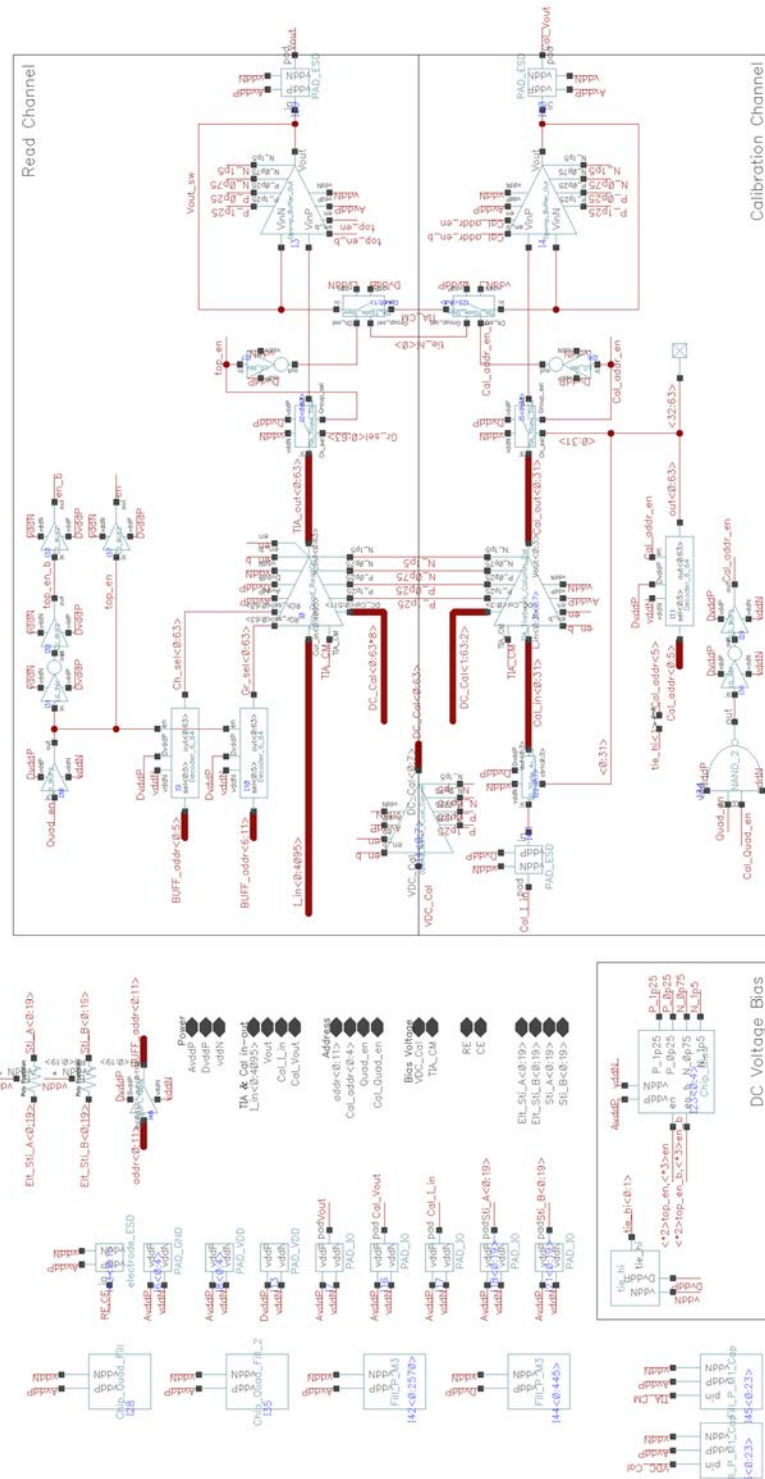
- [129] H. Kurosawa *et al.*, “Development of a new clinically applicable device for embryo evaluation which measures embryo oxygen consumption,” *Hum. Reprod.*, vol. 31, no. 10, pp. 2321–2330, 2016.
- [130] B. Agung *et al.*, “Relationship between oxygen consumption and sex of bovine in vitro fertilized embryos,” *Reprod. Domest. Anim.*, vol. 40, no. 1, pp. 51–56, 2005.
- [131] H. Shiku *et al.*, “Respiration activity of single bovine embryos entrapped in a cone-shaped microwell monitored by scanning electrochemical microscopy,” *Anal. Chim. Acta*, vol. 522, no. 1, pp. 51–58, 2004.
- [132] M. Xu, R. Wang, and Y. Li, “Electrochemical biosensors for rapid detection of *Escherichia coli* O157:H7,” *Talanta*, vol. 162, no. August 2016, pp. 511–522, 2017.
- [133] P. J. Ratcliffe, “Oxygen sensing and hypoxia signalling pathways in animals: The implications of physiology for cancer,” *J. Physiol.*, vol. 591, no. 8, pp. 2027–2042, 2013.
- [134] J. Zhou, M. Sheng, X. Jiang, G. Wu, and F. Gao, “Simultaneous determination of dopamine, serotonin and ascorbic acid at a glassy carbon electrode modified with carbon-spheres,” *Sensors (Basel)*, vol. 13, no. 10, pp. 14029–14040, 2013.



## APPENDIX A: CMOS MICROELECTRODE ARRAY MICROCHIP

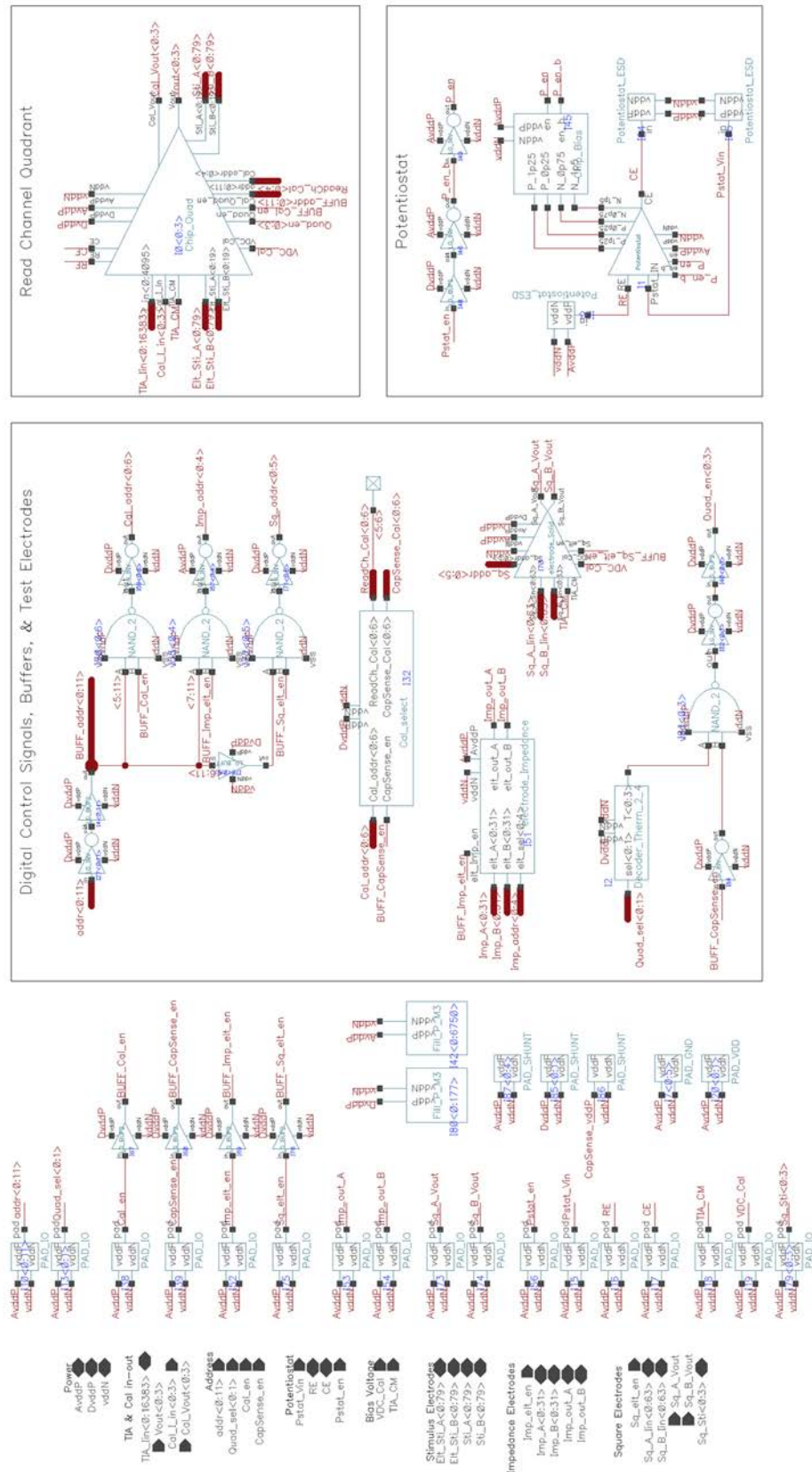
### A.1 Cadence Detailed Schematics

### A.2.1 Entire CMOS Microchip



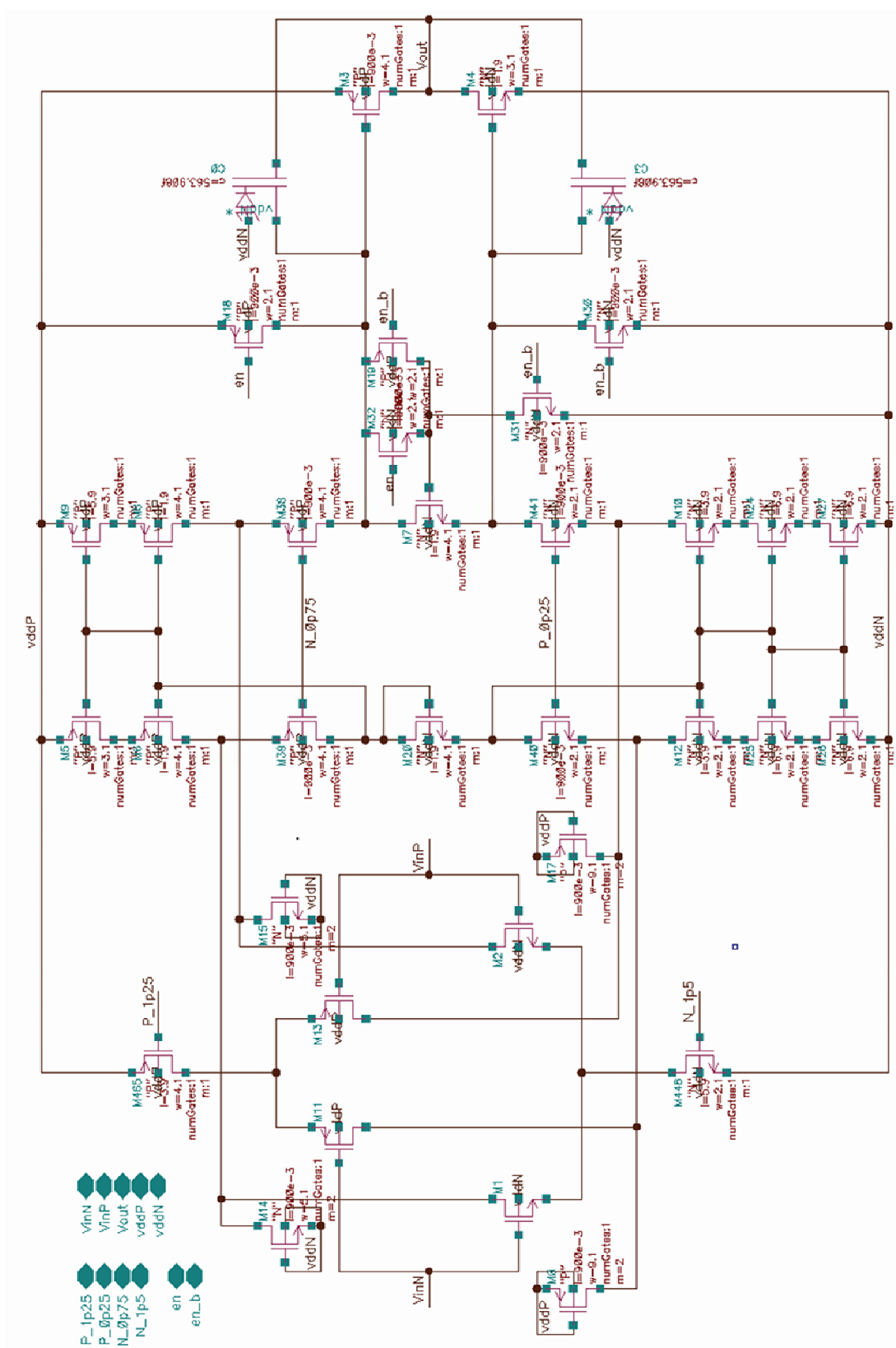


## A.2.2 Quadrant



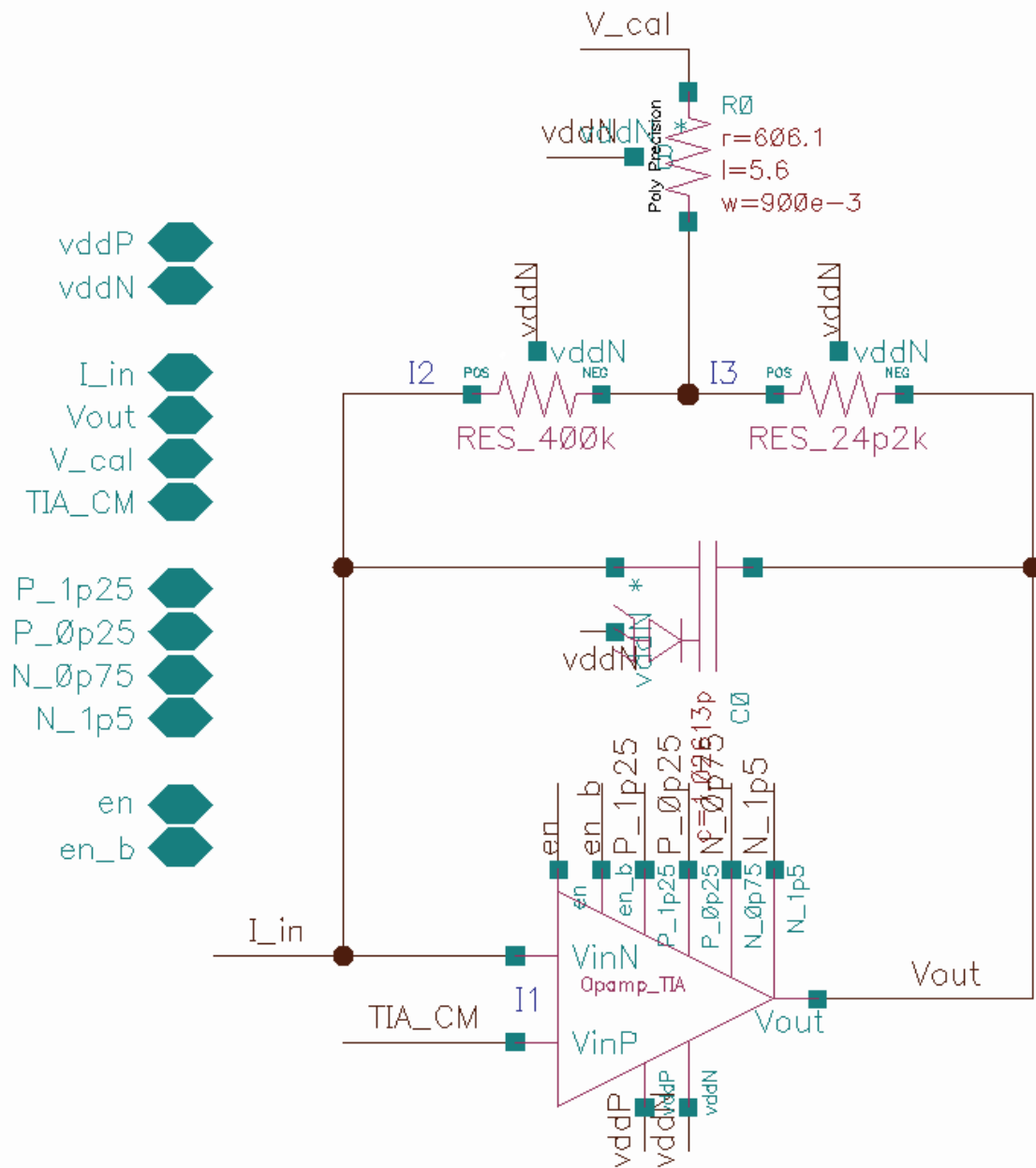


### A.2.3 Transimpedance Amplifier: Operational Amplifier



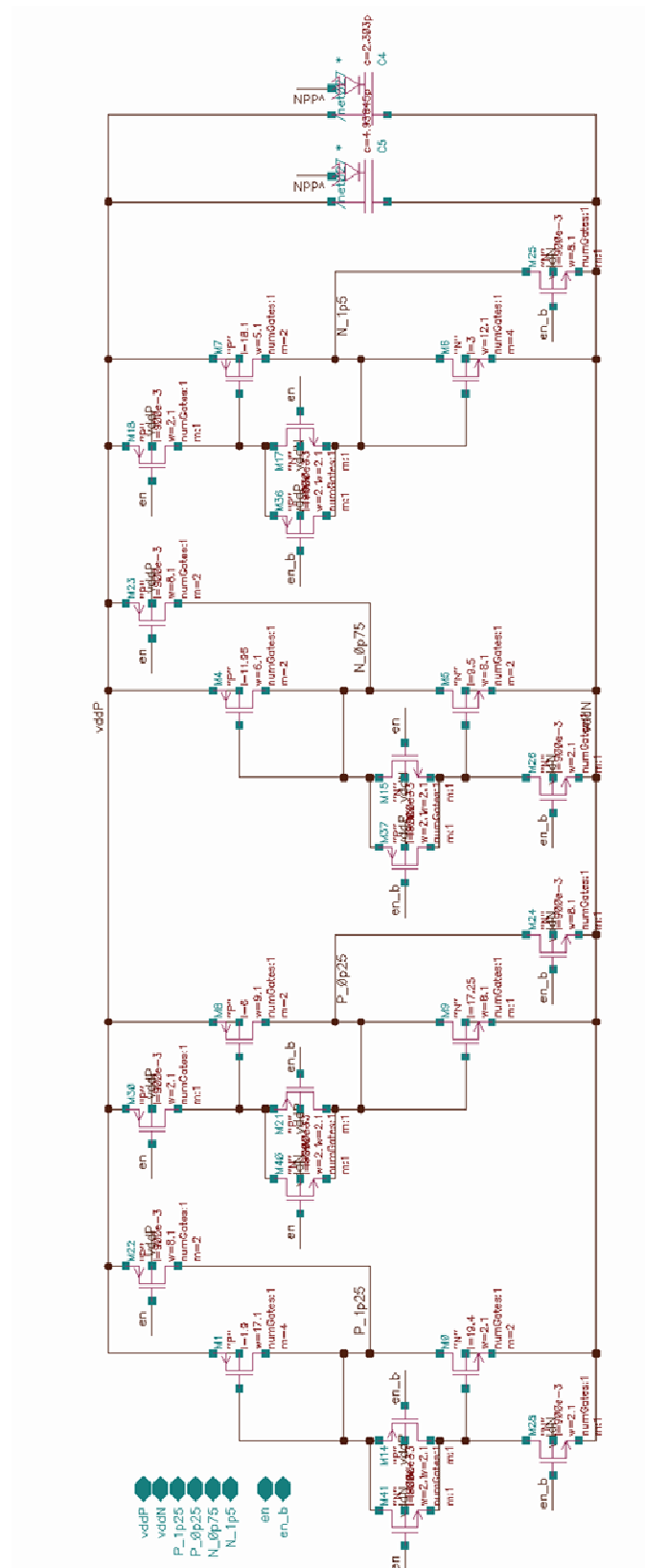


### A.2.4 Transimpedance Amplifier



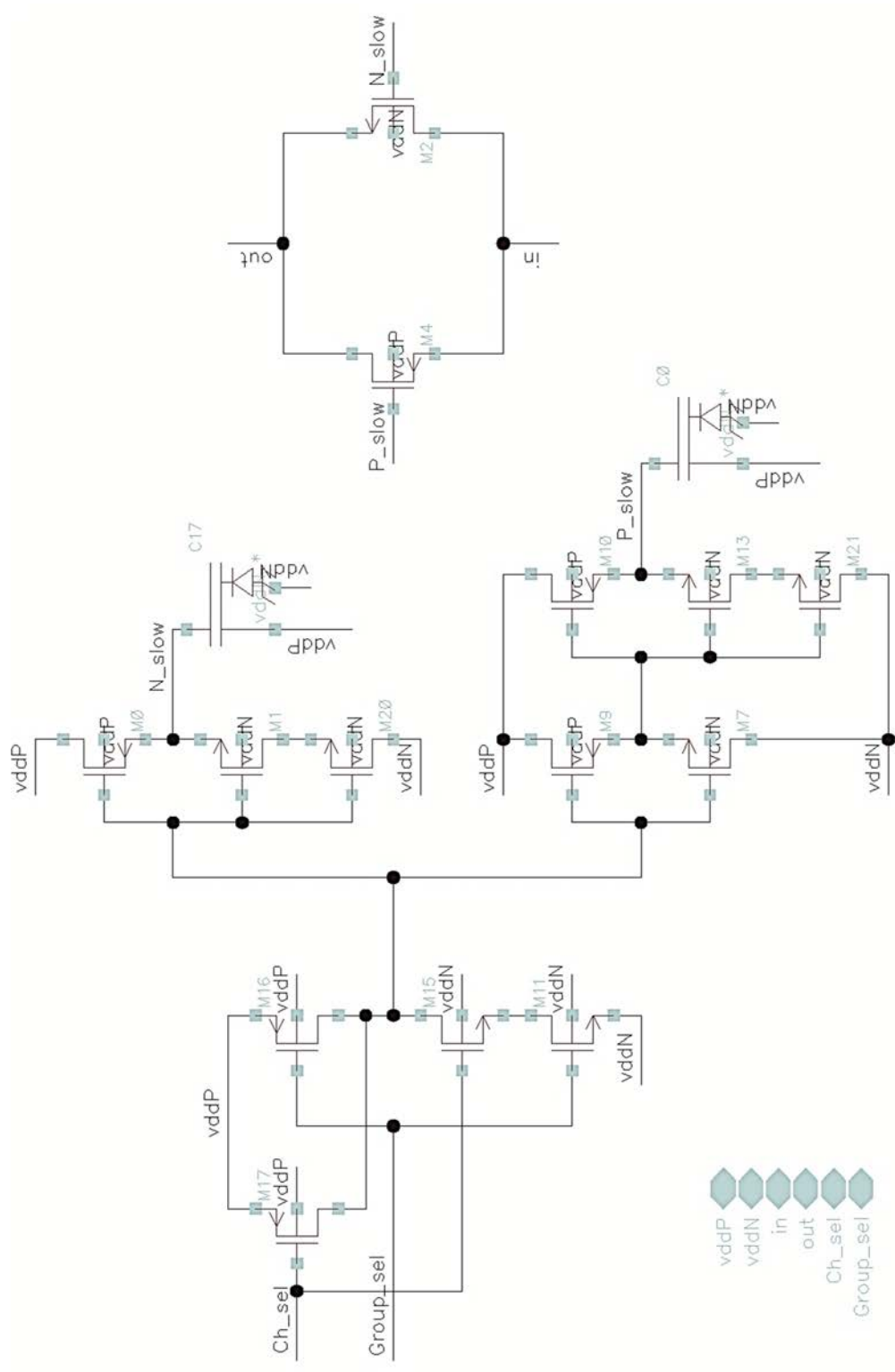


### A.2.5 Global Bias Circuit



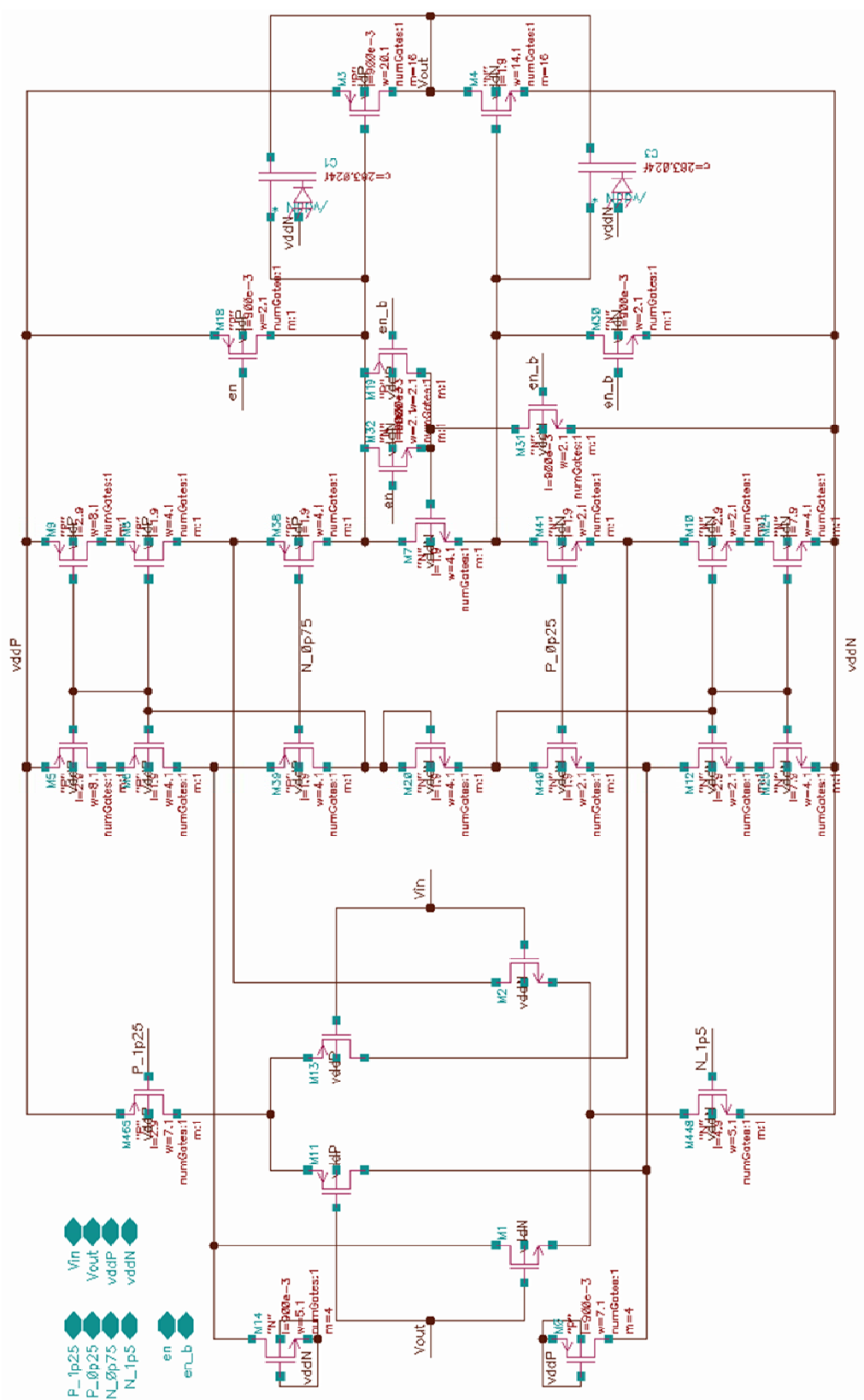


## A.2.6 Transmission Gate (TG)



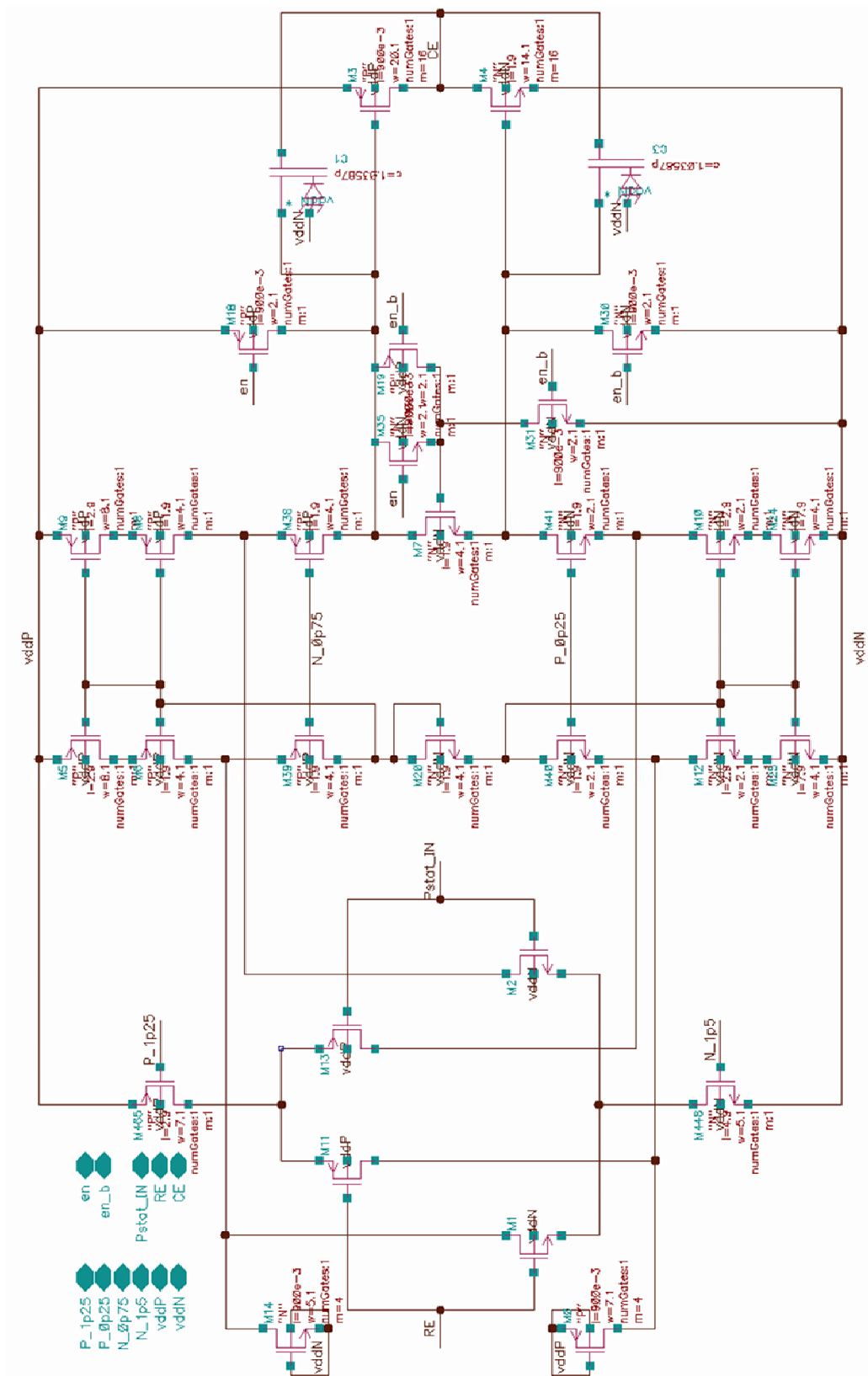


### A.2.7 Voltage Buffer





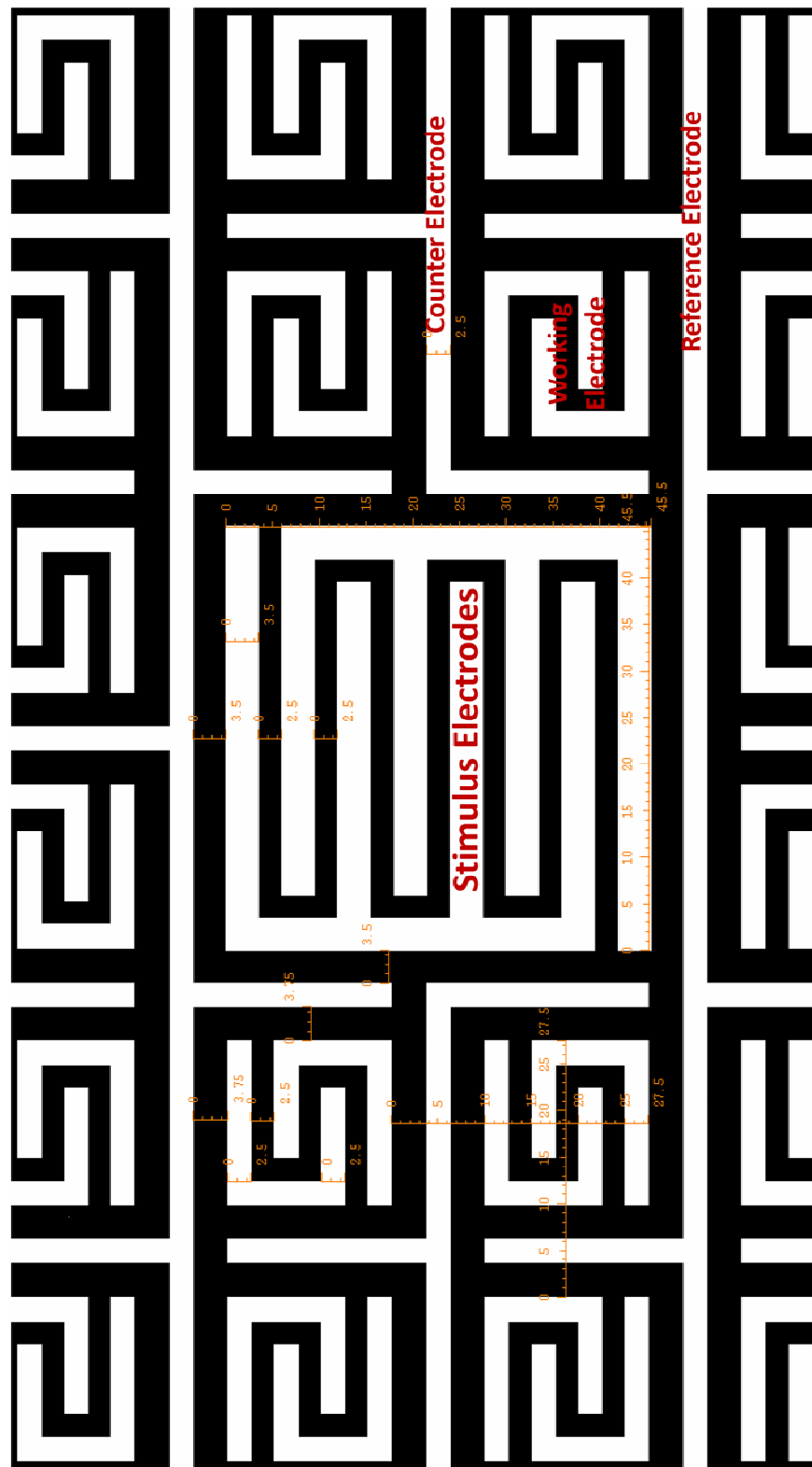
## A.2.8 Potentiostat





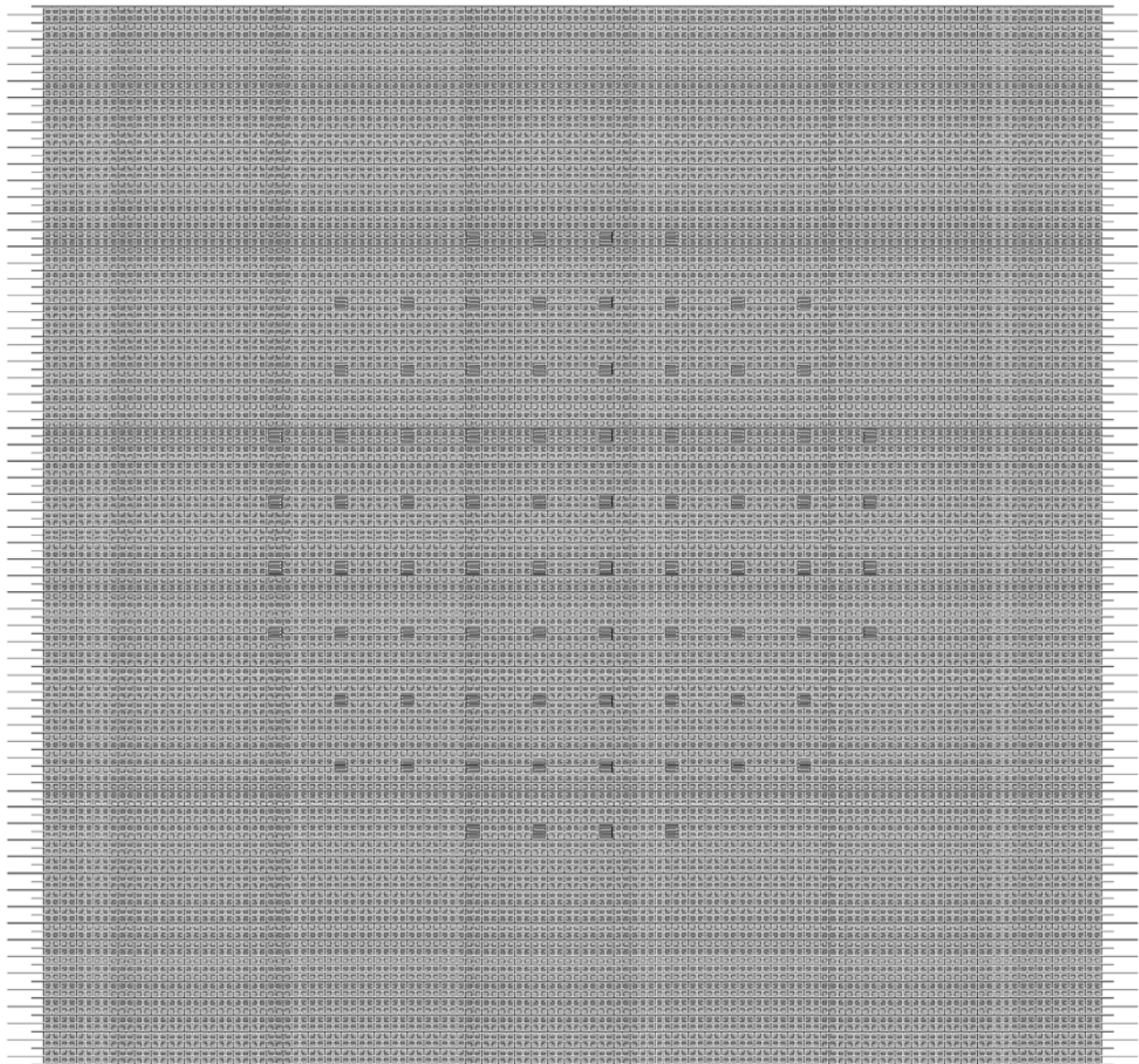
## A.2 Surface MEA

### A.2.1 Dimension of MEA configuration





A.2.2 Entire Surface





### A.3 External Pins

\*Type: A = analog, D = digital | V = voltage, I = current | P = power, I = input, O = output

Pin Name	Type*	#	Description
<b>Power Lines</b>		<b>53</b>	
AvddP	A V P	22	Positive analog power lines (+2.5V)
DvddP	A V P	4	Positive digital power lines (+2.5V)
CapSense_vddP	A V P	1	Positive capacitive sensor power lines (+1.5V)
vddN	A V P	26	Negative power   ground lines (-2.5V -1.5V)
<b>Voltage Bias</b>		<b>2</b>	
VDC_Cal	A V I	1	Read channel output DC offset (default = 0V)
TIA_CM	A V I	1	Read channel common mode (default = 0V)
<b>Enable</b>		<b>5</b>	
Pstat_en	D V I	1	Potentiostat circuit enable
CapSense_en	D V I	1	Capacitive sensor enable, chip disable
Cal_en	D V I	1	Read channel calibration selection enable
Imp_elt_en	D V I	1	Impedance electrode selection enable
Sq_elt_en	D V I	1	Square electrodes circuit enable
<b>Address</b>		<b>14</b>	
addr	D V I	12	Read channel 12-bit least significant digit
Quad_sel	D V I	2	Quadrant select, 2-bit most significant digit
<b>Channel Output</b>		<b>4</b>	
Vout	A V O	4	Read channel voltage output, each per quadrant
<b>Channel Calibration</b>		<b>8</b>	
Cal_I_in	A I I	4	Channel calibration current input, each per quadrant
Cal_Vout	A V O	4	Channel calibration voltage output, each per quadrant
<b>Potentiostat</b>		<b>3</b>	
Pstat_Vin	A V I	1	Potentiostat reference voltage input
RE	A V IO	1	Reference electrode
AE	A V IO	1	Auxiliary electrode



<b>Stimulus Electrodes</b>		<b>160</b>	
Sti_A	A VI IO	80	Stimulus electrode side A
Sti_B	A VI IO	80	Stimulus electrode side B
<b>Impedance Electrodes</b>		<b>2</b>	
Imp_out_A	A VI IO	1	Impedance electrode side A
Imp_out_B	A VI IO	1	Impedance electrode side B
<b>Square Electrodes</b>		<b>6</b>	
Sq_A_Vout	A V O	1	Impedance electrode type A
Sq_B_Vout	A V O	1	Impedance electrode type B
Sq_Sti	A VI IO	4	Stimulus nodes for impedance electrodes
<b>Capacitive Sensor</b>		<b>23</b>	
CapSense_S	D V I	2	Capacitive sensor control signal
CapSense_Update	D V I	1	Capacitive sensor update signal
CapSense_Pad	D V O	20	Capacitive sensor digital output



## A.4 Cadence SKILL Scripts

- Generation of read channel array connections in particular order.

```
procedure(createReadChArray())
    cellID = geGetWindowCellView()
    x1 = 1.9
    y1 = 0
    x2 = 2.5
    y2 = 0.6
    pinCount = 0
    numOfGr = 8
    numOfSixFour = 128

    for(i 1 numOfSixFour
        for(i 1 numOfGr
            sprintf(pinName "Cur_in<d>" pinCount)
            leCreatePin(cellID list("metal3" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 + 1.5
            x2 = x2 + 1.5
            pinCount = pinCount + 1
        )
        x1 = x1 + 7.65
        x2 = x2 + 7.65

        for(i 1 numOfGr
            sprintf(pinName "Cur_in<d>" pinCount)
            leCreatePin(cellID list("metal3" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 + 1.5
            x2 = x2 + 1.5
            pinCount = pinCount + 1
        )

        for(i 1 numOfGr
            sprintf(pinName "Cur_in<d>" pinCount)
            leCreatePin(cellID list("metal3" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 + 1.5
            x2 = x2 + 1.5
            pinCount = pinCount + 1
        )
        x1 = x1 + 7.65
        x2 = x2 + 7.65

        for(i 1 numOfGr
            sprintf(pinName "Cur_in<d>" pinCount)
            leCreatePin(cellID list("metal3" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 + 1.5
            x2 = x2 + 1.5
            pinCount = pinCount + 1
        )
        x1 = x1 + 2.8
        x2 = x2 + 2.8
    )
)
```

- Generation of working electrodes array at Metal 4 and their pins at the corresponding Metal 4 layer.

```
procedure(createElectrodeArray())
    cellID = geGetWindowCellView()
    x1 = 0
    y1 = 0
    x2 = 2.5
    y2 = 2.5
    pinCount = 12288
    numOfRow = 64
    numOfCol = 32

    for(i 1 numOfCol
        for(i 1 numOfRow
            sprintf(pinName "I_in<d>" pinCount)
            leCreatePin(cellID list("metal4" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 + 42.5
            x2 = x2 + 42.5
            pinCount = pinCount + 1

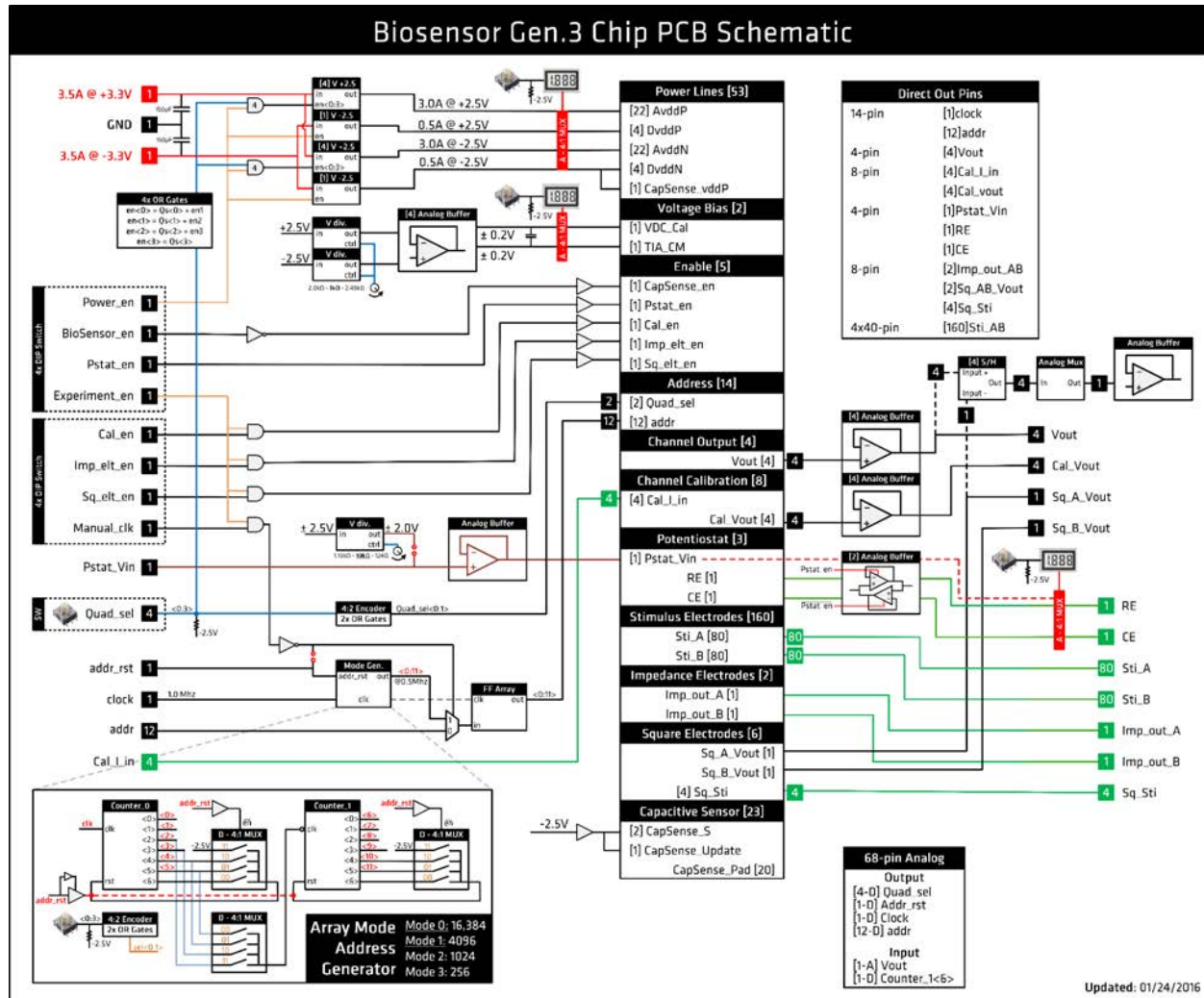
            sprintf(pinName "I_in<d>" pinCount)
            leCreatePin(cellID list("metal4" "pin") "rectangle" list(x1:y1 x2:y2) pinName "inputOutput" list("right" "left"))

            x1 = x1 - 42.5
            x2 = x2 - 42.5
            y1 = y1 + 27.5
            y2 = y2 + 27.5
            pinCount = pinCount + 1
        )
        x1 = 0 + 55*i
        y1 = 0
        x2 = 2.5 + 55*i
        y2 = 2.5
    )
)
```



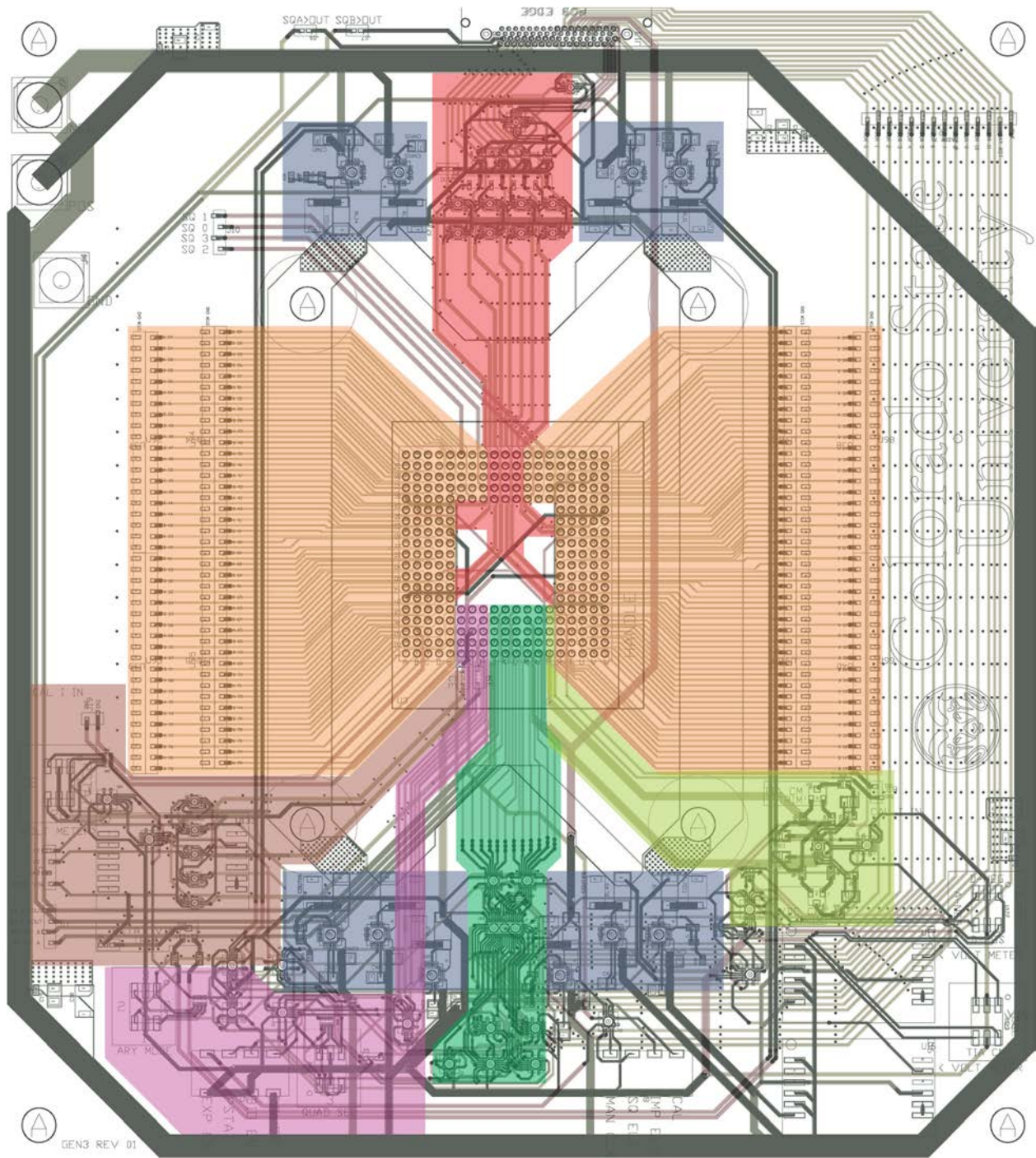
## APPENDIX B: PCB SCHEMATIC & LAYOUT

### B.1. PCB – Prototype board



Functional block diagram of the prototype board.





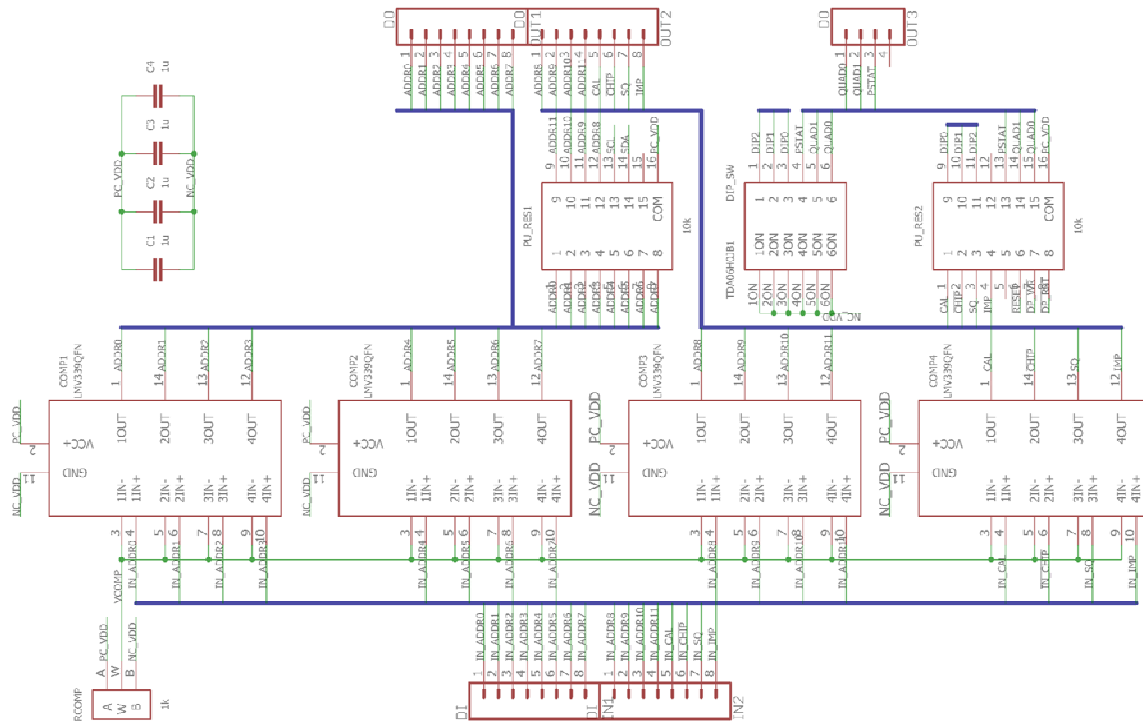
Prototype board layout. Original Design by Nicholas Grant.



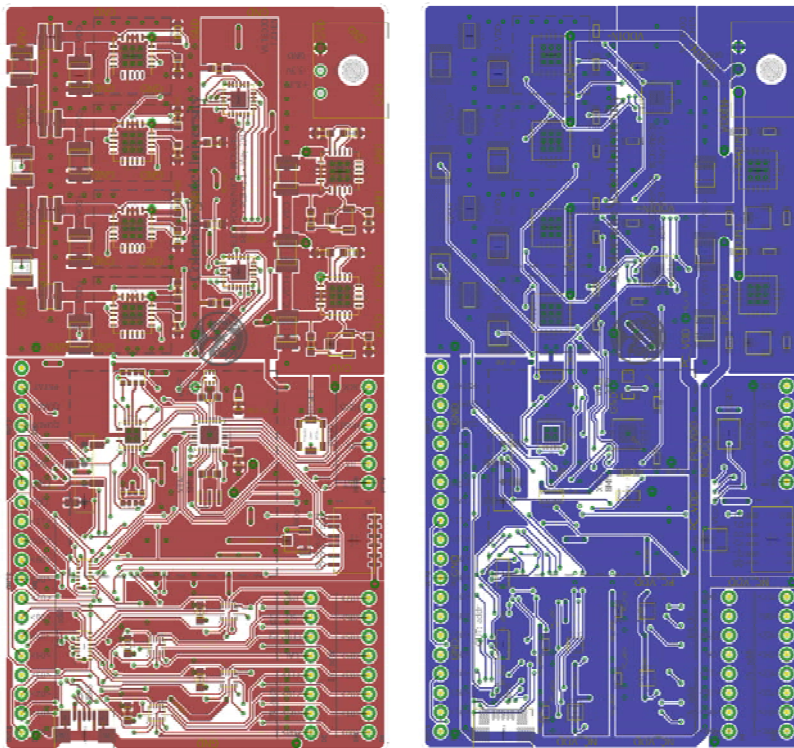
## Main board schematic – version 1.





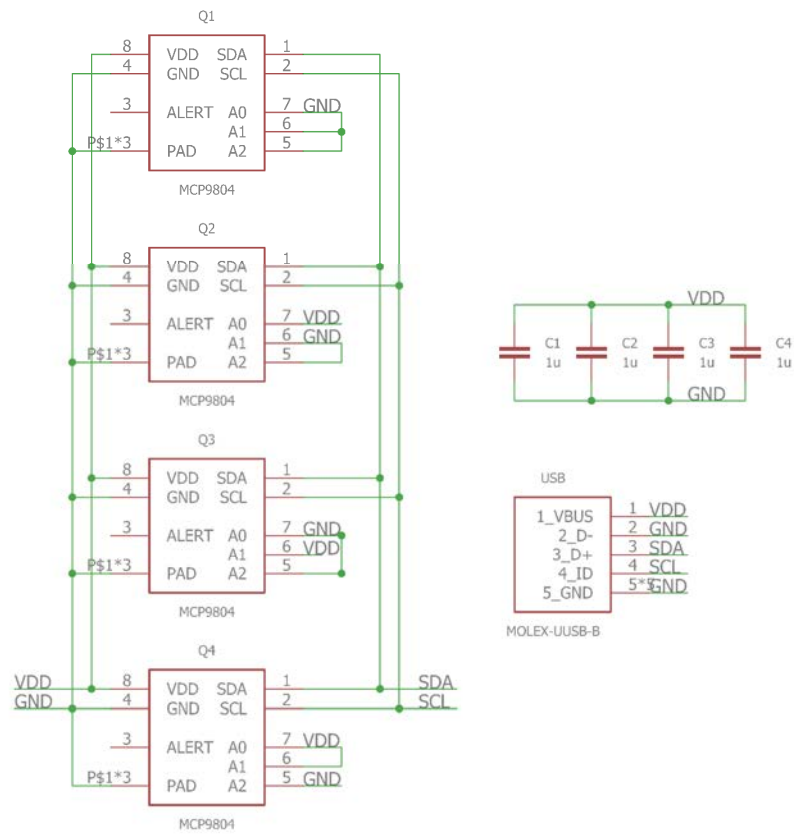


Single to dual rail conversion for the 12-bit array address schematic.

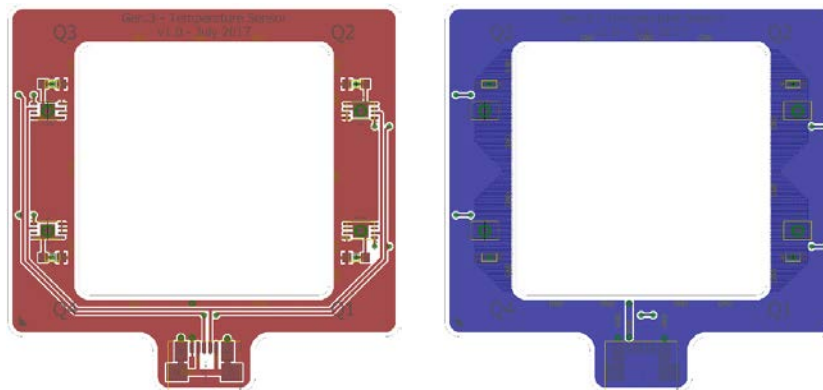


Main board PCB layout – version 1: top and bottom layers.





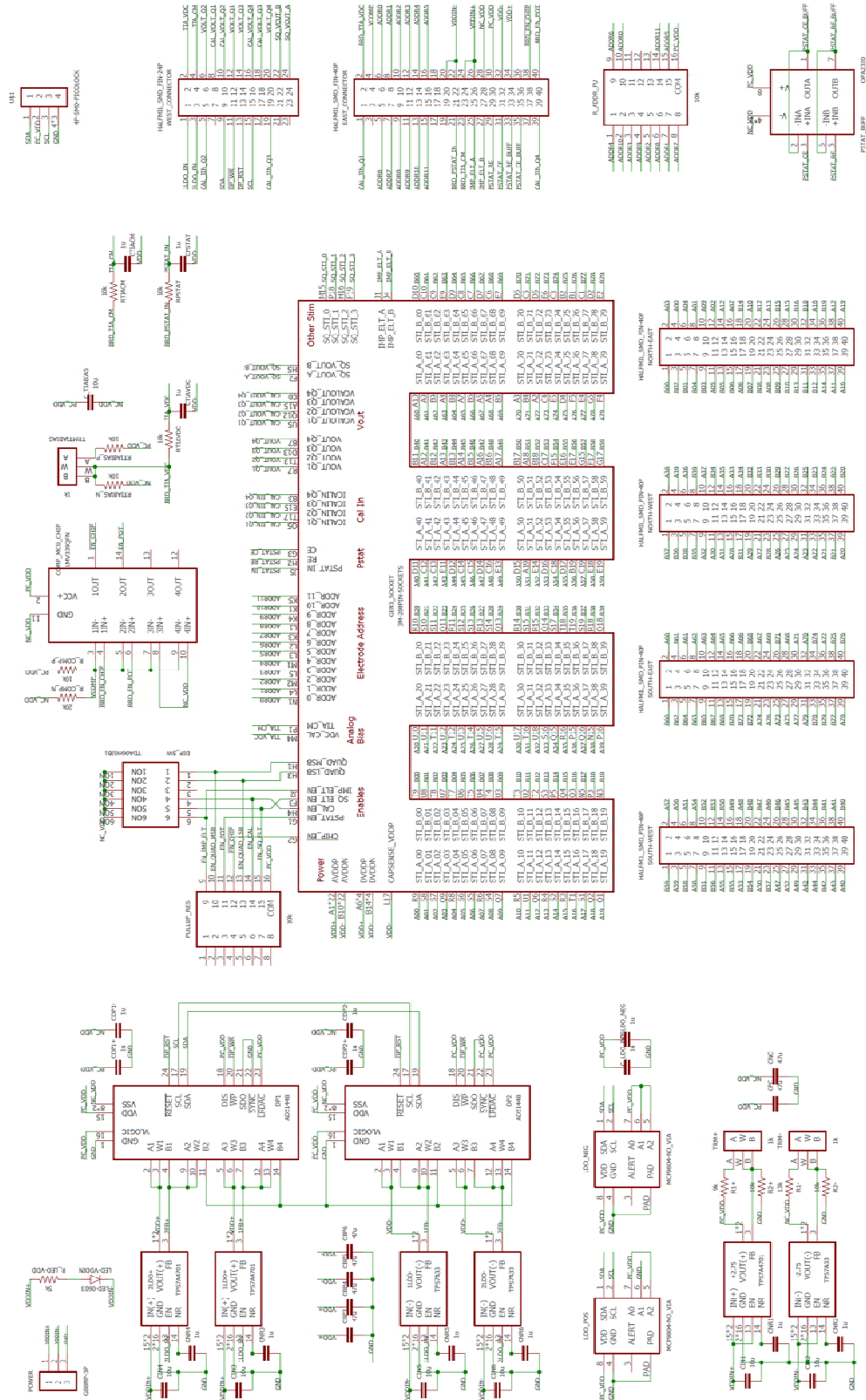
Temperature sensor schematic.



Temperature sensor PCB layout: top and bottom layers.

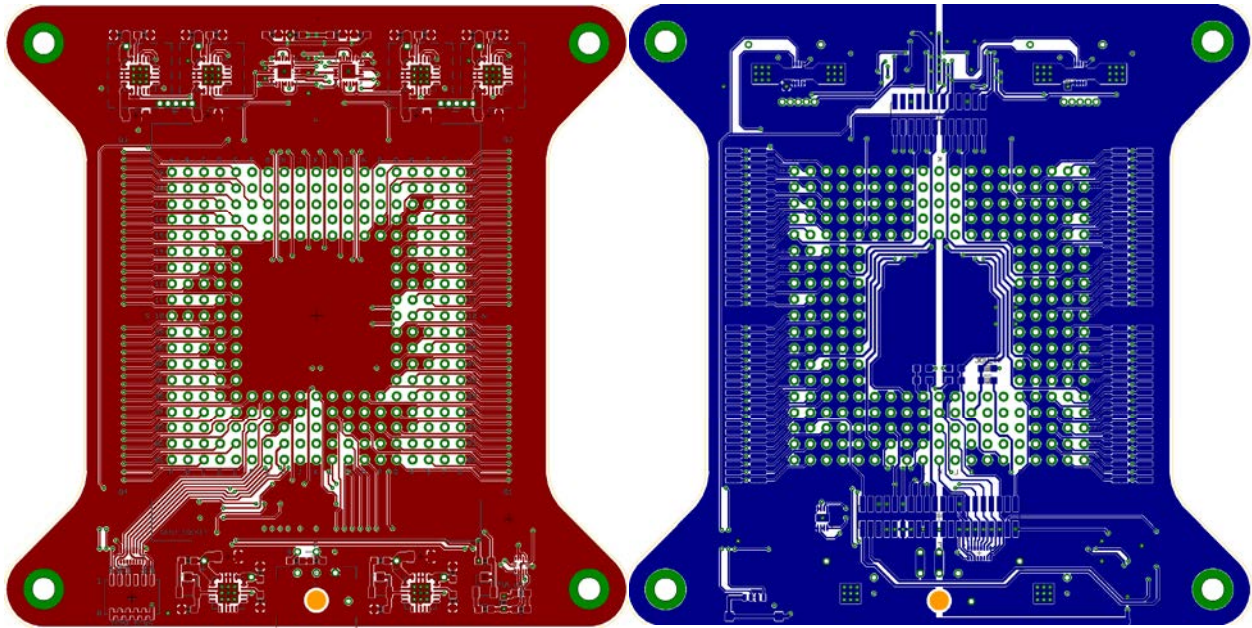


### B.3. PCB – version 2

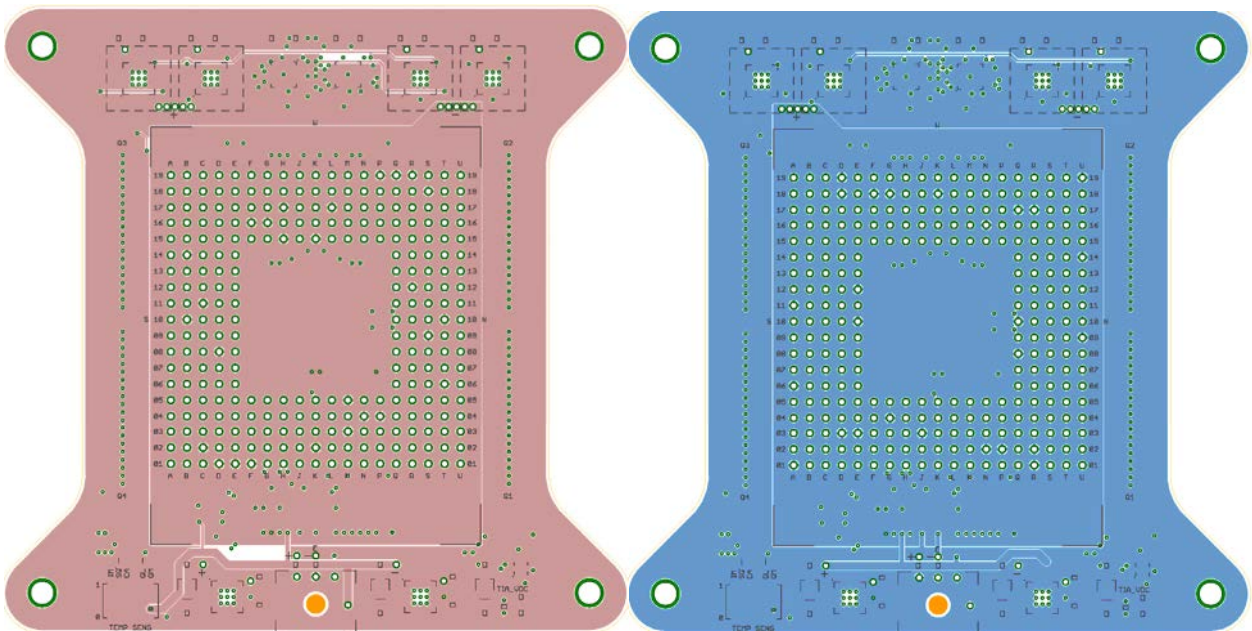


Sensor board schematic – version 2.





Sensor board PCB layout – version 2: top and bottom layers.

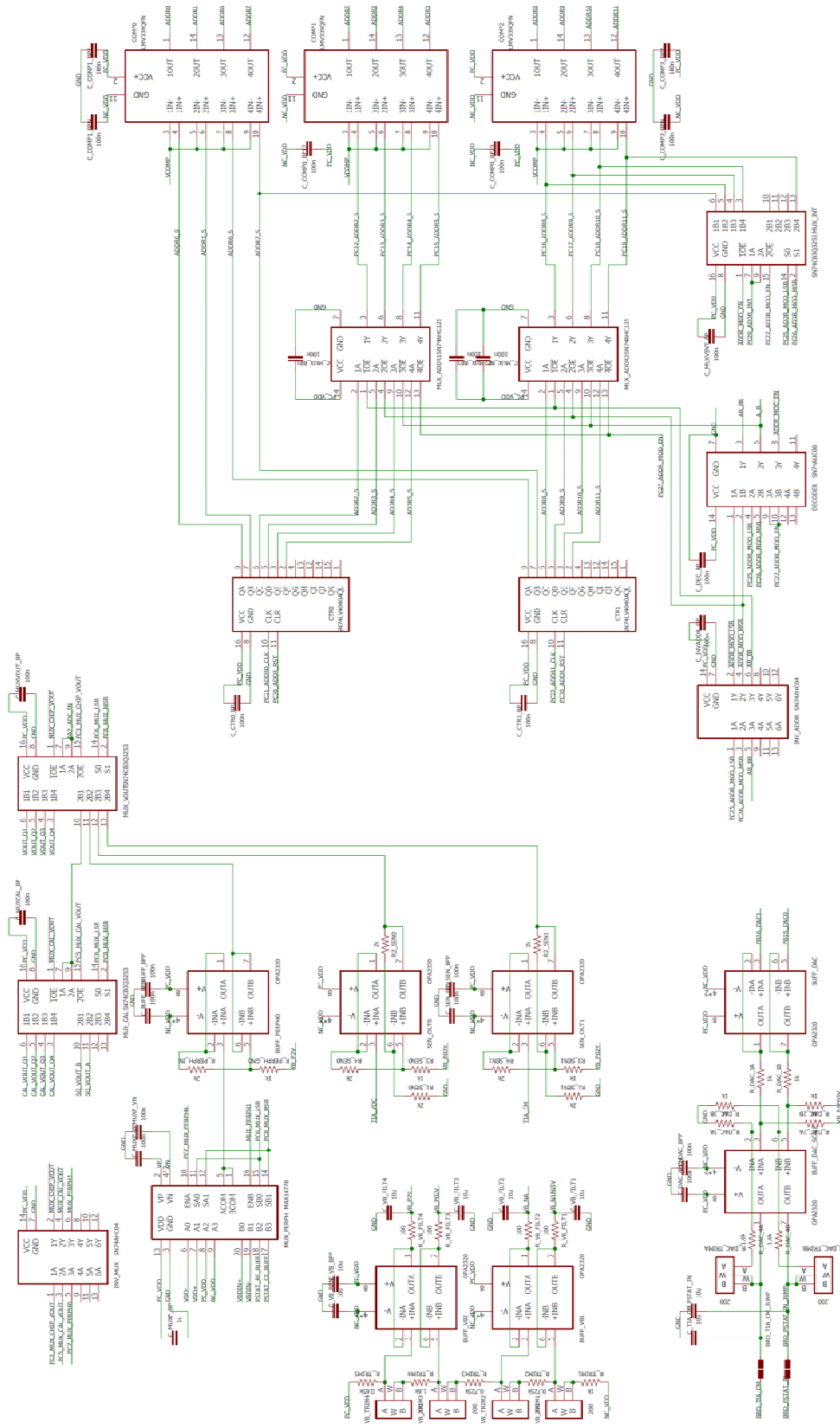


Sensor board PCB Layout – version 2: middle layers.



Control board schematic, MCU – version 2.





Control board schematic: Digital Control – version 2.

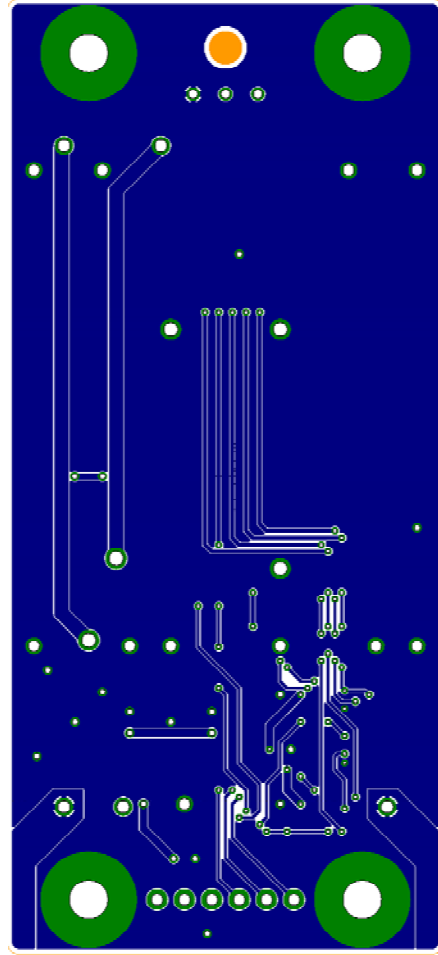
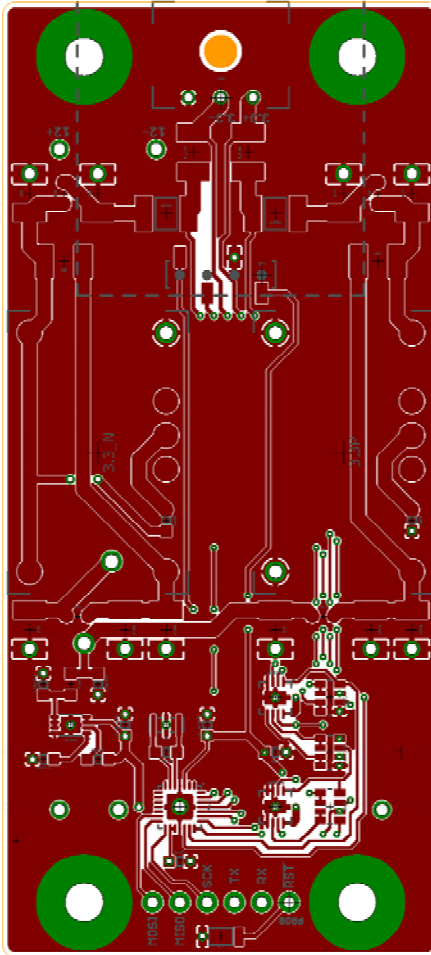






Power supply unit schematic – version 2.

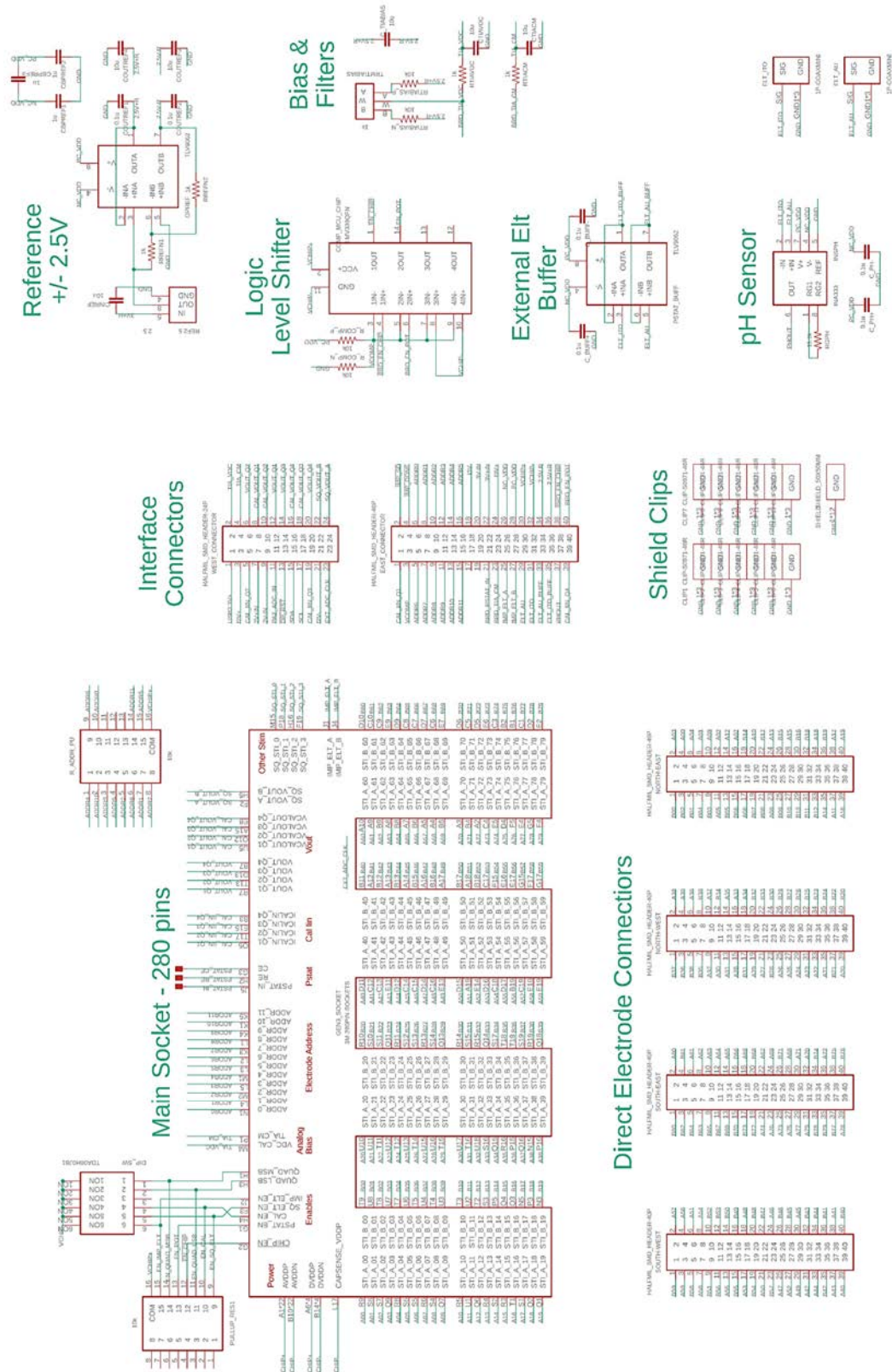




Power supply unit layout – version 2: top and bottom layers.



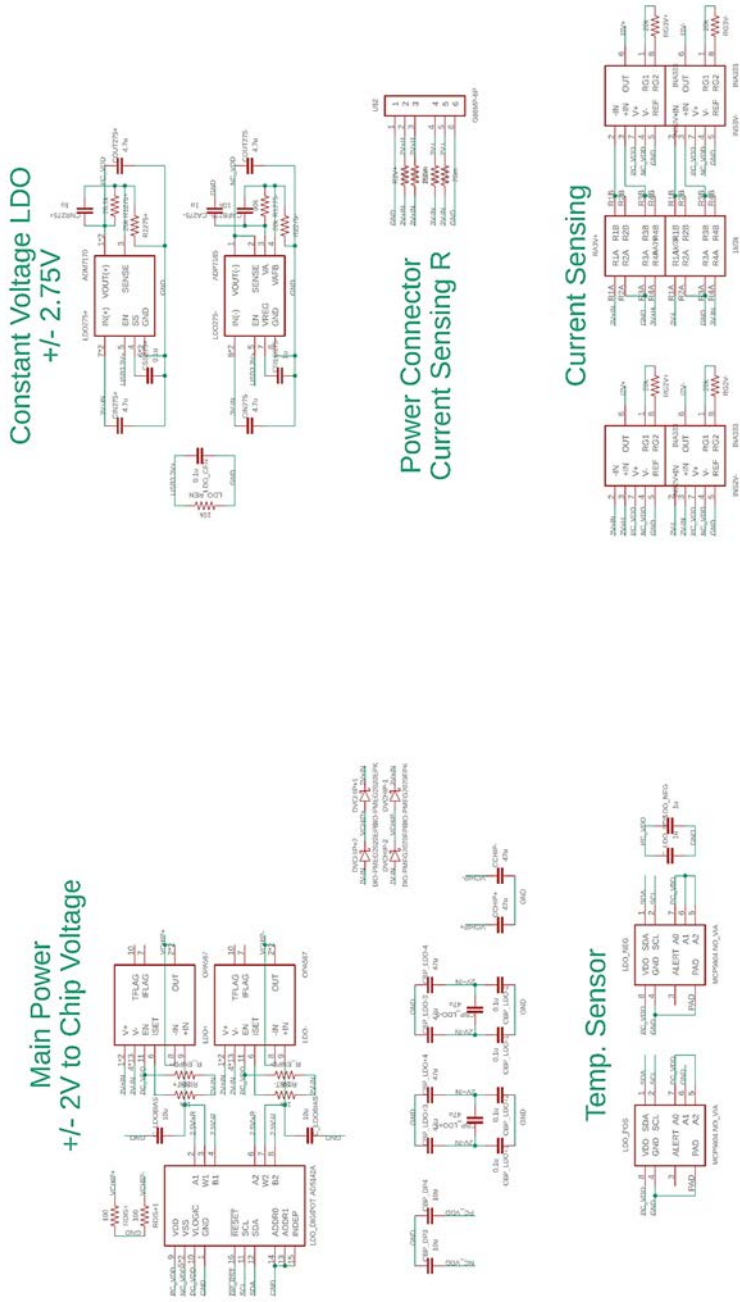
## B.4. PCB – version 3



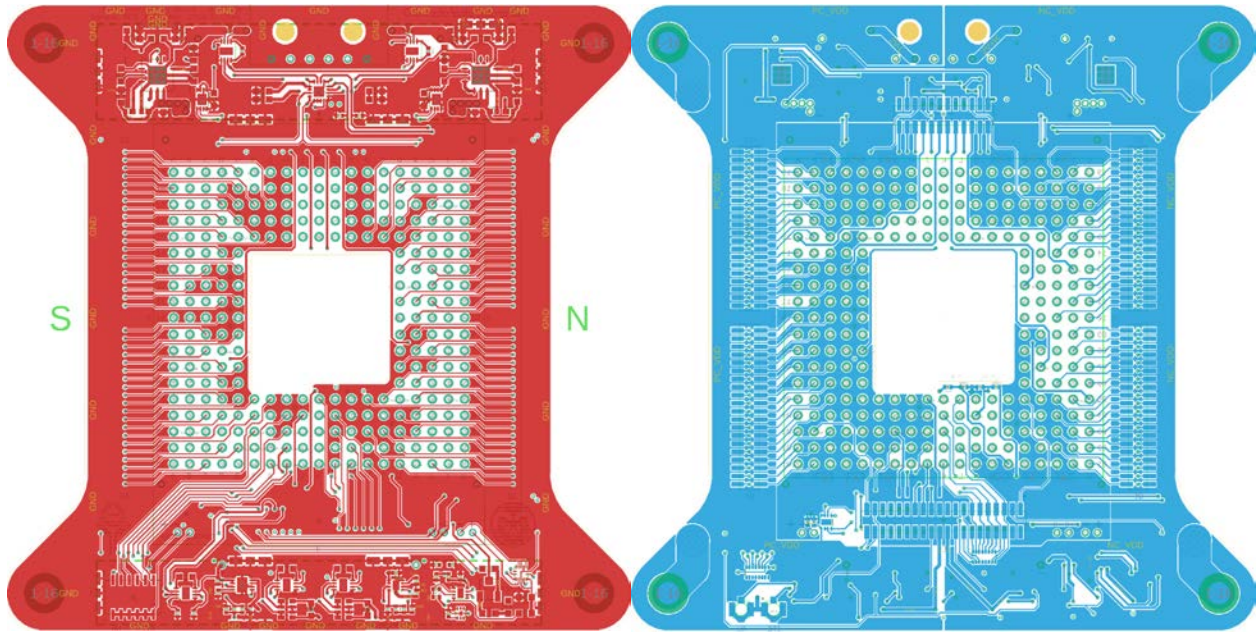
Sensor board schematic – version 3 – Main.



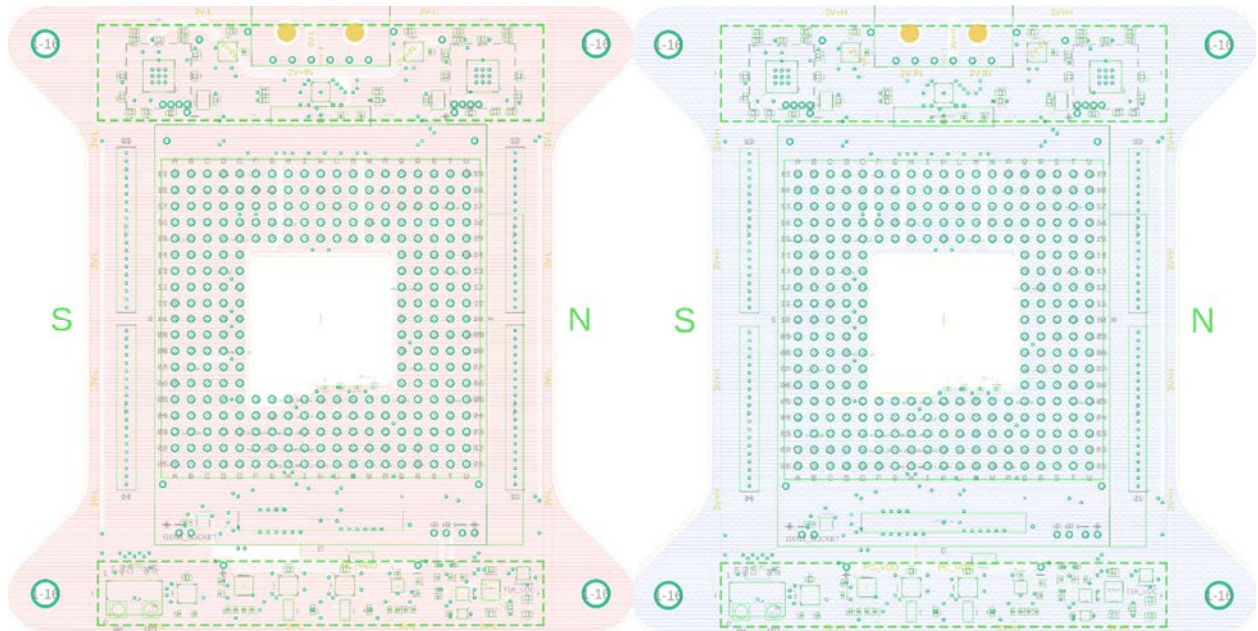
Sensor board schematic – version 3 – Power.







Sensor board PCB layout – version 3: top and bottom layers.



Sensor board PCB Layout – version 3: middle layers.



Control board schematic, MCU – version 3.

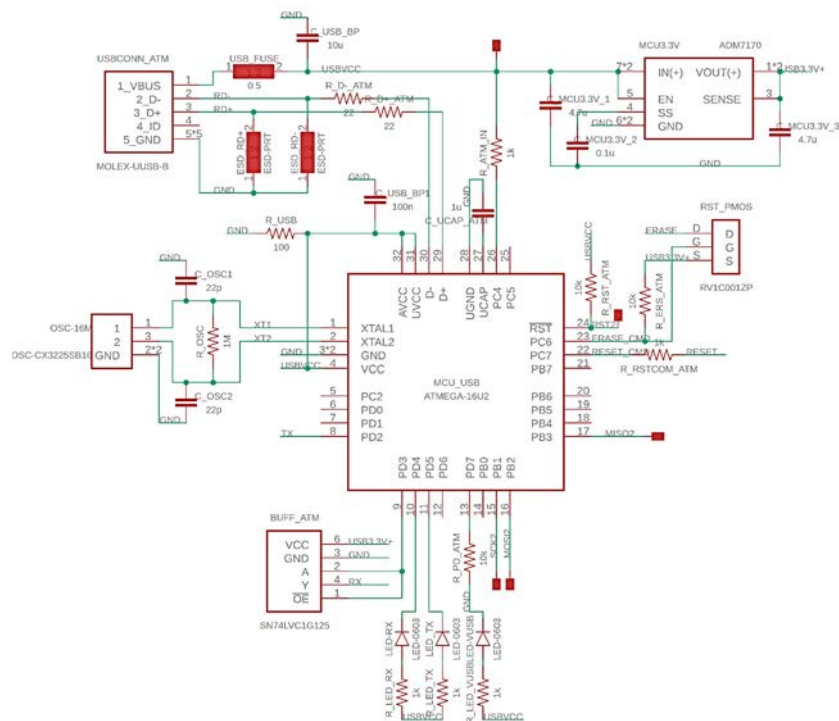


Control board schematic, Analog – version 3.

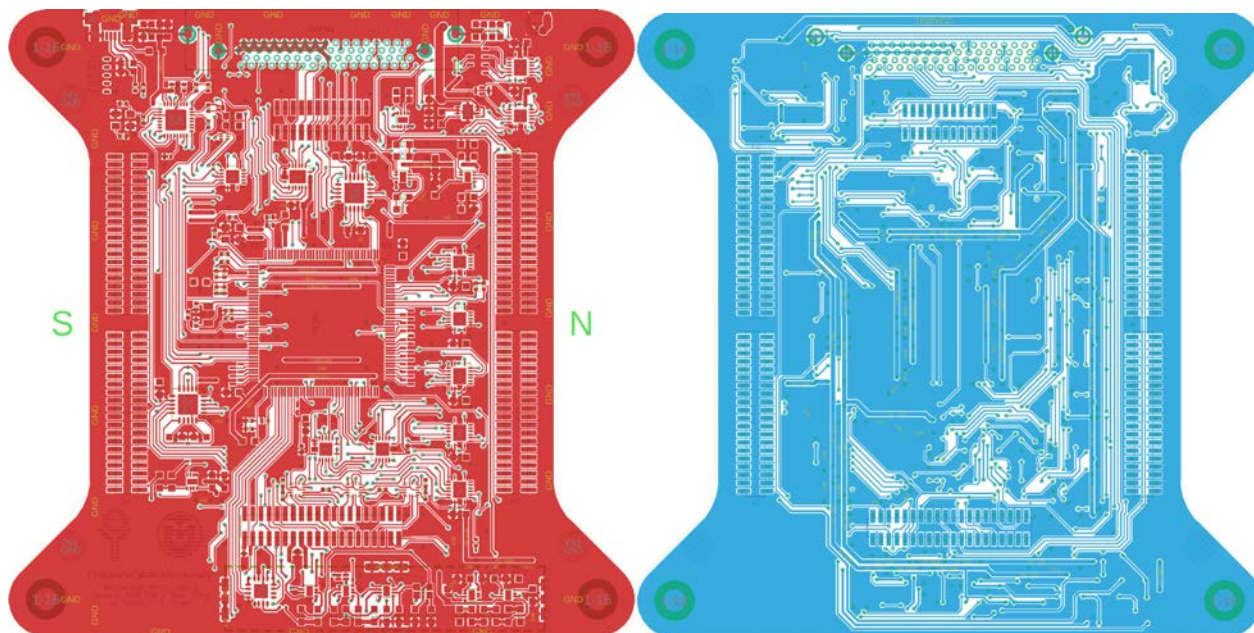


Control board schematic, Digital – version 3.



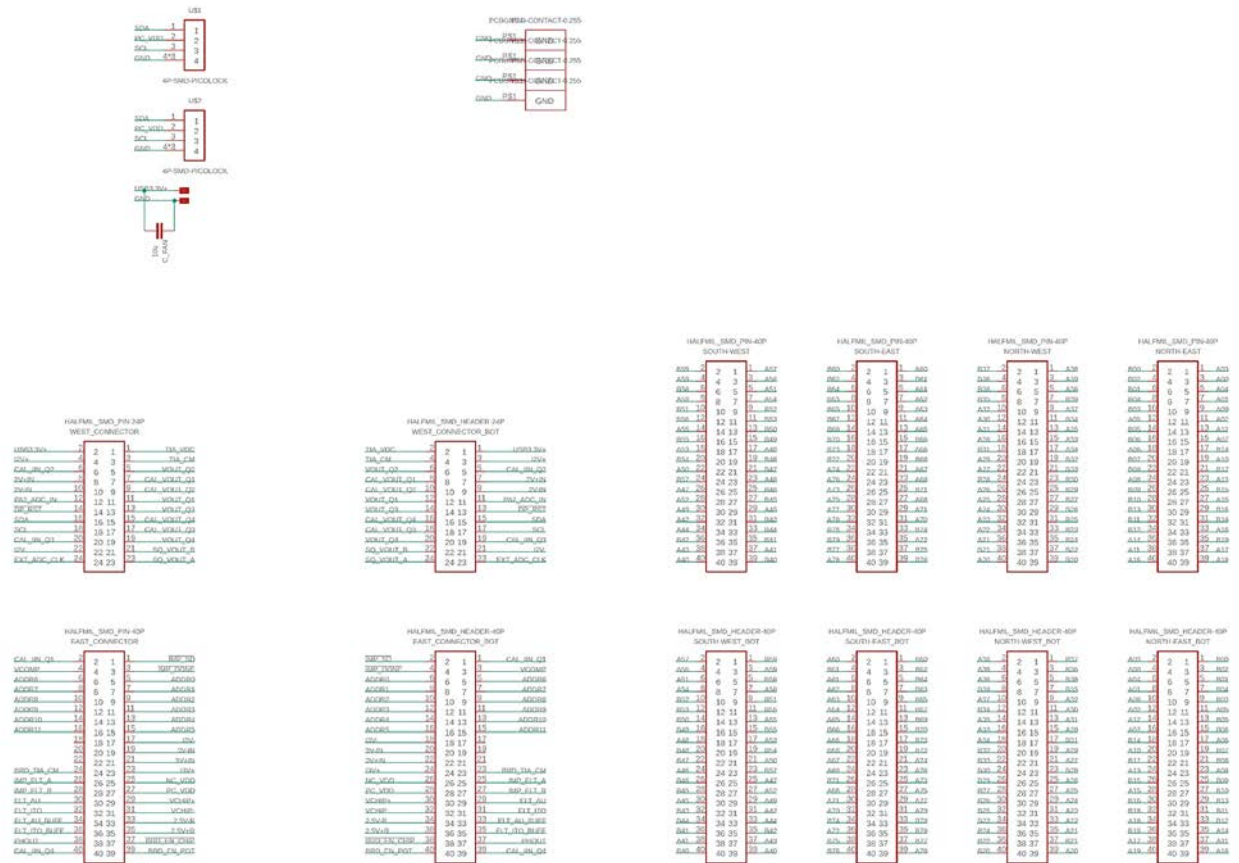


Control board schematic: USB Controller – version 3.

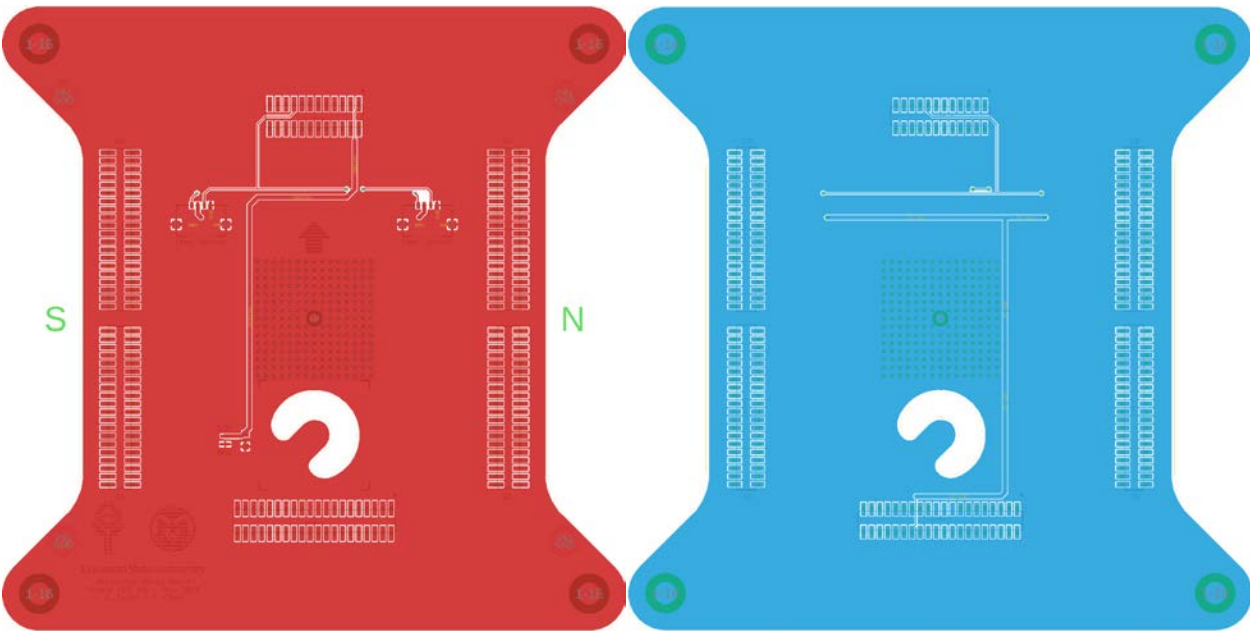


Control board PCB layout – version 3: top and bottom layers.



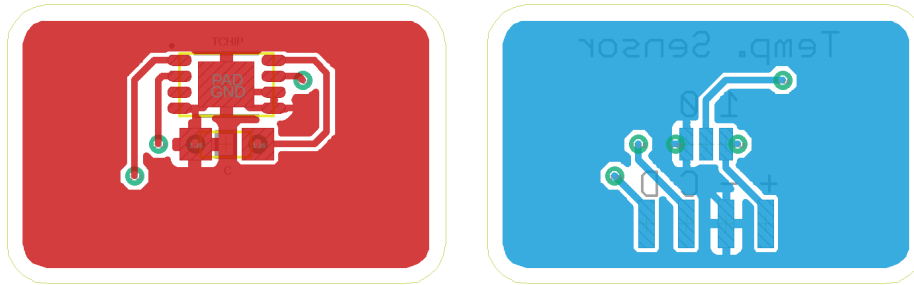
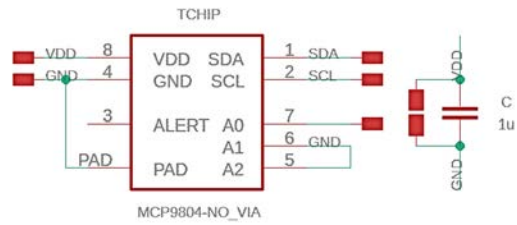


Shield board schematic – version 3.

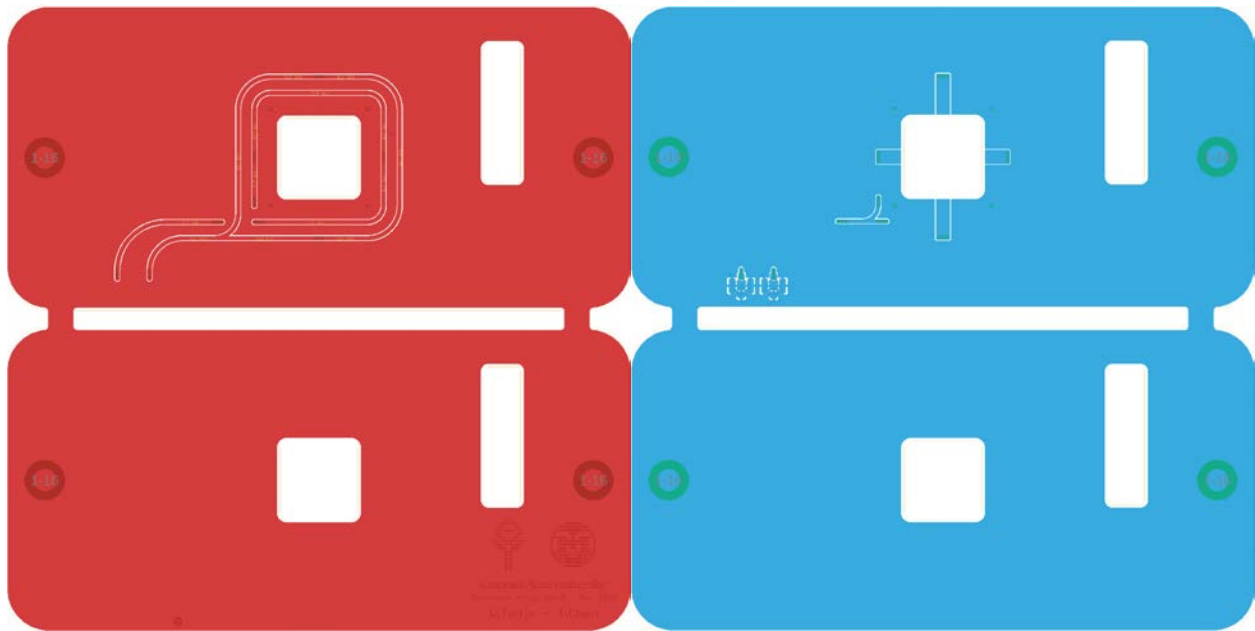


Shield board PCB layout – version 3: top and bottom layers.



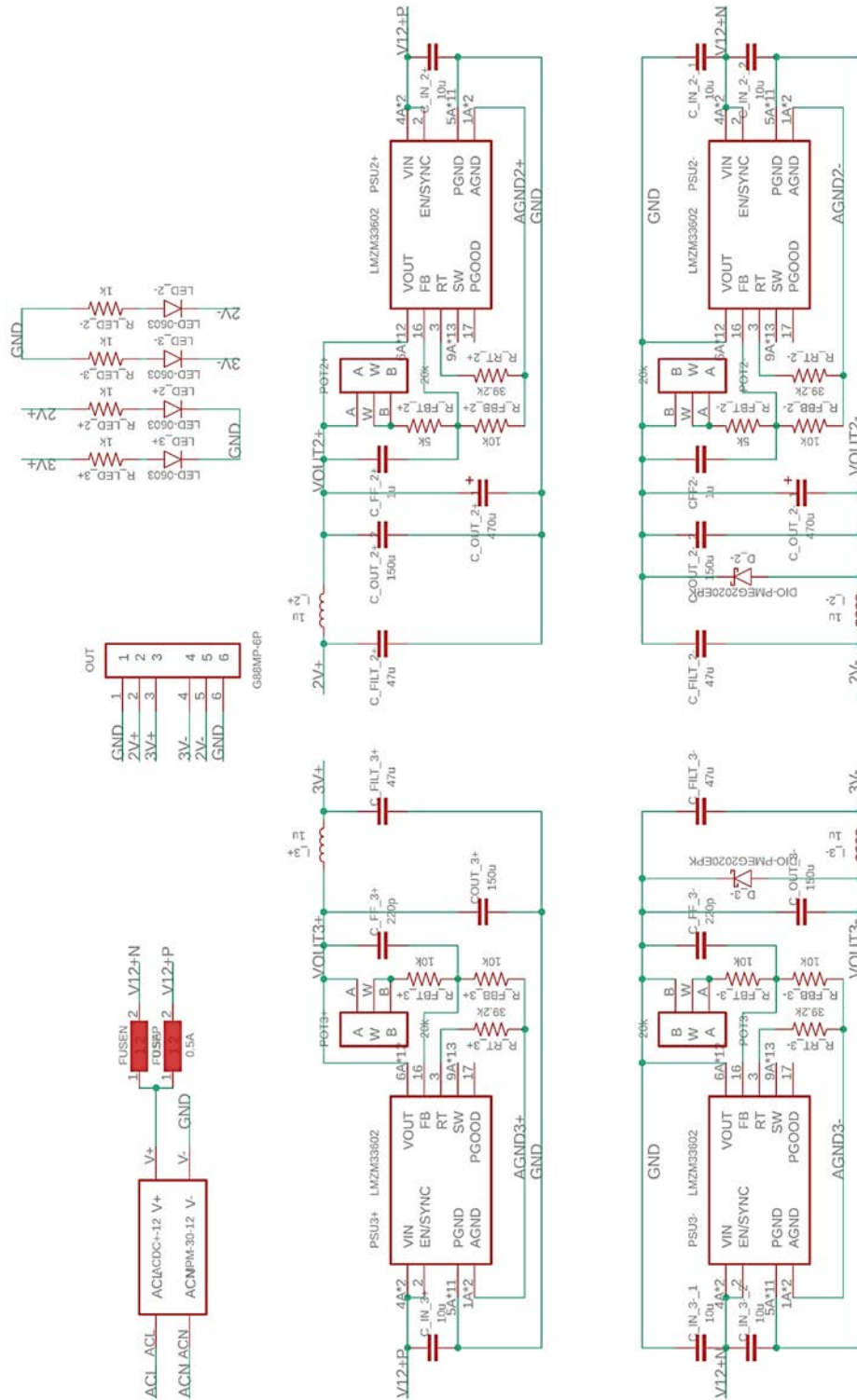


Temperature Sensor schematic, and top and bottom layers – version 3.



Pressure plate PCB layout – version 3: top and bottom layers.





Power supply unit schematic – version 3.



163



## B.5. Bill of Material – version 3

PSU	Function	Part Name	Part #	Qty
1	AC Power	AC Power Entry Modules 1A DUAL STAGE	FN9290-1-06	1
2	AC to DC+12V	AC/DC Power Modules 30W 12V 2.5A	MPM-30-12	1
3	DC/DC switching	Switching Voltage Regulators	LMZM33602RLRR	4
	Cout - 3V	150uF - 1210 - 6.3VDC	JMK325ABJ157MM-T	4
	Cout - 2V	470uF - 10VDC	UWT1A471MNL1GS	2
	Cin	10uF - 1210 - 16 VDC	TMK325BJ106KN-T	8
	Rfht	Trimmer Resistors - SMD 3MM - 20kΩ	3223J-1-203E	4
	Rrt	39.2kΩ - 0603 - 1%	CR0603-FX-3922ELF	4
	Cff	220pF - 10VDC	8.85012E+11	2
4	Power to board	MicroPwrPlus Conn3mm pitch,R/A 1x6pins	G88MPB06002C1EU	2
		MicroPwr Cbl housing 6 pin	G881H0621C3HR	2
		<b>Micro Pwr Cbl term 22-28AwgTinOverNick</b>	<b>G881C0232CEU</b>	<b>12</b>
		Flat Cables RIBBON CABLE 6WAY 24AWG	9.02221E+14	3
6	Conditioning	Fixed Inductors 0603 1.0uH 20% 1.7A	DFE18SAN1R0MG0L	4
7	Shared with other	Schottky Diodes	PMEG2020EPK,315	
8	<b>Additional</b>			
	Fuse	Resettable Fuses - PPTC 0.50A 15V 0.15ohm	652-MF-MSMF050-2	2
PCB1	Function	Part Name	Part #	Qty
1	Chip Socket	3M IC & Component Sockets 19X19 SKT	200-6319-9UN-1900	0
		3M IC & Component Sockets CONTACT	300-6300-CNA-0002B	0
2	LDO CHIP	OPA567AIRHGR	OPA567AIRHGR	2
	DigiPot	Digital Potentiometer ICs 256-pos	AD5142ABCPZ10-RL7	1
3	LDO Periph (-)	LDO Voltage Regulators -500mA	ADP7185ACPZN-R7	2
	VOUT = -0.5 V(1 + R1/R2)	Vout=-2.5V   R2=20kΩ <b>R1=80kΩ</b>	CPF0402B80K6E1	1
		Vout=-2.75V   R2=20kΩ <b>R1=90kΩ</b>	RR0510P-9092-D	1
	LDO Periph (+)	LDO Voltage Regulators 500mA	ADM7170ACPZ-R7	2
	VOUT = 1.2 V(1 + R1/R2)	Vout=2.5V   R2=20kΩ <b>R1=21.66kΩ</b>	RR0510P-223-D	1
		Vout=2.75V   R2=20kΩ <b>R1=25.83kΩ</b>	RR0510P-2612-D	1
	R2	20kΩ - 0402 - 0.1%	RR0510P-203-D	4
	Cin/Cout	4.7uF - 0402 - 6.3VDC	GRM155R60J475ME47D	15
4	Pull up R array	Resistor Networks & Arrays	EXB-Q16P103J	2
	DIP SW	HalfPitch DIP Swt SPST, 6 POS	TDA06H0JB1	1
5	Monitoring	25mΩ - 0603 - 1%	ERJ-3BWFR025V	2
		Resistor Networks & Arrays 0612 100K/1Kohms	CRASA100KF1K00TA	1
6	Shared with other	Op Amps 2-Channel, 10MHz	TLV9062IDSGR	
		Male - 20 Pins	20021121-00026T4LF	
		Male - 40 Pins	20021121-00040T4LF	
		Analog Comparators Quad	LMV339IRUCR	
		47uF - 0603 - 6.3VDC	ZRB18AR60J476ME01L	



		Schottky Diodes	PMEG2020EPK,315	
<b>PCB2</b>	<b>Function</b>	<b>Part Name</b>	<b>Part #</b>	<b>Qty</b>
1	Main MCU	ARM Microcontrollers - MCU	ATSAM3X8EA-AU	1
		Crystals AEC-Q200 12MHz 12pF	CX3225SB12000H0PSTC1	1
		Crystals SMD 32.768kHz 12.5pF	ST3215SB32768H5FPWAA	1
		Tactile Switches Side Actuated	B3U-3000P	1
		LDO Voltage Regulators Automotive 300mA	TLV70233QDSERQ1	1
		39Ω - 0402 - 5%	CRCW040239R0JNED	2
		Resettable Fuses - PPTC	0ZCM0020FF2G	1
2	Programming MCU	8-bit Microcontrollers - MCU AVR USB	ATMEGA16U2-MU	1
		Crystals AEC-Q200 16MHz 12pF	CX3225SB16000H0PSTC1	1
		Buffers & Line Drivers	SN74LVC1G125DSFR	1
		ESD Suppressors 0603 5 Volts	CG0603MLC-05E	5
		Resettable Fuses 0603 0.50A	0603L050SLYR	1
		P MOSFET Small Signal MOSFET	RV1C001ZPT2L	1
		22Ω - 0402 - 5%	ERJ-2GEJ220X	2
3	Addressing	Counter ICs 12-Bit Async Binary	SN74LV4040ARGYR	2
		Buffers & Line Drivers Tri-State Quad	SN74AHC125RGYR	2
		2-Input Pos- NAND Quad	SN74AUC00RGYR	1
		Inverters Hex	SN74AHC04RGYR	1
4	Reference Circuit	Voltage References 2.5V	LT6660HCDC-2.5	1
		0.1kΩ - 0603 - 0.1%	<a href="#">CPF0603B100RE</a>	1
		1.6kΩ - 0603 - 0.1%	<a href="#">CPF0603B1K6E1</a>	1
		3.6kΩ - 0603 - 0.1%	<a href="#">CPF0603B3K6E1</a>	1
5	Analog Mux	Multiplexer Switch ICs 25V Mux	MAX14778ETP+	1
			SN74CB3Q3253RGYR	6
6	Connector	USB Connectors uUSB B	47346-1001	3
		I/O Connectors 68 Ckt	71430-0006	1
7	Shared with other	Op Amps 2-Channel, 10MHz	TLV9062IDSGR	
		Analog Comparators Quad	LMV339IRUCR	
<b>Other</b>	<b>Function</b>	<b>Part Name</b>	<b>Part #</b>	<b>Qty</b>
1	Temp Monitor	Board Mount Temperature Sensors	TMP108AIYFFR	1
		Board Mount Temperature 12-bit	MCP9804-E/MC	2
2	General Opamp	Op Amps 2-Channel, 10MHz	TLV9062IDSGR	7
3	1/2 Pitch Connectors	Male - 20 Pins	20021121-00020T4LF	2
		Male - 40 Pins	20021121-00040T4LF	10
		Female - 20 Pins	20021321-00020C4LF	2
		Female - 40 Pins	20021321-00040T4LF	10
4	Digital	Analog Comparators Quad	LMV339IRUCR	4
5	ESD	Schottky Diodes	PMEG2020EPK,315	6



R	Function	Part Name	Part #	Qty
	General, Monitoring	1k $\Omega$ - 0603 - 0.1%	ERJ-PB3B1001V	25
		2k $\Omega$ - 0603 - 0.1%	ERJ-PB3B2001V	10
	General, PSU R feedback	10k $\Omega$ - 0603 - 0.1%	APC0603B10K0N	25
	Oscillator	1M $\Omega$ - 0402 - 1%	CRCW04021M00FKED	10
	Other	100k $\Omega$ - 0402 - 0.1%	RP73PF1E100KBTDF	10
		10k $\Omega$ - 0402 - 0.1%	CPF0402B10KE1	10
		1k $\Omega$ - 0402 - 0.1%	RP73PF1E1K0BTDF	10
C	Function	Part Name	Part #	Qty
	Filter,Cin/Cout	47uF - 0603 - 6.3VDC	ZRB18AR60J476ME01L	8
		10uF - 0603 - 6.3VDC	GRM188R60J106ME84D	100
		1uF - 0603 - 6.3VDC	GRM188R60J105KA01D	20
		0.1uF - 0603 - 25VDC	C1608X8R1E104M080AA	100
		10uF - 0402 - 4VDC	CC0402MRX5R4BB106	20
		1uF - 0402 - 25VDC	GRT155C80J105KE01D	20
		0.1uF - 0402 - 10VDC	GRM155R61A104MA01D	20
		0.01uF - 0402 - 10VDC	VJ0402Y103KXQCW1BC	20
		22pF - 0402 - 25VDC	GRM0225C1E220GA02L	20



## APPENDIX C: SOFTWARE - FIRMWARE AND SCRIPT

### C.1. MCU Firmware

```
1. #include <Wire.h>
2. // #include <Adafruit_GFX.h>
3. // #include <Adafruit_SSD1306.h>
4. //
5. // #define OLED_RESET 4
6. // Adafruit_SSD1306 display(OLED_RESET);

7. // set temperature and margin where FB loop is activated
8. // float tempSet = 37, tempMargin = 0.1;

9. // Enable pins
10. const int LDO_EN = 25; // LDO enable pin, Pin D0 MCU, digital D25 Arduino
11. const int Chip_ENb = 57, Pot_EN = 56; // Chip and Potentiostat enable pins
12. const int ThreeElt_EN = 55; // Three Electrode System enable pins
13. const int Gnd_Au_ENb = 54; // Ground Au electrode for pH Sensing
14. const int Ext_Con1 = 2, Ext_Con2 = 7; // External shutdown, enable on pin 144 and 134

15. // voltage sensing pins
16. const int MuxChipMSB = 19, MuxChipLSB = 22, MuxChip = 35, MuxCal = 37, MuxPerph = 39, MuxLSB = 38, MuxMSB = 40; // MUX Control pins
17. const int potRstPin = 27;

18. // addressing pins
19. const int PC12 = 51, PC13 = 50, PC14 = 49, PC15 = 48, PC16 = 47, PC17 = 46, PC18 = 45, PC19 = 44; // address pins
20. const int PC24 = 6; // address mode reset
21. const int PC25 = 5, PC26 = 4; // address mode LSB and MSB
22. const int PC28 = 3; // address mode enable
23. const int PC29 = 10; // address interrupt

24. // positive LDO
25. // wiper position of positive 200 to 250 (1.5V to 2.5V)
26. const int RPmin = 206;
27. const int RPmax = 255;
28. int RP = RPmin;

29. // negative LDO
30. // wiper position of negative 0 to 50 (-2.5V to -1.5V)
31. const int RNmin = 49;
32. const int RNmax = 1;
33. int RN = RNmin;

34. // ADC constants
35. int const ADCsamples = 100; // number of samples
36. double ADCvalues[ADCsamples]; // capture values of acquisition here
37. double ADCsum = 0, ADCmean = 0, ADCresult = 0, ADCcalcd = 0;

38. int const MonChar = 112; // number of character for monitoring purpose, serial transfer
39. char ADCcont[MonChar], ADCcontTemp[MonChar];

40. // Address variables
41. int electrodeTime = 64; // in microseconds; range = 8 to 256 with 8 increment, 30 to 1 fps
42. double clkScale = 2.625;
43. int freqClkL = electrodeTime * clkScale; // Least significant clock
44. int freqClkM = electrodeTime * clkScale * 64; // Most significant clock
45. int dutyClkL = freqClkL / 2;
46. int dutyClkM = freqClkM / 2;

47. // Temperature
48. byte temp1, temp2;
49. double temp;

50. // Cyclic Voltammetry
51. int CVvmax = 0, CVvmin = 0, CVdacrater = 0, CVStepNum = 0, CVVal = 0;

52. // other
53. String serialReadStr = "", serialID = "", serialVal = "", CVTemp = "";
54. int ID, Val;

55. void setup() {
56.   Serial.begin(115200);
57.   Wire.begin();
58.   analogWriteResolution(12);

59.   // setup enable values for LDOs
60.   pinMode(LDO_EN, OUTPUT);
61.   digitalWrite(LDO_EN, HIGH);
62.   // pinMode(LDO_EN, INPUT); // pulled down, disabled
63.   vChipSet(); // initialize to close to 0V
```



```

64. initPot(); // initialize digital potential for 1st time.

65. // potentiometer reset pin to high, active low
66. pinMode(potRstPin, OUTPUT);
67. digitalWrite(potRstPin, HIGH);

68. // chip and potentiostat enable
69. pinMode(Chip_ENb, OUTPUT);
70. pinMode(Pot_EN, OUTPUT);
71. pinMode(ThreeElt_EN, OUTPUT);
72. pinMode(Gnd_Au_ENb, OUTPUT);
73. digitalWrite(Chip_ENb, HIGH);
74. digitalWrite(Pot_EN, HIGH);
75. digitalWrite(ThreeElt_EN, LOW);
76. digitalWrite(Gnd_Au_ENb, HIGH);

77. // mux control pins
78. pinMode(MuxChip, OUTPUT);
79. pinMode(MuxCal, OUTPUT);
80. pinMode(MuxPerph, OUTPUT);
81. pinMode(MuxMSB, OUTPUT);
82. pinMode(MuxLSB, OUTPUT);
83. pinMode(MuxChipMSB, OUTPUT);
84. pinMode(MuxChipLSB, OUTPUT);

85. digitalWrite(MuxChip, HIGH);
86. digitalWrite(MuxCal, LOW);
87. digitalWrite(MuxPerph, LOW);
88. digitalWrite(MuxMSB, LOW);
89. digitalWrite(MuxLSB, LOW);
90. digitalWrite(MuxChipMSB, LOW);
91. digitalWrite(MuxChipLSB, LOW);

92. // address pins, all controlled by counter
93. pinMode(PC12, INPUT); pinMode(PC13, INPUT); pinMode(PC14, INPUT); pinMode(PC15, INPUT);
94. pinMode(PC16, INPUT); pinMode(PC17, INPUT); pinMode(PC18, INPUT); pinMode(PC19, INPUT);

95. //pinMode(PC24, INPUT_PULLUP);
96. pinMode(PC25, OUTPUT); // mode LSB
97. pinMode(PC26, OUTPUT); // mode MSB
98. pinMode(PC28, OUTPUT); // mode enable
99. pinMode(PC29, INPUT); // addr interrupt

100. // set to default: all address controlled by counter
101. digitalWrite(PC28, LOW);
102. digitalWrite(PC25, HIGH);
103. digitalWrite(PC26, HIGH);

104. // set external enable and shutdown to HiZ
105. pinMode(Ext_Con1, INPUT);
106. pinMode(Ext_Con2, INPUT);

107. // ADC setups
108. REG_ADC_MR = 272105600; //set free running mode on ADC 272105600
109. // REG_ADC_CR = 2;
110. REG_ADC_CHER = 1; //enable ADC on pin A2(pin), pin A7(arduino), pin 85(chip)

111. // DAC setup
112. analogWrite(DAC0, 2047); // Potentiostat Voltage = 0V
113. analogWrite(DAC1, 2047); // TIA Common Mode Voltage = 0V

114. // Address: PWM setups
115. int32_t mask_PWM_pin = 3 << 21; // select channel P2 (PC21 and PC22) Arduino DUE pin 9 and 8
116. REG_PIOC_PDR |= mask_PWM_pin; // activate peripheral functions for pin (disables all PIO functionality)
117. REG_PIOC_ABSR |= mask_PWM_pin; // choose peripheral option B
118. // REG_PIOC_PUER |= mask_PWM_pin; // pull up resistor enable

119. int32_t mask_PWM_clk_en = 1 << 4;
120. REG_PMC_PCER1 |= mask_PWM_clk_en; // activate clock for PWM controller (PID36 set to 1)

121. REG_PWM_WPCR = 0x50574DFC; // remove PWM write protection

122. REG_PWM_CLK = 0; // choose clock rate, full MLCK

123. int32_t mask_PWM_clk_scale = 5; // prescale: MCLK/2^5 = 84Mhz/32 = 2.625Mhz

124. // REG_PWM_CMR0 = 1 << 9; // select clock and polarity for PWM channel 0 (CPOL = 0)
125. // REG_PWM_CMR0 |= mask_PWM_clk_scale;
126. // REG_PWM_CPRD0 = freqClkL; // initialize PWM period -> T = value/84Mhz (value: up to 16bit),
    value=10 -> 8.4Mhz
127. // REG_PWM_CDTY0 = dutyClkL;

128. REG_PWM_CMR4 = 1 << 9; // select clock and polarity for PWM channel 4 (CPOL = 0)
129. REG_PWM_CMR4 |= mask_PWM_clk_scale;
130. REG_PWM_CPRD4 = freqClkL;
131. REG_PWM_CDTY4 = dutyClkL;

132. REG_PWM_CMR5 = 1 << 9; // select clock and polarity for PWM channel 5 (CPOL = 0)

```



```

133. REG_PWM_CMR5 |= mask_PWM_clk_scale;
134. REG_PWM_CPRD5 = freqClkM;
135. REG_PWM_CDTY5 = dutyClkM;

136. // int32_t mask_PWM_clk_sync = 49;           // synchronize PWM channel 4 and 5
137. // REG_PWM_SCM = mask_PWM_clk_sync;

138. // Interrupt for address enable/disable
139. attachInterrupt(6, clkON, FALLING);
140. attachInterrupt(10, clkOFF, FALLING);
141. }

142. ///////////////////////////////////////////////////
143. /////////////////////////////////////////////////// MAIN LOOP ///////////////////////////////////////////////////
144. void loop() {
145. while (Serial.available() > 0) {
146. char serialRead = Serial.read();
147. serialReadStr += serialRead;

148. if (serialRead == '\n') {
149. serialID = serialReadStr;
150. serialVal = serialReadStr;

151. serialID.remove(1); //grab the first character
152. serialVal.remove(0, 1); //remove the first character

153. ID = serialID.toInt();
154. Val = serialVal.toInt();

155. // Serial.println(ID);
156. // Serial.println(Val);

157. // Case 1: CHIP enable
158. if (ID == 1) {
159. switch (Val) {
160. case 0: digitalWrite(Chip_ENb, HIGH); break; //enable
161. case 1: digitalWrite(Chip_ENb, LOW); break; //disable
162. default: break;
163. }
164. }

165. // Case 2: vChip adjust
166. else if (ID == 2) {
167. RP = (Val - 150) / 2 + RPmin;
168. RN = -1 * (Val - 150) / 2 + RNmin;
169. // Serial.println(RP);
170. // Serial.println(RN);
171. vChipSet();
172. }

173. // Case 3: Potentiostat enables
174. else if (ID == 3) {
175. switch (Val) {
176. case 0: digitalWrite(Pot_EN, LOW); digitalWrite(ThreeElt_EN, LOW); break;
177. case 1: digitalWrite(Pot_EN, LOW); digitalWrite(ThreeElt_EN, HIGH); break;
178. case 2: digitalWrite(Pot_EN, HIGH); digitalWrite(ThreeElt_EN, LOW); break;
179. case 3: digitalWrite(Pot_EN, HIGH); digitalWrite(ThreeElt_EN, HIGH); break;
180. default: break;
181. }
182. }

183. // Case 4: vPot adjust Amperometry
184. else if (ID == 4) {
185. analogWrite(DAC0, Val);
186. }

187. // Case 5: vTIACm adjust
188. else if (ID == 5) {
189. analogWrite(DAC1, Val);
190. }

191. // Case 6: Set address for CV
192. else if (ID == 6) {
193. // set everything to be controlled by MCU
194. digitalWrite(PC28, HIGH);
195. digitalWrite(PC25, LOW);
196. digitalWrite(PC26, LOW);

197. pinMode(PC12, OUTPUT); pinMode(PC13, OUTPUT); pinMode(PC14, OUTPUT); pinMode(PC15, OUTPUT);
198. pinMode(PC16, OUTPUT); pinMode(PC17, OUTPUT); pinMode(PC18, OUTPUT); pinMode(PC19, OUTPUT);

199. REG_PIOC_CODR = 0b11111111 << 12; // clear all address pins to LOW

200. // case for 16 different location of electrodes bit 4(PC14), 5(PC15), 10(PC18), and 11(PC19), bit 2,3,8,9 are set constant to 0.
201. // set 16,17, 20, 21 to adjust the other locations of the electrodes.
202. switch (Val) {
203. case 21: REG_PIOC_SODR = 0b00 << 18; REG_PIOC_SODR = 0b00 << 14; break;
204. case 22: REG_PIOC_SODR = 0b00 << 18; REG_PIOC_SODR = 0b01 << 14; break;

```



```

205. case 23: REG_PIOC_SODR = 0b00 << 18; REG_PIOC_SODR = 0b10 << 14; break;
206. case 24: REG_PIOC_SODR = 0b00 << 18; REG_PIOC_SODR = 0b11 << 14; break;
207. case 25: REG_PIOC_SODR = 0b01 << 18; REG_PIOC_SODR = 0b00 << 14; break;
208. case 26: REG_PIOC_SODR = 0b01 << 18; REG_PIOC_SODR = 0b01 << 14; break;
209. case 27: REG_PIOC_SODR = 0b01 << 18; REG_PIOC_SODR = 0b10 << 14; break;
210. case 28: REG_PIOC_SODR = 0b01 << 18; REG_PIOC_SODR = 0b11 << 14; break;
211. case 29: REG_PIOC_SODR = 0b10 << 18; REG_PIOC_SODR = 0b00 << 14; break;
212. case 30: REG_PIOC_SODR = 0b10 << 18; REG_PIOC_SODR = 0b01 << 14; break;
213. case 31: REG_PIOC_SODR = 0b10 << 18; REG_PIOC_SODR = 0b10 << 14; break;
214. case 32: REG_PIOC_SODR = 0b10 << 18; REG_PIOC_SODR = 0b11 << 14; break;
215. case 33: REG_PIOC_SODR = 0b11 << 18; REG_PIOC_SODR = 0b00 << 14; break;
216. case 34: REG_PIOC_SODR = 0b11 << 18; REG_PIOC_SODR = 0b01 << 14; break;
217. case 35: REG_PIOC_SODR = 0b11 << 18; REG_PIOC_SODR = 0b10 << 14; break;
218. case 36: REG_PIOC_SODR = 0b11 << 18; REG_PIOC_SODR = 0b11 << 14; break;
219. default: break;
220. }
221. }

222. // Case 7: Set potentiostat for CV, one cycle of sweep
223. else if (ID == 7) {
224. // vmin|vmax|dacrte, 3 digits each
225. CVTemp = serialVal.substring(0, 3); //grab the first three characters
226. CVvmin = CVTemp.toInt(); // unit: 1/10 V; example: 203 is 0V | 303 is 1V
227. CVTemp = serialVal.substring(3, 6); // unit: 1/10 V
228. CVvmax = CVTemp.toInt();
229. CVTemp = serialVal.substring(6, 10);
230. CVdacrte = CVTemp.toInt(); // unit: us

231. CVStepNum = (CVvmax - CVvmin) * 10;
232. //Serial.println(CVStepNum);

233. // sweep up
234. for (int i = 0; i <= CVStepNum; i++) {
235. CVVal = CVvmin * 10 + i;
236. analogWrite(DAC0, CVVal);
237. delayMicroseconds(CVdacrte);
238. }

239. // sweep down
240. for (int i = 0; i <= CVStepNum; i++) {
241. CVVal = CVvmax * 10 - i;
242. analogWrite(DAC0, CVVal);
243. delayMicroseconds(CVdacrte);
244. }
245. // end of cycle

246. // set to default: all address controlled by counter
247. digitalWrite(PC28, LOW);
248. digitalWrite(PC25, HIGH);
249. digitalWrite(PC26, HIGH);

250. pinMode(PC12, INPUT); pinMode(PC13, INPUT); pinMode(PC14, INPUT); pinMode(PC15, INPUT);
251. pinMode(PC16, INPUT); pinMode(PC17, INPUT); pinMode(PC18, INPUT); pinMode(PC19, INPUT);
252. }

253. // Case 9: Voltage & Temperature Monitoring
254. else if (ID == 9) {
255. // initialize memory equally to length of MonChar
256. memset(ADCcont, 0, MonChar);
257. memset(ADCcontTemp, 0, MonChar);

258. //vCHIP-
259. digitalWrite(MuxMSB, LOW);
260. digitalWrite(MuxLSB, LOW);
261. digitalWrite(MuxPerph, HIGH);
262. digitalWrite(MuxCal, LOW);
263. digitalWrite(MuxChip, LOW);
264. digitalWrite(MuxChipMSB, LOW);
265. digitalWrite(MuxChipLSB, HIGH);
266. delay(5);
267. ADCPerph();

268. //vCHIP+
269. digitalWrite(MuxLSB, HIGH);
270. delay(5);
271. ADCPerph();

272. //vConst+
273. digitalWrite(MuxMSB, HIGH);
274. digitalWrite(MuxLSB, LOW);
275. delay(5);
276. ADCPerph();

277. //vConst-
278. digitalWrite(MuxLSB, HIGH);
279. delay(5);
280. ADCPerph();

```



```

281. //vIN3+
282. digitalWrite(MuxMSB, LOW);
283. digitalWrite(MuxLSB, LOW);
284. digitalWrite(MuxPerph, LOW);
285. delay(5);
286. ADCPerph();

287. //vIN3-
288. digitalWrite(MuxLSB, HIGH);
289. delay(5);
290. ADCPerph();

291. //RE
292. digitalWrite(MuxMSB, HIGH);
293. digitalWrite(MuxLSB, LOW);
294. delay(5);
295. ADCPerph();

296. //CE
297. digitalWrite(MuxLSB, HIGH);
298. delay(5);
299. ADCPerph();

300. //TIA_VDC
301. digitalWrite(MuxChipMSB, HIGH);
302. digitalWrite(MuxChipLSB, LOW);
303. delay(5);
304. ADCTia();

305. //TIA_CM
306. digitalWrite(MuxChipLSB, HIGH);
307. delay(5);
308. ADCTia();

309. //vIN2-
310. digitalWrite(MuxMSB, HIGH);
311. digitalWrite(MuxLSB, LOW);
312. digitalWrite(MuxCal, HIGH);
313. digitalWrite(MuxChipMSB, LOW);
314. digitalWrite(MuxChipLSB, LOW);
315. delay(10);
316. ADCActual();

317. //vIN2+
318. digitalWrite(MuxLSB, HIGH);
319. delay(5);
320. ADCActual();

321. //i2V+
322. digitalWrite(MuxMSB, LOW);
323. digitalWrite(MuxLSB, LOW);
324. delay(5);
325. ADCCurrent1();

326. //i2V-
327. digitalWrite(MuxLSB, HIGH);
328. delay(5);
329. ADCCurrent1();

330. //i3V-
331. digitalWrite(MuxCal, LOW);
332. delay(5);
333. ADCCurrent2();

334. //i3V+
335. digitalWrite(MuxLSB, LOW);
336. delay(5);
337. ADCCurrent2();

338. //temperature LDO Positive
339. Wire.beginTransmission(0x1D);
340. Wire.write(B00000101);
341. Wire.endTransmission();
342. Wire.requestFrom(0x1F, 2);
343. tempRead();

344. //temperature LDO Negative
345. Wire.beginTransmission(0x1F);
346. Wire.write(B00000101);
347. Wire.endTransmission();
348. Wire.requestFrom(0x1D, 2);
349. tempRead();

350. //temperature Q12
351. Wire.beginTransmission(0x18);
352. Wire.write(B00000101);
353. Wire.endTransmission();
354. Wire.requestFrom(0x18, 2);

```



```

355. tempRead();

356. //temperature Q34
357. Wire.beginTransmission(0x19);
358. Wire.write(B00000101);
359. Wire.endTransmission();
360. Wire.requestFrom(0x19, 2);
361. tempRead();

362. Serial.println(ADCcont);
363. // Serial.println(&ADCcont[66]);
364. // ADCcont format: vCHIP-|vCHIP+|vConst+|vConst-|vIN3+|vIN3-|RE|CE|TIA_VDC|TIA_CM|vIN2-|vIN2+|i2V-|i2V+|i3V-|i3V+|tLDO+|tLDO-
// |tQ12|tQ34|
365. // char length :
6.....|6.....|6.....|6.....|6.....|6.....|6.....|6.....|6.....|5...|5...|5...|5...|5...|5...|5...|5...|
366. // char location :
6.....|7.....|13.....|19.....|25...|31...|37|43|49.....|55.....|61...|67...|73...|78...|83...|88...|93...|98...|103.|108.|

367. // digitalWrite(MuxChip, HIGH); // remove after testing with Caleb's board
368. }
369. serialReadStr = "";
370. }
371. // end of case for serial ID selection
372. }
373. // end of case for serial read
374. }
375. // end of loop

376. //////////// END OF MAIN LOOP ////////////
377. ///////////////////////////////////////////

378. void ADCPerph (void) {
379. // acquire values
380. for (int i = 0; i < ADCsamples; i++) {
381. while ((ADC->ADC_ISR & 1) == 0); // wait for conversion
382. ADCvalues[i] = ADC->ADC_CDR[0]; //get values
383. }

384. // calculate and print values
385. ADCsum = 0;
386. for (int i = 0; i < ADCsamples; i++) {
387. ADCsum = ADCsum + ADCvalues[i];
388. ADCmean = ADCsum / ADCsamples;
389. }
390. ADCresult = ADCmean * 2.5 / 4095;
391. ADCcalcd = 3 * ADCresult - 4; // Vin=3Vout-4 OR Vout=(Vin+4)/3

392. if (ADCcalcd < 0) {
393. ADCcalcd = -1 * ADCcalcd;
394. }
395. sprintf(ADCcont, "%s%0.4f", ADCcontTemp, ADCcalcd); // concatenate calculated value to ADCcont
396. strncpy(ADCcontTemp, ADCcont, MonChar); // copy contents to Temp
397. }

398. void ADCActual(void) {
399. // acquire values
400. for (int i = 0; i < ADCsamples; i++) {
401. while ((ADC->ADC_ISR & 1) == 0); // wait for conversion
402. ADCvalues[i] = ADC->ADC_CDR[0]; //get values
403. }

404. // calculate and print values
405. ADCsum = 0;
406. for (int i = 0; i < ADCsamples; i++) {
407. ADCsum = ADCsum + ADCvalues[i];
408. ADCmean = ADCsum / ADCsamples;
409. }
410. ADCresult = ADCmean * 2.5 / 4095;
411. ADCcalcd = ADCresult;
412. sprintf(ADCcont, "%s%0.4f", ADCcontTemp, ADCcalcd); // concatenate calculated value to ADCcont
413. strncpy(ADCcontTemp, ADCcont, MonChar); // copy contents to Temp
414. }

415. void ADCCurrent1(void) { // for 2V Power supply
416. // acquire values
417. for (int i = 0; i < ADCsamples; i++) {
418. while ((ADC->ADC_ISR & 1) == 0); // wait for conversion
419. ADCvalues[i] = ADC->ADC_CDR[0]; //get values
420. }
421. // calculate and print values
422. ADCsum = 0;
423. for (int i = 0; i < ADCsamples; i++) {
424. ADCsum = ADCsum + ADCvalues[i];
425. ADCmean = ADCsum / ADCsamples;
426. }
427. ADCresult = ADCmean * 2.5 / 4095;
428. ADCcalcd = ADCresult * 40 / 6; //convert to current
429. // ADCcalcd = (ADCcalcd * 1) + 0.025; // Calibrate

```



```

430. sprintf(ADCcont, "%s%0.3f", ADCcontTemp, ADCcalcd);           // concatenate calculated value to ADCcont
431. strncpy(ADCcontTemp, ADCcont, MonChar);
432. }

433. void ADCCurrent2(void) { // for 3V Power supply
434. // acquire values
435. for (int i = 0; i < ADCsamples; i++) {
436. while ((ADC->ADC_ISR & 1) == 0); // wait for conversion
437. ADCvalues[i] = ADC->ADC_CDR[0]; //get values
438. }
439. // calculate and print values
440. ADCsum = 0;
441. for (int i = 0; i < ADCsamples; i++) {
442. ADCsum = ADCsum + ADCvalues[i];
443. ADCmean = ADCsum / ADCsamples;
444. }
445. ADCresult = ADCmean * 2.5 / 4095;
446. ADCcalcd = ADCresult * 3 / 2; //hardware gain compensation
447. ADCcalcd = ADCcalcd * 4 / 6; //convert to current
448. // ADCcalcd = (ADCcalcd * 1) + 0.025; //Calibrate
449. sprintf(ADCcont, "%s%0.3f", ADCcontTemp, ADCcalcd); // concatenate calculated value to ADCcont
450. strncpy(ADCcontTemp, ADCcont, MonChar);
451. }

452. void ADCTia(void) {
453. // acquire values
454. for (int i = 0; i < ADCsamples; i++) {
455. while ((ADC->ADC_ISR & 1) == 0); // wait for conversion
456. ADCvalues[i] = ADC->ADC_CDR[0]; //get values
457. }
458. // calculate and print values
459. ADCsum = 0;
460. for (int i = 0; i < ADCsamples; i++) {
461. ADCsum = ADCsum + ADCvalues[i];
462. ADCmean = ADCsum / ADCsamples;
463. }
464. ADCresult = ADCmean * 2.5 / 4095;
465. ADCcalcd = (ADCresult / 2 * 3) - 0.4;
466. if (ADCcalcd < 0) {
467. ADCcalcd = -1 * ADCcalcd;
468. }
469. sprintf(ADCcont, "%s%0.4f", ADCcontTemp, ADCcalcd); // concatenate calculated value to ADCcont
470. strncpy(ADCcontTemp, ADCcont, MonChar);
471. }

472. void tempRead(void) {
473. temp1 = Wire.read(); // first received byte stored here
474. temp2 = Wire.read();
475. temp = byteToTemp(temp1, temp2);
476. if (temp > 99) {
477. temp = 99.99;
478. }
479. sprintf(ADCcont, "%s%0.2f", ADCcontTemp, temp); // concatenate calculated value to ADCcont
480. strncpy(ADCcontTemp, ADCcont, MonChar);
481. }

482. double byteToTemp(byte x, byte y) {
483. double temp;
484. temp = ((x - 192) * 16) + (y / 16.000);
485. return temp;
486. }

487. void vChipSet (void) {
488. // digipot: 0x2F
489. Wire.beginTransmission(0x2F);
490. Wire.write(B00010000);
491. Wire.write(RP);
492. Wire.endTransmission();

493. Wire.beginTransmission(0x2F);
494. Wire.write(B00010001);
495. Wire.write(RN);
496. Wire.endTransmission();
497. }

498. void clkON() {
499. REG_PWM_ENA = 3 << 4;
500. // REG_PWM_ENA = 49;
501. }

502. void clkOFF() {
503. REG_PWM_DIS = 3 << 4;
504. // REG_PWM_DIS = 49;
505. }

506. //void tempFB(void) {
507. //
508. // // temp too low

```



```

509. // if (tempAve < tempSet - tempMargin) {
510. //     R1P = R1P + 1;
511. //     R1N = R1N + 1;
512. //     // hardlimit value to maximum
513. //     if (R1P > R1maxP) {
514. //         R1P = R1maxP;
515. //     }
516. //     if (R1N > R1maxN) {
517. //         R1N = R1maxN;
518. //     }
519. // }
520. //
521. // // temp too high
522. // else if (tempAve > tempSet + tempMargin) {
523. //     R1P = R1P - 1;
524. //     R1N = R1N - 1;
525. //     // hardlimit value to minimum
526. //     if (R1P < R1minP) {
527. //         R1P = R1minP;
528. //     }
529. //     if (R1N < R1minN) {
530. //         R1N = R1minN;
531. //     }
532. // }
533. //
534. // Wire.beginTransaction(0x2F);
535. // Wire.write(B00010000);
536. // Wire.write(R1P);
537. // Wire.endTransmission();
538. //
539. // Wire.beginTransaction(0x2F);
540. // Wire.write(B00010010);
541. // Wire.write(R1P);
542. // Wire.endTransmission();
543. //
544. // Wire.beginTransaction(0x20);
545. // Wire.write(B00010000);
546. // Wire.write(R1N);
547. // Wire.endTransmission();
548. //
549. // Wire.beginTransaction(0x20);
550. // Wire.write(B00010010);
551. // Wire.write(R1N);
552. // Wire.endTransmission();
553. //}
554. //void voltageRead(void) {
555. // Aread0 = analogRead(A0);
556. // Aread1 = analogRead(A1);
557. // Aread2 = analogRead(A2);
558. // Aread3 = analogRead(A3);
559. //
560. // // convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 2.5V):
561. // AV0 = Aread0 * (2.5 / 1023.0) * 3 / 2;
562. // AV1 = Aread1 * (2.5 / 1023.0) * 2;
563. // AV2 = Aread2 * (2.5 / 1023.0) * 3 / 2;
564. // AV3 = Aread3 * (2.5 / 1023.0) * 2;
565. //}

566. void initPot(void) {
567. // initialize DigiPot Resistors: skip for normal operation
568. delay(250);
569. Wire.beginTransaction(0x2F);
570. Wire.write(B10000000);
571. Wire.write(RP);
572. Wire.endTransmission();
573. delay(250);
574. Wire.beginTransaction(0x2F);
575. Wire.write(B10000010);
576. Wire.write(RN);
577. Wire.endTransmission();
578. delay(250);

579. digitalWrite(potRstPin, LOW);
580. delay(250);
581. digitalWrite(potRstPin, HIGH);
582. }

```



## C.2. MATLAB Real-Time GUI

### C.2.1. MainGUI.m

```
1. function varargout = MainGUI(varargin)
2. % MainGUI MATLAB code for MainGUI.fig

3. % Begin initialization code - DO NOT EDIT
4. gui_Singleton = 1;
5. gui_State = struct('gui_Name',       mfilename, ...
6. 'gui_Singleton',   gui_Singleton, ...
7. 'gui_OpeningFcn', @MainGUI_OpeningFcn, ...
8. 'gui_OutputFcn',  @MainGUI_OutputFcn, ...
9. 'gui_LayoutFcn',  [], ...
10. 'gui_Callback',   []);
11. if nargin && ischar(varargin{1})
12.     gui_State.gui_Callback = str2func(varargin{1});
13. end

14. if nargout
15.     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
16. else
17.     gui_mainfcn(gui_State, varargin{:});
18. end
19. % End initialization code - DO NOT EDIT

20. % --- Executes just before MainGUI is made visible.
21. function MainGUI_OpeningFcn(hObject, eventdata, handles, varargin)
22. % This function has no output args, see OutputFcn.
23. % hObject    handle to figure
24. % eventdata  reserved - to be defined in a future version of MATLAB
25. % handles     structure with handles and user data (see GUIDATA)
26. % varargin    command line arguments to MainGUI (see VARARGIN)

27. % Choose default command line output for MainGUI
28. handles.output = hObject;

29. f = waitbar(0,'Start');
30. waitbar(0,f,'Initializing Constant and Variables');

31. handles.serialCondition = 0;
32. handles.monitorCondition = 0;
33. handles.DAQCondition = 0;
34. handles.DAQConditionV = 0;

35. % buttons initial values
36. handles.ChipEnable = 0;
37. handles.PotentiostatEnableb = 0;
38. handles.ThreeEEEnable = 0;
39. handles.CVSingle = 1;
40. handles.CVBase = 0;
41. handles.CVBaseSig = 0;

42. handles.vChipSet = 1.65;
43. handles.vPotSet = 0;
44. handles.vTIACmSet = 0;

45. handles.measCtrElt = 0;
46. handles.measRefElt = 0;

47. handles.enableAmp = 0;
48. handles.enableCV = 0;

49. % Data Aquisition constants
50. % DAQ full scale range
51. % options: 0-0.1, 0-0.4, 0-0.5, 0-1, 0-2, 0-2.5, 0-4, and the bipolar of each.
52. handles.FSminV = -1;
53. handles.FSmaxV = 1;

54. handles.sampleRate = 100000; % per channel per seconds
55. handles.CVsampleRate = 70000; % sample rate max for CV
56. handles.CVsamplePerTrigger = 7000;
57. handles.electrodeTime = 64;
58. handles.numOfElectrode = 4096;
59. handles.samplePerTrigger = round(handles.electrodeTime*1e-6*handles.numOfElectrode*handles.sampleRate)+2;
60. % samples per frame, 2 for adjusting discrepancy, unknown source

61. % Amperometry initial variables
62. handles.numOffFrame = 10;
63. handles.numOffFPS = 2;
64. handles.heatmapScale = 1; % in nano amperes;
65. set(handles.frameNumber,'String',handles.numOffFrame);
66. set(handles.framePerSecond,'String',handles.numOffFPS);
67. set(handles.runTime,'String',num2str((handles.numOffFrame-1)*(1/handles.numOffFPS)/60,'%1f')); % show calculated runtime
68. set(handles.heatmapMin,'String',handles.heatmapScale);
```



```

69. set(handles.heatmapMax,'String',handles.heatmapScale);

70. % Amperometry sweep initial variables
71. handles.AmpMode = 1;
72. handles.AmpResetPer = 3; % reset period, output equal to 0V
73. handles.AmpSamplePer = 5; % sampling period
74. handles.AmpRecoverPer = 5; % recover period
75. handles.AmpVmin = -0.5; % Amperometry minimum, volt
76. handles.AmpVmax = 0.5; % Amperometry maximum, volt
77. handles.AmpVminTemp = handles.AmpVmin; % temporary
78. handles.AmpVmaxTemp = handles.AmpVmax; % temporary
79. handles.AmpTransRate = 1; % scan rate in V/s
80. handles.AmpPhaseFlag = 1; % phase flag
81. handles.AmpSweepPhase = 1:4; % delay for each phase
82. set(handles.AmpResetTEdit,'String',handles.AmpResetPer);
83. set(handles.AmpSampleTEdit,'String',handles.AmpSamplePer);
84. set(handles.AmpRecoverTEdit,'String',handles.AmpRecoverPer);
85. set(handles.AmpVmaxEdit,'String',handles.AmpVmax);
86. set(handles.AmpVminEdit,'String',handles.AmpVmin);
87. set(handles.AmpTransRateEdit,'String',handles.AmpTransRate);

88. % CV initial variables
89. handles.CVcycle = 4; % num of CV cycle
90. handles.CVperiod = 0.5; % period of CV pulse in second
91. handles.CVvmin = -0.5; % CV minimum, volt
92. handles.CVvmax = 0.5; % CV maximum, volt
93. handles.CVrate = 10; % scan rate in V/s
94. handles.CVdaqrate = 1/(handles.CVrate * 1000); % extra 3us for MCU process
95. handles.CVcurrentCh = 21;
96. handles.CVcycleCtr = 1;
97. set(handles.CVcycleEdit,'String',handles.CVcycle);
98. set(handles.CVperiodEdit,'String',handles.CVperiod);
99. set(handles.CVvminEdit,'String',handles.CVvmin);
100. set(handles.CVvmaxEdit,'String',handles.CVvmax);
101. set(handles.CVrateEdit,'String',handles.CVrate);

102. % save data constant
103. handles.saveFrameNumber = 100; % number of frame in a block of saved data.

104. % saving properties and initial values
105. handles.saveFileName = 'FileName';
106. handles.saveFileDir = pwd;
107. handles.saveDesc = 'FileDescription';
108. set(handles.saveFileNameInput,'String',handles.saveFileName);
109. set(handles.saveFileDirInput,'String',handles.saveFileDir);
110. set(handles.saveFileDescInput,'String',handles.saveDesc);

111. waitbar(0.3,f,'Initializing GUI');
112. % initialize buttons
113. set(handles.serialButton,'String','OFF','foregroundColor','red','fontweight','bold');
114. set(handles.monitorButton,'String','OFF','foregroundColor','red','fontweight','bold');
115. set(handles.enableAmpButton,'String','OFF','foregroundColor','red','fontweight','bold');
116. set(handles.enableCVButton,'String','OFF','foregroundColor','red','fontweight','bold');
117. set(handles.AmperometryPanel,'HighlightColor','red');
118. set(handles.VoltametryPanel,'HighlightColor','red')

119. set(handles.dcCalButton,'Enable','off');
120. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
121. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','off');
122. set(handles.frameNumber,'Enable','off');
123. set(handles.framePerSecond,'Enable','off');
124. set(handles.CVtitle,'Visible','off');

125. set(handles.CVcycleEdit,'Enable','off');
126. set(handles.CVperiodEdit,'Enable','off');
127. set(handles.CVvminEdit,'Enable','off');
128. set(handles.CVvmaxEdit,'Enable','off');
129. set(handles.CVrateEdit,'Enable','off');
130. set(handles.CVSingleButton,'Enable','off');
131. set(handles.CVBaseButton,'Enable','off');
132. set(handles.CVBaseSigButton,'Enable','off');
133. set(handles.runButtonV,'Enable','off');
134. set(handles.CVcalculateButton,'Enable','off');

135. set(handles.AmpResetTEdit,'Enable','off');
136. set(handles.AmpSampleTEdit,'Enable','off');
137. set(handles.AmpRecoverTEdit,'Enable','off');
138. set(handles.AmpVmaxEdit,'Enable','off');
139. set(handles.AmpVminEdit,'Enable','off');
140. set(handles.AmpTransRateEdit,'Enable','off');

141. % initialize vCHIPText
142. set(handles.vChipText,'String',num2str(handles.vChipSet)); % print out the slider value

143. waitbar(0.5,f,'Initializing DAQ board');
144. % warm-up: initialize DAQ and delete

145. try

```



```

146. handles.aiDAQ = analoginput('mwadlink', 0); % Opens the analog input functionality
147. set(handles.aiDAQ, 'SampleRate', handles.sampleRate);
148. set(handles.aiDAQ, 'SamplesPerTrigger', uint16(1000));
149. set(handles.aiDAQ, 'TriggerType', 'Immediate'); % 'External digital' or 'Immediate'
150. set(handles.aiDAQ, 'InputType', 'DIFF');

151. handles.aiCE = addchannel(handles.aiDAQ, 33); %Add channel #33 to ai_device
152. set(handles.aiCE, 'InputRange', [handles.FSminV handles.FSmaxV]);
153. handles.aiRE = addchannel(handles.aiDAQ, 34); %Add channel #34 to ai_device
154. set(handles.aiRE, 'InputRange', [handles.FSminV handles.FSmaxV]);
155. handles.aiVpot = addchannel(handles.aiDAQ, 35); %Add channel #35 to ai_device
156. set(handles.aiVpot, 'InputRange', [handles.FSminV handles.FSmaxV]);
157. handles.ai1 = addchannel(handles.aiDAQ, 40); %Add channel #40 to ai_device
158. set(handles.ai1, 'InputRange', [handles.FSminV handles.FSmaxV]);
159. handles.ai2 = addchannel(handles.aiDAQ, 41); %Add channel #41 to ai_device
160. set(handles.ai2, 'InputRange', [handles.FSminV handles.FSmaxV]);
161. handles.ai3 = addchannel(handles.aiDAQ, 42); %Add channel #42 to ai_device
162. set(handles.ai3, 'InputRange', [handles.FSminV handles.FSmaxV]);
163. handles.ai4 = addchannel(handles.aiDAQ, 43); %Add channel #43 to ai_device
164. set(handles.ai4, 'InputRange', [handles.FSminV handles.FSmaxV]);

165. delete(handles.aiCE);
166. delete(handles.aiRE);
167. delete(handles.aiVpot);
168. delete(handles.ai1);
169. delete(handles.ai2);
170. delete(handles.ai3);
171. delete(handles.ai4);
172. delete(handles.aiDAQ);
173. clear handles.aiCE; %clear ai_channels
174. clear handles.aiRE; %clear ai_channels
175. clear handles.aiVpot;
176. clear handles.ai1; %clear ai_channels
177. clear handles.ai2; %clear ai_channels
178. clear handles.ai3; %clear ai_channels
179. clear handles.ai4; %clear ai_channels
180. clear handles.aiDAQ; %clear ai_device
181. catch % no DAQ is detected, disable experiment functions, keep monitoring ON.
182. handles.NoDAQ = msgbox('DAQ Board is not detected.', 'Error', 'error');
183. set(handles.enableAmpButton, 'String', 'OFF', 'foregroundColor', 'red', 'fontweight', 'bold', 'Enable', 'off');
184. set(handles.enableCVButton, 'String', 'OFF', 'foregroundColor', 'red', 'fontweight', 'bold', 'Enable', 'off');
185. end

186. % initialize images
187. axes(handles.jetImage); cla reset;
188. matlabImage = imread('JetColormap.png'); image(matlabImage); axis off; axis image;
189. axes(handles.legendImage); cla reset;
190. matlabImage = imread('Legend.png'); image(matlabImage); axis off; axis image;
191. axes(handles.ampImage); cla reset;
192. matlabImage = imread('AmpImage.png'); image(matlabImage); axis off; axis image;

193. waitbar(0.8, f, 'Initializing Serial COM');
194. % setup MCU serial
195. handles.due = serial('COM5', 'BAUD', 115200, 'Parity', 'none', 'Databits', 8, 'Stopbits', 1);
196. fopen(handles.due); % turn on serial
197. set(handles.serialButton, 'String', 'ON', 'foregroundColor', 'green', 'fontweight', 'bold');
198. handles.serialCondition = 1;

199. % set initials conditions to PCB and GUI
200. set(handles.vChipSlider, 'Value', handles.vChipSet); % set slider position to rounded off value
201. set(handles.vChipText, 'String', num2str(handles.vChipSet, '%.2f')); % print out the slider value
202. vChipSetSerial = strcat(num2str(2), num2str(handles.vChipSet*100)); % assign ID number 2
203. due = handles.due;
204. fprintf(due, vChipSetSerial);

205. set(handles.MainChipButton, 'Value', handles.ChipEnable);
206. set(handles.PotentiostatButton, 'Value', 0);
207. set(handles.ThreeEButton, 'Value', 0);
208. set(handles.TwoEButton, 'Value', 0);
209. enableSerial = strcat(num2str(3), ...
210. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb), num2str(handles.ThreeEEnable))))); % assign ID number 3
211. due = handles.due;
212. fprintf(due, enableSerial);
213. set(handles.CVSingleButton, 'Value', handles.CVSingle);
214. set(handles.CVBaseButton, 'Value', handles.CVBase, 'foregroundColor', 'black');
215. set(handles.CVBaseSigButton, 'Value', handles.CVBaseSig, 'foregroundColor', 'black');

216. waitbar(1, f, 'Ready');
217. pause(0.5);
218. delete(f);

219. % Update handles structure
220. guidata(hObject, handles);

221. % UIWAIT makes MainGUI wait for user response (see UIRESUME)
222. % uiwait(handles.MainGUI);

223. % --- Outputs from this function are returned to the command line.

```



```

224. function varargout = MainGUI_OutputFcn(hObject, eventdata, handles)
225. % varargout cell array for returning output args (see VARARGOUT);
226. % hObject handle to figure
227. % eventdata reserved - to be defined in a future version of MATLAB
228. % handles structure with handles and user data (see GUIDATA)

229. % Get default command line output from handles structure
230. varargout{1} = handles.output;

231. % --- Executes on button press in serialButton.
232. function serialButton_Callback(hObject, eventdata, handles)
233. handles = guidata(hObject);
234. if (handles.serialCondition == 0)
235. set(handles.serialButton,'String','ON','foregroundColor','green','fontweight','bold');
236. handles.serialCondition = 1;
237. handles.due = serial('COM5','BAUD', 115200, 'Parity', 'none', 'Databits', 8, 'Stopbits', 1);
238. fopen(handles.due);
239. elseif (handles.serialCondition == 1)
240. set(handles.serialButton,'String','OFF','foregroundColor','red','fontweight','bold');
241. set(handles.monitorButton,'String','OFF','foregroundColor','red','fontweight','bold');

242. handles.vChipSet = 1.65;
243. set(handles.vChipSlider, 'Value', handles.vChipSet); % set slider position to rounded off value
244. set(handles.vChipText,'String',num2str(handles.vChipSet,'%2f')); % print out the slider value
245. vChipSetSerial = strcat(num2str(2),num2str(handles.vChipSet*100)); % assign ID number 2
246. due = handles.due;
247. fprintf(due,vChipSetSerial);

248. handles.ChipEnable = 0;
249. handles.PotentiostatEnableb = 1;
250. set(handles.MainChipButton,'Value',0);
251. set(handles.PotentiostatButton,'Value',0);
252. set(handles.ThreeEButton,'Value',0);
253. set(handles.TwoEButton,'Value',0);
254. handles.ThreeEEnable = 0;

255. % Prepare data to send, CHIP enable first bit, MSB
256. enableSerial = strcat(num2str(1), ...
257. num2str(bin2dec(strcat(num2str(handles.ChipEnable))))); % assign ID number 1
258. % Send data to serial
259. due = handles.due;
260. fprintf(due,enableSerial);

261. % Prepare data to send, Potentiostat enable second bit
262. enableSerial = strcat(num2str(3), ...
263. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3
264. % Send data to serial
265. due = handles.due;
266. fprintf(due,enableSerial);

267. handles.serialCondition = 0;
268. if (handles.monitorCondition == 1)
269. handles.monitorCondition = 0;
270. stop(handles.vMonitor);
271. delete(handles.vMonitor);
272. end
273. % remove due as device
274. fclose(handles.due);
275. delete(handles.due);
276. end
277. guidata(hObject, handles);

278. % --- Executes on button press in monitorButton.
279. function monitorButton_Callback(hObject, eventdata, handles)
280. handles = guidata(hObject);
281. if (handles.monitorCondition == 0 && handles.serialCondition == 1)
282. handles.vChipSet = 1.65;
283. set(handles.vChipSlider, 'Value', handles.vChipSet); % set slider position to rounded off value
284. set(handles.vChipText,'String',num2str(handles.vChipSet,'%2f')); % print out the slider value
285. vChipSetSerial = strcat(num2str(2),num2str(handles.vChipSet*100)); % assign ID number 2
286. due = handles.due;
287. fprintf(due,vChipSetSerial);

288. set(handles.monitorButton,'String','ON','foregroundColor','green','fontweight','bold');
289. handles.monitorCondition = 1;
290. handles.vMonitor = timer('Period', 0.5, 'TimerFcn', {@vMonitor, hObject}, 'ExecutionMode','fixedRate');
291. start(handles.vMonitor);

292. elseif (handles.monitorCondition == 1 && handles.serialCondition == 1)
293. set(handles.monitorButton,'String','OFF','foregroundColor','red','fontweight','bold');
294. handles.monitorCondition = 0;

295. handles.vChipSet = 1.65;
296. set(handles.vChipSlider, 'Value', handles.vChipSet); % set slider position to rounded off value
297. set(handles.vChipText,'String',num2str(handles.vChipSet,'%2f')); % print out the slider value
298. vChipSetSerial = strcat(num2str(2),num2str(handles.vChipSet*100)); % assign ID number 2
299. due = handles.due;
300. fprintf(due,vChipSetSerial);

```



```

301. handles.ChipEnable = 0;
302. handles.PotentiostatEnableb = 1;
303. set(handles.MainChipButton,'Value',0);
304. set(handles.PotentiostatButton,'Value',0);
305. set(handles.ThreeEButton,'Value',0);
306. set(handles.TwoEButton,'Value',0);
307. handles.ThreeEEnable = 0;

308. % Prepare data to send, CHIP enable first bit, MSB
309. enableSerial = strcat(num2str(1), ...
310. num2str(bin2dec(strcat(num2str(handles.ChipEnable))))); % assign ID number 1
311. % Send data to serial
312. due = handles.due;
313. fprintf(due,enableSerial);

314. % Prepare data to send, Potentiostat enable second bit
315. enableSerial = strcat(num2str(3), ...
316. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3
317. % Send data to serial
318. due = handles.due;
319. fprintf(due,enableSerial);

320. stop(handles.vMonitor);
321. delete(handles.vMonitor);
322. end
323. guidata(hObject, handles);

324. % --- Executes on slider movement.
325. function vChipSlider_Callback(hObject, eventdata, handles)
326. handles = guidata(hObject);

327. handles.vChipSet = get(handles.vChipSlider,'Value'); % retrieve data from slider
328. handles.vChipSet = round(handles.vChipSet,3,'significant'); % round off this value
329. set(handles.vChipSlider, 'Value', handles.vChipSet); % set slider position to rounded off value
330. set(handles.vChipText,'String',num2str(handles.vChipSet,'%2f')); % print out the slider value

331. % Prepare data to send
332. vChipSetSerial = strcat(num2str(2),num2str(handles.vChipSet*100)); % assign ID number 2

333. % Send data to serial
334. due = handles.due;
335. fprintf(due,vChipSetSerial);

336. guidata(hObject,handles);

337. % --- Executes during object creation, after setting all properties.
338. function vChipSlider_CreateFcn(hObject, eventdata, handles)
339. % Hint: slider controls usually have a light gray background.
340. if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
341. set(hObject,'BackgroundColor',[.9 .9 .9]);
342. end

343. % --- Executes on button press in MainChipButton.
344. function MainChipButton_Callback(hObject, eventdata, handles)
345. handles = guidata(hObject);
346. MainChip = get(handles.MainChipButton,'Value');

347. if (MainChip == 1)
348. handles.ChipEnable = 1;
349. handles.PotentiostatEnableb = 0;
350. set(handles.MainChipButton,'Value',1);
351. set(handles.PotentiostatButton,'Value',1);
352. set(handles.ThreeEButton,'Value',0);
353. set(handles.TwoEButton,'Value',1);
354. handles.ThreeEEnable = 0;

355. elseif (MainChip == 0)
356. handles.ChipEnable = 0;
357. handles.PotentiostatEnableb = 1;
358. set(handles.MainChipButton,'Value',0);
359. set(handles.PotentiostatButton,'Value',0);
360. set(handles.ThreeEButton,'Value',0);
361. set(handles.TwoEButton,'Value',0);
362. handles.ThreeEEnable = 0;
363. end

364. % Prepare data to send, CHIP enable first bit, MSB
365. enableSerial = strcat(num2str(1), ...
366. num2str(bin2dec(strcat(num2str(handles.ChipEnable))))); % assign ID number 1
367. % Send data to serial
368. due = handles.due;
369. fprintf(due,enableSerial);

370. % Prepare data to send, Potentiostat enable second bit
371. enableSerial = strcat(num2str(3), ...
372. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3

```



```

373. % Send data to serial
374. due = handles.due;
375. fprintf(due,enableSerial);

376. guidata(hObject,handles);

377. % --- Executes on button press in PotentiostatButton.
378. function PotentiostatButton_Callback(hObject, eventdata, handles)
379. handles = guidata(hObject);

380. Potentiostat = get(handles.PotentiostatButton,'Value');

381. if (Potentiostat == 1)
382. handles.ChipEnable = 1;
383. handles.PotentiostatEnableb = 0;
384. handles.ThreeEEnable = 0;
385. set(handles.MainChipButton,'Value',1);
386. set(handles.PotentiostatButton,'Value',1);
387. set(handles.ThreeEButton,'Value',0);
388. set(handles.TwoEButton,'Value',1);

389. elseif (Potentiostat == 0)
390. handles.PotentiostatEnableb = 1;
391. handles.ThreeEEnable = 0;
392. set(handles.MainChipButton,'Value',1);
393. set(handles.PotentiostatButton,'Value',0);
394. set(handles.ThreeEButton,'Value',0);
395. set(handles.TwoEButton,'Value',0);
396. end

397. % Prepare data to send, Potentiostat enable second bit
398. enableSerial = strcat(num2str(3), ...
399. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3
400. % Send data to serial
401. due = handles.due;
402. fprintf(due,enableSerial);
403. enableSerial = strcat(num2str(1), ...
404. num2str(bin2dec(strcat(num2str(handles.ChipEnable))))); % assign ID number 1
405. % Send data to serial
406. due = handles.due;
407. fprintf(due,enableSerial);

408. guidata(hObject,handles);

409. % --- Executes on button press in ThreeEButton.
410. function ThreeEButton_Callback(hObject, eventdata, handles)
411. handles = guidata(hObject);

412. handles.ThreeEEnable = 1;
413. set(handles.ThreeEButton,'Value',1);
414. set(handles.TwoEButton,'Value',0);

415. % Prepare data to send, Three Electrode System enable second bit, LSB
416. enableSerial = strcat(num2str(3), ...
417. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3
418. % Send data to serial
419. due = handles.due;
420. fprintf(due,enableSerial);
421. guidata(hObject,handles);

422. % --- Executes on button press in TwoEButton.
423. function TwoEButton_Callback(hObject, eventdata, handles)

424. handles.ThreeEEnable = 0;
425. set(handles.ThreeEButton,'Value',0);
426. set(handles.TwoEButton,'Value',1);
427. % Prepare data to send, Three Electrode System enable third bit, LSB
428. enableSerial = strcat(num2str(3), ...
429. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3
430. % Send data to serial
431. due = handles.due;
432. fprintf(due,enableSerial);
433. guidata(hObject,handles);

434. % --- Executes on button press in enableAmpButton.
435. % Amperometry
436. %

437. % --- Executes on button press in enableAmpButton.
438. function enableAmpButton_Callback(hObject, eventdata, handles)
439. handles = guidata(hObject);
440. if (handles.enableAmp == 0)
441. enableAmp(handles);
442. handles.enableAmp = 1; handles.enableCV = 0;
443. else
444. enableCV(handles);
445. handles.enableAmp = 0; handles.enableCV = 1;

```



```

446. end
447. guidata(hObject,handles);

448. % --- Executes on button press in dcCalButton.
449. function dcCalButton_Callback(hObject, eventdata, handles)
450. handles = guidata(hObject);
451. AmpCalDC(handles);
452. guidata(hObject,handles);

453. % --- Executes on button press in DAQButton.
454. function DAQButton_Callback(hObject, eventdata, handles)
455. handles = guidata(hObject);

456. set(handles.DAQButton,'String','Initiated','foregroundColor','green','fontWeight','bold');
457. set(handles.runButton,'String','Run','foregroundColor','red','fontWeight','bold','Enable','on');

458. % set potentiostat to 0V
459. handles.vPotSet = 0;
460. vPotSetSerial = strcat(num2str(4),num2str((handles.vPotSet + 1.5)*1350)); % assign ID number 4
461. fprintf(handles.due,vPotSetSerial);

462. % time unit initialization
463. handles.timeBasePeriod = 1/handles.sampleRate; % in seconds
464. handles.frameLength = handles.samplePerTrigger*handles.timeBasePeriod; % in seconds
465. handles.timeVectorFrameAve = [];
466. handles.timeVector = [];
467. handles.timeVectorAve = [];

468. % data variables initialization
469. handles.aiDataRaw = int16([]); % all sampling points, multiple points per channel
470. handles.aiDataAve = zeros(handles.numOfElectrode*handles.saveFrameNumber,5,'int16'); % averaged points, single
    point per channel
471. handles.calValue = int16([]); % DC baseline calibration value, single point per channel
472. handles.DataSorted = zeros(handles.numOfElectrode*4,handles.saveFrameNumber,'int16'); % reformat data to single
    column
473. handles.DataFramed = zeros(128,128,handles.saveFrameNumber,'int16'); % reformat data to electrode location

474. timeFrameVector = zeros(handles.numOfFrame-1,1);
475. handles.CtrElt = zeros(handles.numOfFrame-1,1);
476. handles.RefElt = zeros(handles.numOfFrame-1,1);
477. handles.frameQuantile = zeros(handles.numOfFrame-1,3);
478. handles.ChipInfo = zeros(handles.numOfFrame-1,4); % Q12Temp, Q34Temp, VchipN, VchipP
479. handles.timePlot = zeros(handles.numOfFrame-1,1);
480. handles.timePlot(1,1) = 0;

481. % set heatmapScale
482. % handles.heatmapScale = 1;
483. set(handles.heatmapMin,'String',handles.heatmapScale);
484. set(handles.heatmapMax,'String',handles.heatmapScale);

485. % attempt to use data calibration
486. try
487. handles.calData = int16(evalin('base', 'calData'));
488. catch
489. set(handles.MsgPanel,'String',sprintf('Calibration data is missing. Run DC
    Calibration.'),'foregroundColor','red','fontWeight','bold');
490. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontWeight','bold');
491. set(handles.runButton,'String','Run','foregroundColor','red','fontWeight','bold','Enable','off');
492. end

493. set(handles.runTime,'String',num2str((handles.numOfFrame-1)*(1/handles.numOfFPS)/60,'%1f')); % show calculated runtime

494. % create constants for amperometry pulsing
495. handles.AmpSweepPhase(1) = handles.AmpResetPer;
496. handles.AmpSweepPhase(2) = handles.AmpSweepPhase(1) + handles.AmpSamplePer;
497. handles.AmpSweepPhase(3) = handles.AmpSweepPhase(2) + handles.AmpResetPer;
498. handles.AmpSweepPhase(4) = handles.AmpSweepPhase(3) + handles.AmpRecoverPer;
499. handles.AmpSweepPhase(5) = handles.AmpSweepPhase(4);
500. guidata(hObject, handles);

501. % --- Executes on button press in runButton.
502. function runButton_Callback(hObject, eventdata, handles)
503. handles = guidata(hObject);

504. % set buttons condition
505. set(handles.runButton,'String','Run','foregroundColor','red','fontWeight','bold','Enable','off');
506. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontWeight','bold','Enable','off');
507. set(handles.dcCalButton,'Enable','off');
508. set(handles.frameNumber,'Enable','off');
509. set(handles.framePerSecond,'Enable','off');

510. % initialize other variables, preallocate
511. switchingLocsFrame = zeros(handles.numOfElectrode,1,'int16');
512. aiDataFrameAve = zeros(handles.numOfElectrode,5,'int16');

513. % data saving constant
514. saveFileIDCounter = 1;

```



```

515. saveNumberOfFiles = ceil(handles.numOfFrame/handles.saveFrameNumber); % number of file blocks
516. t = datetime('now'); t = datetime('now'); saveDate = sprintf('%02d%02d%02d_%02d%02d', t.Month, t.Day, t.Year, t.Hour, t.Minute);
517. saveFileName = sprintf('%2s_%s.mat', saveDate, handles.saveFileName);
518. saveFullpath = fullfile(handles.saveFileDir, saveFileName);
519. saveDesc = handles.saveDesc;
520. save(saveFullpath, 'saveDesc'); % create save file

521. handles.saveVarName = cell(saveNumberOfFiles, 1); % pre-allocate Variables Name
522. for i = 1:saveNumberOfFiles
523. handles.saveVarName{i, 1} = sprintf('DataSorted_%03d', i);
524. end

525. % initialize DAQ board
526. % Analog channel initialization
527. % DAQ full scale range options: 0-0.1, 0-0.4, 0-0.5, 0-1, 0-2, 0-2.5, 0-4, and the bipolar of each.
528. handles.FSminV = -1;
529. handles.FSmaxV = 1;

530. handles.aiDAQ = analoginput('mwadlink', 0); % Opens the analog input functionality
531. set(handles.aiDAQ, 'SampleRate', handles.sampleRate);
532. set(handles.aiDAQ, 'SamplesPerTrigger', handles.samplePerTrigger);
533. set(handles.aiDAQ, 'TriggerType', 'External digital'); % 'External digital' or 'Immediate'
534. set(handles.aiDAQ, 'DigitalTriggerCondition', 'Negative'); % 'Negative' or 'None'
535. set(handles.aiDAQ, 'InputType', 'DIFF');

536. handles.ai1 = addchannel(handles.aiDAQ, 40); % Add channel #40 to ai_device
537. set(handles.ai1, 'InputRange', [handles.FSminV handles.FSmaxV]);
538. handles.ai2 = addchannel(handles.aiDAQ, 41); % Add channel #41 to ai_device
539. set(handles.ai2, 'InputRange', [handles.FSminV handles.FSmaxV]);
540. handles.ai3 = addchannel(handles.aiDAQ, 42); % Add channel #42 to ai_device
541. set(handles.ai3, 'InputRange', [handles.FSminV handles.FSmaxV]);
542. handles.ai4 = addchannel(handles.aiDAQ, 43); % Add channel #43 to ai_device
543. set(handles.ai4, 'InputRange', [handles.FSminV handles.FSmaxV]);

544. handles.aiClk = addchannel(handles.aiDAQ, 32); % Add channel #0 to ai_device
545. set(handles.aiClk, 'InputRange', [handles.FSminV handles.FSmaxV]);

546. % Digital channel initialization
547. handles.dioDAQ = digitalio('mwadlink', 0);
548. handles.doTRIG = addline(handles.dioDAQ, 39, 'out');
549. putvalue(handles.doTRIG, 1);

550. timeBasePeriod = 1/handles.sampleRate; % in seconds
551. frameLength = handles.samplePerTrigger*timeBasePeriod; % in seconds
552. timeProcessing = 0;
553. frameCounterProc = 0;
554. timeFrameAccumulate = 0;

555. for frameCounter = 0 : handles.numOfFrame
556. % warm up frame
557. if (frameCounter == 0)
558. start(handles.aiDAQ);
559. putvalue(handles.doTRIG, 0); % negative edge to trigger PWM, counter, and DAQ
560. firstFrameRaw = int16(getdata(handles.aiDAQ)*10000); % store sample with resolution of 0.1mV
561. putvalue(handles.doTRIG, 1); % positive level to stop acquisition of a frame
562. end

563. tic

564. % actual acquisition frame
565. if (frameCounter > 0)
566. start(handles.aiDAQ); % initiate acquisition
567. putvalue(handles.doTRIG, 0); % negative edge to trigger PWM, counter, and DAQ

568. % processing stage, skip the first frame
569. if (frameCounter > 1)

570. frameCounterProc = mod((frameCounter-1), handles.saveFrameNumber);
571. if (frameCounterProc == 0)
572. frameCounterProc = handles.saveFrameNumber; end

573. sampleStart = (handles.samplePerTrigger*(frameCounterProc-1))+1;
574. sampleEnd = sampleStart-1+handles.samplePerTrigger;

575. % detect channel switching moments, outputs switching location from last input, LSB the clock signal
576. [~, switchingLocsFrame] = findpeaks(abs(diff(double(handles.aiDataRaw(sampleStart:sampleEnd, 1))))', 'MinPeakHeight', 0.1*handles.FSmaxV*10000);
577. switchingLocsFrame(handles.numOfElectrode) = length(handles.aiDataRaw(sampleStart:sampleEnd)); % add last point

578. samplePerChannel = mean(diff(switchingLocsFrame));

579. % average of per-channel output
580. for locPtr = 1:length(switchingLocsFrame)
581. if locPtr > 1 % after first channel
582. locStart = switchingLocsFrame(locPtr-1);
583. elseif locPtr == 1 % first channel only
584. locStart = 1;

```



```

585. end

586. % average samples from 25% to 75% of a channel
587. locStart = (frameCounterProc-1)*handles.samplePerTrigger + locStart + round(samplePerChannel*0.25);
588. locEnd = locStart + round(samplePerChannel*0.5);

589. aiDataFrameAve(locPtr,:) = int16(mean(handles.aiDataRaw(locStart:locEnd,:)));
590. % timeVectorFrameAve(locPtr,:) = timeVector(switchingLocsFrame(locPtr));
591. end

592. % averaged: concatenate switching location & time values
593. locSwFrameStart = (frameCounterProc-1)*length(switchingLocsFrame)+1;
594. locSwFrameEnd = locSwFrameStart-1+length(switchingLocsFrame);

595. handles.aiDataAve(locSwFrameStart:locSwFrameEnd,:) = aiDataFrameAve; % store data from exiting frame
596. numOfCh = length(switchingLocsFrame); % number of channel

597. % reformatting from four columns to single column
598. for Qd = 1:4
599.     for Ch = 1:numOfCh
600.         handles.DataSorted(Ch+((Qd-1)*numOfCh),frameCounterProc) = handles.aiDataAve(Ch+((frameCounterProc-1)*numOfCh),Qd+1);
601.     end
602. end

603. % use calibration data
604. handles.DataSorted(:,frameCounterProc) = handles.DataSorted(:,frameCounterProc) - handles.calData;

605. % save the quantiles and calculate sigma
606. handles.frameQuantile(frameCounter-1,1:3) = quantile(double(handles.DataSorted(:,frameCounterProc))/1000/-20e6,[0.159, 0.5,
0.841]); % median and 1 sigma around it.
607. handles.frameQuantile(frameCounter-1,4) = (abs(handles.frameQuantile(frameCounterProc,1)
handles.frameQuantile(frameCounterProc,2)) + ...
608. (handles.frameQuantile(frameCounterProc,3) - handles.frameQuantile(frameCounterProc,2)))/2;

609. % reformat based on read channel location
610. eoCtr = 0; % even or odd column counter
611. for col = 0:63
612.     handles.DataFramed(65+col,65:128,frameCounterProc) = handles.DataSorted((((col-eoCtr)*64)+eoCtr)+1:2:((col-
eoCtr+1)*64+64+eoCtr),frameCounterProc); %Q1
613. handles.DataFramed(64-col,65:128,frameCounterProc) = handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh+1:2:((col-
eoCtr+1)*64+64+numOfCh+eoCtr),frameCounterProc); % Q2
614. handles.DataFramed(64-col,1:64,frameCounterProc) = flip(handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh*2+1:2:((col-
eoCtr+1)*64+64+numOfCh*2+eoCtr),frameCounterProc)); % Q3
615. handles.DataFramed(65+col,1:64,frameCounterProc) = flip(handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh*3+1:2:((col-
eoCtr+1)*64+64+numOfCh*3+eoCtr),frameCounterProc)); % Q1
616. if (eoCtr == 1); eoCtr = 0; else eoCtr = 1; end
617. end

618. % passing values from MCU readings
619. handles.CtrElt(frameCounter-1,1) = str2double(get(handles.measVCE,'String'));
620. handles.RefElt(frameCounter-1,1) = str2double(get(handles.measVRE,'String'));
621. handles.ChipInfo(frameCounter-1,1) = str2double(get(handles.measTChipQ12,'String'));
622. handles.ChipInfo(frameCounter-1,2) = str2double(get(handles.measTChipQ34,'String'));
623. handles.ChipInfo(frameCounter-1,3) = str2double(get(handles.measVChipNText,'String'));
624. handles.ChipInfo(frameCounter-1,4) = str2double(get(handles.measVChipPText,'String'));

625. wait(handles.aiDAQ,0.5); % wait for aquisition to be done;

626. % plot heatmap
627. axes(handles.AmpHeatmap);
628. imagesc(-1*handles.DataFramed(:,frameCounterProc),[-handles.heatmapScale*200 handles.heatmapScale*200]);
629. colormap('jet');
630. axis off;

631. % plot transient response
632. handles.timePlot(frameCounter,1) = timeFrameAccumulate;
633. plot(handles.AmpPlot,handles.timePlot(1:frameCounter-1,1),handles.CtrElt(1:frameCounter-1,1),'LineWidth',1,'Color','r');
634. hold(handles.AmpPlot,'on');
635. plot(handles.AmpPlot,handles.timePlot(1:frameCounter-1,1),handles.RefElt(1:frameCounter-1,1),'LineWidth',1,'Color','b');
636. hold(handles.AmpPlot,'off');
637. handles.AmpPlot.Title.String = 'Transient Response';
638. handles.AmpPlot.XLabel.String = 'Time(s)';
639. handles.AmpPlot.YLabel.String = 'Voltage(V)';
640. handles.AmpPlot.XLim = [1 handles.numOffFrame/handles.numOffFPS];
641. handles.AmpPlot.XTick = 0:round(handles.numOffFrame/handles.numOffFPS/20,0):handles.numOffFrame/handles.numOffFPS;
642. handles.AmpPlot.XGrid = 'on';
643. handles.AmpPlot.YLim = [-3 3];
644. handles.AmpPlot.YTick = -3:0.5:3;
645. handles.AmpPlot.YAxisLocation = 'right';
646. handles.AmpPlot.YGrid = 'on';
647. handles.AmpPlot.FontSize = 8;
648. handles.AmpPlot.Title.FontSize = 10;

649. plot(handles.AmpPlot2,handles.timePlot(1:frameCounter-1,1),handles.frameQuantile(1:frameCounter-1,2),'LineWidth',2,'Color','g');
%Q2
650. hold(handles.AmpPlot2,'on');
651. plot(handles.AmpPlot2,handles.timePlot(1:frameCounter-1,1),handles.frameQuantile(1:frameCounter-
1,1),':','LineWidth',1,'Color','g'); %Q1

```



```

652. plot(handles.AmpPlot2,handles.timePlot(1:frameCounter-1,1),handles.frameQuantile(1:frameCounter-
1,3),'-','LineWidth',1,'Color','g'); %Q3
653. hold(handles.AmpPlot2,'off');
654. handles.AmpPlot2.YLabel.String = 'Current(A)';
655. handles.AmpPlot2.YLim = [-2.4e-9 2.4e-9];
656. handles.AmpPlot2.YTick = -2.4e-9:0.4e-9:2.4e-9;
657. handles.AmpPlot2.YAxis.Exponent = -9;
658. handles.AmpPlot2.XLim = [1 handles.numOffFrame/handles.numOffFPS];
659. handles.AmpPlot2.XTick = 0:round(handles.numOffFrame/handles.numOffFPS/20,0):handles.numOffFrame/handles.numOffFPS;
660. handles.AmpPlot2.FontSize = 8;
661. handles.AmpPlot2.Color = 'none';

662. axes(handles.PreProcAx4);
663. cla(handles.PreProcAx4,'reset')
664. % histogram(double(handles.DataSorted(:,frameCounterProc))/10000/-20e6,-1.2e-9:0.05e-9:1.2e-
9,'FaceColor','b','FaceAlpha',0.5);
665. handles.PreProcAx4.Title.String = 'Output Distribution';
666. handles.PreProcAx4.XLabel.String = 'Current(A)';
667. handles.PreProcAx4.YLabel.String = 'Number of Electrodes';
668. handles.PreProcAx4.XLim = [-1.2e-9 1.2e-9];
669. handles.PreProcAx4.XTick = -1.2e-9:0.4e-9:1.2e-9;
670. handles.PreProcAx4.XAxis.Exponent = -9;
671. handles.PreProcAx4.XGrid = 'on';
672. handles.PreProcAx4.YLim = [0 4500];
673. handles.PreProcAx4.YGrid = 'on';
674. handles.PreProcAx4.FontSize = 8;
675. handles.PreProcAx4.Title.FontSize = 10;
676. line([0 0],[0 5000],'Color','red','LineStyle','--','LineWidth',1);
677. text(1.2e-9*0.9,4000,...
678. {'[median = ', num2str(round(handles.frameQuantile(frameCounter-1,2)*1e12,1),'%.0f'), 'pA'], ...
679. ['sigma = ', num2str(round(handles.frameQuantile(frameCounter-1,4)*1e12,1),'%.0f'), 'pA']}, ...
680. 'FontSize',8,'HorizontalAlignment','right','FontSize', 8);
681. end

682. timeProcessing = toc;
683. % end of processing

684. if (frameCounter > 1)
685. sampleStart = sampleStart+handles.samplePerTrigger;
686. sampleEnd = sampleStart-1+handles.samplePerTrigger;
687. end

688. if (frameCounter == 1 || mod((frameCounter),handles.saveFrameNumber) == 1)
689. sampleStart = 1;
690. sampleEnd = handles.samplePerTrigger;
691. end

692. handles.aiDataRaw(sampleStart:sampleEnd,:) = int16(getdata(handles.aiDAQ)*10000); % store sample from DAQ

693. putvalue(handles.doTRIG, 1); % turn off PWM and counter
694. % end of acquisition of a frame

695. end

696. % add delay to adjust time to set FPS time by user
697. pause((1/handles.numOffFPS)-timeProcessing);
698. timeFrame = toc;

699. timeFrameVector(frameCounter+1,1) = timeFrame; % concatenate time of each frame
700. timeFrameAccumulate = timeFrameAccumulate + timeFrame;

701. set(handles.MsgPanel,'String',sprintf(['Amperometry: ', num2str(frameCounter/(handles.numOffFrame-1)*100,'%1f'),'%% | FPS:
',num2str(1/timeFrame,'%2f'), ...
702. ' | Elapsed Time: ', num2str(timeFrameAccumulate/60, '%2f'), 'min.', '\n']], 'foregroundColor','red','fontWeight','bold');

703. % set potentiostat voltage based on sweeping variables
704. % handles.AmpMode; handles.AmpResetPer; handles.AmpSamplePer; handles.AmpRecoverPer
705. % handles.AmpVmin; handles.AmpVmax; handles.AmpTransRate; handles.AmpPhaseFlag
706. % handles.AmpSweepPhase

707. % phase 1: sampling phase, ramp up and flat at Vmax
708. if (timeFrameAccumulate >= handles.AmpSweepPhase(1) && handles.AmpPhaseFlag == 1)
709. handles.AmpPhaseFlag = 2;
710. handles.AmpSweepPhase(1) = handles.AmpResetPer + handles.AmpSweepPhase(4); % phase for the next cycle
711. handles.AmpMode = 1; % ramp up
712. handles.AmpVminTemp = handles.AmpVmin;
713. handles.AmpVmaxTemp = 0;
714. AmpPotSweep(handles);

715. % phase 2: reset phase between sampling and recovery, ramp down and flat at 0V
716. elseif (timeFrameAccumulate >= handles.AmpSweepPhase(2) && handles.AmpPhaseFlag == 2)
717. handles.AmpPhaseFlag = 3;
718. handles.AmpSweepPhase(2) = handles.AmpSweepPhase(2) + handles.AmpSweepPhase(5); % phase for the next cycle
719. handles.AmpMode = 2; % ramp down
720. handles.vPotSet = 0; % set potentiostat to 0V
721. handles.AmpVminTemp = handles.AmpVmin;
722. handles.AmpVmaxTemp = 0;

```



```

723. AmpPotSweep(handles);

724. % phase 3: recovery phase , ramp down and flat at Vmin
725. elseif (timeFrameAccumulate >= handles.AmpSweepPhase(3) && handles.AmpPhaseFlag == 3)
726. handles.AmpPhaseFlag = 4;
727. handles.AmpSweepPhase(3) = handles.AmpSweepPhase(3) + handles.AmpSweepPhase(5); % phase for the next cycle
728. handles.AmpMode = 2; % ramp down
729. handles.AmpVminTemp = 0;
730. handles.AmpVmaxTemp = handles.AmpVmax;
731. AmpPotSweep(handles);

732. % phase 4: reset phase , ramp up and flat at 0V
733. elseif (timeFrameAccumulate >= handles.AmpSweepPhase(4) && handles.AmpPhaseFlag == 4)
734. handles.AmpPhaseFlag = 1;
735. handles.AmpSweepPhase(4) = handles.AmpSweepPhase(4) + handles.AmpSweepPhase(5); % phase for the next cycle
736. handles.AmpMode = 1; % ramp up
737. handles.vPotSet = 0; % set potentiostat to 0V
738. handles.AmpVminTemp = 0;
739. handles.AmpVmaxTemp = handles.AmpVmax;
740. AmpPotSweep(handles);

741. handles.AmpVminTemp = handles.AmpVmin;
742. handles.AmpVmaxTemp = handles.AmpVmax;
743. end

744. % save data in chunks and refresh variable
745. % execute only at the last frame, and when numOffFrame doesn't have remainder after divided by saveFrameNumber
746. if (handles.numOffFrame == frameCounter && mod((frameCounter-1),handles.saveFrameNumber) ~= 0)
747. handles.DataSorted = handles.DataSorted(:,1:mod((frameCounter-1),handles.saveFrameNumber));
748. end

749. % execute every saveFrameNumber and last last frame
750. if ((frameCounter ~= 1 && mod((frameCounter-1),handles.saveFrameNumber) == 0) || frameCounter == handles.numOffFrame)
751. eval([handles.saveVarName{saveFileIDCounter,1} '= handles.DataSorted;']); % create variable with numerical ID
752. save(saveFullpath, handles.saveVarName{saveFileIDCounter,1},'-append');
753. clear(handles.saveVarName{saveFileIDCounter,1});
754. saveFileIDCounter = saveFileIDCounter + 1;
755. handles.DataSorted = zeros(handles.numOfElectrode*4,handles.saveFrameNumber,'int16');
756. end

757. end
758. % end of main loop, go to the next frame

759. % delete ai_channel
760. delete(handles.ai1);
761. delete(handles.ai2);
762. delete(handles.ai3);
763. delete(handles.ai4);
764. delete(handles.aiClk);
765. delete(handles.aiDAQ); %delete ai_device
766. clear handles.ai1; %clear ai_channels
767. clear handles.ai2; %clear ai_channels
768. clear handles.ai3; %clear ai_channels
769. clear handles.ai4; %clear ai_channels
770. clear handles.aiClk; %clear ai_deviceai_device
771. clear handles.aiDAQ; %clear ai_deviceai_device

772. % assignin('base', 'aiDataRaw', handles.aiDataRaw);
773. % assignin('base', 'aiDataAve', handles.aiDataAve);
774. % assignin('base', 'aiDataSorted', handles.DataSorted);
775. % assignin('base', 'aiDataFramed', handles.DataFramed);
776. % assignin('base', 'vCtrElt', handles.CtrElt);
777. % assignin('base', 'vRefElt', handles.RefElt);
778. % assignin('base', 'timevalue', timeFrameVector);

779. % save other information
780. frameQuantile = handles.frameQuantile; save(saveFullpath, 'frameQuantile','-append');
781. vCtrElt = handles.CtrElt; save(saveFullpath, 'vCtrElt','-append');
782. vRefElt = handles.RefElt; save(saveFullpath, 'vRefElt','-append');
783. ChipInfo = handles.ChipInfo; save(saveFullpath, 'ChipInfo','-append');
784. calData = handles.calData; save(saveFullpath, 'calData','-append');

785. timeFrameVector = timeFrameVector(3:end,1);
786. save(saveFullpath, 'timeFrameVector','-append');

787. % end of run commands
788. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontWeight','bold','Enable','on');
789. set(handles.runButton,'String','Run','foregroundColor','red','fontWeight','bold','Enable','off');
790. set(handles.dcCalButton,'Enable','on');
791. set(handles.frameNumber,'Enable','on');
792. set(handles.framePerSecond,'Enable','on');
793. set(handles.MsgPanel,'String',sprintf('Amperometry Run Completed.'),'foregroundColor','green','fontWeight','bold');

794. set(handles.ThreeEButton,'Value',0);
795. set(handles.TwoEButton,'Value',1); % set potentiostat to 2 elt system
796. handles.ThreeEEnable = 0;
797. enableSerial = strcat(num2str(3), ...
798. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnableb),num2str(handles.ThreeEEnable))))); % assign ID number 3

```



```

799. % Send data to serial
800. due = handles.due;
801. fprintf(due,enableSerial);

802. handles.vPotSet = 0;           % set potentiostat to 0V
803. % Prepare data to send
804. vPotSetSerial = strcat(num2str(4),num2str((handles.vPotSet + 1.5)*1350)); % assign ID number 4
805. % Send data to serial
806. fprintf(handles.due,vPotSetSerial);

807. handles.AmpPhaseFlag = 1;     % reset amperometry flag

808. guidata(hObject, handles);

809. % Amperometry reset period
810. function AmpResetTEdit_Callback(hObject, eventdata, handles)
811. handles = guidata(hObject);
812. handles.AmpResetPer = str2double(get(hObject,'String'));
813. guidata(hObject, handles);

814. function AmpResetTEdit_KeyPressFcn(hObject, eventdata, handles)
815. handles = guidata(hObject);
816. handles.AmpResetPer = str2double(get(hObject,'String'));
817. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
818. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
819. guidata(hObject, handles);

820. % Amperometry sampling period
821. function AmpSampleTEdit_Callback(hObject, eventdata, handles)
822. handles = guidata(hObject);
823. handles.AmpSamplePer = str2double(get(hObject,'String'));
824. guidata(hObject, handles);

825. function AmpSampleTEdit_KeyPressFcn(hObject, eventdata, handles)
826. handles = guidata(hObject);
827. handles.AmpSamplePer = str2double(get(hObject,'String'));
828. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
829. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
830. guidata(hObject, handles);

831. % Amperometry recover period
832. function AmpRecoverTEdit_Callback(hObject, eventdata, handles)
833. handles = guidata(hObject);
834. handles.AmpRecoverPer = str2double(get(hObject,'String'));
835. guidata(hObject, handles);

836. function AmpRecoverTEdit_KeyPressFcn(hObject, eventdata, handles)
837. handles = guidata(hObject);
838. handles.AmpRecoverPer = str2double(get(hObject,'String'));
839. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
840. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
841. guidata(hObject, handles);

842. % Amperometry voltage maximum
843. function AmpVmaxEdit_Callback(hObject, eventdata, handles)
844. handles = guidata(hObject);
845. handles.AmpVmin = str2double(get(hObject,'String'))*-1;
846. guidata(hObject, handles);

847. function AmpVmaxEdit_KeyPressFcn(hObject, eventdata, handles)
848. handles = guidata(hObject);
849. handles.AmpVmin = str2double(get(hObject,'String'))*-1;
850. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
851. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
852. guidata(hObject, handles);

853. % Amperometry voltage minimum
854. function AmpVminEdit_Callback(hObject, eventdata, handles)
855. handles = guidata(hObject);
856. handles.AmpVmax = str2double(get(hObject,'String'))*-1;
857. guidata(hObject, handles);

858. function AmpVminEdit_KeyPressFcn(hObject, eventdata, handles)
859. handles = guidata(hObject);
860. handles.AmpVmax = str2double(get(hObject,'String'))*-1;
861. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
862. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
863. guidata(hObject, handles);

864. % Amperometry transition voltage rate
865. function AmpTransRateEdit_Callback(hObject, eventdata, handles)
866. handles = guidata(hObject);
867. handles.AmpTransRate = str2double(get(hObject,'String'));
868. guidata(hObject, handles);

869. function AmpTransRateEdit_KeyPressFcn(hObject, eventdata, handles)
870. handles = guidata(hObject);
871. handles.AmpTransRate = str2double(get(hObject,'String'));

```



```

872. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
873. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
874. guidata(hObject, handles);

875. % number of frame
876. function frameNumber_Callback(hObject, eventdata, handles)
877. handles = guidata(hObject);
878. handles.numOfFrame = str2double(get(hObject,'String'))+1;
879. guidata(hObject, handles);

880. function frameNumber_KeyPressFcn(hObject, eventdata, handles)
881. handles = guidata(hObject);
882. handles.numOfFrame = str2double(get(hObject,'String'))+1;
883. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
884. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
885. guidata(hObject, handles);

886. % frame per second
887. function framePerSecond_Callback(hObject, eventdata, handles)
888. handles = guidata(hObject);
889. handles.numOfFPS= str2double(get(hObject,'String'));
890. guidata(hObject, handles);

891. function framePerSecond_KeyPressFcn(hObject, eventdata, handles)
892. handles = guidata(hObject);
893. handles.numOfFPS= str2double(get(hObject,'String'));
894. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
895. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
896. guidata(hObject, handles);

897. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
898. % Voltametry %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
899. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

900. % --- Executes on button press in enableCVButton.
901. function enableCVButton_Callback(hObject, eventdata, handles)
902. handles = guidata(hObject);
903. if (handles.enableCV == 0)
904. enableCV(handles);
905. handles.enableAmp = 0; handles.enableCV = 1;
906. else
907. enableAmp(handles);
908. handles.enableAmp = 1; handles.enableCV = 0;
909. end
910. guidata(hObject,handles);

911. % Number of Cycle %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
912. function CVcycleEdit_Callback(hObject, eventdata, handles)
913. handles = guidata(hObject);
914. handles.CVcycle = str2double(get(hObject,'String'));
915. guidata(hObject, handles);

916. function CVcycleEdit_KeyPressFcn(hObject, eventdata, handles)
917. handles = guidata(hObject);
918. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
919. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
920. guidata(hObject, handles);

921. % Period of Sweep %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
922. function CVperiodEdit_Callback(hObject, eventdata, handles)
923. handles = guidata(hObject);
924. handles.CVperiod = str2double(get(hObject,'String'));
925. guidata(hObject, handles);

926. function CVperiodEdit_KeyPressFcn(hObject, eventdata, handles)
927. handles = guidata(hObject);
928. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
929. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
930. guidata(hObject, handles);

931. % Voltage Min %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
932. function CVvminEdit_Callback(hObject, eventdata, handles)
933. handles = guidata(hObject);
934. handles.CVvmin = str2double(get(hObject,'String'));
935. guidata(hObject, handles);

936. function CVvminEdit_KeyPressFcn(hObject, eventdata, handles)
937. handles = guidata(hObject);
938. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
939. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
940. guidata(hObject, handles);

941. % Voltage Max %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```



```

942. function CVvmaxEdit_Callback(hObject, eventdata, handles)
943. handles = guidata(hObject);
944. handles.CVvmax = str2double(get(hObject,'String'));
945. guidata(hObject, handles);

946. function CVvmaxEdit_KeyPressFcn(hObject, eventdata, handles)
947. handles = guidata(hObject);
948. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
949. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
950. guidata(hObject, handles);

951. % Voltage Sweeping Rate %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
952. function CVrateEdit_Callback(hObject, eventdata, handles)
953. handles = guidata(hObject);
954. handles.CVrate = str2double(get(hObject,'String'));
955. handles.CVdaqrate = 1/(handles.CVrate * 1000);
956. guidata(hObject, handles);

957. function CVrateEdit_KeyPressFcn(hObject, eventdata, handles)
958. handles = guidata(hObject);
959. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
960. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
961. guidata(hObject, handles);

962. % CV Calculate %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
963. function CVcalculateButton_Callback(hObject, eventdata, handles)
964. handles = guidata(hObject);

965. % other initialization

966. set(handles.AmperometryPanel,'HighlightColor','red');

967. % calculate number of samples per CV ramp up-down
968. % 1MS/s / 2 (differential) / 7 (channels) = ~70000S/s
969. % assume fastest period is 100ms, then 7000S can fit inside a period.
970. handles.CVsampleRate = uint32(round(7000/handles.CVperiod)); % keeping number of sample the same for different period.
971. handles.CVsamplePerTrigger = 7000*(handles.CVcycle+1);

972. % simulate: create and plot CV signal
973. range = round(double((handles.CVvmax - handles.CVvmin)*1e3)); % in mV
974. T = round(handles.CVperiod/handles.CVdaqrate); % period of a signal
975. flatT = T - 2*range; % duration of flat part

976. if (flatT/T < 0.05) % if the flat part is not more than 5% (margin)
977. set(handles.CVcalculateButton,'String','Inc. Period','foregroundColor','red','fontweight','bold');
978. elseif (flatT > 0)
979. for i = 1:range % sweep UP & DOWN
980. CVcurve(i) = handles.CVvmin + (i-1)*1e-3;
981. CVcurve(i+range) = handles.CVvmax - (i-1)*1e-3;
982. end
983. CVcurveDash = CVcurve;
984. for i = 1:flatT % flat part
985. CVcurve(i+range*2) = handles.CVvmin;
986. CVcurveDash(i+range*2) = handles.CVvmin;
987. end
988. for i = 1:T % time vector
989. CVtime(i) = handles.CVdaqrate * i;
990. CVtimeDash(i) = handles.CVdaqrate * i + handles.CVdaqrate * T;
991. end

992. axes(handles.CVsignal);
993. plot(CVtime,CVcurve, 'b', 'LineWidth', 1.5);
994. line([0 max(CVtimeDash)], [0 0], 'Color', 'k', 'LineWidth', 1);
995. hold on;
996. plot(CVtimeDash,CVcurveDash, 'b', 'LineWidth', 2,'LineStyle',':');
997. grid on;
998. hold off;
999. axis([0 max(CVtimeDash) handles.CVvmin-(range/10000) handles.CVvmax+(range/10000)]);
1000. handles.CVsignal.FontSize = 6;

1001. set(handles.CVcycleplot,'String',strcat(num2str(handles.CVcycle),'X'));
1002. set(handles.CVcalculateButton,'String','Initiated','foregroundColor','green','fontweight','bold');
1003. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','on');
1004. set(handles.VoltametryPanel,'HighlightColor','green');
1005. end

1006. % check CV conditions
1007. if (handles.CVcycle < 2) % CVcycle between 2 and 16
1008. set(handles.CVcalculateButton,'String','Inc. #Cycle','foregroundColor','red','fontweight','bold');
1009. set(handles.runButtonV,'Enable','off');
1010. elseif (handles.CVcycle > 16)
1011. set(handles.CVcalculateButton,'String','Dec. #Cycle','foregroundColor','red','fontweight','bold');
1012. set(handles.runButtonV,'Enable','off');
1013. elseif (handles.CVperiod < 0.1) % CVperiod between 0.1 and 20
1014. set(handles.CVcalculateButton,'String','Inc. Period','foregroundColor','red','fontweight','bold');
1015. set(handles.runButtonV,'Enable','off');
1016. elseif (handles.CVperiod > 20)

```



```

1017.set(handles.CVcalculateButton,'String','Dec. Period','foregroundColor','red','fontWeight','bold');
1018.set(handles.runButtonV,'Enable','off');
1019.elseif (handles.CVvmin < -1) % CVvmin between -1 and 1
1020.set(handles.CVcalculateButton,'String','Inc. Vmin','foregroundColor','red','fontWeight','bold');
1021.set(handles.runButtonV,'Enable','off');
1022.elseif (handles.CVvmin > 1)
1023.set(handles.CVcalculateButton,'String','Dec. Vmin','foregroundColor','red','fontWeight','bold');
1024.set(handles.runButtonV,'Enable','off');
1025.elseif (handles.CVvmax < -1) % CVvmax between -1 and 1
1026.set(handles.CVcalculateButton,'String','Inc. Vmax','foregroundColor','red','fontWeight','bold');
1027.set(handles.runButtonV,'Enable','off');
1028.elseif (handles.CVvmax > 1)
1029.set(handles.CVcalculateButton,'String','Dec. Vmax','foregroundColor','red','fontWeight','bold');
1030.set(handles.runButtonV,'Enable','off');
1031.elseif (handles.CVvmin > handles.CVvmax) % CVvmin > CVvmax
1032.set(handles.CVcalculateButton,'String','Vmin > Vmax','foregroundColor','red','fontWeight','bold');
1033.set(handles.runButtonV,'Enable','off');
1034.elseif (handles.CVrate < 0.1) % CVrate between 0.1 and 200
1035.set(handles.CVcalculateButton,'String','Inc. CVRate','foregroundColor','red','fontWeight','bold');
1036.set(handles.runButtonV,'Enable','off');
1037.elseif (handles.CVrate > 200)
1038.set(handles.CVcalculateButton,'String','Dec. CVRate','foregroundColor','red','fontWeight','bold');
1039.set(handles.runButtonV,'Enable','off');
1040.end

1041.guidata(hObject, handles);

1042.function runButtonV_Callback(hObject, eventdata, handles)
1043.handles = guidata(hObject);

1044.% disables buttons
1045.set(handles.CVcycleEdit,'Enable','off');
1046.set(handles.CVperiodEdit,'Enable','off');
1047.set(handles.CVvminEdit,'Enable','off');
1048.set(handles.CVvmaxEdit,'Enable','off');
1049.set(handles.CVrateEdit,'Enable','off');
1050.set(handles.CVsingleButton,'Enable','off');
1051.set(handles.CVbaseButton,'Enable','off');
1052.set(handles.CVbaseSigButton,'Enable','off');
1053.set(handles.runButtonV,'Enable','off');
1054.set(handles.CVcalculateButton,'Enable','off');

1055.set(handles.runButtonV,'String','Running','foregroundColor','green','fontWeight','bold');
1056.stop(handles.vMonitor); % stop vmonitor to avoid serial crash
1057.guidata(hObject, handles);

1058.% initialize DAQ board
1059.% DAQ full scale range options: 0-0.1, 0-0.4, 0-0.5, 0-1, 0-2, 0-2.5, 0-4, and the bipolar of each.
1060.handles.FSminV = -2.5;
1061.handles.FSmaxV = 2.5;

1062.handles.aiDAQ = analoginput('mwdlink', 0);% Opens the analog input functionality
1063.set(handles.aiDAQ,'SampleRate', 70000);
1064.set(handles.aiDAQ,'SamplesPerTrigger', 7000);
1065.set(handles.aiDAQ,'TriggerType','Immediate'); % 'External digital' or 'Immediate'
1066.set(handles.aiDAQ,'InputType','DIFF');

1067.handles.aiCE = addchannel(handles.aiDAQ, 33);%Add channel #33 to ai_device
1068.set(handles.aiCE,'InputRange',[handles.FSminV handles.FSmaxV]);
1069.handles.aiRE = addchannel(handles.aiDAQ, 34);%Add channel #34 to ai_device
1070.set(handles.aiRE,'InputRange',[handles.FSminV handles.FSmaxV]);
1071.handles.aiVpot = addchannel(handles.aiDAQ, 35);%Add channel #35 to ai_device
1072.set(handles.aiVpot,'InputRange',[handles.FSminV handles.FSmaxV]);
1073.handles.ai1 = addchannel(handles.aiDAQ, 40);%Add channel #40 to ai_device
1074.set(handles.ai1,'InputRange',[handles.FSminV handles.FSmaxV]);
1075.handles.ai2 = addchannel(handles.aiDAQ, 41);%Add channel #41 to ai_device
1076.set(handles.ai2,'InputRange',[handles.FSminV handles.FSmaxV]);
1077.handles.ai3 = addchannel(handles.aiDAQ, 42);%Add channel #42 to ai_device
1078.set(handles.ai3,'InputRange',[handles.FSminV handles.FSmaxV]);
1079.handles.ai4 = addchannel(handles.aiDAQ, 43);%Add channel #43 to ai_device
1080.set(handles.ai4,'InputRange',[handles.FSminV handles.FSmaxV]);

1081.% initialization
1082.handles.timeBasePeriod = 1/handles.CVsampleRate; % in seconds

1083.% warm up DAQ run
1084.start(handles.aiDAQ);
1085.handles.aiDataRaw = getdata(handles.aiDAQ)*10000;

1086.% reset sample setting
1087.set(handles.aiDAQ,'SampleRate', handles.CVsampleRate);
1088.set(handles.aiDAQ,'SamplesPerTrigger', uint32(handles.CVsamplePerTrigger));

1089.% setup and run timer for CV
1090.handles.CV = timer('Period', handles.CVperiod, 'TimerFcn', {@CV,hObject}, ...
1091.'TasksToExecute', handles.CVcycle+1, 'ExecutionMode','fixedRate');

1092.start(handles.aiDAQ);

```



```

1093.start(handles.CV);

1094 guidata(hObject, handles);

1095.% --- Executes on button press in CVSingleButton.
1096.function CVSingleButton_Callback(hObject, eventdata, handles)
1097(handles = guidata(hObject);
1098.if (handles.CVSingle == 0)
1099(handles.CVSingle = 1;
1100(handles.CVBase = 0;
1101(handles.CVBaseSig = 0;
1102.set(handles.CVSingleButton,'Value',handles.CVSingle);
1103.set(handles.CVBaseButton,'Value',handles.CVBase,'foregroundColor','black');
1104.set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','black');
1105.elseif (handles.CVSingle == 1)
1106(handles.CVSingle = 0;
1107.set(handles.CVSingleButton,'Value',handles.CVSingle);
1108.end
1109 guidata(hObject, handles);

1110.% --- Executes on button press in CVBaseButton.
1111.function CVBaseButton_Callback(hObject, eventdata, handles)
1112(handles = guidata(hObject);
1113.if (handles.CVBase == 0)
1114(handles.CVSingle = 0;
1115(handles.CVBase = 1;
1116(handles.CVBaseSig = 0;
1117.set(handles.CVSingleButton,'Value',handles.CVSingle);
1118.set(handles.CVBaseButton,'Value',handles.CVBase,'foregroundColor','black');
1119.set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','black');
1120.elseif (handles.CVBase == 1)
1121(handles.CVBase = 0;
1122(handles.CVBaseSig = 0;
1123.set(handles.CVBaseButton,'Value',handles.CVBase);
1124.set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','black');
1125.end
1126 guidata(hObject, handles);

1127.% --- Executes on button press in CVBaseSigButton.
1128.function CVBaseSigButton_Callback(hObject, eventdata, handles)
1129(handles = guidata(hObject);
1130.if (handles.CVBaseSig == 0)
1131(handles.CVSingle = 0;
1132(handles.CVBase = 0;
1133(handles.CVBaseSig = 1;
1134.set(handles.CVSingleButton,'Value',handles.CVBase);
1135.set(handles.CVBaseButton,'Value',handles.CVBase,'foregroundColor','green');
1136.set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','black');
1137.end
1138 guidata(hObject, handles);

1139. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1140.% Saving Files %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
1141. %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

1142.function saveFileNameInput_Callback(hObject, eventdata, handles)
1143(handles = guidata(hObject);
1144(handles.saveFileName = get(hObject,'String');
1145 guidata(hObject, handles);

1146.function saveFileDescInput_Callback(hObject, eventdata, handles)
1147(handles = guidata(hObject);
1148(handles.saveDesc = get(hObject,'String');
1149 guidata(hObject, handles);

1150.function saveFileDirInput_Callback(hObject, eventdata, handles)
1151(handles = guidata(hObject);
1152(handles.saveFileDir = get(hObject,'String');
1153 guidata(hObject, handles);

1154.function saveFileDirBrowse_Callback(hObject, eventdata, handles)
1155(handles = guidata(hObject);
1156(handles.saveFileDir = uigetdir('C:\'));
1157.set(handles.saveFileDirInput,'String',handles.saveFileDir);
1158 guidata(hObject, handles);

1159.% --- Executes on button press in pHSenseButton.
1160.function pHSenseButton_Callback(hObject, eventdata, handles)
1161.% Take screen capture
1162. robot = java.awt.Robot();

```



```

1163.pos = [175, 2160-1575-175, 2100 ,1575]; % [left top width height]
1164.rect = java.awt.Rectangle(pos(1),pos(2),pos(3),pos(4));
1165.cap = robot.createScreenCapture(rect);
1166.% Convert to an RGB image
1167.rgb = typecast(cap.getRGB(0,0,cap.getWidth,cap.getHeight,[],0,cap.getWidth),'uint8');
1168.imgData = zeros(cap.getHeight,cap.getWidth,3,'uint8');
1169.imgData(:,:,1) = reshape(rgb(3:4:end),cap.getWidth,[],);
1170.imgData(:,:,2) = reshape(rgb(2:4:end),cap.getWidth,[],);
1171.imgData(:,:,3) = reshape(rgb(1:4:end),cap.getWidth,[],);
1172.% Show or save to file
1173.imwrite(imgData,'image.png');

1174.% hObject    handle to pHSenseButton (see GCBO)
1175.% eventdata  reserved - to be defined in a future version of MATLAB
1176.% handles     structure with handles and user data (see GUIDATA)

1177.function pHBase_Callback(hObject, eventdata, handles)
1178.% hObject    handle to pHBase (see GCBO)
1179.% eventdata  reserved - to be defined in a future version of MATLAB
1180.% handles     structure with handles and user data (see GUIDATA)

1181.% Hints: get(hObject,'String') returns contents of pHBase as text
1182.%           str2double(get(hObject,'String')) returns contents of pHBase as a double

1183.% --- Executes during object creation, after setting all properties.
1184.function pHBase_CreateFcn(hObject, eventdata, handles)
1185.% hObject    handle to pHBase (see GCBO)
1186.% eventdata  reserved - to be defined in a future version of MATLAB
1187.% handles     empty - handles not created until after all CreateFcns called

1188.% Hint: edit controls usually have a white background on Windows.
1189.%           See ISPC and COMPUTER.
1190.if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
1191.set(hObject,'BackgroundColor','white');
1192.end

1193.function pHSlope_Callback(hObject, eventdata, handles)
1194.% hObject    handle to pHSlope (see GCBO)
1195.% eventdata  reserved - to be defined in a future version of MATLAB
1196.% handles     structure with handles and user data (see GUIDATA)

1197.% Hints: get(hObject,'String') returns contents of pHSlope as text
1198.%           str2double(get(hObject,'String')) returns contents of pHSlope as a double

1199.% --- Executes during object creation, after setting all properties.
1200.function pHSlope_CreateFcn(hObject, eventdata, handles)
1201.% hObject    handle to pHSlope (see GCBO)
1202.% eventdata  reserved - to be defined in a future version of MATLAB
1203.% handles     empty - handles not created until after all CreateFcns called

1204.% Hint: edit controls usually have a white background on Windows.
1205.%           See ISPC and COMPUTER.
1206.if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
1207.set(hObject,'BackgroundColor','white');
1208.end

1209.function pHInitial_Callback(hObject, eventdata, handles)
1210.% hObject    handle to pHInitial (see GCBO)
1211.% eventdata  reserved - to be defined in a future version of MATLAB
1212.% handles     structure with handles and user data (see GUIDATA)

1213.% Hints: get(hObject,'String') returns contents of pHInitial as text
1214.%           str2double(get(hObject,'String')) returns contents of pHInitial as a double

1215.% --- Executes during object creation, after setting all properties.
1216.function pHInitial_CreateFcn(hObject, eventdata, handles)
1217.% hObject    handle to pHInitial (see GCBO)
1218.% eventdata  reserved - to be defined in a future version of MATLAB
1219.% handles     empty - handles not created until after all CreateFcns called

1220.% Hint: edit controls usually have a white background on Windows.
1221.%           See ISPC and COMPUTER.
1222.if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
1223.set(hObject,'BackgroundColor','white');
1224.end

1225.% Heatmap Scale Max
1226.function heatmapMax_Callback(hObject, eventdata, handles)
1227.handles = guidata(hObject);

```



```

1228. handles.heatmapScale = str2double(get(hObject,'String'));
1229. guidata(hObject, handles);

1230. function heatmapMax_KeyPressFcn(hObject, eventdata, handles)
1231. handles = guidata(hObject);
1232. handles.heatmapScale = str2double(get(hObject,'String'));
1233. set(handles.heatmapMin,'String',handles.heatmapScale);
1234. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
1235. guidata(hObject, handles);

1236. % Heatmap Scale Min
1237. function heatmapMin_Callback(hObject, eventdata, handles)
1238. handles = guidata(hObject);
1239. handles.heatmapScale = str2double(get(hObject,'String'));
1240. guidata(hObject, handles);

1241. function heatmapMin_KeyPressFcn(hObject, eventdata, handles)
1242. handles = guidata(hObject);
1243. handles.heatmapScale = str2double(get(hObject,'String'));
1244. set(handles.heatmapMax,'String',handles.heatmapScale);
1245. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
1246. guidata(hObject, handles);

```

## C.2.2. AmpCalDC.m

```

1. function AmpCalDC(handles)
2. % initialize other variables, preallocate
3. handles.aiDataRaw = int16([]);
4. handles.numOffFrame = 11;
5. handles.numOffFPS = 3;

6. switchingLocsFrame = zeros(handles.numOfElectrode,1,'int16');
7. aiDataFrameAve = zeros(handles.numOfElectrode,5,'int16');

8. % data saving constant/variables
9. t = datetime('now'); saveDate = sprintf('%02d%02d%02d', t.Month, t.Day, t.Year, t.Hour, t.Minute);
10. saveFileName = sprintf('%2s_%s.mat', saveDate, 'DCCalibrationFile');
11. saveFullpath = fullfile(handles.saveFileDir, saveFileName);
12. saveDesc = handles.saveDesc;
13. save(saveFullpath, 'saveDesc'); % create save file

14. % disable initiate and run button
15. set(handles.runButton,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');
16. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','off');
17. set(handles.frameNumber,'Enable','off');
18. set(handles.framePerSecond,'Enable','off');

19. % initialize DAQ board
20. % Analog channel initialization
21. % DAQ full scale range options: 0-0.1, 0-0.4, 0-0.5, 0-1, 0-2, 0-2.5, 0-4, and the bipolar of each.
22. handles.FSminV = -1;
23. handles.FSmaxV = 1;

24. handles.aiDAQ = analoginput('mwadlink', 0); % Opens the analog input functionality
25. set(handles.aiDAQ,'SampleRate', handles.sampleRate);
26. set(handles.aiDAQ,'SamplesPerTrigger', handles.samplePerTrigger);
27. set(handles.aiDAQ,'TriggerType', 'External digital'); % 'External digital' or 'Immediate'
28. set(handles.aiDAQ,'DigitalTriggerCondition', 'Negative'); % 'Negative' or 'None'
29. set(handles.aiDAQ,'InputType', 'DIFF');

30. handles.ai1 = addchannel(handles.aiDAQ, 40); %Add channel #40 to ai_device
31. set(handles.ai1, 'InputRange', [handles.FSminV handles.FSmaxV]);
32. handles.ai2 = addchannel(handles.aiDAQ, 41); %Add channel #41 to ai_device
33. set(handles.ai2, 'InputRange', [handles.FSminV handles.FSmaxV]);
34. handles.ai3 = addchannel(handles.aiDAQ, 42); %Add channel #42 to ai_device
35. set(handles.ai3, 'InputRange', [handles.FSminV handles.FSmaxV]);
36. handles.ai4 = addchannel(handles.aiDAQ, 43); %Add channel #43 to ai_device
37. set(handles.ai4, 'InputRange', [handles.FSminV handles.FSmaxV]);

38. handles.aiClk = addchannel(handles.aiDAQ, 32); %Add channel #0 to ai_device
39. set(handles.aiClk, 'InputRange', [handles.FSminV handles.FSmaxV]);

40. % Remove other unused axes
41. axes(handles.PreProcAx1); cla reset;
42. set(handles.PreProcAx1, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx1.Visible = 'off';
43. axes(handles.PreProcAx2); cla reset;
44. set(handles.PreProcAx2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx2.Visible = 'off';
45. axes(handles.PreProcAx3); cla reset;
46. set(handles.PreProcAx3, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx3.Visible = 'off';

47. % Digital channel initialization
48. handles.dioDAQ = digitalio('mwadlink',0);
49. handles.doTRIG = addline(handles.dioDAQ, 39, 'out');
50. putvalue(handles.doTRIG, 1);

```



```

51. timeBasePeriod = 1/handles.sampleRate;           % in seconds
52. frameLength = handles.samplePerTrigger*timeBasePeriod; % in seconds
53. timeProcessing = 0;
54. frameCounterProc = 0;

55. handles.heatmapScale = 1;
56. set(handles.heatmapMin,'String',handles.heatmapScale);
57. set(handles.heatmapMax,'String',handles.heatmapScale);

58. for frameCounter = 0 : handles.numOfFrame

59. % warm up frame
60. if (frameCounter == 0)
61. start(handles.aiDAQ);
62. putvalue(handles.doTRIG, 0); % negative edge to trigger PWM, counter, and DAQ
63. firstFrameRaw = int16(getdata(handles.aiDAQ)*10000); % store sample
64. putvalue(handles.doTRIG, 1); % positive level to stop acquisition of a frame
65. end

66. tic
67. % actual acquisition frames
68. if (frameCounter > 0)
69. start(handles.aiDAQ); % initiate acquisition
70. putvalue(handles.doTRIG, 0); % negative edge to trigger PWM, counter, and DAQ

71. % processing stage, skip the first frame
72. if (frameCounter > 1)
73. frameCounterProc = mod((frameCounter-1),handles.saveFrameNumber);
74. if (frameCounterProc == 0)
75. frameCounterProc = handles.saveFrameNumber; end;

76. sampleStart = (handles.samplePerTrigger*(frameCounterProc-1))+1;
77. sampleEnd = sampleStart-1+handles.samplePerTrigger;

78. % % create time vector for each electrode switching moment
79. % timeVector(sampleStart:sampleEnd,:) ...
80. % = transpose(timeBasePeriod+(frameLength*(frameCounterProc-1)):timeBasePeriod:frameLength+(frameLength*(frameCounterProc-1)));

81. % detect channel switching moments, outputs switching location from last input, LSB the clock signal
82. [-,switchingLocsFrame] = findpeaks(abs(diff(double(handles.aiDataRaw(sampleStart:sampleEnd,1)))),'MinPeakHeight',0.1*handles.FSmaxV*10000);
83. switchingLocsFrame(handles.numOfElectrode) = length(handles.aiDataRaw(sampleStart:sampleEnd)); % add last point

84. samplePerChannel = mean(diff(switchingLocsFrame));

85. % average of per-channel output
86. for locPtr = 1:length(switchingLocsFrame)
87. if locPtr > 1 % after first channel
88. locStart = switchingLocsFrame(locPtr-1);
89. elseif locPtr == 1 % first channel only
90. locStart = 1;
91. end

92. % average samples from 50% to 90% of a channel
93. locStart = (frameCounterProc-1)*handles.samplePerTrigger + locStart + round(samplePerChannel*0.25);
94. locEnd = locStart + round(samplePerChannel*0.5);

95. aiDataFrameAve(locPtr,:) = int16(mean(handles.aiDataRaw(locStart:locEnd,:)));
96. timeVectorFrameAve(locPtr,:) = timeVector(switchingLocsFrame(locPtr));
97. end

98. % averaged: concatenate switching location & time values
99. locSwFrameStart = (frameCounterProc-1)*length(switchingLocsFrame)+1;
100. locSwFrameEnd = locSwFrameStart-1+length(switchingLocsFrame);

101. aiDataFrameAve(:,2:5) = aiDataFrameAve(:,2:5);

102. handles.aiDataAve(locSwFrameStart:locSwFrameEnd,:) = aiDataFrameAve; % store data
103. numOfCh = length(switchingLocsFrame); % number of channel

104. % reformatting from four column to single column
105. for Qd = 1:4
106. for Ch = 1:numOfCh
107. handles.DataSorted(Ch+((Qd-1)*numOfCh),frameCounterProc) = handles.aiDataAve(Ch+((frameCounterProc-1)*numOfCh),Qd+1);
108. end
109. end

110. % save the quantiles and calculate sigma
111. handles.frameQuantile(frameCounterProc,1:3) = quantile(double(handles.DataSorted(:,frameCounterProc)),[0.159, 0.5, 0.841])/10000/20e6;
112. handles.frameQuantile(frameCounterProc,4) = (abs(handles.frameQuantile(frameCounterProc,1) - handles.frameQuantile(frameCounterProc,2)) + ...
113. (handles.frameQuantile(frameCounterProc,3) - handles.frameQuantile(frameCounterProc,2)))/2;

114. % reformat based on read channel location
115. eoCtr = 0; % even or odd column counter

```



```

116. for col = 0:63
117.     handles.DataFramed(65+col,65:128,frameCounterProc) = handles.DataSorted((((col-eoCtr)*64)+eoCtr)+1:2:((col-
        eoCtr+1)*64+64+eoCtr),frameCounterProc); %Q1
118.     handles.DataFramed(64-col,65:128,frameCounterProc) = handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh+1:2:((col-
        eoCtr+1)*64+64+numOfCh+eoCtr),frameCounterProc); % Q2
119.     handles.DataFramed(64-col,1:64,frameCounterProc) = flip(handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh*2+1:2:((col-
        eoCtr+1)*64+64+numOfCh*2+eoCtr),frameCounterProc)); % Q3
120.     handles.DataFramed(65+col,1:64,frameCounterProc) = flip(handles.DataSorted((((col-eoCtr)*64)+eoCtr)+numOfCh*3+1:2:((col-
        eoCtr+1)*64+64+numOfCh*3+eoCtr),frameCounterProc)); % Q1
121.     if (eoCtr == 1); eoCtr = 0; else eoCtr = 1; end
122. end

123. % passing values from MCU readings
124. handles.CtrElt(frameCounter-1) = str2double(get(handles.measVCE,'String'));
125. handles.RefElt(frameCounter-1) = str2double(get(handles.measVRE,'String'));

126. wait(handles.aiDAQ,0.5); % wait for aquisition to be done;

127. % plot heatmap
128. axes(handles.AmpHeatmap);
129. imagesc(handles.DataFramed(:, :, frameCounterProc), [-handles.heatmapScale*20e6/10000 handles.heatmapScale*20e6/10000]);
130. colormap('jet');
131. axis off;

132. % plot histogram
133. axes(handles.PreProcAx4);
134. histogram(double(handles.DataFramed(:, :, frameCounterProc))/10000/20e6, -50e-9:2e-9:50e-9, 'FaceColor', 'b', 'FaceAlpha', 0.5);
135. grid on;
136. handles.PreProcAx4.Title.String = 'Baseline Distribution';
137. handles.PreProcAx4.XLabel.String = 'Current(A)';
138. handles.PreProcAx4.YLabel.String = 'Number of Electrodes';
139. handles.PreProcAx4.XLim = [-50e-9 50e-9];
140. handles.PreProcAx4.XTick = -50e-9:20e-9:50e-9;
141. handles.PreProcAx4.XAxis.Exponent = -9;
142. handles.PreProcAx4.XGrid = 'on';
143. handles.PreProcAx4.YLim = [0 2500];
144. handles.PreProcAx4.YGrid = 'on';
145. handles.PreProcAx4.FontSize = 8;
146. handles.PreProcAx4.Title.FontSize = 10;
147. line([0 0],[0 5000], 'Color', 'red', 'LineStyle', '--', 'LineWidth', 1)
148. text(50e-9*0.95, 2250, ...
149.     ['median = ', num2str(round(handles.frameQuantile(frameCounter-1,2)*1e9,2), '%.2f'), 'nA'], ...
150.     ['sigma = ', num2str(round(handles.frameQuantile(frameCounter-1,4)*1e9,2), '%.2f') 'nA']], ...
151.     'FontSize', 8, 'HorizontalAlignment', 'right', 'FontSize', 8);

152. end

153. timeProcessing = toc;
154. % end of processing

155. if (frameCounter > 1)
156.     sampleStart = sampleStart+handles.samplePerTrigger;
157.     sampleEnd = sampleStart+1+handles.samplePerTrigger;
158. elseif (frameCounter == 1)
159.     sampleStart = 1;
160.     sampleEnd = handles.samplePerTrigger;
161. end
162. % dataout = int16(getdata(handles.aiDAQ)*10000);
163. % assignin('base', 'dataout', dataout);

164. handles.aiDataRaw(sampleStart:sampleEnd,:) = int16(getdata(handles.aiDAQ)*10000); % store sample from DAQ

165. putvalue(handles.doTRIG, 1); % turn off PWM and counter
166. % end of aquisition of a frame
167. end

168. pause((1/handles.numOfFPS)-timeProcessing);
169. timeFrame = toc;

170. timeFrameVector(frameCounter+1) = timeFrame; % concatenante time of each frame

171. percentCalibration = frameCounter/(handles.numOfFrame-1)*100;
172. set(handles.MsgPanel, 'String', sprintf(['Amperometry - Calibrating: ', num2str(percentCalibration, '%.1f'), '%\n']), 'foregroundColor', 'red', 'fontWeight', 'bold');

173. % save data in chunks and refresh variables
174. if (handles.numOfFrame == frameCounter)
175.     handles.DataSorted = handles.DataSorted(:, 1:mod((frameCounter-1), handles.saveFrameNumber));
176. end
177. end
178. % end of main loop, go to the next frame

179. % end of run commands
180. set(handles.runButton, 'String', 'Run', 'foregroundColor', 'red', 'fontWeight', 'bold');

181. % delete ai_channel
182. delete(handles.ai1);

```



```

183. delete(handles.ai2);
184. delete(handles.ai3);
185. delete(handles.ai4);
186. delete(handles.aiClk);
187. delete(handles.aiDAQ); %delete ai_device
188. clear handles.ai1; %clear ai_channels
189. clear handles.ai2; %clear ai_channels
190. clear handles.ai3; %clear ai_channels
191. clear handles.ai4; %clear ai_channels
192. clear handles.aiClk; %clear ai_deviceai_device
193. clear handles.aiDAQ; %clear ai_deviceai_device

194. % storing calibration data
195. calData = mean(handles.DataSorted(:,6:end),2);
196. assignin('base', 'calData', calData);

197. % assignin('base', 'aiDataRaw', handles.aiDataRaw);
198. % assignin('base', 'aiDataAve', handles.aiDataAve);
199. % assignin('base', 'aiDataSorted', handles.DataSorted);
200. % assignin('base', 'aiDataFramed', handles.DataFramed);
201. % assignin('base', 'vCtrElt', handles.CtrElt);
202. % assignin('base', 'vRefElt', handles.RefElt);
203. % assignin('base', 'timevalue', timeFrameVector);

204. % save results
205. frameQuantile = handles.frameQuantile; save(saveFullpath, 'frameQuantile','-append');
206. vCtrElt = handles.CtrElt; save(saveFullpath, 'vCtrElt','-append');
207. vRefElt = handles.RefElt; save(saveFullpath, 'vRefElt','-append');
208. timeFrameVector = timeFrameVector(3:end);
209. save(saveFullpath, 'timeFrameVector','-append');
210. save(saveFullpath, 'calData','-append');
211. DataSorted = handles.DataSorted;
212. save(saveFullpath, 'DataSorted','-append');

213. set(handles.MsgPanel,'String',sprintf(['Amperometry          -          DC          Calibration
Completed.']), 'foregroundColor','green','fontweight','bold');

214. % enable back buttons
215. set(handles.DAQButton,'String','Initiate','foregroundColor','red','fontweight','bold','Enable','on');
216. set(handles.frameNumber,'Enable','on');
217. set(handles.framePerSecond,'Enable','on');

```

### C.2.3. AmpPotSweep.m

```

1. function AmpPotSweep(handles)
2. % conversion from voltage to analogWrite() value
3. % -1V: 0670 | -0.5V:1345 | 0V: 2020 | 0.5V:2695 | 1V:3370
4. % LSB = 2V/2700 = 0.7407 mV

5. Vmin = round(((handles.AmpVminTemp + 1.5)*1350)/10);
6. Vmax = round(((handles.AmpVmaxTemp + 1.5)*1350)/10);

7. AmpStepNum = (Vmax-Vmin)*10; % number of steps in a ramp up OR ramp down scan.
8. AmpScanTime = (handles.AmpVmaxTemp - handles.AmpVminTemp)/handles.AmpTransRate; % time required for a ramp-up or ramp-down scan.

9. rate = round(AmpScanTime/AmpStepNum*1e6)-4; % in microseconds, 4 for compensating delay created by MCU.

10. % configure sweep, ID:8, Val: mode|vmin|vmax|dacrate, 1 digit mode, 3 digits for min and max, 4 digits for rate
11. AmpSerial = strcat(num2str(8), sprintf('%01d',handles.AmpMode), sprintf('%03d',Vmin), sprintf('%03d',Vmax), sprintf('%04d',rate));
12. fprintf(handles.due,AmpSerial);
13. end

```

### C.2.4. CV.m

```

1. function CV(hObject, eventdata, parent_GUI)
2. handles = guidata(parent_GUI);
3. due = handles.due;

4. endFlag = 0;

5. % conversion from voltage to analogWrite() value
6. % -1V: 0670 | -0.5V:1345 | 0V: 2020 | 0.5V:2695 | 1V:3370
7. % LSB = 2V/2700 = 0.7407mV

8. Vmin = round(((handles.CVvmin + 1.5)*1350)/10);
9. Vmax = round(((handles.CVvmax + 1.5)*1350)/10);

10. CVStepNum = (Vmax-Vmin)*10;
11. CVscanTime = (handles.CVvmax - handles.CVvmin)/handles.CVrate; % time required for a ramp-up or ramp-down scan.

12. rate = round(CVscanTime/CVStepNum*1e6)-4; % in microseconds, 4 for compensating delay created by MCU.

13. % configure channel, ID:6, Val:channel to read

```



```

14. CVSerial = strcat(num2str(6), sprintf('%02d',handles.CVcurrentCh));
15. fprintf(due,CVSerial);

16. % configure sweep, ID:7, Val: vmin|vmax|dacrate, 3 digits for min and max, 4 digits for rate
17. CVSerial = strcat(num2str(7), sprintf('%03d',Vmin), sprintf('%03d',Vmax), sprintf('%04d',rate));
18. fprintf(due,CVSerial);

19. % stop and clear CV setups once it reach the defined number of cycle
20. if (handles.CVcycleCtr == handles.CVcycle)
21. endFlag = 1;
22. handles.CVcurrentCh = 21;
23. handles.CVcycleCtr = 1;

24. wait(handles.aiDAQ,handles.CVperiod*2); % wait for 2*period for the last cycle
25. handles.aiDataRaw = getdata(handles.aiDAQ)*10000;
26. handles.aiDataRaw(:,4:7) = handles.aiDataRaw(:,4:7)*-1;

27. % delete ai_channels
28. delete(handles.aiCE);
29. delete(handles.aiRE);
30. delete(handles.aiVpot);
31. delete(handles.ai1);
32. delete(handles.ai2);
33. delete(handles.ai3);
34. delete(handles.ai4);
35. delete(handles.aiDAQ); %delete ai_device
36. clear handles.aiCE; %clear ai_channels
37. clear handles.aiRE; %clear ai_channels
38. clear handles.aiVpot; %clear ai_channels
39. clear handles.ai1; %clear ai_channels
40. clear handles.ai2; %clear ai_channels
41. clear handles.ai3; %clear ai_channels
42. clear handles.ai4; %clear ai_channels
43. clear handles.aiDAQ; %clear ai_deviceai_device

44. % reset addressing to default by calling ID:7
45. CVSerial = strcat(num2str(7), sprintf('%03d',203), sprintf('%03d',205), sprintf('%04d',10));
46. fprintf(due,CVSerial);

47. start(handles.vMonitor); % start monitoring serial
48. set(handles.runButtonV,'String','Run','ForegroundColor','red','fontWeight','bold');

49. % calculate CV sweep on RE
50. CVSmooth = smooth(handles.aiDataRaw(:,3),101,'moving'); % smooth Vpotentiostat
51. CVSmooth(1:1000) = mean(CVSmooth);
52. CVnorm = (CVSmooth - min(CVSmooth)) / (max(CVSmooth)-min(CVSmooth));

53. for i = 1:length(CVnorm)
54. if (CVnorm(i) > 0.8) % buffered to binary at 80% threshold
55. CVsweep(i) = 1;
56. else
57. CVsweep(i) = 0;
58. end
59. end

60. % find peaks points
61. [~,CVlocs(:,1)] = findpeaks(double(diff(CVsweep)));
62. [~,CVlocs(:,2)] = findpeaks(double(-1*diff(CVsweep)));
63. CVlocs(:,3) = CVlocs(:,2) - CVlocs(:,1);
64. CVcyclePeriod = round(median(CVlocs(:,3)));
65. CVlocs = CVlocs(2:end,:);

66. % extend locations points to 100% from 20%
67. for i = 1:size(CVlocs,1)
68. CVlocs(i,1) = CVlocs(i,1) - CVcyclePeriod/2*4;
69. CVlocs(i,2) = CVlocs(i,2) + CVcyclePeriod/2*4;
70. end
71. CVlocs(:,3) = CVlocs(:,2) - CVlocs(:,1);
72. CVcyclePeriod = round(median(CVlocs(:,3)));

73. % finalize and store data only if the CV location (CVRElocs) is correctly identified
74. if (size(CVlocs,1) == handles.CVcycle)
75. % reformat data
76. for q = 1:4
77. for e = 1:handles.CVcycle
78. CVcycleData(1:CVcyclePeriod,e,q) = handles.aiDataRaw(CVlocs(e,1):CVlocs(e,1)+CVcyclePeriod-1,q+3);
79. end
80. end
81. for e = 1:handles.CVcycle
82. CVcycleCE(1:CVcyclePeriod,e) = handles.aiDataRaw(CVlocs(e,1):CVlocs(e,1)+CVcyclePeriod-1,1);
83. CVcycleRE(1:CVcyclePeriod,e) = handles.aiDataRaw(CVlocs(e,1):CVlocs(e,1)+CVcyclePeriod-1,2);
84. CVcycleVpot(1:CVcyclePeriod,e) = handles.aiDataRaw(CVlocs(e,1):CVlocs(e,1)+CVcyclePeriod-1,3);
85. end

86. CVcycleData = double(CVcycleData)/10000;
87. CVcycleCE = double(CVcycleCE)/10000;
88. CVcycleRE = double(CVcycleRE)/10000;
89. CVcycleVpot = double(CVcycleVpot)/10000;

```



```

90. handles.CVVpot = CVcycleVpot;
91. handles.CVData = CVcycleData;
92. handles.CVCE = CVcycleCE;
93. handles.CVRE = CVcycleRE;

94. % plot results takes CVcycleRE and CVcycleData
95. dataMin = floor(min(min(min(CVcycleData(1:CVcyclePeriod,:)))*10)/10;
96. dataMax = ceil(max(max(max(CVcycleData(1:CVcyclePeriod,:)))*10)/10;

97. % Q1
98. for e = 1:handles.CVcycle
99. plot(handles.CVQ1,CVcycleRE(1:CVcyclePeriod,e),CVcycleData(1:CVcyclePeriod,e,1));
100. hold(handles.CVQ1,'on');
101. end
102. handles.CVQ1.XLim = [-1*handles.CVvmax -1*handles.CVvmin];
103. handles.CVQ1.YLim = [dataMin dataMax];
104. handles.CVQ1.XAxisLocation = 'bottom';
105. handles.CVQ1.YAxisLocation = 'right';
106. handles.CVQ1.XGrid = 'on';
107. handles.CVQ1.YGrid = 'on';
108. handles.CVQ1.FontSize = 6;
109. handles.CVQ1.XTick = linspace(-1*handles.CVvmax,-1*handles.CVvmin,5);
110. handles.CVQ1.YTick = linspace(dataMin,dataMax,5);
111. hold(handles.CVQ1,'off');

112. % Q2
113. for e = 1:handles.CVcycle
114. plot(handles.CVQ2,CVcycleRE(1:CVcyclePeriod,e),CVcycleData(1:CVcyclePeriod,e,2));
115. hold(handles.CVQ2,'on');
116. end
117. handles.CVQ2.XLim = [-1*handles.CVvmax -1*handles.CVvmin];
118. handles.CVQ2.YLim = [dataMin dataMax];
119. handles.CVQ2.XAxisLocation = 'top';
120. handles.CVQ2.YAxisLocation = 'right';
121. handles.CVQ2.XGrid = 'on';
122. handles.CVQ2.YGrid = 'on';
123. handles.CVQ2.FontSize = 6;
124. handles.CVQ2.XTick = linspace(-1*handles.CVvmax,-1*handles.CVvmin,5);
125. handles.CVQ2.YTick = linspace(dataMin,dataMax,5);
126. hold(handles.CVQ2,'off');

127. % Q3
128. for e = 1:handles.CVcycle
129. plot(handles.CVQ3,CVcycleRE(1:CVcyclePeriod,e),CVcycleData(1:CVcyclePeriod,e,3));
130. hold(handles.CVQ3,'on');
131. end
132. handles.CVQ3.XLim = [-1*handles.CVvmax -1*handles.CVvmin];
133. handles.CVQ3.YLim = [dataMin dataMax];
134. handles.CVQ3.XAxisLocation = 'top';
135. handles.CVQ3.YAxisLocation = 'left';
136. handles.CVQ3.XGrid = 'on';
137. handles.CVQ3.YGrid = 'on';
138. handles.CVQ3.FontSize = 6;
139. handles.CVQ3.XTick = linspace(-1*handles.CVvmax,-1*handles.CVvmin,5);
140. handles.CVQ3.YTick = linspace(dataMin,dataMax,5);
141. hold(handles.CVQ3,'off');

142. % Q4
143. for e = 1:handles.CVcycle
144. plot(handles.CVQ4,CVcycleRE(1:CVcyclePeriod,e),CVcycleData(1:CVcyclePeriod,e,4));
145. hold(handles.CVQ4,'on');
146. end
147. handles.CVQ4.XLim = [-1*handles.CVvmax -1*handles.CVvmin];
148. handles.CVQ4.YLim = [dataMin dataMax];
149. handles.CVQ4.XAxisLocation = 'bottom';
150. handles.CVQ4.YAxisLocation = 'left';
151. handles.CVQ4.XGrid = 'on';
152. handles.CVQ4.YGrid = 'on';
153. handles.CVQ4.FontSize = 6;
154. handles.CVQ4.XTick = linspace(-1*handles.CVvmax,-1*handles.CVvmin,5);
155. handles.CVQ4.YTick = linspace(dataMin,dataMax,5);
156. hold(handles.CVQ4,'off');

157. % reset Vpot to 0V
158. vPotSetSerial = strcat(num2str(4),num2str(2030));
159. fprintf(due,vPotSetSerial);

160. % plot detailed data
161. CVMin = floor(min(min(handles.CVVpot))*10)/10;
162. CVMax = ceil(max(max(handles.CVVpot))*10)/10;
163. CVTime = linspace(0,size(handles.CVVpot,1)/double(handles.CVsampleRate),size(handles.CVVpot,1));

164. % Pre-processed result 1: Counter Electrode
165. plot(handles.PreProcAx1,CVTime,-1*handles.CVVpot(:,1), 'Color', [.75 .75 .75], 'LineWidth', 2,'LineStyle','-' );
166. hold(handles.PreProcAx1,'on');
167. for e = 1:handles.CVcycle
168. plot(handles.PreProcAx1,CVTime,handles.CVCE(:,e));

```



```

169. end
170. handles.PreProcAx1.Title.String = 'Actual Value: AE';
171. handles.PreProcAx1.XLabel.String = 'Time (s)';
172. handles.PreProcAx1.YLabel.String = 'Voltage (V)';
173. handles.PreProcAx1.XLim = [0 max(CVTime)];
174. handles.PreProcAx1.YLim = [-2.5 2.5];
175. handles.PreProcAx1.XGrid = 'on';
176. handles.PreProcAx1.YGrid = 'on';
177. handles.PreProcAx1.FontSize = 8;
178. handles.PreProcAx1.Title.FontSize = 10;
179. handles.PreProcAx1.YTick = linspace(-2.5,2.5,6);
180. hold(handles.PreProcAx1,'off');

181. % Pre-processed result 2: Reference Electrode
182. plot(handles.PreProcAx2,CVTime,-1*handles.CVPot(:,1), 'Color', [.75 .75 .75], 'LineWidth', 2,'LineStyle','-' );
183. hold(handles.PreProcAx2,'on');
184. for e = 1:handles.CVcycle
185. plot(handles.PreProcAx2,CVTime,handles.CVRE(:,e));
186. end
187. handles.PreProcAx2.Title.String = 'Actual Value: RE';
188. handles.PreProcAx2.XLabel.String = 'Time (s)';
189. handles.PreProcAx2.YLabel.String = 'Voltage (V)';
190. handles.PreProcAx2.XLim = [0 max(CVTime)];
191. handles.PreProcAx2.YLim = [-1*CVMax -1*CVMin ];
192. handles.PreProcAx2.XGrid = 'on';
193. handles.PreProcAx2.YGrid = 'on';
194. handles.PreProcAx2.FontSize = 8;
195. handles.PreProcAx2.Title.FontSize = 10;
196. hold(handles.PreProcAx2,'off');

197. % Pre-processed result 3: Raw Data
198. for Q = 1:4
199. for e = 1:handles.CVcycle
200. plot(handles.PreProcAx3,CVTime,handles.CVData(:,e,Q));
201. hold(handles.PreProcAx3,'on');
202. end
203. end
204. handles.PreProcAx3.Title.String = 'All Quad: WE vs. Time';
205. handles.PreProcAx3.XLabel.String = 'Time (s)';
206. handles.PreProcAx3.YLabel.String = 'WE Signal (V)';
207. handles.PreProcAx3.XLim = [0 max(CVTime)];
208. handles.PreProcAx3.YLim = [dataMin dataMax];
209. handles.PreProcAx3.XGrid = 'on';
210. handles.PreProcAx3.YGrid = 'on';
211. handles.PreProcAx3.FontSize = 8;
212. handles.PreProcAx3.Title.FontSize = 10;
213. hold(handles.PreProcAx3,'off');

214. % Pre-processed result 4: Data vs RE
215. for Q = 1:4
216. for e = 1:handles.CVcycle
217. plot(handles.PreProcAx4,handles.CVRE(:,e),handles.CVData(:,e,Q));
218. hold(handles.PreProcAx4,'on');
219. end
220. end
221. handles.PreProcAx4.Title.String = 'All Quad: WE vs. RE';
222. handles.PreProcAx4.XLabel.String = 'RE (V)';
223. handles.PreProcAx4.YLabel.String = 'WE (V)';
224. handles.PreProcAx4.XLim = [-1*handles.CVvmax -1*handles.CVvmin ];
225. handles.PreProcAx4.YLim = [dataMin dataMax];
226. handles.PreProcAx4.XGrid = 'on';
227. handles.PreProcAx4.YGrid = 'on';
228. handles.PreProcAx4.FontSize = 8;
229. handles.PreProcAx4.Title.FontSize = 10;
230. handles.PreProcAx4.XTick = linspace(-1*handles.CVvmax,-1*handles.CVvmin,5);
231. handles.PreProcAx4.YTick = linspace(dataMin,dataMax,5);
232. hold(handles.PreProcAx4,'off');
233. end
234. end

235. % end of a CV sweep, go to the next sweep
236. if (endFlag == 0)
237. handles.CVcycleCtr = handles.CVcycleCtr + 1; % counting sweep moments
238. handles.CVcurrentCh = handles.CVcurrentCh + 1; % increment channel sequentially
239. if (handles.CVcurrentCh > 36)
240. handles.CVcurrentCh = 21; % reset if hits the max
241. end
242. end

243. set(handles.MsgPanel,'String',sprintf(['Cyclic Voltametry: ', num2str((handles.CVcycleCtr-1)/handles.CVcycle*100,'%1f'),'%%',
'\n']), 'foregroundColor','red', 'fontWeight','bold');

244. % end of CV run
245. if (endFlag == 1)
246. % reset counter
247. handles.CVcurrentCh = 21;

```



```

248. handles.CVcycleCtr = 1;
249. set(handles.MsgPanel,'String',sprintf('Cyclic Voltametry Run Completed.'),'foregroundColor','green','fontweight','bold');
250. set(handles.CVcalculateButton,'Enable','on','String','Initiate','foregroundColor','red','fontweight','bold');
251. set(handles.CVcycleEdit,'Enable','on');
252. set(handles.CVperiodEdit,'Enable','on');
253. set(handles.CVvminEdit,'Enable','on');
254. set(handles.CVvmaxEdit,'Enable','on');
255. set(handles.CVrateEdit,'Enable','on');
256. set(handles.CVSingleButton,'Enable','on');
257. set(handles.CVBaseButton,'Enable','on');
258. set(handles.CVBaseSigButton,'Enable','on');

259. % reset electrode system
260. handles.ThreeEEnable = 0;
261. set(handles.ThreeEButton,'Value',0);
262. set(handles.TwoEButton,'Value',1); % set potentiostat to 2 elt system
263. enableSerial = strcat(num2str(3), ...
264. num2str(bin2dec(strcat(num2str(handles.PotentiostatEnable),num2str(handles.ThreeEEnable)))); % assign ID number 3
265. % Send data to serial
266. due = handles.due;
267. fprintf(due,enableSerial);

268. % store data according to CV mode, baseline, baseline+signal, or signal
269. if (handles.CVSingle == 1)
270. assignin('base','CVcycleVpot', CVcycleVpot);
271. assignin('base','CVData', handles.CVData);
272. assignin('base','CVCE', handles.CVCE);
273. assignin('base','CVRE', handles.CVRE);
274. assignin('base','CVVpot', handles.CVVpot);
275. assignin('base','CVRESweep', CVSweep);
276. assignin('base','CVRElocs', CVlocs);
277. assignin('base','CVaiDataRaw', handles.aiDataRaw);
278. assignin('base','CVTime', CVTime);

279. elseif (handles.CVBase == 1)
280. handles.CVBaseData = CVcycleData;
281. handles.CVBaseRE = CVcycleRE;
282. handles.CVBase = 0;
283. handles.CVBaseSig = 1;
284. set(handles.CVBaseButton,'Value',handles.CVBase,'foregroundColor','green','fontweight','bold');
285. set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','red','fontweight','bold');
286. assignin('base','CVBaseData', handles.CVBaseData);
287. assignin('base','CVBaseRE', handles.CVBaseRE);

288. elseif (handles.CVBaseSig == 1)
289. handles.CVBaseSigData = CVcycleData;
290. handles.CVBaseSigRE = CVcycleRE;
291. handles.CVBase = 0;
292. handles.CVBaseSig = 0;
293. set(handles.CVBaseButton,'Value',handles.CVBase,'foregroundColor','green','fontweight','bold');
294. set(handles.CVBaseSigButton,'Value',handles.CVBaseSig,'foregroundColor','green','fontweight','bold');
295. set(handles.CVcalculateButton,'String','Initiate','foregroundColor','red','fontweight','bold');
296. set(handles.runButtonV,'String','Run','foregroundColor','red','fontweight','bold','Enable','off');

297. % adjust size of matrix before and after signal, make them the same size
298. if (size(handles.CVBaseRE,1) > size(handles.CVBaseSigRE,1))
299. handles.CVBaseData(size(handles.CVBaseSigRE,1)+1:size(handles.CVBaseRE,1),:,:) = [];
300. handles.CVBaseRE(size(handles.CVBaseSigRE,1)+1:size(handles.CVBaseRE,1),:,:) = [];
301. elseif (size(handles.CVBaseSigRE,1) > size(handles.CVBaseRE,1))
302. handles.CVBaseSigData(size(handles.CVBaseRE,1)+1:size(handles.CVBaseSigRE,1),:,:) = [];
303. handles.CVBaseSigRE(size(handles.CVBaseRE,1)+1:size(handles.CVBaseSigRE,1),:,:) = [];
304. end
305. CVcyclePeriod = size(handles.CVBaseSigRE,1);

306. % subtract and save as the signal data
307. handles.CVSigData = handles.CVBaseSigData-handles.CVBaseData;
308. handles.CVSigRE = handles.CVBaseSigRE;

309. CVcycleData = handles.CVSigData;
310. CVcycleRE = handles.CVSigRE;

311. assignin('base','CVBaseSigData', handles.CVBaseSigData);
312. assignin('base','CVBaseSigRE', handles.CVBaseSigRE);
313. assignin('base','CVSigData', handles.CVSigData);
314. assignin('base','CVSigRE', handles.CVSigRE);
315. end
316. end

317. guidata(parent_GUI,handles);

```

## C.2.5. enableAmp.m

```

1. function enableAmp(handles)

```



```

2. handles.samplePerTrigger = round(handles.electrodeTime*1e-6*handles.numOfElectrode*handles.sampleRate)+2;

3. % warm-up: initialize DAQ and delete
4. handles.aiDAQ = analoginput('mwadlink', 0);% Opens the analog input functionality
5. set(handles.aiDAQ,'SampleRate', handles.sampleRate);
6. set(handles.aiDAQ,'SamplesPerTrigger', uint16(1000));
7. set(handles.aiDAQ,'TriggerType','Immediate'); % 'External digital' or 'Immediate'
8. set(handles.aiDAQ,'InputType', 'DIFF');

9. handles.aiCE = addchannel(handles.aiDAQ, 33);%Add channel #33 to ai_device
10. set(handles.aiCE, 'InputRange', [handles.FSminV handles.FSmaxV]);
11. handles.aiRE = addchannel(handles.aiDAQ, 34);%Add channel #34 to ai_device
12. set(handles.aiRE, 'InputRange', [handles.FSminV handles.FSmaxV]);
13. handles.aiVpot = addchannel(handles.aiDAQ, 35);%Add channel #35 to ai_device
14. set(handles.aiVpot, 'InputRange', [handles.FSminV handles.FSmaxV]);
15. handles.ai1 = addchannel(handles.aiDAQ, 40);%Add channel #40 to ai_device
16. set(handles.ai1, 'InputRange', [handles.FSminV handles.FSmaxV]);
17. handles.ai2 = addchannel(handles.aiDAQ, 41);%Add channel #41 to ai_device
18. set(handles.ai2, 'InputRange', [handles.FSminV handles.FSmaxV]);
19. handles.ai3 = addchannel(handles.aiDAQ, 42);%Add channel #42 to ai_device
20. set(handles.ai3, 'InputRange', [handles.FSminV handles.FSmaxV]);
21. handles.ai4 = addchannel(handles.aiDAQ, 43);%Add channel #43 to ai_device
22. set(handles.ai4, 'InputRange', [handles.FSminV handles.FSmaxV]);

23. delete(handles.aiCE);
24. delete(handles.aiRE);
25. delete(handles.aiVpot);
26. delete(handles.ai1);
27. delete(handles.ai2);
28. delete(handles.ai3);
29. delete(handles.ai4);
30. delete(handles.aiDAQ);
31. clear handles.aiCE; %clear ai_channels
32. clear handles.aiRE; %clear ai_channels
33. clear handles.aiVpot;
34. clear handles.ai1; %clear ai_channels
35. clear handles.ai2; %clear ai_channels
36. clear handles.ai3; %clear ai_channels
37. clear handles.ai4; %clear ai_channels
38. clear handles.aiDAQ; %clear ai_device

39. % plotting initialization
40. axes(handles.CVQ1); cla reset;
41. set(handles.CVQ1, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ1.Visible = 'off';
42. axes(handles.CVQ2); cla reset;
43. set(handles.CVQ2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ2.Visible = 'off';
44. axes(handles.CVQ3); cla reset;
45. set(handles.CVQ3, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ3.Visible = 'off';
46. axes(handles.CVQ4); cla reset;
47. set(handles.CVQ4, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ4.Visible = 'off';
48. axes(handles.PreProcAx1); cla reset;
49. set(handles.PreProcAx1, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx1.Visible = 'off';
50. axes(handles.PreProcAx2); cla reset;
51. set(handles.PreProcAx2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx2.Visible = 'off';
52. axes(handles.PreProcAx3); cla reset;
53. set(handles.PreProcAx3, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx3.Visible = 'off';
54. axes(handles.PreProcAx4); cla reset;
55. set(handles.PreProcAx4, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx4.Visible = 'on';
56. axes(handles.AmpHeatmap); cla reset;
57. set(handles.AmpHeatmap, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.AmpHeatmap.Visible = 'on';
58. axes(handles.AmpPlot); cla reset;
59. set(handles.AmpPlot, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.AmpPlot.Visible = 'on';
60. axes(handles.AmpPlot2); cla reset;
61. set(handles.AmpPlot2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.AmpPlot2.Visible = 'on';
62. axes(handles.legendImage); cla reset;
63. set(handles.legendImage, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.legendImage.Visible = 'on';
64. matlabImage = imread('Legend.png'); image(matlabImage); axis off; axis image;
65. axes(handles.jetImage); cla reset;
66. set(handles.jetImage, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.jetImage.Visible = 'on';
67. matlabImage = imread('JetColormap.png'); image(matlabImage); axis off; axis image;

68. set(handles.heatmapMin,'Visible','on');
69. set(handles.heatmapMax,'Visible','on');
70. set(handles.heatmapMinLbl,'Visible','on');
71. set(handles.heatmapMaxLbl,'Visible','on');
72. set(handles.heatmapScaleBar,'Visible','on');
73. set(handles.CVtitle,'Visible','off');

74. set(handles.dcCalButton,'Enable','on');
75. set(handles.runButton,'String','Run','ForegroundColor','red','fontWeight','bold','Enable','off');
76. set(handles.DAQButton,'String','Initiate','ForegroundColor','red','fontWeight','bold','Enable','on');
77. set(handles.frameNumber,'Enable','on');
78. set(handles.framePerSecond,'Enable','on');

79. set(handles.CVcycleEdit,'Enable','off');
80. set(handles.CVperiodEdit,'Enable','off');
81. set(handles.CVvminEdit,'Enable','off');
82. set(handles.CVvmaxEdit,'Enable','off');

```



```

83. set(handles.CVrateEdit,'Enable','off');
84. set(handles.CVSingleButton,'Enable','off');
85. set(handles.CVBaseButton,'Enable','off');
86. set(handles.CVBaseSigButton,'Enable','off');
87. set(handles.runButtonV,'Enable','off');
88. set(handles.CVcalculateButton,'Enable','off');

89. set(handles.AmpResetTEdit,'Enable','on');
90. set(handles.AmpSampleTEdit,'Enable','on');
91. set(handles.AmpRecoverTEdit,'Enable','on');
92. set(handles.AmpVmaxEdit,'Enable','on');
93. set(handles.AmpVminEdit,'Enable','on');
94. set(handles.AmpTransRateEdit,'Enable','on');

95. set(handles.AmperometryPanel,'HighlightColor','green');
96. set(handles.VoltametryPanel,'HighlightColor','red');

97. set(handles.enableAmpButton,'Value',1,'String','ON','foregroundColor','green','fontweight','bold');
98. set(handles.enableCVButton,'String','OFF','foregroundColor','red','fontweight','bold');
99. set(handles.enableCVButton,'Value',0);

100. end

```

## C.2.6. enableCV.m

```

1. function enableCV(handles)
2. % warm-up: initialize DAQ and delete
3. handles.FSminV = -2.5;
4. handles.FSmaxV = 2.5;

5. handles.aiDAQ = analoginput('mwadlink', 0);% Opens the analog input functionality
6. set(handles.aiDAQ,'SampleRate', handles.CVsampleRate);
7. set(handles.aiDAQ,'SamplesPerTrigger', uint32(handles.CVsamplePerTrigger));
8. set(handles.aiDAQ,'TriggerType','Immediate'); % 'External digital' or 'Immediate'
9. set(handles.aiDAQ,'InputType','DIFF');

10. handles.aiCE = addchannel(handles.aiDAQ, 33);%Add channel #33 to ai_device
11. set(handles.aiCE, 'InputRange', [handles.FSminV handles.FSmaxV]);
12. handles.aiRE = addchannel(handles.aiDAQ, 34);%Add channel #34 to ai_device
13. set(handles.aiRE, 'InputRange', [handles.FSminV handles.FSmaxV]);
14. handles.aiVpot = addchannel(handles.aiDAQ, 35);%Add channel #35 to ai_device
15. set(handles.aiVpot, 'InputRange', [handles.FSminV handles.FSmaxV]);
16. handles.ai1 = addchannel(handles.aiDAQ, 40);%Add channel #40 to ai_device
17. set(handles.ai1, 'InputRange', [handles.FSminV handles.FSmaxV]);
18. handles.ai2 = addchannel(handles.aiDAQ, 41);%Add channel #41 to ai_device
19. set(handles.ai2, 'InputRange', [handles.FSminV handles.FSmaxV]);
20. handles.ai3 = addchannel(handles.aiDAQ, 42);%Add channel #42 to ai_device
21. set(handles.ai3, 'InputRange', [handles.FSminV handles.FSmaxV]);
22. handles.ai4 = addchannel(handles.aiDAQ, 43);%Add channel #43 to ai_device
23. set(handles.ai4, 'InputRange', [handles.FSminV handles.FSmaxV]);

24. % delete ai_channels
25. delete(handles.aiCE);
26. delete(handles.aiRE);
27. delete(handles.aiVpot);
28. delete(handles.ai1);
29. delete(handles.ai2);
30. delete(handles.ai3);
31. delete(handles.ai4);
32. delete(handles.aiDAQ); %delete ai_device
33. clear handles.aiCE; %clear ai_channels
34. clear handles.aiRE; %clear ai_channels
35. clear handles.aiVpot; %clear ai_channels
36. clear handles.ai1; %clear ai_channels
37. clear handles.ai2; %clear ai_channels
38. clear handles.ai3; %clear ai_channels
39. clear handles.ai4; %clear ai_channels
40. clear handles.aiDAQ; %clear ai_deviceai_device

41. % plotting initialization
42. axes(handles.CVQ1); cla reset;
43. set(handles.CVQ1, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ1.Visible = 'on';
44. axes(handles.CVQ2); cla reset;
45. set(handles.CVQ2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ2.Visible = 'on';
46. axes(handles.CVQ3); cla reset;
47. set(handles.CVQ3, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ3.Visible = 'on';
48. axes(handles.CVQ4); cla reset;
49. set(handles.CVQ4, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.CVQ4.Visible = 'on';
50. axes(handles.PreProcAx1); cla reset;
51. set(handles.PreProcAx1, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx1.Visible = 'on';
52. axes(handles.PreProcAx2); cla reset;
53. set(handles.PreProcAx2, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx2.Visible = 'on';
54. axes(handles.PreProcAx3); cla reset;
55. set(handles.PreProcAx3, 'box','off','XTickLabel',[],'XTick',[],'YTickLabel',[],'YTick',[]); handles.PreProcAx3.Visible = 'on';

```



```

56. axes(handles.PreProcAx4); cla reset;
57. set(handles.PreProcAx4, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.PreProcAx4.Visible = 'on';
58. axes(handles.AmpHeatmap); cla reset;
59. set(handles.AmpHeatmap, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.AmpHeatmap.Visible = 'off';
60. axes(handles.AmpPlot); cla reset;
61. set(handles.AmpPlot, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.AmpPlot.Visible = 'off';
62. axes(handles.AmpPlot2); cla reset;
63. set(handles.AmpPlot2, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.AmpPlot2.Visible = 'off';
64. axes(handles.legendImage); cla reset;
65. set(handles.legendImage, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.legendImage.Visible = 'off';
66. axes(handles.jetImage); cla reset;
67. set(handles.jetImage, 'box', 'off', 'XTickLabel', [], 'XTick', [], 'YTickLabel', [], 'YTick', []); handles.jetImage.Visible = 'off';

68. set(handles.heatmapMin, 'Visible', 'off');
69. set(handles.heatmapMax, 'Visible', 'off');
70. set(handles.heatmapMinLbl, 'Visible', 'off');
71. set(handles.heatmapMaxLbl, 'Visible', 'off');
72. set(handles.heatmapScaleBar, 'Visible', 'off');
73. set(handles.CVtitle, 'Visible', 'on');

74. set(handles.dcCalButton, 'Enable', 'off');
75. set(handles.runButton, 'String', 'Run', 'foregroundColor', 'red', 'fontweight', 'bold', 'Enable', 'off');
76. set(handles.DAQButton, 'String', 'Initiate', 'foregroundColor', 'red', 'fontweight', 'bold', 'Enable', 'off');
77. set(handles.frameNumber, 'Enable', 'off');
78. set(handles.framePerSecond, 'Enable', 'off');

79. set(handles.CVcycleEdit, 'Enable', 'on');
80. set(handles.CVperiodEdit, 'Enable', 'on');
81. set(handles.CVvminEdit, 'Enable', 'on');
82. set(handles.CVvmaxEdit, 'Enable', 'on');
83. set(handles.CVrateEdit, 'Enable', 'on');
84. set(handles.CVSingleButton, 'Enable', 'on');
85. set(handles.CVBaseButton, 'Enable', 'on');
86. set(handles.CVBaseSigButton, 'Enable', 'on');
87. set(handles.runButtonV, 'Enable', 'off');
88. set(handles.CVcalculateButton, 'String', 'Initiate', 'Enable', 'on', 'foregroundColor', 'red', 'fontweight', 'bold');

89. set(handles.AmpResetTEdit, 'Enable', 'off');
90. set(handles.AmpSampleTEdit, 'Enable', 'off');
91. set(handles.AmpRecoverTEdit, 'Enable', 'off');
92. set(handles.AmpVmaxEdit, 'Enable', 'off');
93. set(handles.AmpVminEdit, 'Enable', 'off');
94. set(handles.AmpTransRateEdit, 'Enable', 'off');

95. set(handles.AmperometryPanel, 'HighlightColor', 'red');
96. set(handles.VoltametryPanel, 'HighlightColor', 'green');

97. set(handles.enableAmpButton, 'String', 'OFF', 'foregroundColor', 'red', 'fontweight', 'bold');
98. set(handles.enableAmpButton, 'Value', 0);
99. set(handles.enableCVButton, 'Value', 1, 'String', 'ON', 'foregroundColor', 'green', 'fontweight', 'bold');

100. end

```

## C.2.7. vMonitor.m

```

1. function vMonitor(hObject, eventdata, parent_GUI)
2. handles = guidata(parent_GUI);
3. due = handles.due;

4. fprintf(due, num2str(9)); % Send data to serial
5. vValue = fscanf(due, '%s'); % receive reading result

6. % ADCcont format: vCHIP-|vCHIP+|vConst+|vConst-|vIN3+|vIN3-|RE|CE|TIA_VDC|TIA_CM|vIN2-|vIN2+|i2V+|i2V-|i3V-|i3V+|tLDO+|tLDO-
   % |tQ12|tQ34|
7. % char length :
8. % char length :
9. % if -isempty(vValue)
10. % Voltage Monitor
11. % Vin = 3Vout-4
12. % Vout =(Vin+4)/3
13. A = -0.015;
14. B = 0.995;
15. set(handles.measVChipNText, 'String', num2str((str2double(vValue(1:6)))/B-A, '%.3f')); % vCHIP-
16. set(handles.measVChipPText, 'String', num2str((str2double(vValue(7:12)))/B+A, '%.3f')); % vCHIP+
17. set(handles.measVConstPText, 'String', num2str((str2double(vValue(13:18)))/B+A, '%.3f')); % vConst+
18. set(handles.measVConstNText, 'String', num2str((str2double(vValue(19:24)))/B-A, '%.3f')); % vConst-
19. set(handles.meas3VPsuPText, 'String', num2str((str2double(vValue(25:30)))/B+A, '%.3f')); % vPSU3V+
20. set(handles.meas3VPsuNText, 'String', num2str((str2double(vValue(31:36)))/B-A, '%.3f')); % vPSU3V-

21. set(handles.measVRE, 'String', num2str((str2double(vValue(37:42))-3)/B+A, '%.3f')); % vRE
22. set(handles.measVCE, 'String', num2str((str2double(vValue(43:48))-3)/B+A, '%.3f')); % vCE

```



```

23. %      Vin = 2Vout-2.1
24. %      Vout =(Vin+2.1)/2
25. A = 0;
26. B = 1;
27. set(handles.measVTIADc,'String',num2str((str2double(vValue(49:54))+A-3)/B, '%.3f')); % vTIA_VDC
28. set(handles.measVTIACm,'String',num2str((str2double(vValue(55:60))+A-3)/B, '%.3f')); % vTIA_CM

29. A = 0;
30. B = 0.995;
31. C = 0.13; % baseline calibration
32. set(handles.meas2VPsuNText,'String',num2str((str2double(vValue(61:66)))/B+A, '%.3f')); % vPSU2V-
33. set(handles.meas2VPsuPText,'String',num2str((str2double(vValue(67:71)))/B-A, '%.3f')); % vPSU2V+
34. set(handles.meas2IPsuPText,'String',num2str((str2double(vValue(73:77)))/B-C, '%.2f')); % iPSU2V+
35. set(handles.meas2IPsuNText,'String',num2str((str2double(vValue(78:82)))/B-C, '%.2f')); % iPSU2V-
36. set(handles.meas3IPsuNText,'String',num2str((str2double(vValue(83:87)))/B+A, '%.2f')); % iPSU3V-
37. set(handles.meas3IPsuPText,'String',num2str((str2double(vValue(88:92)))/B-A, '%.2f')); % iPSU3V+

38. %      Temperature Monitor
39. set(handles.measTLDOPos,'String',num2str(str2double(vValue(93:97)), '%.2f')); % tLDO_Pos
40. set(handles.measTLDONeg,'String',num2str(str2double(vValue(98:102)), '%.2f')); % tLDO_Neg
41. set(handles.measTChipQ12,'String',num2str(str2double(vValue(103:107)), '%.2f')); % tChip_Q12
42. set(handles.measTChipQ34,'String',num2str(str2double(vValue(108:112)), '%.2f')); % tChip_Q34
43. end

44. guidata(parent_GUI,handles);

```



## C.3. MATLAB Post-Processing GUI

### C.3.1. prePlotProcess.m

```
1. %% STEP 1 - Combine data
2. samplingLength = uint16(size(ChipInfo,1));
3. sampleGroupLength = uint16(size(DataSorted_001,2));
4. numOfGroup = samplingLength/sampleGroupLength;
5. if (samplingLength == sampleGroupLength)
6.     numOfGroup = 1;
7. end
8. for i = 1:numOfGroup
9.     eval(['aiDataSorted' sprintf('(:,((%d-1)*100)+1:((%d-1)*100)+sampleGroupLength)',i,i) ']= ' sprintf('DataSorted_%03d', i) ']);
10. end
11. clearvars -except aiDataSorted calData ChipInfo DataFramedRaw procFrameQ procFrameSpa timeFrameVector vCtrElt vRefElt frameQuantile
    ProcessQuantile;
12. %% STEP 2 - Remove Frames
13. % detect and remove recovery frames
14. RETresh = 0.5; % RE voltage threshold
15. indexMarginBefore = 2; % frame to remove before switching point
16. indexMarginAfter = 8; % frame to remove after switching point
17. indexSampling = find(vRefElt > RETresh); % find index location that has RE voltage threshold
18. indexSamplingMargin = find(diff(indexSampling) > 2)+1; % find the starting of transition
19. indexSamplingNew = indexSampling;
20. % detect and remove the frame, length indicated by indexMargin, after the transition point
21. for itrans = 1:length(indexSamplingMargin)
22.     for i = 1:indexMarginBefore
23.         indexSamplingNew(indexSamplingMargin(itrans)-i) = 0;
24.     end
25.     for i = 1:indexMarginAfter
26.         indexSamplingNew(indexSamplingMargin(itrans)+i-1) = 0;
27.     end
28. end
29. indexSamplingNew(indexSamplingNew==0)=[];
30. indexSampling = indexSamplingNew;
31. for i = 1:length(indexSampling)
32.     indexSelect = indexSampling(i);
33.     aiDataSorted_RecoveryCleaned(:,i) = aiDataSorted(:,indexSelect);
34.     ChipInfo_RecoveryCleaned(i,:) = ChipInfo(indexSelect,:);
35.     % DataFramed_RecoveryCleaned(:,i) = DataFramedRaw(:,indexSelect);
36.     frameQuantile_RecoveryCleaned(i,:) = frameQuantile(indexSelect,:);
37.     % procFrameSpa_RecoveryCleaned(:,i) = procFrameSpa(:,indexSelect);
38.     timeFrameVector_RecoveryCleaned(i,:) = timeFrameVector(indexSelect,:);
39.     vCtrElt_RecoveryCleaned(i,:) = vCtrElt(indexSelect,:);
40.     vRefElt_RecoveryCleaned(i,:) = vRefElt(indexSelect,:);
41. end
42. aiDataSorted = aiDataSorted_RecoveryCleaned;
43. ChipInfo = ChipInfo_RecoveryCleaned;
44. % DataFramedRaw = DataFramedRaw_RecoveryCleaned;
45. frameQuantile = frameQuantile_RecoveryCleaned;
46. % procFrameSpa = procFrameSpa_RecoveryCleaned;
47. timeFrameVector = timeFrameVector_RecoveryCleaned;
48. vCtrElt = vCtrElt_RecoveryCleaned;
49. vRefElt = vRefElt_RecoveryCleaned;
50. clearvars -except aiDataSorted calData ChipInfo DataFramedRaw procFrameQ procFrameSpa timeFrameVector vCtrElt vRefElt frameQuantile
    ProcessQuantile;
51. %% manual frame removal
52. startRemove = 1;
53. endRemove = 4;
54. ChipInfo = ChipInfo([1:startRemove-1, endRemove+1:end],:);
55. aiDataSorted = aiDataSorted(:,[1:startRemove-1, endRemove+1:end]);
56. frameQuantile = frameQuantile([1:startRemove-1, endRemove+1:end],:);
57. % procFrameQ = procFrameQ(:,[1:startRemove-1, endRemove+1:end]);
58. timeFrameVector = timeFrameVector([1:startRemove-1, endRemove+1:end]);
59. vCtrElt = vCtrElt([1:startRemove-1, endRemove+1:end]);
60. vRefElt = vRefElt([1:startRemove-1, endRemove+1:end]);
61. clearvars -except calData ChipInfo aiDataSorted procFrameQ procFrameSpa timeFrameVector vCtrElt vRefElt frameQuantile
    ProcessQuantile;
62. %% STEP 3 - Process data
```



```

63. % constants to use
64. smoothing = 1;
65. constEltTh = 800; % Threshold for replacing the stuck electrodes, 200 signifies 20mV or 1nA.
66. constSmoothFrame = 10; % Constant for smoothing between frames, number of frame to consider for smoothing, span for moving
    averaging.
67. constSpaFilt = 4; % Constant for smoothing spatially, number of pixel around a pixel to consider for smoothing, span.

68. numOffFrame = size(aiDataSorted,2);
69. numOffChSorted = size(aiDataSorted,1);
70. numOffCh = 4096;
71. counter = 0;

72. % Map the 16k data to it's actual location, for raw data
73. f = waitbar(0,'Start');
74. for fr = 1:numOffFrame
75.     eoCtr = 0;
76.     for col = 0:63
77.         DataFramed(65+col,65:128,fr) = aiDataSorted((((col-eoCtr)*64)+eoCtr)+1:2:((col-eoCtr+1)*64+64+eoCtr),fr); %Q1
78.         DataFramed(64-col,65:128,fr) = aiDataSorted((((col-eoCtr)*64)+eoCtr)+numOffCh+1:2:((col-eoCtr+1)*64+64+numOffCh+eoCtr),fr);
            % Q2
79.         DataFramed(64-col,1:64,fr) = flip(aiDataSorted((((col-eoCtr)*64)+eoCtr)+numOffCh*2+1:2:((col-eoCtr+1)*64+64+numOffCh*2+eoCtr),fr));
            % Q3
80.         DataFramed(65+col,1:64,fr) = flip(aiDataSorted((((col-eoCtr)*64)+eoCtr)+numOffCh*3+1:2:((col-eoCtr+1)*64+64+numOffCh*3+eoCtr),fr));
            % Q1
81.         if (eoCtr == 1); eoCtr = 0; else eoCtr = 1; end
82.     end
83.     DataFramedRow(:,fr) = double(DataFramed(:,fr));
84. end
85. waitbar(1,f,{'Processing Data (1/5)':'RAW data mapped'});

86. % Map calibration data
87. eoCtr = 0;
88. for col = 0:63
89.     CalFramed(65+col,65:128) = calData((((col-eoCtr)*64)+eoCtr)+1:2:((col-eoCtr+1)*64+64+eoCtr)); %Q1
90.     CalFramed(64-col,65:128) = calData((((col-eoCtr)*64)+eoCtr)+numOffCh+1:2:((col-eoCtr+1)*64+64+numOffCh+eoCtr)); % Q2
91.     CalFramed(64-col,1:64) = flip(calData((((col-eoCtr)*64)+eoCtr)+numOffCh*2+1:2:((col-eoCtr+1)*64+64+numOffCh*2+eoCtr)); % Q3
92.     CalFramed(65+col,1:64) = flip(calData((((col-eoCtr)*64)+eoCtr)+numOffCh*3+1:2:((col-eoCtr+1)*64+64+numOffCh*3+eoCtr)); % Q1
93.     if (eoCtr == 1); eoCtr = 0; else eoCtr = 1; end
94. end

95. procFrameQ = DataFramedRow;

96. if (smoothing == 1)
97.     % Replace railing data with neighboring values
98.     for fr = 1:numOffFrame
99.         for i = 1: numOffChSorted
100.            if (aiDataSorted(i,fr) > constEltTh || aiDataSorted(i,fr) < -1*constEltTh)
101.                aiDataSorted(i,fr) = 0; % replace railed signal with 0
102.            counter = counter +1;
103.            end
104.        end
105.        waitbar(fr/numOffFrame,f,{'Processing Data (2/5)':'Replacing stuck electrodes with 0s'});
106.    end

107. % Replaces zeroes with neighborhood channels
108. for fr = 1:numOffFrame
109.     for i = 1: numOffChSorted
110.         if (aiDataSorted(i,fr) == 0)
111.             % replace the first address with the 2nd address
112.             aiDataSorted(i,fr) = aiDataSorted(i+1,fr);
113.         elseif (i==numOffChSorted) % replace the last address with the 2nd to lastaddress
114.             aiDataSorted(i,fr) = aiDataSorted(i-1,fr);
115.         else % replace the last address average between two electrodes
116.             aiDataSorted(i,fr) = (aiDataSorted(i-1,fr)+aiDataSorted(i+1,fr))/2;
117.         end
118.     end
119. end
120. waitbar(fr/numOffFrame,f,{'Processing Data (3/5)':'Replacing 0s with surrounding pixel'});
121. end

122. % Smooth transient, channel-to-channel smoothing, horizontally
123. ProcessQuantile(:,1) = quantile(double(aiDataSorted(:,1)),0.5); % median of raw

124. parfor ch = 1:16384
125.     aiDataSortedTrans(ch,:) = smooth(double(aiDataSorted(ch,:)),constSmoothFrame);
126. end
127. ProcessQuantile(:,2) = quantile(aiDataSortedTrans,0.5); % median of after frame to frame filtering,
    each electrode
128. ProcessQuantile(:,3) = smooth(double(ProcessQuantile(:,2)),numOffFrame*0.01); % smoothed median, or target median
129. ProcessQuantile(:,4) = ProcessQuantile(:,3)-ProcessQuantile(:,2); % distance to raw median

130. parfor fr = 1:numOffFrame
131.     aiDataSortedTransFrame(:,fr) = aiDataSortedTrans(:,fr) + ProcessQuantile(fr,4);
132. end
133. ProcessQuantile(:,5) = quantile(aiDataSortedTransFrame,0.5); % median of after frame to frame filtering
134. % waitbar(ch/16384,f,{'Processing Data (4/5)':'Smoothing between frame'});
135. waitbar(1,f,{'Processing Data (4/5)':'Smoothing between frame'});

```







### C.3.2. PlotGUI.m

```

1. function varargout = PlotGUI(varargin)
2. % PLOTGUI MATLAB code for PlotGUI.fig

3. % Begin initialization code - DO NOT EDIT
4. gui_Singleton = 1;
5. gui_State = struct('gui_Name',       mfilename, ...
6. 'gui_Singleton',   gui_Singleton, ...
7. 'gui_OpeningFcn', @PlotGUI_OpeningFcn, ...
8. 'gui_OutputFcn',  @PlotGUI_OutputFcn, ...
9. 'gui_LayoutFcn',  [], ...
10. 'gui_Callback',   []);
11. if nargin && ischar(varargin{1})
12.     gui_State.gui_Callback = str2func(varargin{1});
13. end

14. if nargout
15.     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
16. else
17.     gui_mainfcn(gui_State, varargin{:});
18. end
19. % End initialization code - DO NOT EDIT

20. % --- Executes just before PlotGUI is made visible.
21. function PlotGUI_OpeningFcn(hObject, eventdata, handles, varargin)
22. % initialize variable
23. handles.R = 20e6*1e4;
24. handles.frameNum = 1;
25. handles.heatmapMaximum = 0.5;
26. handles.heatmapOffset = 0;

27. % initialize overlay images
28. handles.imageOverlayFlag = 0;
29. set(handles.overlayOpticalButton,'String','Overlay: OFF','foregroundColor','red','fontWeight','bold');
30. handles.myImage = imread('image.jpg');

31. handles.legendPlotImage = imread('LegendPlot.png');
32. axes(handles.LegendPlot);
33. imshow(handles.legendPlotImage);

34. % import data
35. handles.DataFramedRaw = double(evalin('base', 'DataFramedRaw'));
36. % handles.DataFramedSmooth = double(evalin('base', 'DataFramedRaw'));
37. handles.DataFramedSmooth = double(evalin('base', 'procFrameSpa'));
38. handles.timeVector = evalin('base', 'timeFrameVector');
39. handles.CtrElt = evalin('base', 'vCtrElt');
40. handles.RefElt = evalin('base', 'vRefElt');

41. % transform framed data to vector
42. for frame = 1:size(handles.DataFramedRaw,3)
43.     for i = 1:128 % row
44.         handles.DataSortedRaw(1+((i-1)*128):128+((i-1)*128),frame) = handles.DataFramedRaw(i,:,frame);
45.         handles.DataSortedSmooth(1+((i-1)*128):128+((i-1)*128),frame) = handles.DataFramedSmooth(i,:,frame);
46.     end
47. end

48. % create vectors of quantiles
49. handles.frameRowQuantile(1:size(handles.DataFramedRaw,3),1:3) = quantile(double(handles.DataSortedRaw(:,:))/handles.R,[0.159,
0.5, 0.841]);
50. handles.frameRowQuantile(1:size(handles.DataFramedRaw,3),2) = mean(handles.DataSortedRaw(:,:)/handles.R);
51. handles.frameRowQuantile(1:size(handles.DataFramedRaw,3),4) = (abs(handles.frameRowQuantile(:,1) - handles.frameRowQuantile(:,2))
+ ...
(handles.frameRowQuantile(:,3) - handles.frameRowQuantile(:,2)))/2;% mean and 1 sigma around it.

53. handles.frameSmoothQuantile(1:size(handles.DataFramedRaw,3),1:3) =
quantile(double(handles.DataSortedSmooth(:,:))/handles.R,[0.159, 0.5, 0.841]);
54. handles.frameSmoothQuantile(1:size(handles.DataFramedRaw,3),2) = mean(handles.DataSortedSmooth(:,:)/handles.R);
55. handles.frameSmoothQuantile(1:size(handles.DataFramedRaw,3),4) =
(abs(handles.frameSmoothQuantile(:,1) -
handles.frameSmoothQuantile(:,2)) + ...
(handles.frameSmoothQuantile(:,3) - handles.frameSmoothQuantile(:,2)))/2;% median and 1 sigma around it.

57. % set constant base on imported data;
58. handles.frameLength = size(handles.DataFramedRaw,3);

59. % create time vector
60. handles.timeVector(1,2) = 0;
61. for i = 2:handles.frameLength
62.     handles.timeVector(i,2) = handles.timeVector(i-1,2) + handles.timeVector(i-1,1);
63. end

64. % initiate button and sliders
65. set(handles.frameNumSlider,'Value',1); % set slider position to rounded off value
66. set(handles.frameNumSlider,'Min',1); % set slider position to rounded off value
67. set(handles.frameNumSlider,'Max',handles.frameLength); % set slider position to rounded off value

```



```

68. set(handles.frameNumSlider,'SliderStep',[1/handles.frameLength, 0.025]);
69. set(handles.heatmapOffset,'Value', handles.heatmapOffset); % set slider position to rounded off value
70. set(handles.heatmapOffsetText,'String',num2str(handles.heatmapOffset,'%1f')); % print out the slider value
71. set(handles.heatmapMax,'Value', handles.heatmapMaximum); % set slider position to rounded off value
72. set(handles.heatmapMaxText,'String',num2str(handles.heatmapMaximum,'%1f')); % print out the slider value
73. set(handles.scaleN1,'String',num2str(handles.heatmapMaximum/2,'%2f'));
74. set(handles.scaleN2,'String',num2str(handles.heatmapMaximum/2,'%2f'));
75. set(handles.scaleP1,'String',num2str(handles.heatmapMaximum/2,'%2f'));
76. set(handles.scaleP2,'String',num2str(handles.heatmapMaximum/2,'%2f'));

77. % initiate axes
78. axes(handles.Heatmap1); axis off;
79. axes(handles.Heatmap2); axis off;
80. axes(handles.overlayAxes); axis off;

81. % initialize jet colormap
82. axes(handles.jetImage1); cla reset;
83. matlabImage = imread('JetColormap.png'); image(matlabImage);
84. axis off; axis image;
85. axes(handles.jetImage2); cla reset;
86. matlabImage = imread('JetColormap.png'); image(matlabImage);
87. axis off; axis image;

88. % initialize other variables
89. handles.spaOriX = 1; set(handles.spatialOriginX,'String',num2str(handles.spaOriX));
90. handles.spaOriY = 1; set(handles.spatialOriginY,'String',num2str(handles.spaOriY));
91. handles.spaDimX = 10; set(handles.spatialDimX,'String',num2str(handles.spaDimX));
92. handles.spaDimY = 10; set(handles.spatialDimY,'String',num2str(handles.spaDimY));

93. handles.spaOriXSig = 11; set(handles.spatialOriginXSig,'String',num2str(handles.spaOriXSig));
94. handles.spaOriYSig = 11; set(handles.spatialOriginYSig,'String',num2str(handles.spaOriYSig));
95. handles.spaDimXSig = 10; set(handles.spatialDimXSig,'String',num2str(handles.spaDimXSig));
96. handles.spaDimYSig = 10; set(handles.spatialDimYSig,'String',num2str(handles.spaDimYSig));

97. handles.output = hObject;

98. % Update handles structure
99. guidata(hObject, handles);
100. % UIWAIT makes PlotGUI wait for user response (see UIRESUME)
101. % uiwait(handles.figure1);

102. % --- Outputs from this function are returned to the command line.
103. function varargout = PlotGUI_OutputFcn(hObject, eventdata, handles)
104. % varargout cell array for returning output args (see VARARGOUT);
105. % hObject handle to figure
106. % eventdata reserved - to be defined in a future version of MATLAB
107. % handles structure with handles and user data (see GUIDATA)

108. % Get default command line output from handles structure
109. varargout{1} = handles.output;

110. % --- Executes on slider movement.
111. function frameNumSlider_Callback(hObject, eventdata, handles)
112. handles = guidata(hObject);

113. handles.frameNum = get(handles.frameNumSlider,'Value'); % retrieve data from slider
114. handles.frameNum = round(handles.frameNum,0); % round off this value

115. heatmapPlot(handles); % plot heatmap, call function

116. guidata(hObject,handles);

117. % --- Executes on button press in autoRunButton.
118. function autoRunButton_Callback(hObject, eventdata, handles)
119. handles = guidata(hObject);

120. handles.autoFlag = 1;
121. handles.frameNum = 1;
122. set(handles.frameNumSlider,'Value',handles.frameNum); % set slider position to rounded off value

123. for i=1:size(handles.DataFramedRaw,3)
124. handles.frameNum = i;
125. set(handles.frameNumSlider,'Value',handles.frameNum); % set slider position to rounded off value
126. heatmapPlot(handles); % plot heatmap, call function
127. end
128. guidata(hObject,handles);

129. % --- Executes on slider movement.
130. function heatmapOff_Callback(hObject, eventdata, handles)
131. handles = guidata(hObject);

132. handles.heatmapOffset = get(handles.heatmapOff,'Value'); % retrieve data from slider
133. handles.heatmapOffset = round(handles.heatmapOffset,2); % round off this value

```



```

134. set(handles.heatmapOff, 'Value', handles.heatmapOffset); % set slider position to rounded off value
135. set(handles.heatmapOffText, 'String', num2str(handles.heatmapOffset, '%.2f')); % print out the slider value
136. heatmapPlot(handles); % replot heatmap, call function

137. guidata(hObject, handles);

138. % --- Executes on slider movement.
139. function heatmapMax_Callback(hObject, eventdata, handles)
140. handles = guidata(hObject);

141. handles.heatmapMaximum = get(handles.heatmapMax, 'Value'); % retrieve data from slider
142. handles.heatmapMaximum = round(handles.heatmapMaximum, 1); % round off this value
143. heatmapPlot(handles); % replot heatmap, call function

144. guidata(hObject, handles);

145. % --- Executes on button press in spatialCalculateButton.
146. function spatialCalculateButton_Callback(hObject, eventdata, handles)
147. handles = guidata(hObject);

148. % reset reference values
149. handles.DataFramedSmoothRef = []; % reference spot defined by user, origin and dimension
150. handles.DataSortedSmoothRef = []; % sorted version of the framed
151. handles.DataFramedSmoothSig = []; % signal spot defined by user, origin and dimension
152. handles.DataSortedSmoothSig = []; % sorted version of the framed - signal
153. handles.frameSmoothRefQuantile = []; % quantiles of sorted data

154. % extract the Specified spot values - Reference
155. handles.DataFramedSmoothRef(:, :, :) = handles.DataFramedSmooth(handles.spaOriY:handles.spaOriY+handles.spaDimY-1, handles.spaOriX:handles.spaOriX+handles.spaDimX-1, :);
156. % handles.spaOriY:handles.spaOriY+handles.spaDimY-1
157. % handles.spaOriX:handles.spaOriX+handles.spaDimX-1

158. % transform framed data to vector
159. for frame = 1:size(handles.DataFramedSmooth, 3)
160. for i = 1:handles.spaDimY % row
161. handles.DataSortedSmoothRef(1+((i-1)*handles.spaDimX):handles.spaDimX+((i-1)*handles.spaDimX), frame) = handles.DataFramedSmoothRef(i, :, frame);
162. end
163. end

164. % extract the Specified spot values - Signal
165. % handles.spaOriYSig = 85;
166. handles.DataFramedSmoothSig(:, :, :) = handles.DataFramedSmooth(handles.spaOriYSig:handles.spaOriYSig+handles.spaDimYSig-1, handles.spaOriXSig:handles.spaOriXSig+handles.spaDimXSig-1, :);
167. % handles.spaOriYSig = 35;
168. % handles.DataFramedSmoothSig(:, :, 68:199) = handles.DataFramedSmooth(handles.spaOriYSig:handles.spaOriYSig+handles.spaDimYSig-1, handles.spaOriXSig:handles.spaOriXSig+handles.spaDimXSig-1, 68:199);
169. % handles.spaOriYSig:handles.spaOriYSig+handles.spaDimYSig-1
170. % handles.spaOriXSig:handles.spaOriXSig+handles.spaDimXSig-1

171. % transform framed data to vector
172. for frame = 1:size(handles.DataFramedSmooth, 3)
173. for i = 1:handles.spaDimYSig % row
174. handles.DataSortedSmoothSig(1+((i-1)*handles.spaDimXSig):handles.spaDimXSig+((i-1)*handles.spaDimXSig), frame) = handles.DataFramedSmoothSig(i, :, frame);
175. end
176. end

177. % calculate quantiles Reference
178. handles.frameSmoothRefQuantile(1:size(handles.DataFramedSmoothRef, 3), 1:3) = quantile(double(handles.DataSortedSmoothRef(:, :))/handles.R, [0.159, 0.5, 0.841]);
179. handles.frameSmoothRefQuantile(1:size(handles.DataFramedSmoothRef, 3), 2) = mean(handles.DataSortedSmoothRef(:, :)/handles.R);
180. handles.frameSmoothRefQuantile(1:size(handles.DataFramedSmoothRef, 3), 4) = (abs(handles.frameSmoothRefQuantile(:, 1) - handles.frameSmoothRefQuantile(:, 2)) + ...
181. (handles.frameSmoothRefQuantile(:, 3) - handles.frameSmoothRefQuantile(:, 2)))/2; % median and 1 sigma around it.

182. % calculate quantiles Signal
183. handles.frameSmoothSigQuantile(1:size(handles.DataFramedSmoothSig, 3), 1:3) = quantile(double(handles.DataSortedSmoothSig(:, :))/handles.R, [0.159, 0.5, 0.841]);
184. handles.frameSmoothSigQuantile(1:size(handles.DataFramedSmoothSig, 3), 2) = mean(handles.DataSortedSmoothSig(:, :)/handles.R);
185. handles.frameSmoothSigQuantile(1:size(handles.DataFramedSmoothSig, 3), 4) = (abs(handles.frameSmoothSigQuantile(:, 1) - handles.frameSmoothSigQuantile(:, 2)) + ...
186. (handles.frameSmoothSigQuantile(:, 3) - handles.frameSmoothSigQuantile(:, 2)))/2; % median and 1 sigma around it.
187. handles.frameSmoothSigQuantile(1:size(handles.DataFramedSmoothSig, 3), 5) = quantile(double(handles.DataSortedSmoothSig(:, :))/handles.R, 0.95); %use for max consumption calibration

188. % calculate processed frames
189. for frame = 1:size(handles.DataFramedSmooth, 3)
190. % handles.DataFramedSmoothProc(:, :, frame) = double(handles.DataFramedSmooth(:, :, frame)) - handles.frameSmoothRefQuantile(frame, 2)*handles.R;
191. % handles.DataFramedSmoothProc(:, :, frame) = double(handles.DataFramedSmooth(:, :, frame)) - handles.frameSmoothSigQuantile(frame, 5)*handles.R;
a. handles.DataFramedSmoothProc(:, :, frame) = double(handles.DataFramedSmooth(:, :, frame));
192. end

```



```

193. heatmapPlot(handles);           % replot heatmap, call function
194. guidata(hObject,handles);

195. function spatialOriginX_Callback(hObject, eventdata, handles)
196. handles = guidata(hObject);
197. handles.spaOriX = str2double(get(hObject,'String'));
198. % reset value if more than 128, the max
199. if (handles.spaOriX > 128)
200. handles.spaOriX = 128;
201. set(handles.spatialOriginX,'String',num2str(handles.spaOriX));
202. set(handles.spatialDimX,'String',num2str(0));
203. end
204. % reset value if is than 1, the min
205. if (handles.spaOriX < 1)
206. handles.spaOriX = 1;
207. set(handles.spatialOriginX,'String',num2str(handles.spaOriX));
208. end
209. guidata(hObject,handles);

210. function spatialOriginY_Callback(hObject, eventdata, handles)
211. handles = guidata(hObject);
212. handles.spaOriY = str2double(get(hObject,'String'));
213. % reset value if more than 128, the max
214. if (handles.spaOriY > 128)
215. handles.spaOriY = 128;
216. set(handles.spatialOriginY,'String',num2str(handles.spaOriY));
217. set(handles.spatialDimY,'String',num2str(0));
218. end
219. % reset value if is than 1, the min
220. if (handles.spaOriY < 1)
221. handles.spaOriY = 1;
222. set(handles.spatialOriginY,'String',num2str(handles.spaOriY));
223. end
224. guidata(hObject,handles);

225. function spatialDimX_Callback(hObject, eventdata, handles)
226. handles = guidata(hObject);
227. handles.spaDimX = str2double(get(hObject,'String'));
228. % reset value if more than 128-Origin
229. if (handles.spaDimX > 129-handles.spaOriX)
230. handles.spaDimX = 129-handles.spaOriX;
231. set(handles.spatialDimX,'String',num2str(handles.spaDimX));
232. end
233. if (handles.spaDimX < 0)
234. handles.spaDimX = 0;
235. set(handles.spatialDimX,'String',num2str(handles.spaDimX));
236. end
237. guidata(hObject,handles);

238. function spatialDimY_Callback(hObject, eventdata, handles)
239. handles = guidata(hObject);
240. handles.spaDimY = str2double(get(hObject,'String'));
241. % reset value if more than 128-Origin
242. if (handles.spaDimY > 129-handles.spaOriY)
243. handles.spaDimY = 129-handles.spaOriY;
244. set(handles.spatialDimY,'String',num2str(handles.spaDimY));
245. end
246. if (handles.spaDimY < 0)
247. handles.spaDimY = 0;
248. set(handles.spatialDimY,'String',num2str(handles.spaDimY));
249. end
250. guidata(hObject,handles);

251. function spatialOriginXSig_Callback(hObject, eventdata, handles)
252. handles = guidata(hObject);
253. handles.spaOriXSig = str2double(get(hObject,'String'));
254. % reset value if more than 128, the max
255. if (handles.spaOriXSig > 128)
256. handles.spaOriXSig = 128;
257. set(handles.spatialOriginXSig,'String',num2str(handles.spaOriXSig));
258. set(handles.spatialDimXSig,'String',num2str(0));
259. end
260. % reset value if is than 1, the min
261. if (handles.spaOriXSig < 1)
262. handles.spaOriXSig = 1;
263. set(handles.spatialOriginXSig,'String',num2str(handles.spaOriXSig));
264. end
265. guidata(hObject,handles);

266. function spatialOriginYSig_Callback(hObject, eventdata, handles)
267. handles = guidata(hObject);
268. handles.spaOriYSig = str2double(get(hObject,'String'));
269. % reset value if more than 128, the max
270. if (handles.spaOriYSig > 128)
271. handles.spaOriYSig = 128;

```



```

272. set(handles.spatialOriginYSig,'String',num2str(handles.spaOriYSig));
273. set(handles.spatialDimYSig,'String',num2str(0));
274. end
275. % reset value if is than 1, the min
276. if (handles.spaOriYSig < 1)
277. handles.spaOriYSig = 1;
278. set(handles.spatialOriginYSig,'String',num2str(handles.spaOriYSig));
279. end
280. guidata(hObject,handles);

281. function spatialDimXSig_Callback(hObject, eventdata, handles)
282. handles = guidata(hObject);
283. handles.spaDimXSig = str2double(get(hObject,'String'));
284. % reset value if more than 128-Origin
285. if (handles.spaDimXSig > 129-handles.spaOriX)
286. handles.spaDimXSig = 129-handles.spaOriX;
287. set(handles.spatialDimXSig,'String',num2str(handles.spaDimXSig));
288. end
289. if (handles.spaDimXSig < 0)
290. handles.spaDimXSig = 0;
291. set(handles.spatialDimXSig,'String',num2str(handles.spaDimXSig));
292. end
293. guidata(hObject,handles);

294. function spatialDimYSig_Callback(hObject, eventdata, handles)
295. handles = guidata(hObject);
296. handles.spaDimYSig = str2double(get(hObject,'String'));
297. % reset value if more than 128-Origin
298. if (handles.spaDimYSig > 129-handles.spaOriYSig)
299. handles.spaDimYSig = 129-handles.spaOriYSig;
300. set(handles.spatialDimYSig,'String',num2str(handles.spaDimYSig));
301. end
302. if (handles.spaDimYSig < 0)
303. handles.spaDimYSig = 0;
304. set(handles.spatialDimYSig,'String',num2str(handles.spaDimYSig));
305. end
306. guidata(hObject,handles);

307. % --- Executes on button press in overlayOpticalButton.
308. function overlayOpticalButton_Callback(hObject, eventdata, handles)
309. handles = guidata(hObject);
310. if (handles.imageOverlayFlag == 0)
311. handles.imageOverlayFlag = 1;
312. set(handles.overlayOpticalButton,'String','Overlay: ON','foregroundColor','green','fontWeight','bold');
313. set(handles.uipanel1,'Title','Optical Overlay');
314. axes(handles.overlayAxes);
315. imshow(handles.myImage);
316. alpha(1); %transparency of image
317. heatmapPlot(handles); % plot heatmap, call function

318. elseif (handles.imageOverlayFlag == 1)
319. handles.imageOverlayFlag = 0;
320. set(handles.overlayOpticalButton,'String','Overlay: OFF','foregroundColor','red','fontWeight','bold');
321. set(handles.uipanel1,'Title','Raw Data');
322. heatmapPlot(handles); % plot heatmap, call function
323. end

324. guidata(hObject,handles);

325. % --- Executes on button press in extractDataButton.
326. function extractDataButton_Callback(hObject, eventdata, handles)
327. handles = guidata(hObject);

328. refSorted = double(handles.DataSortedSmoothRef)/handles.R;
329. refFramed = double(handles.DataFramedSmoothRef)/handles.R;
330. refInfo(1) = handles.spaOriX;
331. refInfo(2) = handles.spaOriY;
332. refInfo(3) = handles.spaDimX;
333. refInfo(4) = handles.spaDimY;

334. sigSorted = double(handles.DataSortedSmoothSig)/handles.R;
335. sigFramed = double(handles.DataFramedSmoothSig)/handles.R;
336. sigInfo(1) = handles.spaOriXSig;
337. sigInfo(2) = handles.spaOriYSig;
338. sigInfo(3) = handles.spaDimXSig;
339. sigInfo(4) = handles.spaDimYSig;

340. rate = handles.frameSmoothRefQuantile(:,2)-handles.frameSmoothSigQuantile(:,2);
341. time = handles.timeVector(:,2);
342. handles.fitted = polyfit(time,rate,1);
343. rateFitted = handles.fitted;
344. vRefElt = handles.RefElt(:,1);
345. vCtrElt = handles.CtrElt(:,1);
346. SigQuantileCal = handles.frameSmoothSigQuantile(:,5)*handles.R;
347. RefQuantileCal = handles.frameSmoothRefQuantile(:,2)*handles.R;
348. SigQuantile = handles.frameSmoothSigQuantile(:,2)*handles.R;

```



```

349. save('extractedData','refSorted','refFramed', 'refInfo', 'sigSorted', 'sigFramed', 'sigInfo', 'rate', 'time',
'rateFitted','vRefElt','vCtrElt', 'SigQuantileCal', 'RefQuantileCal', 'SigQuantile');
350. guidata(hObject,handles);

```

### C.3.3. heatmapPlot.m

```

1. function heatmapPlot(handles)

2. % set frame# and time info
3. set(handles.frameCurrent1,'String',num2str(handles.frameNum,'%0f')); % print current frame
4. set(handles.frameCurrent2,'String',num2str(handles.frameNum,'%0f')); % print current frame
5. set(handles.frameFinal1,'String',num2str(handles.frameLength,'%0f')); % print final frame
6. set(handles.frameFinal2,'String',num2str(handles.frameLength,'%0f')); % print final frame
7. set(handles.timeCurrent1,'String',num2str(handles.timeVector(handles.frameNum,2),'%0f')); % print current time
8. set(handles.timeCurrent2,'String',num2str(handles.timeVector(handles.frameNum,2),'%0f')); % print current time
9. set(handles.timeFinal1,'String',num2str(handles.timeVector(end,2),'%0f')); % print final frame
10. set(handles.timeFinal2,'String',num2str(handles.timeVector(end,2),'%0f')); % print final frame

11. % set scaling values
12. handles.scaleMin = ((-1*(handles.heatmapMaximum/2))+handles.heatmapOffset)*1e-9;
13. handles.scaleMax = ((1*(handles.heatmapMaximum/2))+handles.heatmapOffset)*1e-9;

14. % create rounded tick marks
15. for i = 1:13
16. handles.scaleRange(i) = handles.scaleMin + ((i-1) * round((handles.scaleMax-handles.scaleMin)/13),2,'significant'));
17. end

18. %%%%%%%%% Heatmap axis 2 %%%%%%%%%
19. imagesc(handles.DataFramedSmoothProc(:, :, handles.frameNum)/handles.R*1, 'Parent', handles.Heatmap2, [handles.scaleRange(1)
handles.scaleRange(end)]);
20. colormap('jet'); axis off;
21. handles.Heatmap2.XTick = [];
22. handles.Heatmap2.YTick = [];

23. % % create reference box
24. % % location of corner in relative to quadrant
25. % handles.spaQ1 = [handles.spaOriX+handles.spaDimX-1 handles.spaOriY+handles.spaDimY-1];
26. % handles.spaQ2 = [handles.spaOriX+handles.spaDimX-1 handles.spaOriY];
27. % handles.spaQ3 = [handles.spaOriX handles.spaOriY];
28. % handles.spaQ4 = [handles.spaOriX handles.spaOriY+handles.spaDimY-1];
29. %
30. % line(handles.Heatmap2,[handles.spaQ1(1) handles.spaQ2(1)],[handles.spaQ1(2) handles.spaQ2(2)],'Color',[0.65, 0.65,
0.65],'LineWidth',2); % line from Q1 to Q2 corners
31. % hold(handles.Heatmap2,'on');
32. % line(handles.Heatmap2,[handles.spaQ2(1) handles.spaQ3(1)],[handles.spaQ2(2) handles.spaQ3(2)],'Color',[0.65, 0.65,
0.65],'LineWidth',2); % line from Q2 to Q3 corners
33. % line(handles.Heatmap2,[handles.spaQ3(1) handles.spaQ4(1)],[handles.spaQ3(2) handles.spaQ4(2)],'Color',[0.65, 0.65,
0.65],'LineWidth',2); % line from Q3 to Q4 corners
34. % line(handles.Heatmap2,[handles.spaQ4(1) handles.spaQ1(1)],[handles.spaQ4(2) handles.spaQ1(2)],'Color',[0.65, 0.65,
0.65],'LineWidth',2); % line from Q4 to Q1 corner
35. %
36. % % create signal box
37. % % location of corner in relative to quadrant
38. % handles.spaQ1s = [handles.spaOriXSig+handles.spaDimXSig-1 handles.spaOriYSig+handles.spaDimYSig-1];
39. % handles.spaQ2s = [handles.spaOriXSig+handles.spaDimXSig-1 handles.spaOriYSig];
40. % handles.spaQ3s = [handles.spaOriXSig handles.spaOriYSig];
41. % handles.spaQ4s = [handles.spaOriXSig handles.spaOriYSig+handles.spaDimYSig-1];
42. %
43. % line(handles.Heatmap2,[handles.spaQ1s(1) handles.spaQ2s(1)],[handles.spaQ1s(2) handles.spaQ2s(2)],'Color','m','LineWidth',2);
% line from Q1 to Q2 corners
44. % line(handles.Heatmap2,[handles.spaQ2s(1) handles.spaQ3s(1)],[handles.spaQ2s(2) handles.spaQ3s(2)],'Color','m','LineWidth',2);
% line from Q2 to Q3 corners
45. % line(handles.Heatmap2,[handles.spaQ3s(1) handles.spaQ4s(1)],[handles.spaQ3s(2) handles.spaQ4s(2)],'Color','m','LineWidth',2);
% line from Q3 to Q4 corners
46. % line(handles.Heatmap2,[handles.spaQ4s(1) handles.spaQ1s(1)],[handles.spaQ4s(2) handles.spaQ1s(2)],'Color','m','LineWidth',2);
% line from Q4 to Q1 corner
47. %%%%%%%%% Heatmap axis 1 %%%%%%%%%
48. axes(handles.Heatmap1);
49. handles.Heatmap2.XTick = [];
50. handles.Heatmap2.YTick = [];

51. if (handles.imageOverlayFlag == 1)
52. cla(handles.Heatmap1);
53. end

54. if (handles.imageOverlayFlag == 0)
55. imagesc(handles.DataFramedRaw(:, :, handles.frameNum)/handles.R*1, 'Parent', handles.Heatmap1, [handles.scaleRange(1)
handles.scaleRange(end)]);
56. colormap('jet'); axis off;
57. end

58. % % create reference box

```



```

59. % % location of corner in relative to quadrant
60. % line(handles.Heatmap1,[handles.spaQ1(1) handles.spaQ2(1)], [handles.spaQ1(2) handles.spaQ2(2)], 'Color', [0.65, 0.65,
0.65], 'LineWidth', 2); % line from Q1 to Q2 corners
61. % hold(handles.Heatmap1, 'on');
62. % line(handles.Heatmap1,[handles.spaQ2(1) handles.spaQ3(1)], [handles.spaQ2(2) handles.spaQ3(2)], 'Color', [0.65, 0.65,
0.65], 'LineWidth', 2); % line from Q2 to Q3 corners
63. % line(handles.Heatmap1,[handles.spaQ3(1) handles.spaQ4(1)], [handles.spaQ3(2) handles.spaQ4(2)], 'Color', [0.65, 0.65,
0.65], 'LineWidth', 2); % line from Q3 to Q4 corners
64. % line(handles.Heatmap1,[handles.spaQ4(1) handles.spaQ1(1)], [handles.spaQ4(2) handles.spaQ1(2)], 'Color', [0.65, 0.65,
0.65], 'LineWidth', 2); % line from Q4 to Q1 corner
65. %
66. % % create signal box
67. % % location of corner in relative to quadrant
68. % line(handles.Heatmap1,[handles.spaQ1s(1) handles.spaQ2s(1)], [handles.spaQ1s(2) handles.spaQ2s(2)], 'Color', 'm', 'LineWidth', 2);
% line from Q1 to Q2 corners
69. % line(handles.Heatmap1,[handles.spaQ2s(1) handles.spaQ3s(1)], [handles.spaQ2s(2) handles.spaQ3s(2)], 'Color', 'm', 'LineWidth', 2);
% line from Q2 to Q3 corners
70. % line(handles.Heatmap1,[handles.spaQ3s(1) handles.spaQ4s(1)], [handles.spaQ3s(2) handles.spaQ4s(2)], 'Color', 'm', 'LineWidth', 2);
% line from Q3 to Q4 corners
71. % line(handles.Heatmap1,[handles.spaQ4s(1) handles.spaQ1s(1)], [handles.spaQ4s(2) handles.spaQ1s(2)], 'Color', 'm', 'LineWidth', 2);
% line from Q4 to Q1 corner

72. % fill(handles.Heatmap2,[100 100 0 0],[100 100 0 0],[0.65, 0.65, 0.65], 'FaceAlpha', .2, 'LineStyle', 'none');
73. hold(handles.Heatmap1, 'off');

74. % Time Domain Plot
75. % plot raw data quantiles
76. % plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameRawQuantile(:, 2)*1, 'LineWidth', 2, 'Color', [1, 0.65, 0]); % Q2
77. % hold(handles.timePlot2, 'on');
78. % plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameRawQuantile(:, 1)*1, 'LineWidth', 1, 'Color', [1, 0.65, 0]); % Q1
79. % plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameRawQuantile(:, 3)*1, 'LineWidth', 1, 'Color', [1, 0.65, 0]); % Q3

80. plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameSmoothQuantile(:, 2)*1, 'LineWidth', 1, 'Color', [0.2, 0.2, 0.2]); %
Q2 - Smoothed original
81. hold(handles.timePlot2, 'on');
82. % plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameSmoothQuantile(:, 1), 'LineWidth', 1, 'Color', [1, 0.65, 0]); % Q1
83. % plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameSmoothQuantile(:, 3), 'LineWidth', 1, 'Color', [1, 0.65, 0]); % Q3

84. plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameSmoothRefQuantile(:, 2)-
handles.frameSmoothSigQuantile(:, 5)*1, 'LineWidth', 2, 'Color', [0.65, 0.65, 0.65]); % Q2 - Reference
85. plot(handles.timePlot2, handles.timeVector(:, 2), handles.frameSmoothSigQuantile(:, 2)-
handles.frameSmoothSigQuantile(:, 5)*1, 'LineWidth', 2, 'Color', 'm'); % Q2 - Signal
86. plot(handles.timePlot2, handles.timeVector(:, 2), (handles.frameSmoothRefQuantile(:, 2)-
handles.frameSmoothSigQuantile(:, 2))*1, 'LineWidth', 2, 'Color', 'g'); % Rate

87. % plot fitted value
88. rate = handles.frameSmoothRefQuantile(:, 2)-handles.frameSmoothSigQuantile(:, 2);
89. time = handles.timeVector(:, 2);
90. handles.fitted = polyfit(time, rate, 1);
91. fittedValue = polyval(handles.fitted, time);
92. plot(handles.timePlot2, handles.timeVector(:, 2), fittedValue, 'LineWidth', 2, 'Color', [0, 0.65, 0]); % fitted rate

93. hold(handles.timePlot2, 'off');

94. handles.timePlot2.XLim = [0 (handles.timeVector(end, 2))];
95. handles.timePlot2.XTick = [];
96. handles.timePlot2.YAxisLocation = 'left';
97. handles.timePlot2.YLabel.String = 'Current (A)';
98. handles.timePlot2.YLim = [handles.scaleRange(1) handles.scaleRange(end)];
99. handles.timePlot2.YTick = handles.scaleRange;
100. handles.timePlot2.YAxis.TickLabelFormat = '%.2f';

101. handles.timePlot2.YAxis.Exponent = -9;
102. handles.timePlot2.FontSize = 8;

103. % plot time domain results
104. plot(handles.timePlot1, handles.timeVector(:, 2), handles.CtrElt(:, 1), 'LineWidth', 1, 'Color', 'r');
105. hold(handles.timePlot1, 'on');
106. plot(handles.timePlot1, handles.timeVector(:, 2), handles.RefElt(:, 1), 'LineWidth', 1, 'Color', 'b');
107. line(handles.timePlot1, [handles.timeVector(handles.frameNum, 2) handles.timeVector(handles.frameNum, 2)], [-3 3], 'Color', [0 0
0], 'LineStyle', ':', 'LineWidth', 2); % plot the marker line % moving line

108. hold(handles.timePlot1, 'off');
109. handles.timePlot1.XLabel.String = 'Time (s)';
110. handles.timePlot1.XLim = [0 (handles.timeVector(end, 2))];
111. handles.timePlot1.XTick = 0:round((handles.timeVector(end, 2))/20, 0):(handles.timeVector(end, 2));
112. handles.timePlot1.XGrid = 'on';
113. handles.timePlot1.YAxisLocation = 'right';
114. handles.timePlot1.YLabel.String = 'Voltage (V)';
115. handles.timePlot1.YLim = [-0.6 3];
116. handles.timePlot1.YTick = linspace(-0.6, 3, 13);
117. handles.timePlot1.YGrid = 'on';
118. handles.timePlot1.FontSize = 8;
119. handles.timePlot1.Color = 'none';

120. % Histogram Plot

```



```

121. % axes(handles.histPlot0);
122. % cla(handles.histPlot0,'reset');
123. histogram(double(handles.DataSortedSmooth(:,handles.frameNum))/10000/20e6, 'Parent', handles.histPlot0, -0.4e-9:0.01e-9:0.4e-
9,'FaceColor','k','FaceAlpha',0.1,'EdgeAlpha',0.1);
124. hold(handles.histPlot0,'on');
125. histogram(double(handles.DataSortedSmoothRef(:,handles.frameNum))/10000/20e6-handles.frameSmoothSigQuantile(handles.frameNum,5),
'Parent', handles.histPlot0, -0.4e-9:0.01e-9:0.4e-9,'FaceColor',[0.5, 0.5, 0.5],'FaceAlpha',0.8,'EdgeAlpha',0.1);
126. histogram(double(handles.DataSortedSmoothSig(:,handles.frameNum))/10000/20e6-handles.frameSmoothSigQuantile(handles.frameNum,5),
'Parent', handles.histPlot0, -0.4e-9:0.01e-9:0.4e-9,'FaceColor','m','FaceAlpha',0.8,'EdgeAlpha',0.1);
127. % handles.histPlot0.Title.String = 'Output Distribution';

128. handles.histPlot0.XLabel.String = 'Current(A)';
129. handles.histPlot0.YLabel.String = 'Number of Electrodes';
130. handles.histPlot0.XLim = [-0.4e-9 0.4e-9];
131. handles.histPlot0.XTick = -0.4e-9:0.1e-9:0.4e-9;
132. handles.histPlot0.XAxis.Exponent = -9;
133. handles.histPlot0.XGrid = 'on';
134. handles.histPlot0.YLim = [0 100];
135. handles.histPlot0.YGrid = 'on';
136. handles.histPlot0.FontSize = 8;
137. handles.histPlot0.Title.FontSize = 10;

138. rateValues = (handles.frameSmoothRefQuantile(handles.frameNum,2)-handles.frameSmoothSigQuantile(handles.frameNum,2));
139. line([rateValues rateValues],[0 5000],'Color','g','LineStyle','--','LineWidth',1);
140. text(0.4e-9*0.9,95, ...
141. ['Overall: \mu=',num2str(round(handles.frameRawQuantile(handles.frameNum,2)*1e12,1),'%.0f'),'pA',' \sigma='
num2str(round(handles.frameRawQuantile(handles.frameNum,4)*1e12,1),'%.0f') 'pA']], ...
142. 'Parent', handles.histPlot0,'FontSize',8,'HorizontalAlignment','right','FontSize', 8,'Color',[0, 0, 0]);
143. text(0.4e-9*0.9,90, ...
144. ['Reference: \mu=',num2str(round(handles.frameSmoothRefQuantile(handles.frameNum,2)*1e12,1),'%.0f'),'pA',' \sigma='
num2str(round(handles.frameSmoothRefQuantile(handles.frameNum,4)*1e12,1),'%.0f') 'pA']], ...
145. 'Parent', handles.histPlot0,'FontSize',8,'HorizontalAlignment','right','FontSize', 8,'Color',[0.5, 0.5, 0.5]);
146. text(0.4e-9*0.9,85, ...
147. ['Signal: \mu=',num2str(round(handles.frameSmoothSigQuantile(handles.frameNum,2)*1e12,1),'%.0f'),'pA',' \sigma='
num2str(round(handles.frameSmoothSigQuantile(handles.frameNum,4)*1e12,1),'%.0f') 'pA']], ...
148. 'Parent', handles.histPlot0,'FontSize',8,'HorizontalAlignment','right','FontSize', 8,'Color','m');
149. text(0.4e-9*0.9,80, ...
150. ['\muRef - \muSig (\Delta t)= ',num2str(round(rateValues*1e12,1),'%.0f'),'pA']], ...
151. 'Parent', handles.histPlot0,'FontSize',8,'HorizontalAlignment','right','FontSize', 8,'Color','g');
152. text(0.4e-9*0.9,75, ...
153. ['Fitted Rate = ',num2str(round(handles.fitted(1)*1e12,3),'%.3f'),'pA/s']], ...
154. 'Parent', handles.histPlot0,'FontSize',8,'HorizontalAlignment','right','FontSize', 8,'Color',[0, 0.65, 0]);
155. hold(handles.histPlot0,'off');

156. set(handles.scaleN1,'String',num2str(handles.scaleRange(1)*1e9,'%.2f'));
157. set(handles.scaleN2,'String',num2str(handles.scaleRange(1)*1e9,'%.2f'));
158. set(handles.scaleP1,'String',num2str(handles.scaleRange(end)*1e9,'%.2f'));
159. set(handles.scaleP2,'String',num2str(handles.scaleRange(end)*1e9,'%.2f'));

160. % set(handles.heatmapMax, 'Value', (handles.scaleRange(end)-handles.scaleRange(1))*1e9); % set slider
position to rounded off value
161. set(handles.heatmapMaxText,'String',num2str((handles.scaleRange(end)-handles.scaleRange(1))*1e9,'%.2f')); % print out the
slider value
162. end

```

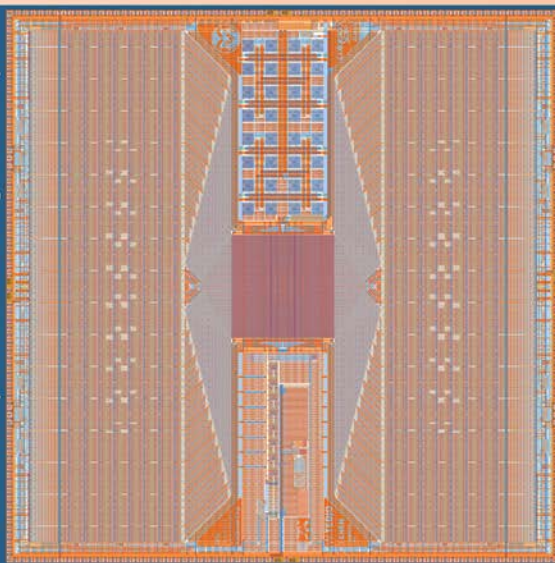


# A Lab-on-a-Chip Design of 128x128 Ultra-Microelectrode Array



## PHD CANDIDATE - ELECTRICAL ENGINEERING

## CSU Biosensor Chip Generation 3 - CMOS Integrated Circuit Layout



## Classification of Biosensor



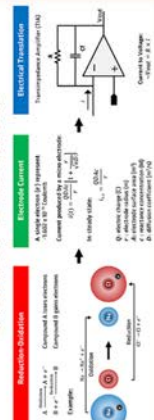
## What is ElectroChemical Biosensor?

It is a self-contained device that is capable of attaining chemical signal from a biological sample, translating it to electrical components, analyzing, and interpreting the data in a real time manner.

## How does it Work?



## Chemical to Electrical



## Reference

[illegible]

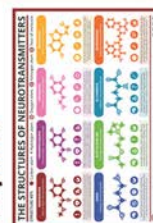
### Impact on other Disciplines

### Biochemical Study in Living Cells:

- Cellular growth and metabolism detection
- Cell migration and metastasis
- Neural networking detection

### Biopharmaceutical Research & Development:

- Neurodegenerative: Alzheimer and Parkinson
- Partial or distributed seizures: Epilepsy

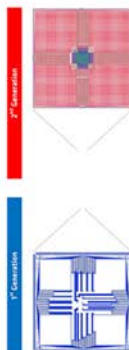


**CSU Biosensor Project**

The CSU biosensor project was first introduced back in 2009 as an interdisciplinary research project conducted by Biomedical, Chemical, and Electrical Engineering departments. The project vision is to build a device that accurately senses, analyzes, and interprets chemical activity in a living organism, or often called *in-vitro* measurement.

Based on numerous published works and literatures, an *in-vitro* detection would be a significant advancement in medical field. This would enable us recognizing early signs of disease and biological anomalies without performing invasive diagnostic work-up.

### Previous Generation Biosensor Chip

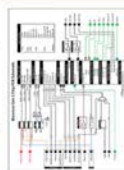


### Biosensor Chip Generation 3

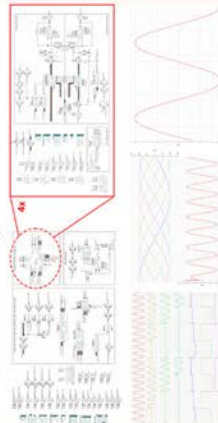
This latest generation biosensor chip has been improved in order to obtain higher signal to noise ratio and it includes other sensing features, such as: capacitance and impedance measurement array.

### Technical Specifications:

- Electrode coverage area: 3.5mm x 3.5mm  
Spatial: 16,032 Working Electrodes, 275µm pitch.  
Temporal: 250 Frame per second update rate.  
Input current range: 50pA - 100nA  
Power consumption: ±2.5V at 2.85A  
Operational temperature: 0C° - 90C°  
160-point current/voltage stimulation.



## Electrical Design and Simulations



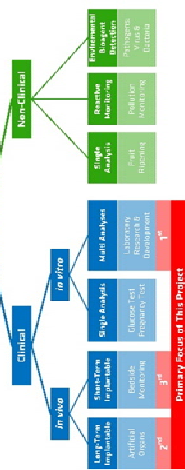




# A BIOSENSOR DESIGN

## FOR HIGH RESOLUTION ELECTROCHEMICAL IMAGING

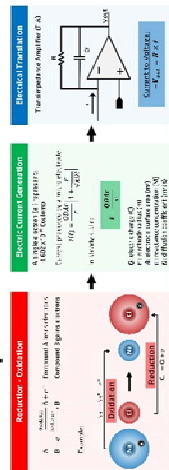
### Classification of Biosensor



### What is ElectroChemical Biosensor?

It is a self-contained device that is capable of attaining chemical signal excreted by biological sample, translating it to electrical components, analyzing, and interpreting the data in a real time manner.

### Principles: Chemical to Electrical



### How does it Work?



### Reference

1. [Reference text]

2. [Reference text]

3. [Reference text]

4. [Reference text]

5. [Reference text]

6. [Reference text]

7. [Reference text]

8. [Reference text]

9. [Reference text]

10. [Reference text]

### Acknowledgment

AVAGO TECHNOLOGIES

### WILLIAM TEDJO PHD CANDIDATE ELECTRICAL ENGINEERING

Researcher in Biosensor Design, Colorado State University, Fort Collins, CO, USA. Email: tedjo@colorado.edu

### Electronics and System

The **Biosensor Chip** is fabricated in a similar way as microprocessor used in computers. The sensing surface contains 8,192 contact points, specially shaped and coated in platinum, a conductive but inert material.

The custom made **Printed Circuit Board** contains supporting circuit and amplifiers for the Biosensor Chip.

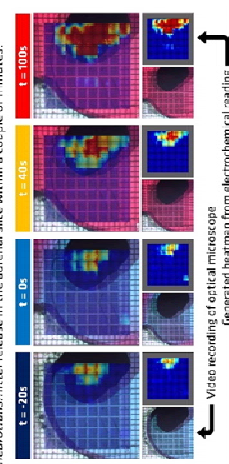
The board is progressively improved based on necessities specified by chemical and biomedical sciences researchers.

An **Optical Microscope** with environmental chamber (temperature & humidity controlled) in biomedical sciences laboratory. It is the current platform for CSU biosensor electronics and system development.

### Experiment Accomplishment

The experiment results were achieved through stimulation of a slice of mouse adrenal gland, which is known to release high concentration of **neurotransmitter** (a chemical compound responsible to increase body activity).

A stimulation by caffeine (red dyed solution) occurs at  $t = 0s$ , shows increasing neurotransmitter release in the adrenal slice within a couple of minutes.



### CSU Biosensor Project

The CSU Biosensor is a **multidisciplinary research project** conducted by Biomedical Sciences, Chemistry, and Electrical & Computer Engineering departments. The project emphasizes on device and method development for handling organism tissue sample, transforming its chemical activity to electrical value, and process the data into informative research matters.

The semiconductor based biosensor chip has been developed by CSU Electrical & Computer Engineering, the **Very Large Scale Integration (VLSI) Circuit Design** laboratory and manufactured by **Avago Technologies**, Fort Collins, CO.

### Significance to Others

The traditional optical microscope imaging method require fluorescence staining of target cells and molecules to provide visibility. It is well known that fluorescence staining method requires specific label for different targets. Furthermore, some labels have adverse effect to healthy cells for extended use. The electrochemical method offers label-free detection.

The electrochemical information would also provide cellular physiology details that were not observable in the past, such as chemical release and its gradient patterns over time.

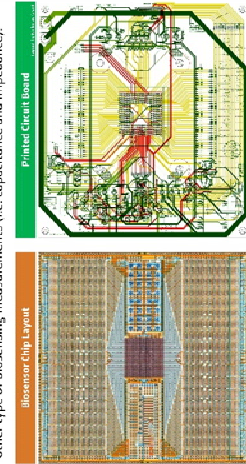
In conjunction with optical microscope, the electrochemical method would assist researchers and scientists in:

### Biopharmaceutical Research & Drug Development

- Personalized Medicine
- Biomedical Study of Living Cells
- Cellular growth and metabolism detection
- Cell communication, migration, and metastasis

### Future Generation

The next generation biosensor design has been significantly improved to obtain greater space & time resolution, higher signal quality, and so include other type of biosensing measurements (i.e. capacitance and impedance).







# DESIGN OF AN INTEGRATED MICROELECTRODE ARRAY SYSTEM FOR HIGH SPATIOTEMPORAL RESOLUTION CHEMICAL IMAGING

## Colorado State University Biosensor



**The Colorado State University (CSU) Biosensor project** is a multidisciplinary research conducted by:

- Electrical & Computer Engineering
- Biomedical Science
- Chemistry

The project emphasizes on developing **electrochemical sensing device** and experiment methods with live bio-sample.

The semiconductor based biosensor chip and its surface electrodes array has been developed by CSU Electrical & Computer Engineering (Bio-analytical Integrated Systems Laboratory) and manufactured by Imadatom (U.S.A., Fort Collins, CO, USA).

### Objectives

**In live tissue imaging**, optical microscopy requires fluorescence staining of target cells and molecules to provide visibility.

**In spite of its successes**, there are limitations in fluorescence microscopy:

- Applicable only to target analytes that possess native fluorescent property or can be modified with fluorescent marker.
- Risk of photobleaching and phototoxicity.

**As a complementary method**, the electrochemical imaging:

- Provides the ability to monitor extracellular chemical releases (cell-cell communication) and activity (chemotaxis).
- Offers label-free detection without fluorescent illumination.

### Principles: Chemical to Electrical



### References

1. J. Wang, "Electrochemical detection of glucose using a glucose oxidase-modified carbon paste electrode," *J. Electroanal. Chem.*, vol. 386, pp. 1-8, 1995.

2. J. Wang, "Electrochemical detection of glucose using a glucose oxidase-modified carbon paste electrode," *J. Electroanal. Chem.*, vol. 386, pp. 1-8, 1995.

3. J. Wang, "Electrochemical detection of glucose using a glucose oxidase-modified carbon paste electrode," *J. Electroanal. Chem.*, vol. 386, pp. 1-8, 1995.

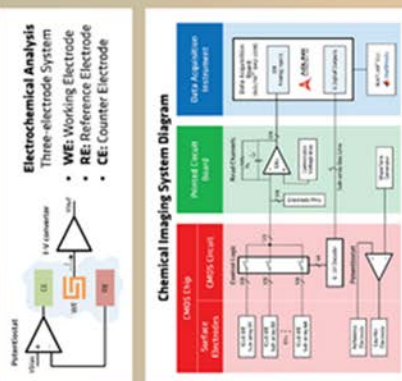
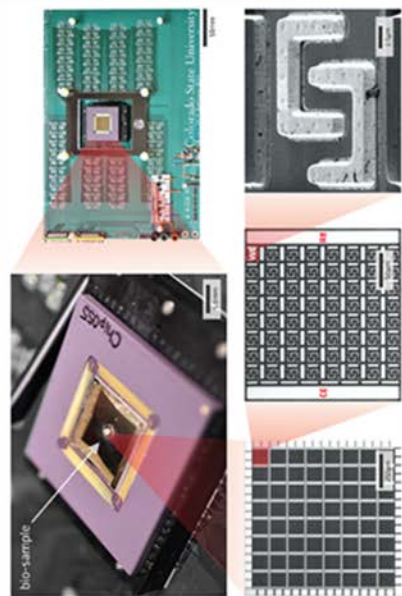
### Acknowledgments

BROADCOM, INC.  
GDE-0841259

**WILLIAM TEDJO<sup>1</sup>**, Rachel Feeny<sup>2</sup>, Chad Eitel<sup>1</sup>, Luke Schwerdtfeger<sup>3</sup>, Stacy Willett<sup>1</sup>, Charles Henry<sup>2</sup>, Stuart Tobet<sup>1</sup>, Tom Chen<sup>1</sup>

<sup>1</sup>Department of Electrical & Computer Engineering - <sup>2</sup>Department of Chemistry - <sup>3</sup>School of Biomedical Engineering  
Colorado State University, Fort Collins, Colorado, USA

## CMOS MICROELECTRODE ARRAY & ELECTRONICS



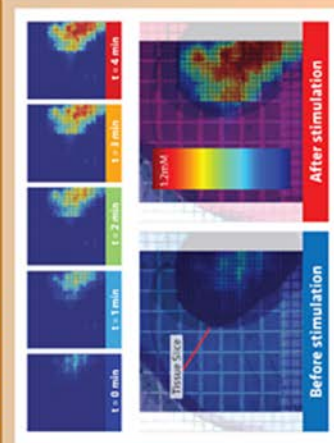
## EX VIVO EXPERIMENT & RESULTS

Ex vivo experiment results are achieved through **caffeine stimulation** of 200µm-thick murine adrenal gland. Caffeine in medulla region of adrenal gland is specifically known to release high concentration of catecholamine (a group of neurotransmitter), organic compounds responsible to increase body activity in sympathetic nervous system of mammals.



The **Environmental Chamber** and **Microfluidic Channel** provides live tissue with:

- Constant dark conditions at 37°C and 95% humidity
- Flow of culture media with 7µM level (Neurobasal® media + HEPES)





## **D.2. Conference and Journal Publications**

- [1] Tedjo, W., Catandi, G., Obeidat, Y., Carnevale, E., Chen, T., 2019. Design of a Microelectrode Sensor Array for High-Resolution Imaging of Oxygen Consumption in Small Biological Samples. Lab on a Chip (September 2019 - Under review)
- [2] Tedjo, W., Chen, T., 2019. An Integrated Biosensor System with High-Density Microelectrode Array for Real-Time Electrochemical Imaging. IEEE Trans. on Biomedical Circuits and Systems.
- [3] Obeidat, Y., Evans, A., Tedjo, W., Chicco, A., Carnevale, E., Chen, T., 2018. Monitoring Oocyte/Embryo Respiration Using Electrochemical-Based Oxygen Sensors. Sensors Actuators B Chem.
- [4] Tedjo, W., Nejad, J.E., Feeny, R., Yang, L., Henry, C.S., Tobet, S., Chen, T., 2018. Electrochemical biosensor system using a CMOS microelectrode array provides high spatially and temporally resolved images. Biosens. Bioelectron. 114, 78-88.
- [5] Tedjo, W., Feeny, R., Eitel, C., Schwerdtfeger, L., Willett, S., Henry, C., Tobet, S., Chen, T., 2016. Design of an integrated microelectrode array system for high spatiotemporal resolution chemical imaging, in: 2016 IEEE Biomedical Circuits and Systems Conference (BioCAS). IEEE, pp. 46-49.



## LIST OF ABBREVIATIONS

AE	: Auxiliary Electrode
ART	: Assisted Reproductive Technology
CMOS	: Complementary-Metal-Oxide-Semiconductor
COC	: Cumulus Oocytes Complexes
CV	: Cyclic Voltammetry
DA	: Dopamine
DAQ	: Data Acquisition
DI-H <sub>2</sub> O	: Deionized Water
DOC:	: Dissolved Oxygen Concentration
GUI	: Graphical User Interface
I-V	: Current-to-Voltage
ITO	: Indium-Tin-Oxide
LDO	: Low dropout
LoD	: Limit of Detection
MCU	: Microcontroller Unit
MEA	: Microelectrode Array
NE	: Norepinephrine
OCR	: Oxygen Consumption Rate
Opamp	: Operational Amplifier
OXPHOS	: Oxidative Phosphorylation
PCB	: Printed Circuit Board
PDF	: Probability Density Function
PDMS	: Polydimethylsiloxane
PMMA	: Polymethylmethacrylate
pO <sub>2</sub>	: Oxygen partial Pressure
PSU:	: Power Supply Unit



PVC	: Polyvinylchloride
PVT	: Process, Voltage, and Temperature
RE	: Reference Electrode
Redox	: Reduction-Oxidation
SCSI	: Small Computer System Interface
SECM	: Scanning Electrochemical Microscopy
SNR	: Signal-to-Noise Ratio
TG	: Transmission Gate
TIA	: Transimpedance Amplifier
UE	: Utility Electrodes
VHDCI	: Very High-Density Cable Interconnect
WE	: Working Electrode
ZIF	: Zero Insertion Socket