5-2021

Very Low Frequency Electrical Impedance Tomography Image Reconstruction System Using FPGA Software-Hardware Co-design

Monali Sinare

Very Low Frequency Electrical Impedance Tomography Image Reconstruction System Using FPGA Software-Hardware Co-design

by

Monali Sinare

A Thesis

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfilment of the Requirements

for the Degree of

Master of Science

in Electrical Engineering

May, 2021

Thesis Committee:
Yi Zheng, Chairperson
Mahbub Hossain
Shensheng Tang

Abstract

Electrical Impedance Tomography (EIT) is an imaging technique which is noninvasive and uses the internal conductivity distribution of the object of interest to form a tomographic image. It is performed by applying electrodes to the surface of the object. An alternating current up to frequency 10kHz is applied through a pair of electrodes, and the induced voltage is measured on other electrodes. These current and voltage values are used to reconstruct the internal conductivity distribution. The EIT imaging is increasingly getting used in clinical applications, as it is safer, portable, and low cost if compared with available imaging technologies used in clinical settings.

The goal of this project is to develop a low frequency Zynq SoC-based EIT system. A Zynq 7020 device-based development board, Zedboard, interfaced with a customized hardware circuit, is used to develop a complete EIT system. A graphical user interface is developed using C# Graphic User Interface (GUI) application to control the hardware and visualize the results. It is a twelve-electrode system, and current injection and voltage measurement is performed through Zynq SoC. There are two image reconstruction algorithms developed, Gauss Newton One Step and Total variation. The algorithms are implemented in Zynq SoC using software-hardware co-design. The algorithms are also implemented in C#. The image reconstruction performance between the two algorithms is compared. The computation performance between Zynq SoC implementation and C# implementation is also compared to understand the feasibility of FPGA implementation of EIT image reconstruction algorithms.

Acknowledgements

I would like to thank Dr. Yi Zheng, my advisor, and mentor for this thesis. I am grateful to Dr. Zheng for allowing me to work on this project. He has provided valuable insights throughout the development of this project. I would like to especially thank Dr. Zheng for his continuous support and encouragement throughout my entire study time at St. Cloud State University.

I would like to thank Dr. Mahbub Hossain and Dr. Shensheng Tang for being on the thesis committee. I would also like to thank Robert Dubois for helping me during the initial phase of this project.

Finally, I would like to thank my family, especially my son, for their patience and support.

Table of Contents

Chapter Page

Chapter Page

List of Tables

Table Page

List of Figures

Figure Page

Figure Page

Figure Page

Chapter 1: Introduction

Electrical Impedance Tomography (EIT) is the imaging technique that uses internal conductivity distribution to form an image of the medium. The EIT technology is being studied and improved for the last four decades[1]. In EIT, internal conductivity is estimated with the help of electrodes attached to the surface of the subject. A specific current is injected through one pair of the electrodes, and induced voltages are measured through the remaining electrodes. These current and voltage measurements are used to calculate the conductivity distribution over the internal area, which then can be used to form a tomographic image.

Electrical Impedance Tomography has various applications in the fields such as biomedical, geophysical, and industrial. EIT application is also being studied for semiconductor manufacturing, nanotechnology, and chemical engineering [2]. In the biotechnology field, the EIT is used for biological cell imaging.

1.1 EIT in Biomedical Applications

Electrical Impedance Tomography imaging usage in healthcare is due to the fact that biological tissues have electrical properties. In EIT imaging, the tissue conductivity property is studied. A simple electrical model of a tissue constitutes a resistance and a capacitance; thus, the tissue conductivity varies with the frequency at lower range (<100kHz). Moreover, tissue conductivity depends on fluid content and ion concentration [3], hence different tissues show different conductivity. This makes it feasible to form the tomographic image of the internal of the body part of interest.

Table 1.1 shows typical resistivity of some of the body tissues measured at different frequencies [4], [5].

Table 1. 1 Typical Tissue Resistivity [4], [5]

Tissue	**Resistivity (Ω-m)**	**Frequency**
Muscle	2-4	10kHz
Fat	20	10kHz
Lungs	10	10kHz
Blood	1.6	10kHz
Bone	>40	10kHz
Cerebrospinal Fluid	0.80	50kHz
Gray Matter	3.51	50kHz
White Matter	3.91	50kHz

Moreover, the EIT imaging is performed non-invasively and uses low alternating currents. If compared to the available imaging techniques such as CT or PET scan which uses ionizing radiation, or MRI which uses strong magnetic fields, EIT is much safer for the patient. EIT, unlike CT or MRI, can be used at the bedside of the patient. The instrumentation required to perform EIT makes it feasible to apply it to the patient on the bed and hence the visualization of tomographic image of the body part can be performed continuously. This advantage makes long term monitoring of the condition possible [4]. The most studied application of EIT is lung ventilation monitoring [3], [6]. The impedance change within lungs during the ventilation cycle is used to monitor the lung functioning. Another important application is brain imaging. The conditions such as cerebral ischemia , brain hemorrhage can be monitored using EIT, since the change in blood flow changes the conductivity[7]. Other notable applications of EIT which are under research are breast imaging, pediatric lung and cranial imaging[2]. During recent years, applications of EIT are also being explored in wearable healthcare systems[8].

Despite the potential benefits of EIT imaging in biomedical applications, it does not provide the spatial resolution as provided by more popular imaging techniques such as CT or MRI. There is an intensive research work being performed continuously in this area since last few decades. There are multiple algorithms proposed and implemented for improving the quality of reconstrued image. Moreover, there are various medical equipment manufacturers who are providing commercial EIT imaging systems [3], for instance, PulmoVista 500 by Drager or LuMon by Sentec.

1.2 FPGA for EIT Image reconstruction System

FPGAs are increasingly being used for implementing computationally complex designs. FPGAs provide high speed programmable logic arrays which can be used for parallel processing logic implementations. The Xilinx provided Zynq SoCs includes a single or dual core high performance ARM Cortex A9 based processor (PS) and high-performance programmable logic. It allows users to combine the computational power of processor and FPGA to develop efficient designs. Additionally, the SoCs provide a cost-effective solution as compared to fast processors and DSPs.

The SoC based EIT image reconstruction system[9] may provide a cost-effective solution for the parallel processing required by a EIT system.

1.3 Objectives

In this project, the feasibility of implementing EIT image reconstruction algorithms using parallel processing provided by FPGA is explored. A standalone and efficient EIT system for

biomedical applications can be developed. A phantom made using animal skin gelatin is used as a body of interest, for the test purpose. A custom designed Analog Front End, designed by one of the senior design teams in Department of Electrical Engineering, St. Cloud State University is used. A Xilinx Zynq SoC development board, Zedboard is used as a controller and the image data processor. A graphical user interface (GUI) is designed using C# windows form application. For image reconstruction, two algorithms, Gauss Newton One Step and Total Variation are developed. The complete work can be divided as follows

1) Study EIT Image Reconstruction Methods: It includes the study of Electrical Impedance Tomography methods and algorithms. It also includes the study of finite element method and its application in EIT. Two image reconstruction algorithms, Gauss Newton One Step and Total Variation are studied.
2) Study and Modify the Available Analog Front End Board: The AFE board designed by a senior design team, is designed to work with a frequency range of 100Hz to 100kHz. The AFE is modified to be able to work at lower frequency range from 1Hz to 30Hz. The EIT imaging at lower frequencies can be helpful in the cases like tissue edema.
3) Implement the Signal Generation and Acquisition Logic in FPGA: The AFE board includes a DAC to transmit the driving signal to the phantom and ADC to read the induced potential on the surface potential. The FPGA design for sinusoidal signal generation at various frequencies and data acquisition at different sampling rates is designed.
4) Develop a Communication Protocol for Serial Communication: The FPGA design requires few parameter settings transferred from GUI as well as the resulting image data is to be sent

to GUI for display. There is a serial communication link between Zynq SoC and GUI. A communication protocol is designed to transfer the data between GUI and Zynq SoC.

5) Implement the Gauss Newton One Step Algorithm: The one step image reconstruction algorithm, Gauss Newton One Step is developed in FPGA. The design is PS/PL co-design. The reconstruction algorithm is stored in PS DDR and transferred to PL for computation as per requirement.
6) Implement the Total Variation Algorithm: An iterative algorithm, Total Variation is developed in Zynq PS and PL. It is also a PS/PL co-design. Some part of the computation is developed in PS and some part of the computation is performed in PL.
7) Design a C# Graphical User Interface: A GUI is developed using C# windows form application. The GUI can communicate with FPGA over serial port for transferring data to and from FPGA. It also includes the display of image data using finite element mesh.
8) Implement the Gauss Newton One Step and Total Variation Algorithms in C# Software: The Gauss Newton One Step and Total Variation algorithms are developed in software. This way, the FPGA can only be used for data acquisition. Also, further computationally complicated reconstruction algorithms can be developed in software.
9) Test the Complete System for Different settings: Experiments using different parameters such as different frequencies, number of samples, type of algorithms etc. are performed and results are analyzed.

1.4 Thesis Organization

Chapter 1, Introduction, includes a brief introduction of EIT imaging technique and its applications in biomedical. It also includes the project objectives description.

Chapter 2, Electrical Impedance Tomography, includes mathematical background for an EIT image reconstruction problem. It includes the finite element model, Jacobian Matrix, Prior techniques, and the image reconstruction algorithms. Two algorithms, Gauss Newton One Step and Total Variation algorithm are introduced.

Chapter 3, Previous work study, includes the review of the EIT projects performed by previous St. Cloud State University students. Two of the projects are software implementation using C# and one of the projects is an FPGA based standalone EIT system development.

Chapter 4, EIT EIDORS Implementation, includes a design description of a MATLAB code designed to study the EIT image reconstruction.

Chapter 5, Xilinx Development Hardware and Software includes the details of the FPGA development board used in this project. The Xilinx Zynq SoC based Zedboard is used. The Zynq SoC architecture and the Xilinx provided development software, VIVADO IDE is also discussed.

Chapter 6, EIT System Architecture, includes a block wise description of the complete EIT system developed in this project.

Chapter 7, Analog Front End, includes the description of the analog front-end hardware design developed by a senior design team. It also includes the design upgrade implementation done for the AFE to work at lower frequencies.

Chapter 8, FPGA Design Development, includes a detailed design report of the FPGA design. The Zynq SoC constitutes of FPGA and processor, this chapter covers the FPGA design as well as processor design developed in this project.

Chapter 9, Graphical User Interface Design and Development includes the detailed GUI front end design. The software design and communication protocol are discussed.

Chapter 10, Experiments and Results, includes the tests performed for validation of the various design aspects of the project.

Chapter 11, Discussion and Conclusion includes the discussion about the complete project work and the results obtained.

At the end, the SDK code and C# GUI code flowcharts are given in the appendix. It also includes a MATLAB code which was designed for a study during the preliminary work

Chapter 2: Electrical Impedance Tomography

In this chapter Electrical Impedance Tomography (EIT) technique is introduced with a typical EIT system and the mathematical description. The finite element method, Gauss Newton One Step and Total Variation algorithms are also introduced.

2.1 Electrical Impedance Tomography Background

In Electrical Impedance Tomography imaging technique, number of electrodes are placed around the body part of interest, an alternating current of less than ten milliamperes [10] at a frequency of up to 100kHz is applied through a pair of electrodes and resulting potential is recorded on the remaining electrodes. This process is repeated by using all electrodes one pair at a time as a current source. The recorded data is then passed through a mathematical algorithm to reconstruct an image of the internal impedance distribution of the body part of interest. The alternating current level used in EIT should follow the "patient auxiliary current" limit decided by the International Electrotechnical Commission (IEC) [10]. According to this standard, for frequency range from 0.1 Hz to 1 kHz, the current should be limited to 100 µA, from 1kHz to 100kHz it should be $100f$ µA, where f is the frequency in kHz and from 100kHz and above it should be 10mA [4], [10].

The challenge in EIT image reconstruction problem is that the electric current passed through one electrode pair travels three dimensionally along the path of least resistivity [11]. This may affect the resulting voltage measurements at the other electrodes. In most available literatures, even though the data is collected from a three-dimensional body, an EIT image was reconstructed with an assumption that the body was two dimensional [11]. Thus, the reconstructed image does

not really represent the true distribution of the conductivities of the body. In addition, the algorithm solved an inverse problem, thus solution is not unique. Calculating the conductivity distribution of the interior of the body of interest with known applied current and measured voltages at the boundary, makes the image reconstruction as inverse problem[11]. However, if the applied current and impedance distribution are known, the voltage can be calculated, it is called as the forward problem. In EIT image reconstruction, at first, an internal impedance distribution is assumed, and a forward model is generated, which can estimate the boundary voltages with given applied current and impedance distribution. The calculated voltages are compared to the measured voltages. Depending upon the difference between measured voltages and calculated voltage the impedance distribution is adjusted, and the process is repeated until the difference between actual and calculated voltages reduces to the acceptable level.

The forward problem is governed by partial differential equation. Hence, to solve the forward problem, numerical techniques such as finite element method (FEM), finite difference method (FDM) or boundary element method (BEM) can be used [12] [13]. The FEM divides a large system into smaller parts called as finite elements which can be polygons, whereas, the BEM requires for the object under investigation to have homogeneous conductivity distribution and the FDM uses the rectangular geometry. Hence, in clinical EIT, finite element method is preferred as it can model irregular and nonhomogeneous geometries more precisely.

With the help of forward model, a regularized non-linear solver can be used to obtain a unique and stable inverse solution. In last few decades various image reconstruction techniques and algorithms for EIT image reconstruction have been proposed and implemented. Those algorithms include modified newton Raphson [13], back projection, Gauss-Newton one step

algorithm, Graz consensus Reconstruction algorithm for EIT (GREIT), conjugate gradient method, Total Variation etc. Artificial Neural Networks and Deep Learning are also getting studied for EIT image reconstruction in order to improve image quality [14]. There is an open source software EIDORS (Electrical Impedance and Diffused Optical Reconstruction Software) developed using MATLAB which can be used for forward and inverse modelling for EIT image reconstruction [15].

There are two imaging techniques in EIT, difference imaging and static imaging [4], [13]. In difference imaging, there are two subcategories, time difference and frequency difference. In time difference technique, the change in tissue conductivity measured at two different instances is calculated. This change is back projected to form an image showing changes in the conductivity distribution happened between two instances. In frequency difference imaging, the change in tissue conductivity at two different frequencies is calculated. Frequency difference technique is based on the fact that different tissues have different frequency response [14]. In static imaging, the absolute values of conductivity distribution are calculated and used to form an image. As mentioned above, there have been numerous algorithms studied and implemented to reconstruct the impedance distribution image. The imaging algorithms can be classified as iterative and non-iterative. The impedance image reconstruction is a non-linear problem since the internal impedance distribution is unknown, hence an iterative method could give the best result [4], but it would be computationally more expensive. In this project, two algorithms using time difference imaging are implemented, a one step that is non-iterative Gauss Newton One Step and iterative Total variation algorithm.

2.1.1 Advantages/Disadvantages of EIT

The Electrical Impedance Tomography is performed non-invasively. It uses up to ten milliamperes of alternating current [10] to inject through the body of interest to measure the induced potential and build the relative impedance image. This technique is less harmful as compared to the available imaging techniques such as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI). CT uses ionizing radiation whereas MRI uses strong magnetic fields, both of which can be harmful for longer exposures to the patient. Because the EIT imaging technique uses few milliamperes current, it can be used for long term patient monitoring, for instance in cases where continuous lung ventilation needs to be monitored. The EIT systems mostly include hardware to generate low level constant current source, a data acquisition hardware and a computer for processing acquired data from multiple channels. This makes EIT systems portable. Another very important advantage is that EIT technique has faster imaging capabilities. Some of the commercially available EIT systems provide 40 images/seconds frame rate [3]. Real time imaging can be made possible due to such temporal resolution.

The major disadvantage of EIT systems is the technique has lower spatial resolution. In the EIT image reconstruction problem there can be a very small change on the surface potential for a large change in conductivity inside the body of interest. Also the final solution is very sensitive to noise introduced due to measurement artifacts [16].

2.1.2 Typical EIT System

This section includes the description of typical EIT system. Figure 2.1 shows the block diagram

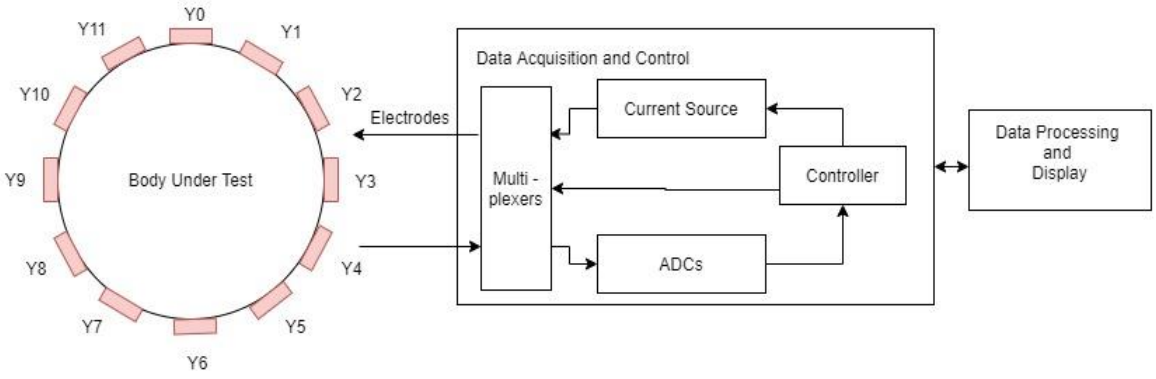


Figure 2. 1 Typical EIT System Block Diagram

The EIT system can be divided into two parts such as data acquisition and data processing. Following is the description of both parts in brief.

1. Data Acquisition and Control

As described above, in EIT, number of electrodes are placed on the surface of the subject of interest. To retrieve the internal conductivity distribution, an alternating current is applied to a pair of electrodes and resulting voltages on remaining electrodes are recorded. The current application is repeated in a pattern for other electrodes and respective voltages are recorded. For achieving this functionality, a current source is required. The current source is to be connected to different electrodes, in a pattern, hence a multiplexer can be used to connect one current source to a pair of electrodes. Similarly, the voltage readout from the remaining electrodes also needs to be taken from different electrodes in a pattern, hence a multiplexer can be used to choose which

electrode to be connected to ADC. The multiplexers can be controlled through a microcontroller or a microprocessor. The controller needs to be used to control the system functionality as well as to transfer the acquired measurement data to the software for processing. The link between controller and software system can be wired or wireless.

2. Data Processing and Display

In EIT, the software system or a Graphical User Interface is required to set the parameters such as number of electrodes and current application pattern for the acquisition. The main functionality of the software should be to store the voltage measurements, run the image reconstruction algorithms and display the results. The typical EIT system software operational flow would include setting up the system parameters, creating a forward model, acquiring, and storing the measurement data, solving for the inverse solution, and displaying the reconstructed image.

2.1.3 Stimulation and Measurement Patterns

When a current is applied using two electrodes on a surface of a body with homogeneous impedance distribution, it forms equipotential distribution inside the body. If there is a change in the impedance distribution the potential distribution varies accordingly, hence changing the potential on the boundary. Due to this the injected current and measured boundary potential can be used to approximate the impedance distribution of the internal of the body.

There are two current injection patterns, multiple electrode current drive and pair electrode current drive [4], [14]. In multiple electrode current drive pattern, the current is applied

to more than two electrodes and the potential is measured on all the electrodes. This current pattern is repeated till the current is injected through all the electrodes. This way the current density inside the body increases which also increases the sensitivity towards the impedance distribution measurements[4]. However, this method is not cost effective as well as it complicates the hardware design.

The pair electrode current drive is typically implemented in two ways, adjacent current drive, and opposite current drive. In adjacent current drive, the current is injected between two adjacent electrodes, as shown in figure 2.2 (a) and voltage is measured between remaining adjacent electrode pairs. This sequence is repeated till the current is injected through all the electrodes. Another pair electrode current drive type is opposite or pseudo-polar drive pattern [17]. In this drive pattern the current electrode pair is selected as opposite, as shown in figure 2.2 (b). The voltage is measured at the adjacent electrodes. In this project the opposite current drive pattern is used.

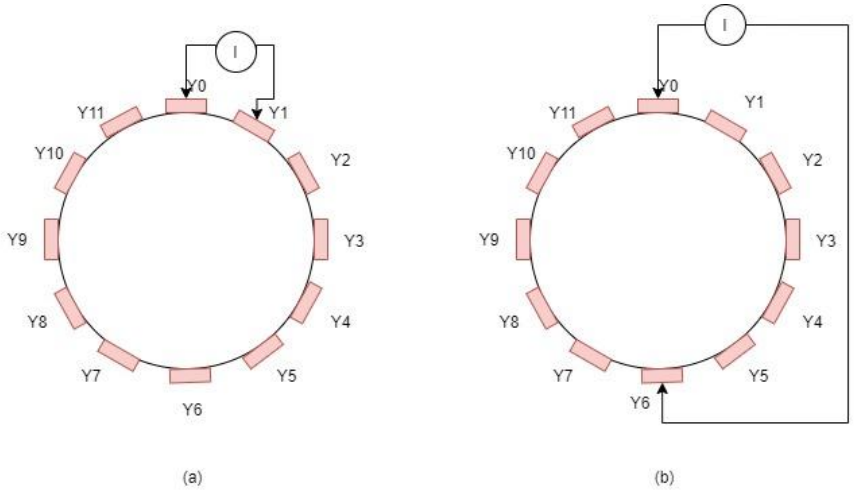


Figure 2. 2 Stimulation Pattern for Electrode Pair

2.2 Mathematical Background

In EIT, the problem is to reconstruct the internal conductivity distribution with the help of known applied current and measured voltages on the surface. In this section, the mathematical model required to solve the said problem is discussed.

In case of clinical EIT, the applied current is known, and the induced voltages are measured. From available current and voltage values, internal conductivity distribution is to be calculated, this makes it as an inverse problem. Moreover, according to Hadamard [14], a mathematical model of a physical problem is well posed if it satisfies the following conditions

1. A solution exists for all admissible data
2. The solution is unique for all the admissible data
3. The solution continuously depends on the data

In case of the EIT problem, there are more unknowns than the known parameters. Because of this a unique solution may not exist. Also, for any given measurement precision, the change in conductivity precision may be arbitrarily large and may go undetected. This makes the problem ill-posed[11]. To find a practical solution for this problem it is regularized using regularization methods. Also, for improving the situation of data dependency, some additional information about conductivity distribution is introduced. This a priori information constraints the solution and brings stability to the solution.

Mostly, the conductivity calculation is approached in two steps, forward problem, and inverse problem. If the current applied to the body is known along with conductivity distribution and voltage is to be calculated, it is called as forward problem. If the current applied and voltage measured is known and conductivity distribution is to be calculated, it is called as inverse problem.

In the following section, the mathematical expression behind forward and inverse problem are discussed.

2.2.1 Forward Problem

For solving EIT forward problem, a forward model is built in which the initial conductivity of the body is assumed. With the help of applied current and assumed conductivity, the boundary voltage is calculated. This computation is based on electromagnetism [7], [18].

In EIT computation, the low frequency signals are used so that the static field assumptions can be used[7], [18], [19]. This is because at low frequencies the wavelength is much greater than the dimension of a body of interest [7]. It is also assumed that the internal conductivity distribution is not affected by the electromagnetic field density, hence the conductivity anywhere can be treated as scalar [19] .

The Maxwell's equations [19] for electric field E and magnetic field H in point format are given as follows

$$\nabla X\ E = -\frac{\partial B}{\partial t} \tag{2.1}$$

$$\nabla X\ H = \frac{\partial D}{\partial t} + J \tag{2.2}$$

where,

B = Magnetic flux

D = Electric Displacement

J = Current density

Using Ohm's law, the internal current density J in terms of electric field [20]is given by

$$J = \sigma E \tag{2.3}$$

where, σ is the internal conductivity distribution.

In case of EIT, the dimension of the object under investigation is much smaller than the wavelength of the signal applied, for example, if the applied signal frequency is 10 kHz the wavelength would be $3x10^4$ meters. This makes the dimension of the object under investigation <1% of the signal wavelength. Hence, the internal electric field can be considered as static and can be defined in terms of the scalar potential (V) as follows

$$\mathrm{E} = -\nabla \mathrm{V} \tag{2.4}$$

where, V represents the scalar potential distribution within the field. Combining equations 2.4 and 2.3, we get

$$\mathrm{J} = -\sigma \nabla \mathrm{V} \tag{2.5}$$

If it is considered that there are no internal current sources [20], we get

$$\nabla . \mathrm{J} = 0 \tag{2.6}$$

Combining equations 2.5 and 2.6, we get

$$\nabla . \sigma \nabla \mathrm{V} = 0 \tag{2.7}$$

Equation 2.7 is the partial differential equation (PDE) which dominates the EIT problem [21].

For a domain $\Omega(x, y, z)$, the equation 2.7 can also be written as

$$\frac{\partial}{\partial x}\sigma \frac{\partial V}{\partial x} + \frac{\partial}{\partial y}\sigma \frac{\partial V}{\partial y} + \frac{\partial}{\partial z}\sigma \frac{\partial V}{\partial z} = 0 \tag{2.8}$$

For equation 2.7 to have a unique solution, the boundary condition needs to be defined [19].

If $\partial\Omega$ represents the boundary, applying the Gauss's theorem for conservation of current [19], [20], we can write

$$\int_{\partial\Omega} j = 0 \tag{2.9}$$

where, j is the current density at the boundary. The current density at the boundary is given as [20]

$$\mathrm{j} = -\mathrm{J} \,.\, \mathrm{n} \tag{2.10}$$

where, n is an outward unit normal to the boundary $\partial\Omega$. Using equation 2.5, equation 2.10 can also be written as

$$\mathrm{j} = \ \sigma\nabla \mathrm{V}.\,\mathrm{n} \tag{2.11}$$

Equation 2.7 and 2.11 represent the EIT forward problem [19], with the help of known conductivity and known applied current, the potential distribution V can be calculated.

2.2.2 Inverse Problem

The goal behind solving the inverse problem is to get the unknown internal conductivity distribution with the help of measured voltages and simulated voltages. By solving the forward problem, the simulated voltages at the boundary can be calculated. If V_m is the measured voltages at the boundary, then the solution can be approached by minimizing the difference between measured voltages and simulated voltages

$$V_m - F(\sigma) \ \approx 0 \tag{2.12}$$

where, F(.) is the forward operator which gives the simulated voltages which depends upon the conductivity distribution σ.

If $F(\sigma)$ can be approximated by using Taylor series and considering only first order approximation, we get

$$F(\sigma) \approx F(\sigma_0) + \frac{\boldsymbol{\partial} F(\sigma_0)}{\boldsymbol{\partial}\sigma}(\sigma - \sigma_0) \tag{2.13}$$

where σ_0 is the assumed initial estimation. Equation 2.13 can be written as

$$F(\sigma) \approx F(\sigma_0) + \boldsymbol{J}(\sigma - \sigma_0)$$

As $\boldsymbol{J} = \frac{\boldsymbol{\partial} F(\sigma_0)}{\boldsymbol{\partial}\sigma}$ is the Jacobian matrix of F calculated at σ_0. Hence, equation 2.12 can be re-written as

$$V_m - F(\sigma_0) - \boldsymbol{J}(\sigma - \sigma_0) \approx 0 \tag{2.14}$$

Putting $\Delta \mathrm{V} = V_m - F(\sigma_0)$, which shows the variation in the measured potentials for change in conductivity [17], and $\Delta\sigma = \sigma - \sigma_0$, which shows the change in conductivity, then equation 2.14 becomes

$$\Delta \mathrm{V} \approx \mathbf{J}\Delta\sigma \tag{2.15}$$

Which gives

$$\Delta\sigma \approx \mathbf{J^{-1}}\Delta \mathrm{V} \tag{2.16}$$

Since Jacobian matrix is mostly not invertible, equation 2.16 is solved by minimizing the least squares errors as follows [19]

$$\Delta\sigma_{LS} = argmin(\|\mathbf{J}\Delta\sigma - \Delta \mathrm{V}\|^2) \tag{2.17}$$

Which gives, [19]

$$\Delta\sigma_{LS} = (\boldsymbol{J^T J})^{-1}\boldsymbol{J^T}\Delta \mathrm{V} \tag{2.18}$$

Here, since there are a greater number of unknown conductivity elements than the known boundary measurements, the problem is ill-conditioned[19]. For getting a unique solution to this, regularization is needed. Regularization is a technique used to find more stable but approximate solution by applying a prior information. There are several regularization methods, but in EIT

application Tikhonov regularization is commonly used[11]. It adds a regularization parameter with prior information.

Hence, equation 2.18 becomes [17]

$$\Delta\sigma = (\boldsymbol{J}^T\boldsymbol{J} + \lambda^2 R^T R)^{-1}\boldsymbol{J}^T\Delta \mathrm{V} \tag{2.19}$$

Where, λ is called as the hyperparameter or regularization parameter and R is the regularization matrix, which controls the smoothness of the solution [17]. There are various methods to calculate the regularization matrix R, for example, NOSER, Laplace, Tikhonov etc.

2.3 Finite Element Method

Finite element method (FEM) is a numerical method used to solve problems that are described by partial differential equations [22], [23]. In EIT, forward problem is solved with the help of finite element method [11]. An area under investigation is discretized as a group of non-overlapping finite elements such as triangles or quadrilaterals. In other words, a continuous physical problem is converted into discretized finite element problem with unknown nodal values. The process of analyzing any problem with finite element method [19], [23] involves

- Dividing the solution region into finite elements, also called as finite element mesh. The output of mesh generation consists of arrays of nodal coordinates.
- Selecting the interpolation functions to interpolate the field variables over the element.
- Deriving the equations for elements which relates the nodal values of the unknown function to other parameters.
- Finding the global equation system for the complete solution region by assembling the element equations.

- Solving the global equation system

Consider, a domain is divided into finite number of triangles and the triangle shown in figure 2.3 represents an element in the finite element model with vertices denoted by node1 (x1, y1), node2 (x2, y2) and node3 (x3, y3). The problem statement here is to find the approximation for the potential V_e within an element e. The potential for each element then can be interrelated to find the approximate solution for the complete region.

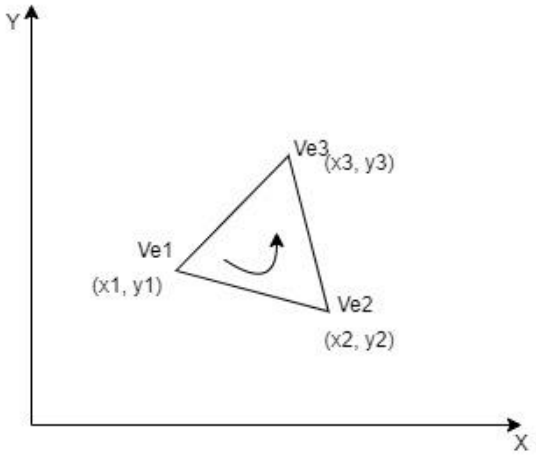


Figure 2. 3 Typical Triangular Element in 2D problem

If the solution region is divided into N number of triangular elements, the approximate potential for the whole region can be represented as[24]

$$V(x,y) = \sum_{e=1}^{N} V_e \tag{2.20}$$

The common form of approximation for V within a triangular element is polynomial approximation as follows

$$V_e(x,y) = a + bx + cy \tag{2.21}$$

From equation 2.21, the potential at $V_{e1,}$ V_{e2}, V_{e3} at the triangular element nodes 1,2 and 3 can be written as

$$\begin{bmatrix} V_{e1} \\ V_{e2} \\ V_{e3} \end{bmatrix} = \begin{bmatrix} 1 & x1 & y1 \\ 1 & x2 & y2 \\ 1 & x3 & y3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{2.22}$$

The coefficients a, b and c can be determined using equation 2.22 as follows

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 & x1 & y1 \\ 1 & x2 & y2 \\ 1 & x3 & y3 \end{bmatrix}^{-1} \begin{bmatrix} V_{e1} \\ V_{e2} \\ V_{e3} \end{bmatrix} \tag{2.23}$$

Solving for inverse of the coordinate matrix and substituting resulting equation into equation 2.21, we get

$$V_e(x,y) = [1 \quad x \quad y] \frac{1}{2A} \begin{bmatrix} (x1y3 - x3y2) & (x3y1 - x1y3) & (x1y2 - x2y1) \\ (y2 - y3) & (y3 - y1) & (y1 - y2) \\ (x3 - x2) & (x1 - x3) & (x2 - x1) \end{bmatrix} \begin{bmatrix} V_{e1} \\ V_{e2} \\ V_{e3} \end{bmatrix} \tag{2.24}$$

where, A is the area of the element and given as

$$2A = \begin{vmatrix} 1 & x1 & y1 \\ 1 & x2 & y2 \\ 1 & x3 & y3 \end{vmatrix} \tag{2.25}$$

For the value of area A to be positive, the nodes numbering should be done counter clock wise as shown in figure 3 with arrow [23].

Equation 2.24 can also be written as

$$V_e = \sum_{i=1}^{3} \alpha_i(x,y) V_{ei} \tag{2.26}$$

where,

$$\alpha_1 = \frac{1}{2A}\ [(x2y3 - x3y2) + (y2 - y3)\,x + (x3 - x2)\,y] \tag{2.27a}$$

$$\alpha_2 = \frac{1}{2A}\ [(x3y1 - x1y3) + (y3 - y1)\,x + (x1 - x3)\,y] \tag{2.27b}$$

$$\alpha_3 = \frac{1}{2A}\ [(x1y2 - x2y1) + (y1 - y2)\,x + (x2 - x1)\,y] \tag{2.27c}$$

Here, α_i are the linear interpolation functions and called as element shape functions [24]. Equation 2.26 defines the potential within the triangular element as a function of the values of the potential at the element's three nodes.

From equation 2.7, EIT partial differential equation, if V_E is the actual solution and V_{FEM} is the approximate solution found by FEM, then [24]

$$\nabla.\sigma\nabla V_{FEM} - \nabla.\sigma\nabla V_E = R \tag{2.28}$$

where, R is the residual. Hence,

$$\nabla.\sigma\nabla V_{FEM} = R \tag{2.29}$$

As, $\nabla.\sigma\nabla V_E = 0$ and $\nabla.\sigma\nabla V_{FEM} \approx 0$.

In the weighted residuals method, the weighted sum of the residuals over all the elements can be zero if proper weight can be found. Hence, the integral of the weighted residual over the medium Ω will be zero [24].

$$\int_{\Omega} WRd\Omega = 0 \tag{2.30}$$

where, W is defined [24] as

$$W = \sum_{i=1}^{n} w_i\,\alpha_i \tag{2.31}$$

where, α_i represents the shape functions for the element, and w_i are the weight coefficients, and n is the dimension of the element. Putting equation 2.29 in 2.30, we get

$$\int_{\Omega} W(\nabla.\sigma\nabla V_{FEM})d\Omega = 0 \tag{2.32}$$

Rearranging and solving equation 2.32 by following the procedure given in [24], equation 2.32 can be written for k[th] element, assuming conductivity σ_k is constant along the element, we get

$$\int_{E_k} \sigma_k \nabla V_e.\nabla \mathrm{W} d\Omega = \sigma_k \sum_{i=1}^{3} V_{ei} \sum_{j=1}^{3} w_j \; C_{ij}^k \tag{2.33}$$

where, C_{ij}^k is called as the stiffness matrix for element k.

$$C_{ij}^k = \int_{E_k} \nabla\alpha_i.\nabla\alpha_j d\Omega \tag{2.34}$$

For entire mesh, equation 2.33 would be [24]

$$\int_{\Omega} \sigma_k \nabla V_e.\nabla \mathrm{W} d\Omega = \sum_{E=1}^{N} \sigma_k \sum_{i=1}^{3} V_{ei} \sum_{j=1}^{3} w_j \; C_{ij}^k \tag{2.35}$$

where, N is the total number of elements.

To calculate stiffness matrix, using equations 2.27a, 2.27b and 2.27c, we have

$$\alpha_1 = \frac{1}{2A} [a_1 + b_1 x + c_1 y] \tag{2.36a}$$

$$\alpha_2 = \frac{1}{2A} [a_2 + b_2 x + c_2 y] \tag{2.36b}$$

$$\alpha_3 = \frac{1}{2A} [a_3 + b_3 x + c_3 y] \tag{2.36c}$$

where,

a_1 = x2y3 – x3y2; b_1 = y2-y3; and c_1 = x1 - x3

a_2 = x3y1 – x1y3; b_2 = y3 – y1; and c_2 = x1 – x3

a_3 = x1y2 – x2y1; b_3 = y1 – y2; and c_3 = x2 – x1

Using equation 2.36a, 2.36b and 2.36c, the elemental stiffness matrix K^e is given as follows [24]

$$K^e = \frac{\sigma_e}{4A_e}\begin{bmatrix} b_1^2 + c_1^2 & b_1b_2 + c_1c_2 & b_1b_3 + c_1c_3 \\ b_2b_1 + c_2c_1 & b_2^2 + c_2^2 & b_2b_3 + c_2c_3 \\ b_3b_1 + c_3c_1 & b_3b_2 + c_3c_2 & b_3^2 + c_3^2 \end{bmatrix} \tag{2.37}$$

Assembling all the elemental stiffness matrices the Global Stiffness matrix K is formed and following system of equations is derived [24], [25]

$$[\mathrm{K}]_{nxn}[\mathrm{V}]_{nx1} = [\mathrm{I}]_{nx1} \tag{2.38}$$

where, $[\mathrm{V}]_{nx1}$ is the matrix of potentials at all the nodes and $[\mathrm{I}]_{nx1}$ is the matrix of currents at all the nodes. Equation 2.38 can be written as

$$[\mathrm{V}]_{nx1} = [\mathrm{K}]^{-1}_{nxn}[\mathrm{I}]_{nx1} \tag{2.39}$$

2.4 Jacobian Matrix

Jacobian matrix, in case of EIT, provides the relationship between voltage distribution on the region of interest's surface and the regions internal conductivity distribution[26]. Jacobian matrix is calculated using FEM, current injection pattern and electrode position[27]. A standard method to calculate Jacobian matrix is proposed in [25]. In this method, the Jacobian matrix is derived by differentiating equation 2.39 with respect to conductivity σ [25] , we get

$$\mathbf{J} = \frac{\partial \mathrm{V}}{\partial \sigma_j} = -\mathrm{K}^{-1}.\frac{\partial \mathrm{K}}{\partial \sigma_j}.\mathrm{K}^{-1}\mathrm{I} \tag{2.40}$$

Which can be written as

$$\mathbf{J} = \frac{\partial \mathrm{V}}{\partial \sigma_j} = -\mathrm{K}^{-1}.\frac{\partial \mathrm{K}}{\partial \sigma_j}.\mathrm{V} \tag{2.41}$$

If there are M number of boundary measurements and N number of FEM elements, then $\mathbf{J} \in R^{MxN}$. In this method only stiffness matrix derivative is to be calculated to get **J.**

2.5 EIT Algorithms

There are various image reconstruction algorithms used for EIT problem solving. There are two imaging techniques in EIT, difference imaging and static imaging. In this project, two algorithms based on difference imaging are implemented, Gauss Newton One step and Total Variation. In this section, both the algorithms are discussed.

2.5.1 Gauss Newton One Step Algorithm

The Gauss Newton One Step solver is based on Gauss-Newton algorithm described in [25] . It is an improvement based on Gauss-Newton algorithm along with statistical consideration and prior information. It is used as a difference method. It uses regularized model to solve EIT inverse problem. The Gauss Newton approach uses the noise variance of the measurements and the covariance of the conductivity distribution for linearized image reconstruction[6]. Using this method, the solution to the inverse problem in EIT is estimated by minimizing [28]

$$\|\boldsymbol{J}\Delta\sigma - \Delta V\|^2_{\Sigma_n^{-1}} + \|\Delta\sigma - \Delta\sigma_*\|^2_{\Sigma_m^{-1}} \quad (2.42)$$

Where, $\Delta\sigma_*$ is the initial estimation which is 0 in case of difference EIT [17].

$\sum_n$ is the covariance matrix of the measurement noise n.

$\sum_m$ is the expected image covariance. The covariance matrices are normally modeled heuristically from a priori considerations as follows

$$W = \xi_n^2 \sum_n^{-1} \tag{2.43}$$

And

$$R = \xi_m^2 \sum_m^{-1} \tag{2.44}$$

where, ξ_n is the average measurement noise amplitude and ξ_m is the a priori amplitude of conductivity change.

Here, W models the measurement accuracy and for difference EIT it is Identity matrix (I). R represents the regularization matrix. Various prior calculation methods are available to calculate the regularization matrix R, for example, NOSER, Laplace, Tikhonov etc.

Solving equation 2.42 gives the linearized, one step inverse solution as follows

$$\Delta\sigma = \left(\boldsymbol{J}^T \frac{1}{\xi_n^2} W\boldsymbol{J} + \frac{1}{\xi_m^2} R\right)^{-1} \boldsymbol{J}^T \frac{1}{\xi_n^2} W\Delta V \tag{2.45}$$

Let's put $\lambda = \frac{\xi_n}{\xi_m}$, then equation 2.45 can be written as

$$\Delta\sigma = (\boldsymbol{J}^T W\boldsymbol{J} + \lambda^2 R)^{-1} \boldsymbol{J}^T W\Delta V \tag{2.46}$$

The hyperparameter λ controls the tradeoff between the resolution and noise attenuation in the resulting image. Usually the hyperparameter is selected heuristically [29]. Few of the reconstructed results generated over a range of hyperparameter values are observed and the one with best result is selected. In this project, the hyperparameter value is used as 0.003, this value is used by the previous students who worked on the EIT image reconstruction project in St. Cloud State University.

2.5.2 Total Variation Algorithm

The Total Variation (TV) technique is a regularization method which allows the regularization of discontinuous parameters. The commonly used regularization techniques fall short of describing sharp variations in the reconstructed image whereas TV preserves the edges in reconstructions[30]. In many regularization matrices L2 norm is used which makes the solution biased towards smoother functions. TV uses L1 norm which makes it discontinuous and not differentiable at every point. The total variation of a conductivity image is defined as follows [31]

$$\mathrm{TV}(\sigma) = \int_{\Omega} |\nabla\sigma| d\Omega \tag{2.47}$$

Where, Ω is the region of interest. Let us consider the discretized version of the region (FEM), the conductivity is assumed to be piecewise constant over each element [31]. This also means the conductivity $\nabla\sigma$ is non-zero only on the edges between the elements.

Hence, for the i^{th} edge, shared by the FEM elements m(i) and n(i), the difference in conductivity is

$$\left|\sigma_{m(i)} - \sigma_{n(i)}\right| \tag{2.48}$$

The Total Variation of the entire image can be written as the sum of the TV of all the edges

$$\mathrm{TV}(\sigma) = \sum_{i} l_i \left|\sigma_{m(i)} - \sigma_{n(i)}\right| \tag{2.49}$$

Where l_i is the length of the i^{th} edge in the mesh.

Equation 2.49 can also be expressed as

$$\mathrm{TV}(\sigma) = \sum_i |L_i \sigma| \tag{2.50}$$

Where *L* is the sparse matrix with one row for each edge and each row has two non-zero elements in the columns m(i) and n(i). Hence, the TV is the sum of the absolute values.

TV algorithm for EIT

There are various algorithms developed for solving TV regularized problems. A few algorithms are described in [30]. For EIT application, Primal and Dual Interior Point Method (PDIPM) and Iteratively Reweighted Least Squares (IRLS) methods are mostly used. PDIPM introduces a smoothness parameter and a dual variable using the Cauchy-Schwartz inequality. [31] provides detailed explanation of PDIPM algorithm implementation for clinical EIT. The IRLS method solves a weighted least squares problem iteratively [32]. Both methods are available in EIODRS.

In this project, the Total Variation algorithm shown by Rabi Gartaula [33] is used. It uses one step Gauss Newton approximation step to find initial solution, σ_0, as follows

$$\Delta\sigma = (\boldsymbol{J}^T W \boldsymbol{J} + R^T R)^{-1} \boldsymbol{J}^T W \Delta V \tag{2.51}$$

For further iterations, a TV regularization parameter is calculated as follows

$$\mathrm{W_B} = \frac{0.5}{\sqrt{(R * \Delta\sigma)^2 + \beta}} \tag{2.52}$$

Where, β is a smoothness parameter and kept constant throughout iterations. This parameter is incorporated in equation 2.51 for further iterations, as shown in equation 2.53.

$$\Delta\sigma = \frac{J^T * W * \Delta V}{(J^T * W * J + R^T * W_B * R)} \quad (2.53)$$

Equation 2.52 and 2.53 are repeated for the set number of iterations.

2.6 Prior Techniques

As mentioned in section 2.2, a prior information is introduced to stabilize the solution of the ill-posed problem[28]. The technique of applying prior information is called as regularization. There are different methods of prior calculation. In this section, three methods such as Tikhonov, NOSER and Laplace are discussed.

2.6.1 Tikhonov Prior

It is the most used method of regularization in EIT inverse solution. The Tikhonov regularization uses differential operators which provides smooth solutions. If the elements of image are considered as independent with identical expected magnitudes, the regularization matrix [19] becomes,

$$R = I \quad (2.54)$$

Where, I is the identity matrix. This is called as zeroth-order Tikhonov regularization. This regularization provides the smooth solutions but does not preserve sudden changes in the image [34]. In this project, Tikhonov prior is used to reduce the complexity of the computation.

2.6.2 NOSER Prior

NOSER prior, stands for Newton's One Step Error Reconstruction, is also known as weighted diagonal prior[35]. In NOSER prior the regularization matrix [28] is calculated as

$$R = [diag(\boldsymbol{J}^T\boldsymbol{J})]^P \tag{2.55}$$

Where **J** is the Jacobian matrix. The exponent p needs to be selected carefully to keep balance between solution stability and image contrast [17] and p Є [0,1].

2.6.3 Laplace Prior

Laplace prior is a second order high pass filter[36]. For a finite element mesh, it is defined as -1 for each adjacent element and 3 for the element itself[28]. Hence, edges are preserved in Laplace prior as compared to Tikhonov and NOSER [36]. In other words, it preserves high frequency components such as edges in the image.

Chapter 3: Previous Work Study

There are few EIT image reconstruction project works performed by past St. Cloud State University students. Three of the projects were studied as a preliminary study for this project. Two of the projects were C# implementation of the EIT image reconstruction algorithms. Third project was FPGA based implementation which included complete EIT data acquisition as well as data processing system[9]. The hardware designed in FPGA based project is used in this project, with some modifications. In this chapter, a brief description of each project functionality and design is given.

3.1 Study on human brain Activity and electrical impedance tomography [19]

3.1.1 Design Description

It is a C# implementation of Gauss-Newton one step algorithm for EIT image reconstruction. Tikhonov, Laplace and NOSER priors are implemented. A graphical user interface (GUI) provides user with the options to set the available system parameters. The parameters required for FEM settings, stimulation, measurement data, prior selection and image reconstruction algorithm can be set through GUI. The measurement data that is the homogeneous and inhomogeneous data taken from a phantom and stored in text files is used to test the code. The C# implementation is designed by following the open source MATLAB implementation EIDORS.

For building finite element model (FEM), number of electrodes and number of rings is taken as input. The FEM is designed as a 2D circular model and discretized according to the number of rings entered by the user. There is a class defined to handle FEM model generation and

it provides the model's nodes coordinate data, element assembly data and electrode positioning matrix.

The experiment setup is entered in the form of a stimulation pattern. It is created with the help of user entered data which is the current injection pattern, opposite or adjacent, and voltage measurement pattern and amplitude of the applied current. User can also set if the measurement from the drive pattern is to be used or need to be removed. This process interprets the stimulation information into a vector form.

Next, homogeneous data and non-homogeneous data are loaded. It is read through text files; hence user needs to browse and select the specific text file. The data is read from the file and stored in a matrix for further processing. If the measurement from drive pattern to be excluded, it is removed at this stage. Then, a difference data is calculated by subtracting homogeneous data from in-homogeneous data.

As Gauss Newton One step algorithm is implemented for image reconstruction, a prior needs to be selected. Depending upon user selection the prior R is calculated. At this stage, measurement noise covariance W and Jacobian Matrix J is calculated. User also needs to enter hyperparameter before starting reconstruction. The image reconstruction algorithm is designed according to its EIDORS implementation. Once the reconstruction button is clicked, the algorithm is run, and the resulting image is drawn on the GUI.

A brief flowchart of the design is created. Also, the code is tested with the given homogeneous and inhomogeneous data and the resulting images are shown in following section.

3.1.2 Flowchart

Figure 3.1 shows the workflow which summarizes C# code in [19]

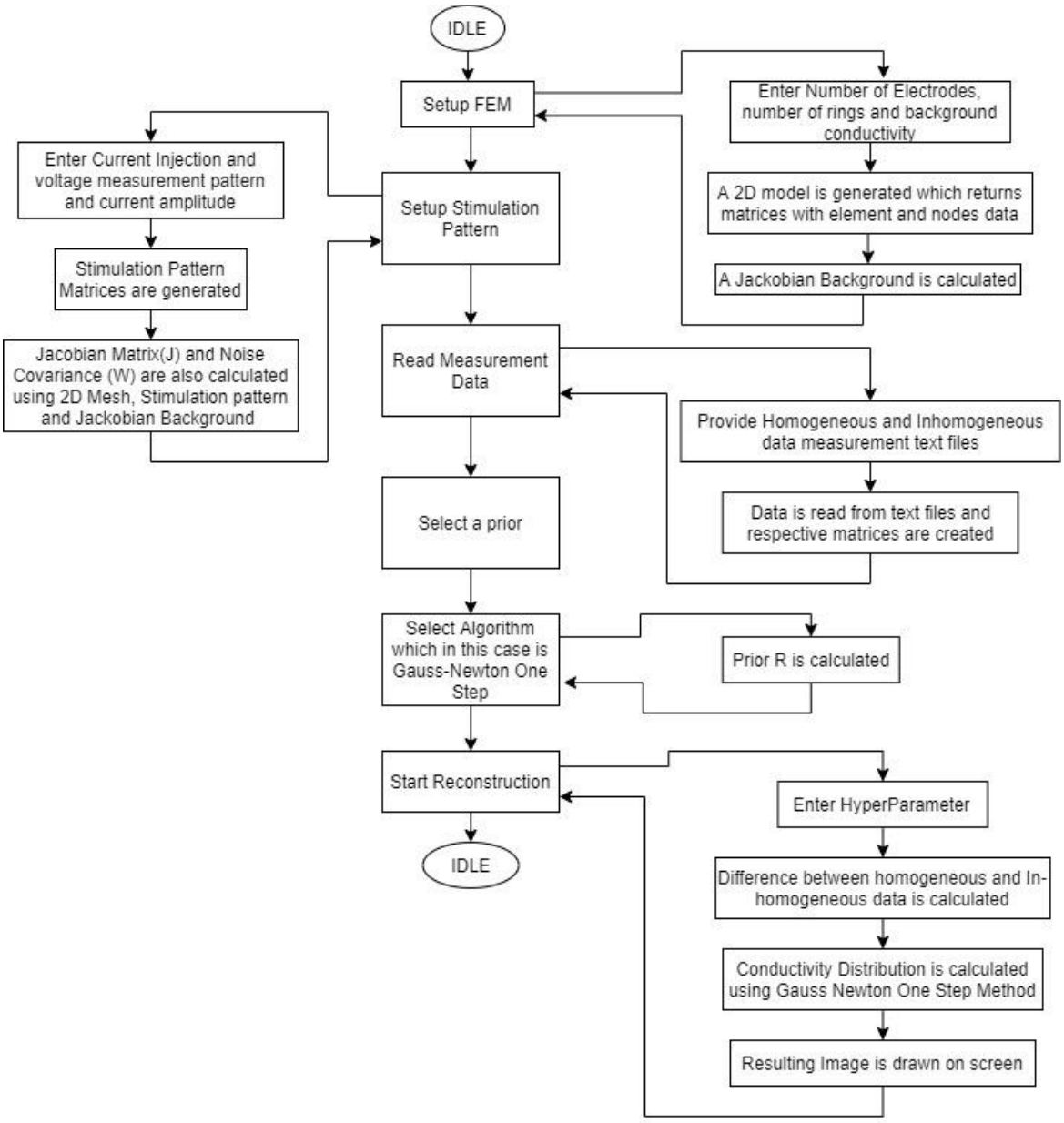


Figure 3. 1 C# Implementation Flowchart (Chen)

3.1.3 Test Results

The C# implementation in this project is tested using an 8 electrode measurement data. The text files for homogeneous measurement and inhomogeneous measurement are available. The code is tested for different number of rings, the options for rings are 4,8,12 and 16, and different priors, the options for priors are NOSER, Tikhonov and Laplace. The result with 16 rings and Tikhonov prior is shown below in figure 3.2.

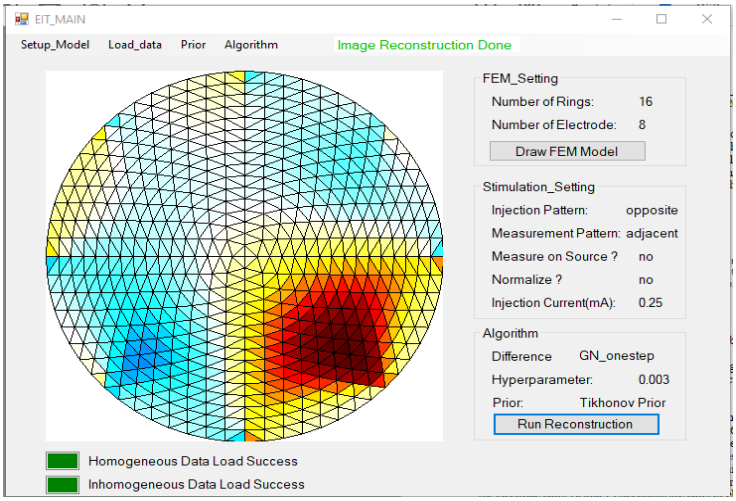


Figure 3. 2 Image for 16 rings and Tikhonov Prior (Chen)

3.2 Image Reconstruction and EIT [33]

3.2.1 Design Description

In this project, three EIT image reconstruction algorithms are implemented in C. The algorithms are Gauss Newton one step, Total Variation and Conjugate Gradient. A GUI is designed to allow user to enter the system parameters and measurement data files. Multithreading is also implemented in order to make the design more time efficient. The parameters that can be set from

the GUI are measurement data files, algorithm selection, FEM parameters, stimulation parameters, prior selection, hyperparameter and number of iterations.

The homogeneous and inhomogeneous data can be loaded from the menu option *Load* on GUI. The files can be loaded in two formats, txt and xls or xlsx. The data is stored in matrix format for further use.

The parameters used to build finite element model such as number of electrodes, number of rings and background conductivity can be entered from the textboxes available on GUI. On the click of Setup FEM button, a 2D Mesh is generated and FEM model is displayed on the GUI. Later, if the number of electrodes or number of rings are changed, the FEM model is updated accordingly. The Jacobian background matrix is also created.

The design implements three priors, Laplace, NOSER and Tikhonov. Default prior is set as Laplace and can be changed from GUI. If the prior is changed, the prior matrix is updated accordingly.

The algorithm selection can be done from the menu bar available on GUI. To reconstruct the image, an algorithm needs to be selected. The hyperparameter value can also be set from GUI. The maximum iteration parameter required in case of TV and conjugate gradient is set as 3 and 50 respectively, by default. There is a Re-generate button on GUI, if pressed the set algorithm gets executed and resulting image gets drawn on screen. A timer is used to check if the calculations are done and image is ready to be drawn.

3.2.2 Flowchart

Figure 3.3 shows the working flowchart of the C# design

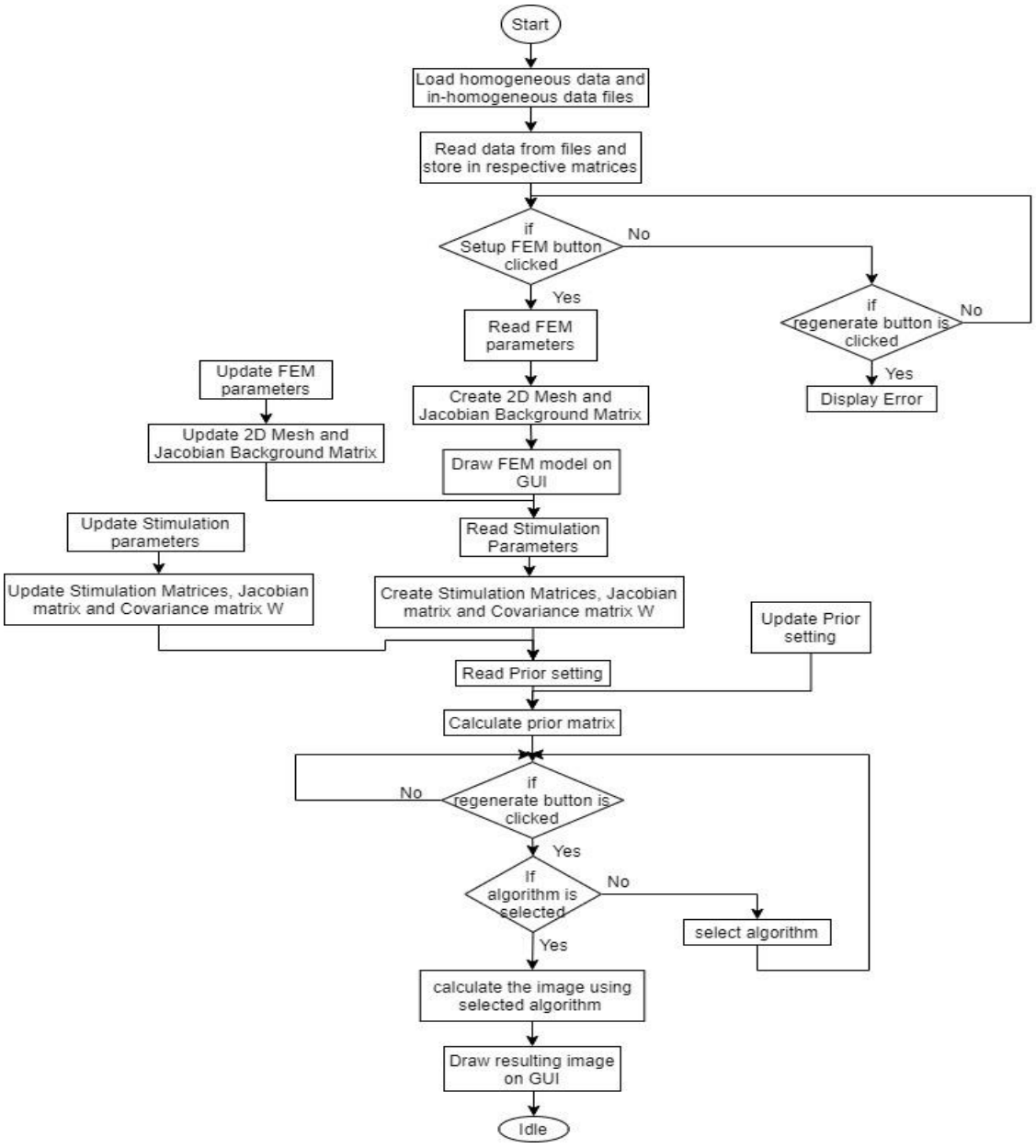


Figure 3. 3 C# Implementation Flowchart (Rabi)

3.2.3 Test Results

The code was tested using available 8-electrode homogeneous and inhomogeneous data. All three algorithms for different rings and prior settings are tested and results are captured. The result with conjugate gradient algorithm, 24 rings and Laplace prior is shown below in figure 3.4.

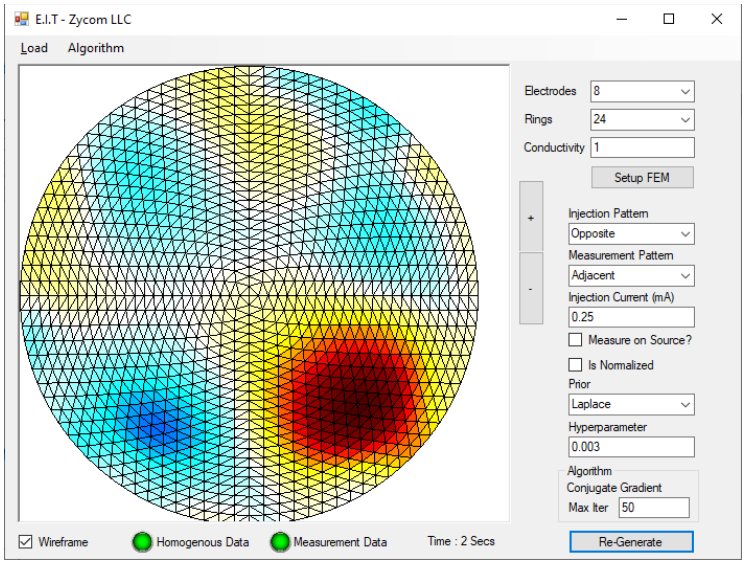


Figure 3. 4 Conjugate Gradient Image result with 24 rings, Laplace Prior (Rabi)

3.3 Real-Time Electrical Impedance Tomography [9]

3.3.1 Design Description

This project includes a complete system design for real time Electrical Impedance Tomography for head imaging. A FPGA based system is developed which can drive the electrodes, read the voltages at the electrodes, process the acquired data, and send the resulting image data to a C# based GUI for display. A 12-electrode system is used to connect to a circular phantom. Analog circuitry is developed to drive and read data to and from the electrodes. A

Zynq 7020 device-based development board, Zedboard, is used to interface the analog circuit to the onboard Zynq FPGA. The Zynq FPGA controls the electrode interface as well as process the acquired data. Gauss Newton One step algorithm along with Tikhonov prior is implemented for image reconstruction. The C# GUI is designed to provide user with control over system operation, such as start, stop acquisition and processing etc. The GUI communicates with Zynq FPGA over serial port. It receives the processed data and displays it on the screen. The GUI also provides an option of one-time measurement and continuous measurement. With the continuous measurement option selected, the system is designed to be able to produce the image data continuously in real-time.

3.3.2 Hardware Description

Figure 3.5 shows the block level representation of analog circuitry, Analog Front End board designed in this project to drive and read to and from the electrodes.

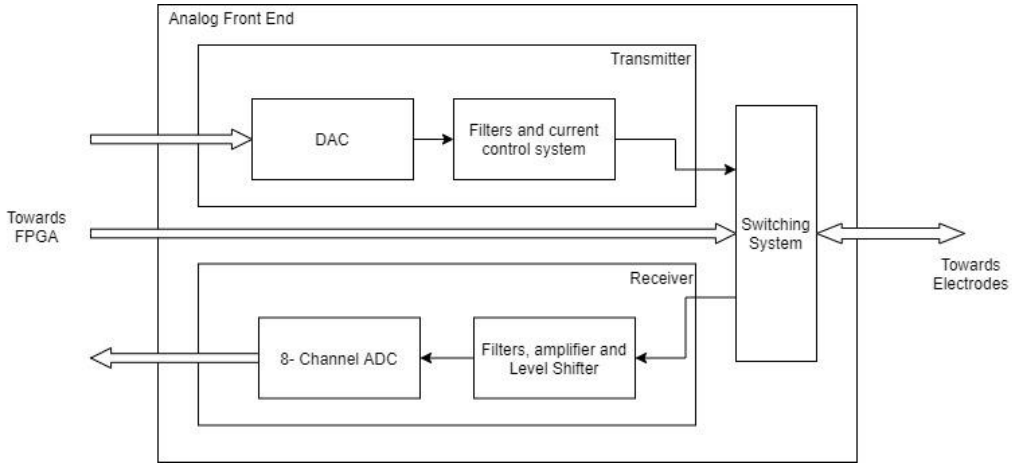


Figure 3. 5 Block Diagram of Analog Front End Hardware

The devices (ICs) used in AFE board are as follows

- DAC LTC1668
- Operational Amplifier OPA 2192
- Multiplexer ADG 1606
- Digital Potentiometer AD5293
- Instrumentation Amplifier AD8429
- ADC AD7761
- FMC Connector

As shown in the block diagram, the AFE has two main sections, transmitter, and receiver. The transmitter section includes DAC, filters, and the constant current circuit to generate the sinusoidal signal to drive the electrodes. A digital data (16-bit) for a sinusoidal signal at different frequencies is generated in the FPGA. This digital data is converted to analog signal using Linear Technology LTC1668 DAC chip. The Linear Technology LTC1668 IC is a 16-bit 50Msps digital to analog converter. The DAC takes 16-bit input data and gives differential current output. The differential current output is filtered through an active bandpass filter designed for a range of 100Hz-100kHz. The filtered signal is passed through a current source before passing it to the electrodes through multiplexing switches. To limit the output current, a digital potentiometer, AD5293 is used. The value setting for digital potentiometer is controlled through FPGA.

In EIT systems, the current needs to be injected through each electrode in a pattern, hence a multiplexer arrangement using Analog Devices ADG1606 multiplexer IC is incorporated to connect the current signal to electrodes. The ADG1606 IC is 16:1 multiplexer. There are twelve such multiplexers used in the design and each multiplexer is connected to all twelve electrodes.

Two of the multiplexers are connected to the differential current input coming from DAC, and other ten multiplexers are connected to the receiver circuit. This way, using the four select signals of the multiplexers, each multiplexer gets connected to one electrode each time. The select signals are controlled from FPGA to control the current application and voltage measurement pattern.

The induced voltage measurement is connected to the receiver circuit through the multiplexer section. As mentioned above, at every switch rotation, there would be two electrodes connected to inject the current and other ten electrodes to measure induced voltage. These ten electrodes are used in pairs to generate 8 measurements. Signal from each electrode is passed through a passive bandpass filter designed for a range of 150Hz to 300kHz. The filtered signal from adjacent electrodes are then passed through an instrumentation amplifier to generate one measurement output. The instrumentation amplifier stage is set with a gain of 7 [37]. The single ended signal is passed through an active bandpass filter designed for a range 70Hz to 398kHz. The filter also provides a gain of 2. Hence, effectively, the signal is applied with a gain of 14 before giving it to the ADC.

The Analog Devices AD7761 ADC is an 8-channel 16-bit ADC, with individual serial output for each channel. This ADC converts 8 channel analog data to digital and provides it at 8 serial outputs in 16bit format. The output of ADC is passed to the FPGA for further processing. The AFE is connected to the Zedboard through an FPGA Mezzanine Card (FMC) connector.

3.3.3 FPGA Implementation

Zynq FPGA design constitutes two parts, programmable logic design and processor software design. The processor is mainly responsible for communicating with the GUI over serial port for getting the system configuration and sending the image reconstruction output data. The processor also reads digitized voltage measurement data from PL and processes it through DFT to get the magnitude of the measurement. This magnitude data is passed to the PL logic to use in reconstruction process.

The Programmable logic is mainly responsible for reconstruction algorithm implementation. The PL also generates the DAC signal as per the configuration received from processor and stores the ADC data readout for all 8 channels to pass to the processor. For image reconstruction, since Gauss Newton One Step algorithm is used, following equation governs the reconstruction process

$$\Delta\sigma = (\boldsymbol{J}^T W \boldsymbol{J} + \lambda^2 R)^{-1} \boldsymbol{J}^T W \Delta V \tag{3.1}$$

Where, $\Delta\sigma$ = change in impedance, to be calculated

$\boldsymbol{J}$ = Jacobian Matrix

W = variance matrix

λ = hyperparameter

R = Prior Matrix

ΔV = Difference voltage

Above equation can be written in the following format

$$\Delta\sigma = B\ \Delta V \tag{3.2}$$

where B is called as reconstruction matrix and given as

$$B = (\boldsymbol{J}^T W \boldsymbol{J} + \lambda^2 R)^{-1} \boldsymbol{J}^T W \quad (3.3)$$

In the present system, most of the part of the reconstruction matrix can be kept constant. For instance, Tikhonov prior is fixed for this implementation. Additionally, the parameters required to calculate FEM such as number of electrodes, number of rings, current injection pattern, measurement pattern are also fixed. Hence, the reconstruction matrix is precalculated using MATLAB and stored as constant in the FPGA code. The difference voltage matrix is calculated using magnitude data provided by processor and the image data is calculated by multiplying reconstruction matrix and voltage difference matrix. The calculated data output is passed to processor where it transmits the data to GUI over serial port. Figure 3.6 shows the block diagram which summarizes the FPGA functionality

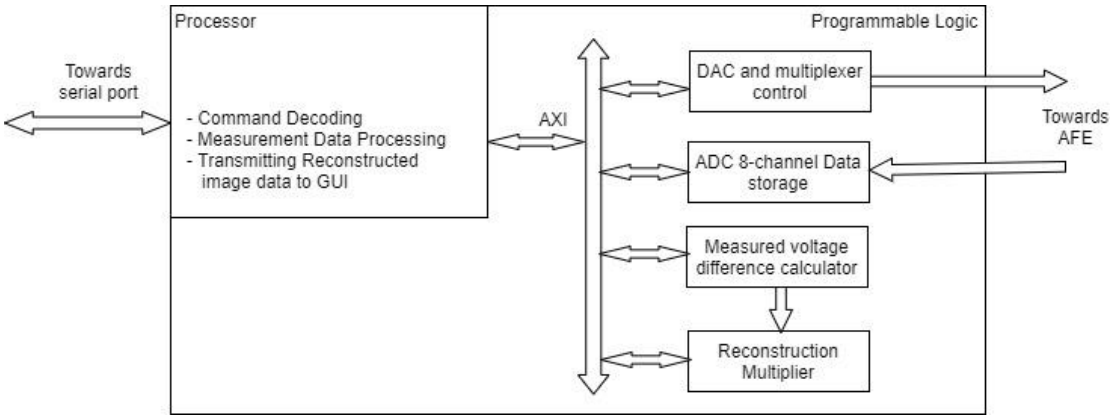


Figure 3. 6 FPGA Block Level Implementation (Senior Design Team)

3.3.4 FPGA Design Analysis

In this section, a detailed analysis of FPGA design is performed. The FPGA implementation is designed to drive and read data from 12 electrodes attached to a circular phantom, process the measurements in FPGA for impedance image reconstruction and transmit

the resulting image to GUI over serial port to display. In this design the reconstruction matrix in equation 3.3, is derived using MATLAB before programming the FPGA and used as a constant for solving equation 3.2. The reconstruction matrix size is 576X96, which is due to 12 rings, 12 electrode model. The homogeneous and inhomogeneous measurement data are retrieved in PS and passed to PL. The measurement matrix is 96X1 as there are 8 measurements and 12 total electrodes. The difference vector, which is 96X1 matrix, is calculated in PL. Hence, effectively, the row-wise multiplication of 576X96 matrix and 96X1 vector is performed in PL, that is equation 3.2, to get the output image. This multiplication output is transferred to GUI through processor and serial port. In this implementation, there are 96 multiplications which needs to be done one time to get one output. The present code implements two such multipliers and uses 192 DSP slices out of 220 available. Also, in this design the processor is used for serial communication and DFT implementation for measurement data magnitude calculation.

Figure 3.7 shows the timing summary of the present implementation. It is observed that the present design has the total negative slack of almost 1.7 us and 410 failing endpoints. Negative slack implies that the signal may not reach to the destination within the desired time. Total negative slack is the addition of all the worst negative slack violations [38].

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	-18.883 ns	Worst Hold Slack (WHS):	0.022 ns
Total Negative Slack (TNS):	-1739.044 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	410	Number of Failing Endpoints:	0
Total Number of Endpoints:	7844	Total Number of Endpoints:	7844

Timing constraints are not met.

Figure 3. 7 Timing Summary of FPGA design (Senior Design Team)

Figure 3.8 shows the device utilization report of FPGA implementation. It is observed that, in the design most of the LUTs are used for data storage which makes total LUT utilization up to 82.28%. Also, in the design, two 96 input multipliers are implemented, which makes DSP utilization as 87.27%.

Resource	Utilization	Available	Utilization %
LUT	43773	53200	82.28
LUTRAM	96	17400	0.55
FF	9174	106400	8.62
BRAM	64	140	45.71
DSP	192	220	87.27
IO	47	200	23.50
MMCM	1	4	25.00
PLL	1	4	25.00

Figure 3. 8 Device Utilization Report (Senior Design Team)

3.3.5 GUI Design Description

A graphical user interface is designed using C#, which communicates with Zynq FPGA over serial port. The GUI provides user with controls such as initiate reading homogeneous and inhomogeneous measurements. It also allows user to set the frequency of the injection current to be set as one of the following - 500Hz, 1kHz, 2.5kHz, 5kHz, 10kHz. The number of samples to be used to get magnitude of the measured signal through DFT can also be set as 512, 1024, 2048, 4096, 8192 samples, through GUI. A command structure is designed to communicate the instructions between GUI and Zynq FPGA.

Once measurements are done, the reconstruction calculations are performed inside FPGA and the resulting data is sent to GUI. The reconstruction data is stored in proper format and displayed on the GUI. A 2D Mesh calculation is performed to draw the reconstruction image on

the GUI. This operation of acquiring the data to printing the image on screen can be performed once or continuously, depending upon user selection. A handshaking command is used to initiate the communication repeatedly in case continuous operation is selected. Flowchart in figure 3.9 summarizes the working of C# code.

3.3.6 System Performance

According to [9], this system could generate 1.4 frames per second in continuous mode. The frame rate was measured for system setting as 512 number of samples, 128ksps sampling frequency and 115200 baud rate. The time required to complete the process for image reconstruction starting from image data acquisition is measured or calculated to verify the frame rate. It takes 48ms for data acquisition for twelve switch positions, with 512 number of samples per channel and 128ksps sampling frequency. For the FPGA computation, that is image reconstruction computation, it takes around 100ms. At 115200 baud rate, the time taken for transmission of image data to PC over serial port is 200ms. The time taken for image to refresh on screen in C# takes 350ms. This way, complete data acquisition to image display time takes 700ms, which gives frame rate of 1.4.

Figure 3.9 shows the working flow of C# code.

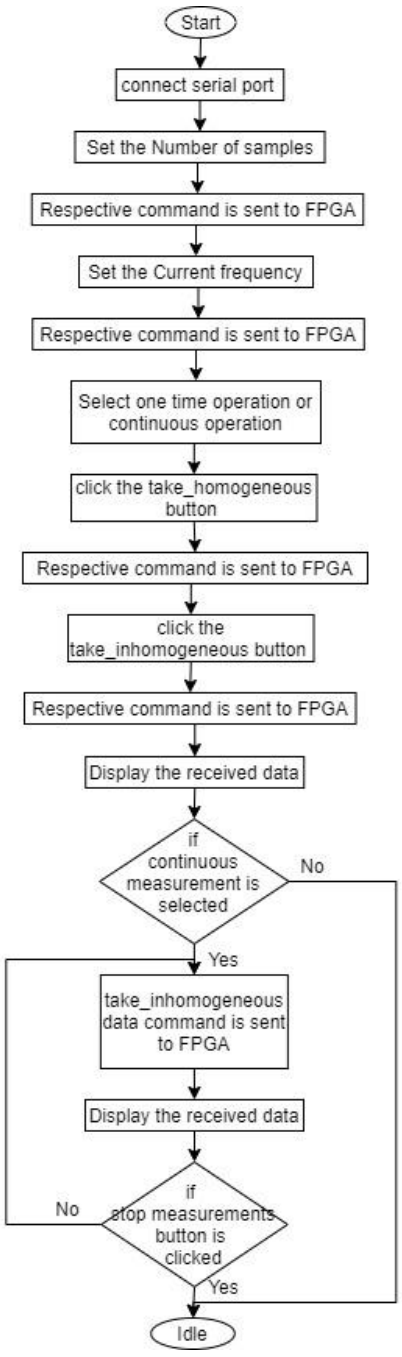


Figure 3. 9 Flowchart for GUI Implementation (Senior Design Team)

Chapter 4: EIT MATLAB Implementation

A MATLAB based open source software called as EIDORS provides a wide range of functions which can be used to solve an EIT reconstruction problem. As a part of preliminary work, a MATLAB code using EIDORS is developed. This chapter includes the description of the MATLAB code design and test results.

4.1 About EIDORS

The Electrical Impedance and Diffused Optical Reconstruction Software (EIDORS) provides a software library for 2D as well as 3D reconstruction for electrical impedance tomography as well as optical tomography [15]. It is developed using MATLAB. EIDORS provided software libraries can be used to reconstruct an EIT image using complex algorithms. There are various FEM models, algorithms and prior methods implemented in EIDORS. In this project, a MATLAB code using EIDORS is developed to understand the forward and inverse modeling in EIT reconstruction process.

The EIDORS software download and help can be found on the following website

http://eidors3d.sourceforge.net/

Once the software is downloaded, the folder needs to be unzipped. In order to use EIDORS libraries in MATLAB, user needs to run the following command in MATLAB command window or at the starting of the code which uses EIDORS functions

run C:/path/to/eidors/startup.m

4.2 Design Description

In EIT image reconstruction, first a forward model is built depending upon the number of electrodes and number of rings. Number of rings is the specification for generating a finite element model. The forward model and measurement data are used to obtain the inverse solution. In this code the measurement data that is homogeneous and inhomogeneous data are read from text files. At the starting of this project only eight electrode data were available in text files, hence, the code is designed for eight number of electrodes.

At first, the measurement data is loaded. EIDORS provides various forward model building options for complex geometries. A function *mk_common_model()* provides FEM model for geometries such as 2D circle, 2D thorax, cylindrical etc. A 2D circular model from 64 elements to 6400 elements, which corresponds to 4 to 40 equally spaced rings, can be constructed. However, as the number of elements increases, the computation time increases. Hence, in this project, four options for number of rings are provided, 4, 8, 12 and 16. User can select any one option and corresponding FEM model will be created.

The computation requires the stimulation pattern and parameters used at the time of taking the measurements. In the case of the eight-electrode data which is used, the stimulation is given in opposite pattern, and measurements are taken from adjacent electrodes. Also, the stimulation current is used as 0.25mA. These parameters are used to create a stimulation pattern.

EIDORS provides various algorithms to solve the inverse problem. In this code, Gauss Newton One Step and Total Variation algorithms are used. User can select any one of the algorithms for further computation. For Total Variation, EIDORS provides two methods, Iteratively Reweighted Least Squares (IRLS) and Primal Dual – Interior Point Method (PDIPM)

[31]. In this project IRLS method is used. Next, three prior application options such as Laplace, Noser and Tikhonov are provided. User can select any one of the priors for further computation. Using the forward model, selected algorithm and prior and measurement data the inverse solution is obtained using inv_solve() function. Finally, the image is displayed on the screen.

Flowchart

Figure 4.1 shows the flowchart for the MATLAB code designed for EIT image reconstruction. The code is given in the appendix section in this document.

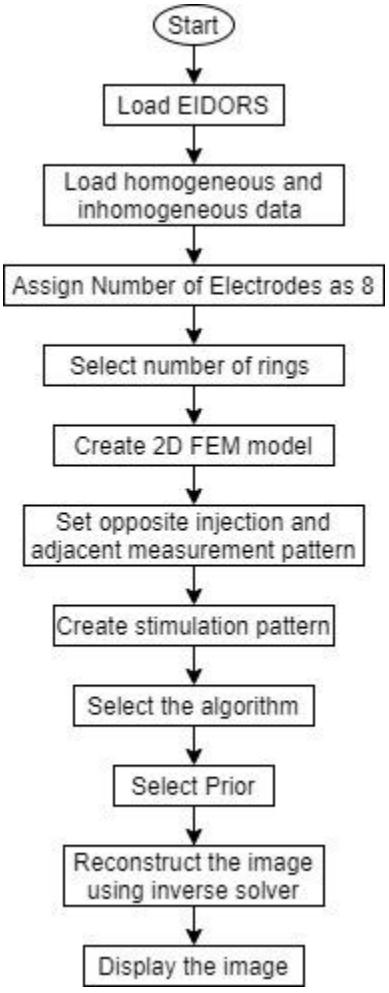


Figure 4. 1 EIDORS Code Flowchart

4.3 Test Results

The above designed code is tested using available eight electrode measurement data. All the options were tried to observe the changes in the reconstructed image. Following images, figure 4.2 and figure 4.3, show some of the results. The blue area in the image is a low impedance anomaly which is introduced while taking inhomogeneous measurements. It was observed that, the Total Variation algorithm, as it is iterative, provides a more precise solution with respect to the one step Gauss Newton algorithm. Also, as the number of elements increased, the low impedance area in the reconstructed image started to get precise. Also, the Noser prior was observed to give a better result.

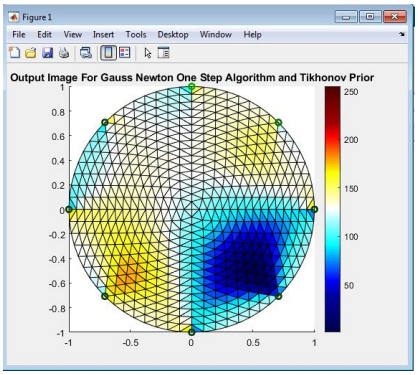


Figure 4. 2 Image with Gauss Newton One Step, 16 rings and Tikhonov Prior

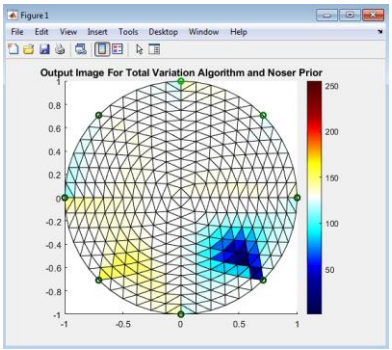


Figure 4. 3 Image with Total Variation, 12 rings and Noser Prior

Chapter 5: XILINX Development Hardware and Software

In this project, the Xilinx provided Zynq 7020 SoC based development board is used as a controller and processor for the EIT system. This chapter introduces the Xilinx Zynq SoC architecture, a brief introduction of the development board used in this project and the Xilinx provided VIVADO design suite for FPGA design development.

5.1 Device Architecture

The Xilinx Zynq-7000 SoC family integrates dual core ARM cortex A9 processor and high performance, low power programmable logic [39]. The programmable logic as well as processor can be programmed independently or to work together, depending upon the requirement of the end application. The availability of ARM based processors allows usage of operating systems like Linux.

Figure 5.1 shows the block diagram of architecture of Zynq SoC.

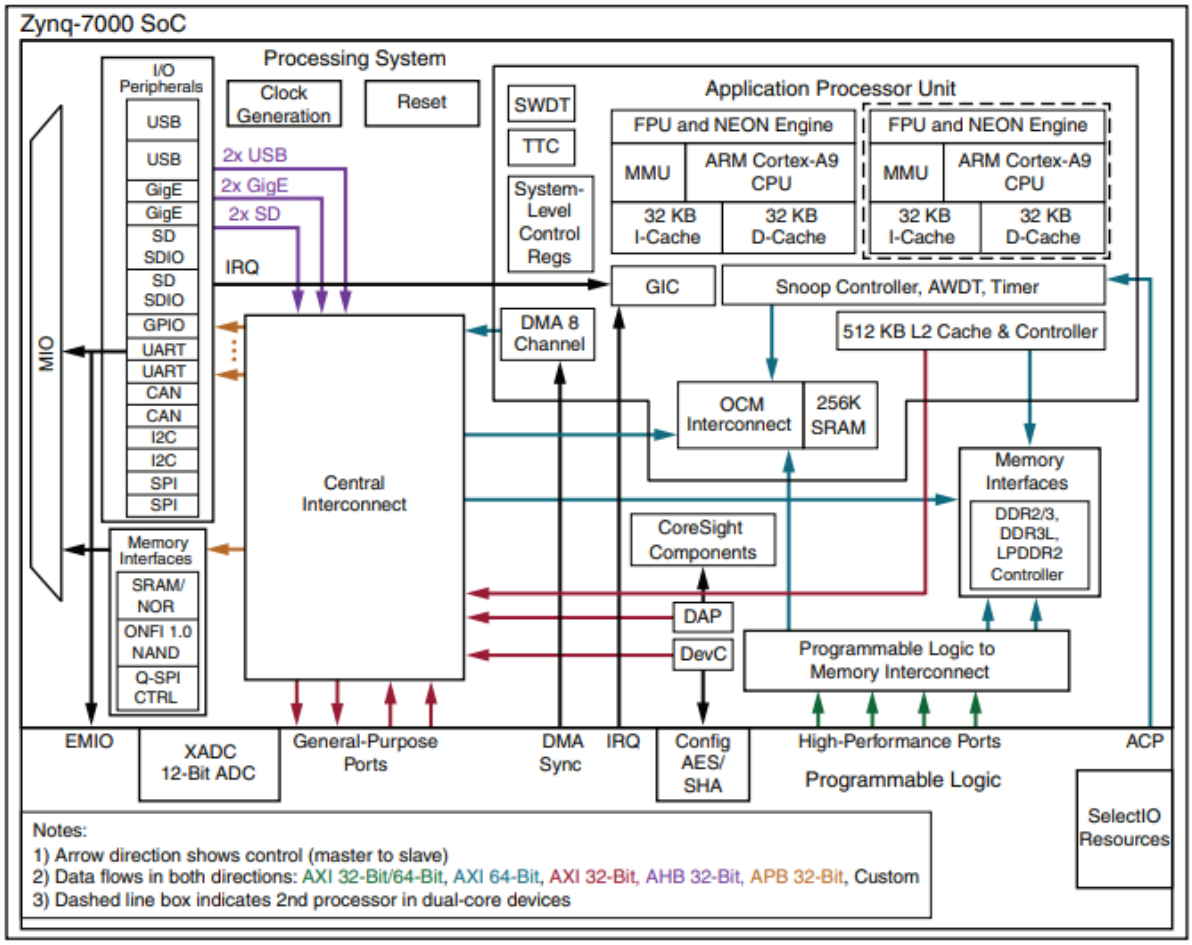


Figure 5. 1 Zynq FPGA SoC Architecture *[39]*

As can be seen in above figure, the Zynq SoC device is divided into two parts, processing system and programmable logic. The processing system (PS) includes ARM cortex A9 processor along with I/O peripherals such as UART, USB, Ethernet, CAN, SDIO, I2C and SPI and 54 general purpose IOs. It also includes on chip memory and external memory interfaces such as DDR controller, quad SPI controller, NAND controller and SRAM controller.

The programmable logic includes configurable logic block (CLBs), block RAMs, DSP48E1 slices for digital signal processing, configurable I/Os and Analog to Digital converter. To interface PS and PL side designs, two types of interfaces can be used, functional interfaces and configuration interfaces. Functional interfaces include AXI (Advanced eXtensible Interface), Extended MIO interface (EMIO), interrupts, DMA flow control, clocks, and debug interfaces. Configuration interface signals are the signals which are connected to a fixed logic within the PL configuration block in order to provide control to PS. It includes Processor Configuration Access Port (PCAP), Configuration Status, Single event Update (SEU) and program/done/init signals.

Due to the high performance and scalable architecture of Zynq SoCs, they are increasingly becoming popular for various applications such as automation, networking, digital communication, biomedical instrumentation etc.[39]. With the availability of high-performance interconnect interfaces between PS and PL, a complex design can be partitioned in Hardware (PL) and Software (PS) for efficient performance. More detailed information about Zynq-7000 devices can be found in [39] and [40]. Depending upon application requirement and algorithm complexity, the Zynq-7000 family device can be selected.

5.1.1 Zynq 7020 Device Specifications

- Processor Specification
    - Processor: Dual core ARM Cortex-A9 MPCore
    - Maximum Frequency: 667MHz
    - L1 Cache: 32 KB Instruction, 32 KB data cache
    - L2 Cache: 512 KB

- On Chip Memory: 256 KB
- External Memory Support: DDR3, Qaud-SPI, NAND, NOR
- Peripherals: UART, CAN, I2C, SPI, USB, Gigabit Ethernet, SDIO

- Programmable Logic Specification
  - FPGA: Artix-7
  - Programmable Logic Cells: 85k
  - Look-Up Tables (LUTs): 53,200
  - Flip-Flops: 106,400
  - Block RAM (36Kb Blocks): 140
  - DSP Slices: 220
  - ADC

5.2 Development Board

In this project, a Zynq 7020 based development board that is Zedboard is used. This Zynq evaluation and development board includes variety of on-board peripherals and expansion connectors.

Figure 5.2 shows the hardware block diagram of Zedboard [41].

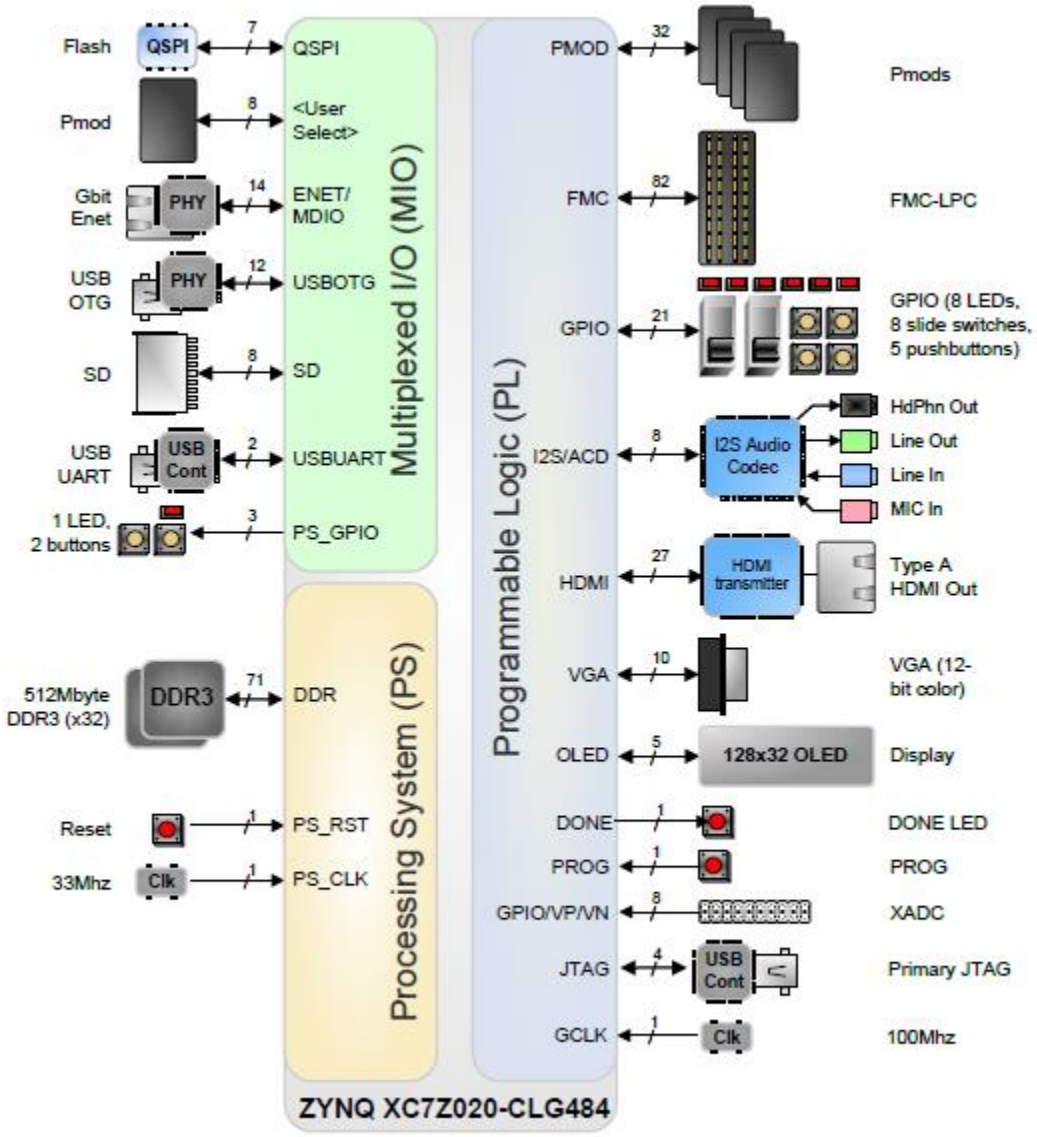


Figure 5. 2 Zedboard Block Diagram *[41]*

As shown in above figure, Zedboard provides many onboard peripherals which makes it preferred platform for various designs. It provides onboard memory such as DDR3 and QSPI. It

also provides various interfaces such as USB, Ethernet, UART, SDIO, PMOD connectors, LEDs and switches. It also has HDMI output port, VGA port and Audio line in, line out, headphone and microphone in. The Zynq FPGA provides few different boot methods. The boot process can be performed through NOR flash memory, NAND flash memory, Quad-SPI memory, SD card or JTAG. The boot mode can be selected by varying the inputs to boot mode configuration IO pins. Zedboard provides jumpers JP7,8,9,10 and 11 using which the boot mode can be selected. In this project, a JTAG boot mode is set by connecting the JTAG configuration pins to GND using the jumper connections. The USB-JTAG port is used to program the FPGA and PS.

5.3 Development Software

The Xilinx provided VIVADO Design Suite includes various design tools required to develop an embedded system along with FPGA design. VIVADO Design Suite includes [42]

- VIVADO Integrated Design Environment (IDE)
- Software Development kit (SDK)
- Intellectual property cores

The VIVADO IDE allows user to develop a customized FPGA design including embedded processor hardware. The VIVADO IDE provides a graphical user interface for Xilinx FPGA designs. It provides following features [43]

- Register Transfer Level (RTL) design using VHDL, Verilog and System Verilog or a block wise integration of intellectual property blocks.
- Behavioral, Functional and Timing Simulation
- Synthesis, Place and Route implementation, floor planning

- Logic Analyzer for debugging the design
- Constraints entry
- Timing Analysis
- Bitstream generation

The FPGA hardware design developed using VIVADO IDE is used as a hardware platform for the software development. The Software Development Kit (SDK) allows user to develop the software application. The SDK provides following features

- Design a C/C++ based standalone application
- Design a Linux based application
- Build the design
- Program the FPGA and Debug

In this project VIVADO IDE 2018.3 and SDK 2018.3 are used for PL and PS design and development respectively. More detailed information about VIVADO Design Suite can be found in [39]–[43].

Chapter 6: EIT System Architecture

In this chapter, a block wise architecture of the complete EIT system implemented in this project is discussed. It also includes the design specifications and a brief description of the image reconstruction algorithms implementation.

6.1 System Architecture

Figure 6.1 shows the block level implementation of the EIT system. It includes a graphical user interface, data acquisition hardware and a phantom created for testing the EIT system.

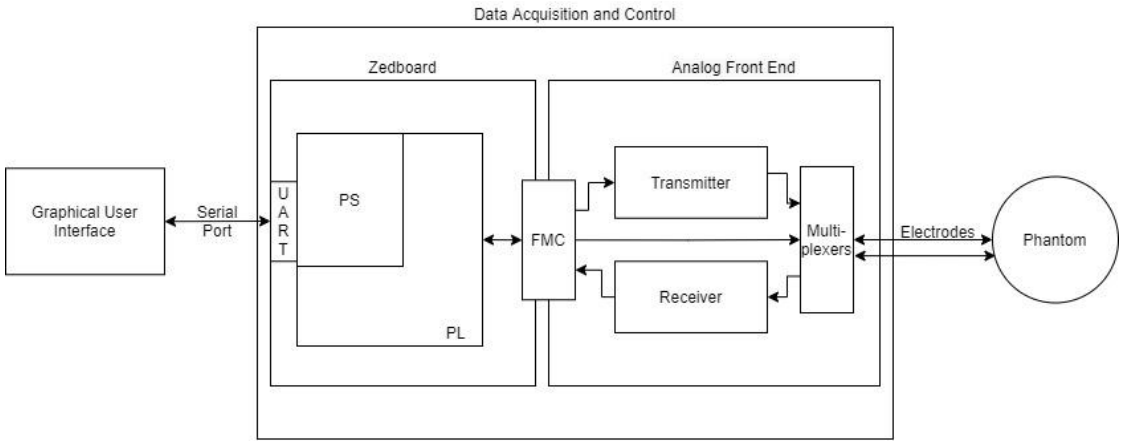


Figure 6. 1 Block Diagram of EIT System Implementation

As shown in the figure, the data acquisition and control system communicate with the graphical user interface through a serial port. Also, the data acquisition system provides the current signal to the phantom and reads the voltage measurement from the phantom through electrodes. The phantom tissue is created using animal hide gelatin. It is a twelve-electrode system, which means that the data acquisition system is designed for controlling and acquiring

data from twelve electrodes. The injected current frequency can be set from the GUI. There are two image reconstruction algorithms implemented in this system, Gauss Newton One Step and Total Variation.

In the following section, the functionality and architecture of GUI and data acquisition system is described. The complete system specifications are also listed.

6.1.1 Data Acquisition and Control

The data acquisition system constitutes of a Zedboard and the analog front-end (AFE) hardware. As mentioned in chapter 3, the AFE is designed by one of the senior design teams [9] as a part of their project. The detailed architecture of AFE is described in chapter 3. The AFE is originally designed to work at high frequency with the range 500Hz to 10kHz. In this project, the AFE hardware is modified to work with low frequency range from 1Hz to 30Hz. The detailed design modifications are given in chapter 6.

The Zedboard is a development board designed around Xilinx Zynq SoC. The Zynq SoC includes an ARM Cortex A9 processor (PS) and a high-speed FPGA (PL). The Zynq SoC architecture and device specifications are given in chapter 4. The Zynq SoC is responsible for

- Communicating with GUI for receiving system specifications
- Generating sinusoidal signal for current injection
- Acquiring the voltage measurement data through ADC
- Process the data through DFT
- Use the voltage data to reconstruct the image using the selected algorithm
- Transfer the image data to GUI for display

The design to implement above mentioned functionalities is developed using both PS and PL in Zynq SoC. The processor initializes UART and all the hardware designed in PL. The processor receives the system configuration from GUI and configures respective sub systems in FPGA accordingly. For instance, if the signal frequency is set as 30Hz from GUI, the FPGA logic designed to generate sinusoidal signal is configured to output 30 Hz. The processor also configures the ADC through SPI communication. It also reads the sampled ADC data and processes it through DFT to get the measured voltage amplitude. The reconstruction and Jacobian matrices required for image reconstruction algorithms' computation are stored in the SD-card. The processor, at initialization, reads the files stored in SD-card and copies the data in DDR memory for later use.

The FPGA (PL) includes the logic to generate the sinusoidal signal for the DAC on the AFE. The PL also includes the logic to control digital potentiometer. The ADC on AFE board is configured by PS for setting the sampling frequency, however, the sampling frequency is varied as per the requirement within the PL logic. The ADC provides the digitized data as 8 channel serial input. The serial input is parallelized and stored in FIFOs inside FPGA. The data storage in FIFOs is controlled by PS. The FIFO data is read by PS and processed through DFT to get the voltage measurement data. The Gauss Newton One Step algorithm computation is a matrix and vector multiplication, that is reconstruction matrix and voltage measurement data. This computation is implemented in the FPGA. The voltage measurement data and reconstruction matrix data are passed by PS to PL and the parallel computation is performed. The resulting output is stored in DDR to transfer to GUI for display. The detailed implementation is described in chapter 7.

The AFE is connected to Zedboard through an FMC connector available on Zedboard. The FMC connector provides 68 single ended FPGA IOs [41].

6.1.2 Graphical User Interface

The graphical user interface (GUI) is designed using C# windows application form. The GUI is mainly responsible for allowing user to set the system parameters, communicating with Zynq SoC over serial port to transfer the parameters and receive the image or voltage difference data, and displaying the processed image.

There is a communication protocol set for data transfer between GUI and Zynq SoC. The system parameters that can be set from GUI are signal frequency, sampling frequency and number of samples. As previously mentioned, there are two image reconstruction algorithms implemented, the GUI allows the algorithm selection and the parameters related to algorithm such as hyperparameter and number of iterations. The image reconstruction algorithms are implemented in Zynq SoC as well as C# GUI, the user can select whether to use Zynq SoC hardware only for data acquisition and process the data in software (C#) or to process the data in SoC and only display the reconstructed image on GUI. If Zynq SoC is only used for data acquisition, it transfers the difference voltage data to GUI. This data is processed through selected algorithm on GUI. This data is also stored in text files, such up to 100 frames can be stored. This data can be used later, that is offline to reconstruct and observe the image. The reconstruction matrix and Jacobian matrix required for computations are calculated once and stored in text files for later use. This way, the matrices need not be calculated each time reconstruction computation is done. Also, to display the image on GUI, the 2D FEM mesh is

computed in C#. The mesh computation and Jacobian matrix computation is used from the C# implementation developed by one of the previous students in St. Cloud State University [19]. The detailed implementation of C# GUI developed in this project is described in chapter 8.

6.1.3 System Specification

Following is the list of specifications provided by the EIT system

- Twelve electrode EIT hardware
- Twelve rings forward model (mesh)
- Opposite current injection system
- Fixed 0.25 mA current injection
- 2Hz to 10kHz current signal frequency range
- Sampling Frequency options 250Hz, 500Hz, 1kHz, 2kHz and 128kHz
- Number of samples for data acquisition can be set as 256, 512, 1024, 2048, 4096 and 8192
- Image reconstruction algorithms
    - Gauss Newton One Step
    - Total Variation

- Tikhonov prior
- Number of iterations can be set from 1 to 6
- Hyper parameter can be set as 0.3,0.03 or 0.003
- Single or continuous acquisition
- Option for data processing in Zynq SoC or C# GUI
- Up to 100 measurement data storage
- Image reconstruction from database

6.2 Algorithm Implementation

As mentioned above, the image reconstruction algorithms are implemented in the Zynq SoC as well as in C# (GUI). In this section the reconstruction algorithms' implementation is discussed. A one step algorithm that is Gauss Newton One Step and an iterative algorithm that is Total Variation are implemented. As both the algorithms are time difference method, the voltage measurements are taken as homogeneous at one instant and inhomogeneous at another instant. The voltage difference between homogeneous and inhomogeneous is calculated and used in the conductivity reconstruction computation.

6.2.1 Gauss Newton One Step

As explained in the chapter 2, Gauss Newton One Step algorithm is represented using following equation

$$\Delta\sigma = (\boldsymbol{J}^T W \boldsymbol{J} + \lambda^2 R)^{-1} \boldsymbol{J}^T W \Delta V \tag{6.1}$$

Where, $\Delta\sigma$ = change in conductivity, to be calculated

$\boldsymbol{J}$ = Jacobian Matrix

W = variance matrix

λ = hyperparameter

R = Prior Matrix

ΔV = Difference voltage

Above equation can be written in the following format

$$\Delta\sigma = B\,\Delta V \tag{6.2}$$

Where B is called as reconstruction matrix and given as

$$\mathrm{B} = (\boldsymbol{J}^T W \boldsymbol{J} + \lambda^2 R)^{-1} \boldsymbol{J}^T W \tag{6.3}$$

In equation 6.3, the variance matrix, W, is identity matrix since it is a difference method. If a Tikhonov prior method is used, then R becomes identity matrix. The hyperparameter is kept as 0.003 throughout the computation.

The Jacobian matrix computation depends upon the forward model, that is number of electrodes, stimulation method and number of rings. Since, in this system all the system parameters are constant the Jacobian matrix need to be calculated only once. This way, equation 6.3, that is reconstruction matrix can be calculated once and can be used for computation for every new difference voltage measurement.

Zynq SoC Implementation

Using all the system specifications, the reconstruction matrix is calculated in C# and stored in a text file. The text file is stored in a SD card. On initialization of PS, the reconstruction matrix from SD card is read and copied to the DDR memory on board. The Gauss Newton One Step implementation requires equation 6.2 to be implemented. This requires the difference voltage data. The homogeneous and inhomogeneous data is acquired, and the difference voltage is calculated.

The twelve electrodes and twelve rings give 576 triangular elements in the 2D FEM model. This makes the reconstruction matrix size as 576x96. Also, with twelve number of electrodes and opposite stimulation method, there are 96 total measurements for one homogeneous or inhomogeneous data acquisition. This makes equation 6.2 as a multiplication between 576x96 matrix and 96x1 voltage difference vector. This can be achieved by implementing the row wise multiplication in FPGA. The voltage difference data is transferred from PS to FPGA and stored in an array. Later each row of reconstruction matrix is transferred from DDR to FPGA for a parallel multiplication and accumulation operation. The result is transferred to DDR for storage. Moreover, the Zynq SoC FPGA has 220 DSP slices, if one row multiply and accumulate computation is implemented in FPGA to be executed parallelly, it takes 96 DSP slices. Hence, two rows multiply and accumulate operation is achieved using 192 DSP slices. Two rows are transferred to FPGA at one time from DDR, the computation is performed, and result is stored in DDR. After the computation completes for all the rows, the resulting image data of size 576x1 is transferred to GUI over serial port for display. The logic implementation is detailed in chapter 8.

Software Implementation

The Gauss Newton One Step equation, equation 5.2 is also implemented in C#. As mentioned above, user can set the Zynq SoC for only data acquisition. This way, once homogeneous and inhomogeneous measurements are done, the voltage difference data is transferred to GUI. The precalculated reconstruction matrix is read from a text file and used to compute equation 5.2. The resulting 576x1 data is used to display the conductivity change image on the GUI. The implementation logic is detailed in chapter 9.

6.2.2 Total Variation

As explained in chapter 2, the total variation is an iterative method and hence is performed in following steps.

Gauss Newton one step is to be used to find initial solution

$$\Delta\sigma_0 = (\boldsymbol{J}^T W \boldsymbol{J} + R^T R)^{-1} \boldsymbol{J}^T W \Delta V \tag{6.4}$$

For further iterations, a TV regularization parameter is to be calculated as follows

$$\mathrm{W}_B = \frac{0.5}{\sqrt{(R * \Delta\sigma_0)^2 + \beta}} \tag{6.5}$$

where, β is smoothness parameter and kept constant throughout iterations

This parameter is to be incorporated in equation 6.4 to get,

$$\Delta\sigma = (\boldsymbol{J}^T W \boldsymbol{J} + R^T\, W_B R)^{-1} \boldsymbol{J}^T W \Delta V \tag{6.6}$$

The calculations in equation 6.5 and 6.6 are repeated for few iterations to get the result.

To implement TV algorithm, the previously implemented Gauss Newton One Step computation is used to calculate initial solution that is equation 6.4. The TV regularization parameter is calculated using initial solution, Tikhonov prior R and, $\beta = 0.0004$.

Next, to compute equation 6.6, the Jacobian and variance matrices need to be used. As mentioned earlier, the Jacobian can be calculated only once. Also the variance matrix is identity matrix, hence, the numerator in equation 6.6 that is $(J^T * W)$ matrix and part of the denominator that is $(J^T * W * J)$ matrix are precalculated and used for computation for every new difference voltage measurement.

Zynq SoC Implementation

The matrices $(J^T * W)$ and $(J^T * W * J)$ are calculated in C# and stored in text files. These text files are then stored in SD card. At the initialization of processor, it reads the above said files and stores the data into onboard DDR. If Total Variation algorithm is selected, the initial solution is calculated using the reconstruction matrix stored in DDR and FPGA implementation for Gauss Newton One Step, as described in previous section. Next, for the calculation for TV regularization parameter, that is equation 6.5, the size of prior matrix is 576x576 and the initial solution is 576x1. This computation is implemented in PS.

The TV regularization parameter and $(J^T * W * J)$ matrix is used to calculate the denominator of equation 6.6, in PS. Now, to calculate resulting conductivity change in equation 6.6, the matrix inversion for denominator needs to be done. The inverse is calculated using row elimination method. The calculated inverse is multiplied with $(J^T * W)$ matrix to generate new reconstruction matrix. This new reconstruction matrix and difference voltage data are used to

calculate new conductivity change value, using the Gauss Newton One Step multiplier implemented in FPGA. If the number of iterations is set as more than one, equation 6.5 and 6.6 are repeated to calculate new conductivity change values. Once the computation is completed for all iterations, the resulting image data is transferred to GUI for display. The detailed implementation is given in chapter 8.

Software Implementation

The Total Variation algorithm is also implemented in C#. If the hardware is set to be used for only data acquisition and TV algorithm is selected, the received voltage difference data is used to compute the conductivity change by using the C# TV implementation. Again, the precalculated matrices $(J^T * W)$ and $(J^T * W * J)$ are read from text files and used for computation. To compute initial solution that is equation 6.4, the C# Gauss Newton implementation is used. The computation for equation 6.5 and 6.6 is implemented through matrix operations and repeated for set number of iterations. The resulting 576x1 conductivity change matrix is used to display the image on GUI. The implementation logic is detailed in chapter 9.

Chapter 7: Analog Front End Design Study and Modifications

This chapter gives the design analysis of the analog front-end hardware and the description of the changes implemented in the AFE hardware, in this project. As mentioned in chapter 3, the AFE board is developed by a senior design team [9] to work with higher frequency range 100Hz to 10kHz. In this project, the AFE is modified to work with low frequency range 1Hz to 30Hz.

7.1 Design Analysis

The AFE board design and its functionality are studied to use it in this project. The brief design description of the analog front-end board is given in chapter 3. The AFE is designed to generate the current injection signal and acquire the voltage measurement data. It involves three main sections, transmitter, switching circuitry and receiver.

7.1.1 Transmitter Section

Figure 7.1 shows the functional blocks in the transmitter section.

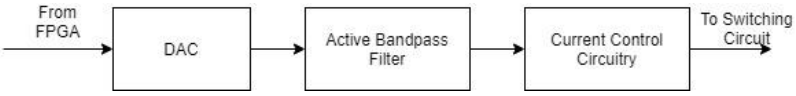


Figure 7. 1 AFE Transmitter Section

The FPGA generates digitized data for selected sinusoidal signal frequency. The digitized data is given to a digital to analog converter, LTC 1668. The Linear Technology LTC1668 IC is a 16-bit 50Msps digital to analog converter. The DAC takes 16-bit input data and gives

differential current output. The differential current output is converted to single ended voltage signal using a differential current to voltage circuitry. There are some corrections done on the board for this circuit which are not there in the available schematic. Figure 7.2 shows the updated schematic for the section.

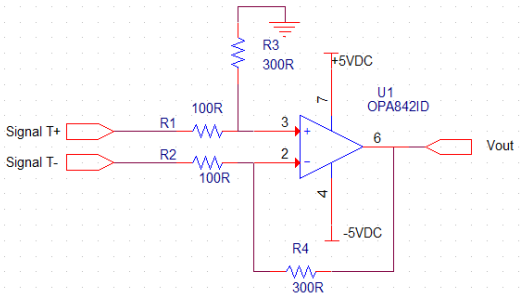


Figure 7. 2 Differential to Single Ended DAC out

Signal T+ and Signal T- are the output coming from DAC. The single ended voltage output of above circuit is passed through a band limiting active filter. The active bandpass filter is designed for 100Hz to 100kHz frequency band. The filtered signal is passed through a current control circuitry as shown in figure 7.3.

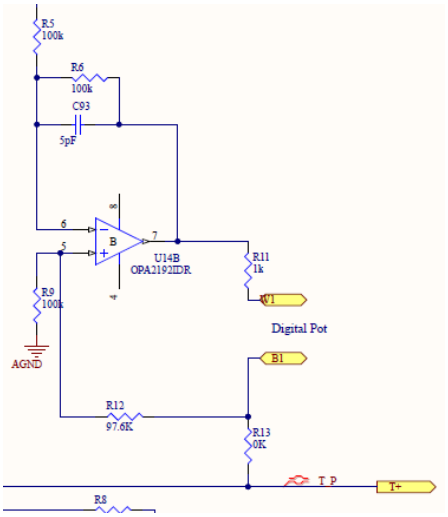


Figure 7. 3 Current limiting circuit

There is a digital pot connected between the feedback loop of Opamp U14. The digital potentiometer AD5293 is a 20kohm potentiometer with 1024 programmable steps. It can be controlled through SPI interface. The output current can be calculated as

$$I_{rms} = V_{rms}/(DigitalPot + R11) \quad (7.1)$$

Where, V_{rms} is the input voltage and I_{rms} is the output current.

7.1.2 Switching circuit

The switching circuit is designed in such a way that the current signal can be connected to an opposite pair of electrodes in a pattern and at the same time voltage measurement from adjacent electrode pairs can be taken. To achieve this a ADG 1606 multiplexer IC is used. It is a 16:1 multiplexer, which can be controlled by 4 selector switch inputs. There are twelve electrodes. There are twelve multiplexers and all of them are connected to the twelve electrodes in a pattern. Two of the multiplexers are connected to the differential current input coming from DAC, and other ten multiplexers are connected to the receiver circuit. This way, using the four select signals of the multiplexers, each multiplexer gets connected to one electrode each time. The select signals are controlled from FPGA to control the current application and voltage measurement pattern. The voltage is measured using a pair of adjacent electrodes. Hence, there are eight voltage measurements generated using ten electrodes. These eight measurements are connected to an eight channel ADC for converting the analog signal into digital and pass it to FPGA for further processing.

Figure 7.4 shows the electrode sequence connected to a phantom.

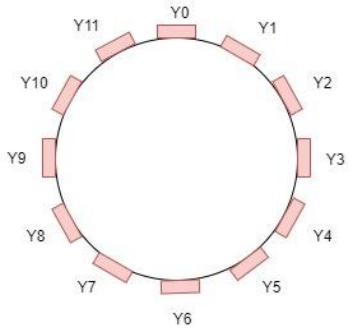


Figure 7. 4 Electrode Connection with Phantom

Table 7.1 shows the sequence of current injection and voltage measurement with respect to switch position, electrode pair and its corresponding ADC channel

Table 7. 1 Electrode stimulation and measurement pattern

Switch Position	***Current Injection Electrode pair***	***Voltage Measured Electrode pair***	***ADC Channel***
0000 (0)	Y0-Y6	Y1-Y2	AN0
		Y2-Y3	AN1
		Y3-Y4	AN2
		Y4-Y5	AN3
		Y7-Y8	AN4
		Y8-Y9	AN5
		Y9-Y10	AN6
		Y10-Y11	AN7
0001(1)	Y1-Y7	Y2-Y3	AN0
		Y3-Y4	AN1
		Y4-Y5	AN2
		Y5-Y6	AN3
		Y8-Y9	AN4
		Y9-Y10	AN5
		Y10-Y11	AN6
		Y11-Y0	AN7
0010(2)	Y2-Y8	Y3-Y4	AN0
		Y4-Y5	AN1

Table 7.1 Continued

Switch Position	*Current Injection Electrode pair*	*Voltage Measured Electrode pair*	*ADC Channel*
0010(2)	Y2-Y8	Y5-Y6	AN2
		Y6-Y7	AN3
		Y9-Y10	AN4
		Y10-Y11	AN5
		Y11-Y0	AN6
		Y0-Y1	AN7
0011(3)	Y3-Y9	Y4-Y5	AN0
		Y5-Y6	AN1
		Y6-Y7	AN2
		Y7-Y8	AN3
		Y10-Y11	AN4
		Y11-Y0	AN5
		Y0-Y1	AN6
		Y1-Y2	AN7
0100(4)	Y4-Y10	Y5-Y6	AN0
		Y6-Y7	AN1
		Y7-Y8	AN2
		Y8-Y9	AN3
		Y11-Y0	AN4
		Y0-Y1	AN5
		Y1-Y2	AN6
		Y2-Y3	AN7
0101(5)	Y5-Y11	Y6-Y7	AN0
		Y7-Y8	AN1
		Y8-Y9	AN2
		Y9-Y10	AN3
		Y0-Y1	AN4
		Y1-Y2	AN5
		Y2-Y3	AN6
		Y3-Y4	AN7
0110(6)	Y6-Y0	Y7-Y8	AN0
		Y8-Y9	AN1
		Y9-Y10	AN2
		Y10-Y11	AN3
		Y1-Y2	AN4
		Y2-Y3	AN5
		Y3-Y4	AN6
		Y4-Y5	AN7

Table 7.1 Continued

Switch Position	*Current Injection Electrode pair*	*Voltage Measured Electrode pair*	*ADC Channel*
0111(7)	Y7-Y1	Y8-Y9	AN0
		Y9-Y10	AN1
		Y10-Y11	AN2
		Y11-Y0	AN3
		Y2-Y3	AN4
		Y3-Y4	AN5
		Y4-Y5	AN6
		Y5-Y6	AN7
1000(8)	Y8-Y2	Y9-Y10	AN0
		Y10-Y11	AN1
		Y11-Y0	AN2
		Y0-Y1	AN3
		Y3-Y4	AN4
		Y4-Y5	AN5
		Y5-Y6	AN6
		Y6-Y7	AN7
1001(9)	Y9-Y3	Y10-Y11	AN0
		Y11-Y0	AN1
		Y0-Y1	AN2
		Y1-Y2	AN3
		Y4-Y5	AN4
		Y5-Y6	AN5
		Y6-Y7	AN6
		Y7-Y8	AN7
1010(10)	Y10-Y4	Y11-Y0	AN0
		Y0-Y1	AN1
		Y1-Y2	AN2
		Y2-Y3	AN3
		Y5-Y6	AN4
		Y6-Y7	AN5
		Y7-Y8	AN6
		Y8-Y9	AN7
1011(11)	Y11-Y5	Y0-Y1	AN0
		Y1-Y2	AN1
		Y2-Y3	AN2
		Y3-Y4	AN3
		Y6-Y7	AN4
		Y7-Y8	AN5
		Y8-Y9	AN6
		Y9-Y10	AN7

Receiver Section

The receiver section functional blocks are shown in figure 7.5

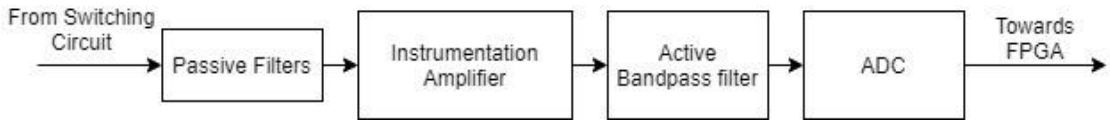


Figure 7. 5 AFE Receiver Section

The signal from each electrode is passed through a passive bandpass filter. The pass band for passive filter is 144Hz to 320kHz. The voltage measurement for EIT is performed using adjacent electrode pairs. Hence, the filtered signal from adjacent electrodes are then passed through an instrumentation amplifier, AD8429, to generate one measurement output. There are ten such instrumentation amplifiers used to generate ten single ended voltage measurements. The output of instrumentation amplifier is passed through an active bandpass filter with a frequency band of 72Hz to 398kHz. Out of the ten single ended voltage signals only eight are required for the EIT computation. Hence, the eight single ended voltage measurements are connected to an eight channel ADC, AD7761. This ADC converts 8 channel analog data to digital and provides it at 8 serial outputs in 16bit format. The output of ADC is passed to the FPGA for further processing. Table 1 shows the electrode pairs set to form eight ADC inputs.

7.2 Modifications to Operate on Low Frequency

The tissue impedance mainly includes extracellular impedance, cell membrane impedance and intracellular impedance [44]. The intracellular and extracellular parts of tissue

pass higher frequencies same way, however, at low frequencies the cell membrane provide high capacitance [45]. Moreover, at the time of injuries, fluid level increases in intracellular part of the tissue. Hence, using lower frequencies for EIT, conditions like tissue edema can be detected.

The AFE hardware is originally designed to work with frequency range 100Hz to 10kHz. To create an EIT set up for low frequency EIT study, the AFE needed to be modified to work with lower frequencies. From the above analysis, it can be said that the active and passive bandpass filters needed to be modified for a lower frequency range. There were already some modifications done on the AFE board by senior design team, however the modifications were not documented because of which the capacitor values in the filter designs were unknown. To analyze the changes, the receiver section was tested for 10Hz input frequency. It was observed that the passive filter was attenuating the input signal by a factor of 10. The output of passive filter is given to the instrumentation amplifier which has a gain setting of seven. The signal was observed to be amplified by the factor of seven at the output of instrumentation amplifier. This signal is passed to active bandpass filter. It was observed that the resistor combination of bandpass filter was again designed to introduce the attenuation of a factor of ten. Hence, the signal was very low at the output of active bandpass filter.

Figure 7.6 shows receiver circuit for channel 0, showing passive filters and active filters. The component values in the following schematic are that of for high frequency filter design, as described in previous section

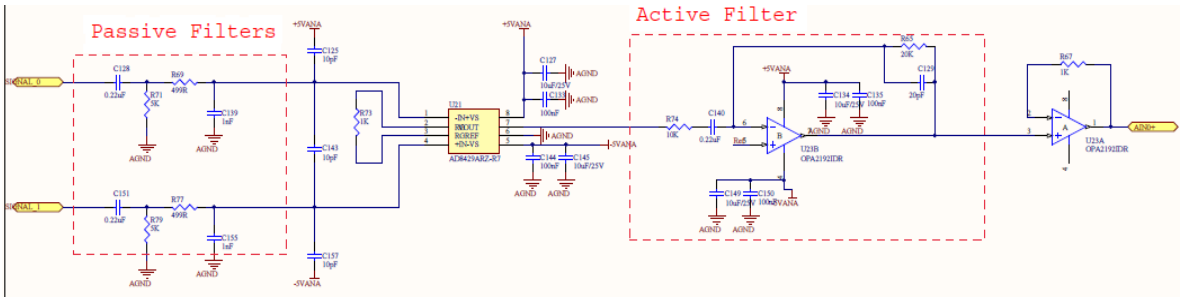


Figure 7. 6 AFE receiver channel for ADC channel AN0

To identify the issues in the filter design, the capacitors on the board in passive and active filter designs were removed and the values were measured using a Fluke meter. With the known values of capacitors, the passive and active filter designs were simulated. The Cadence Orcad PSpice software was used for simulations. Following are the simulation results for passive filter and active filter designs.

- *Passive Filter*

The measured values for resistors and capacitors in the passive circuit are as follows

C3 = 0.22uf R4 = 174kohm

C4 = 2.2uf R3 = 4kohm

After calculation using the R-C values of passive filter, the passband for the design comes as *4Hz to 18Hz.*

Figure 7.7 shows the schematic drawn for simulation

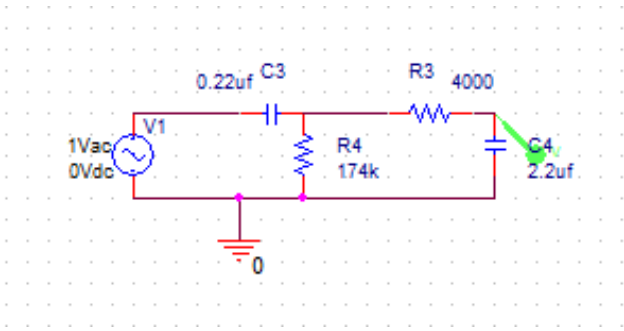


Figure 7. 7 Passive filter schematic for simulation

The filter design was simulated using a frequency sweep simulation from 0.01Hz to 100kHz. Figure 7.8 shows the frequency response achieved

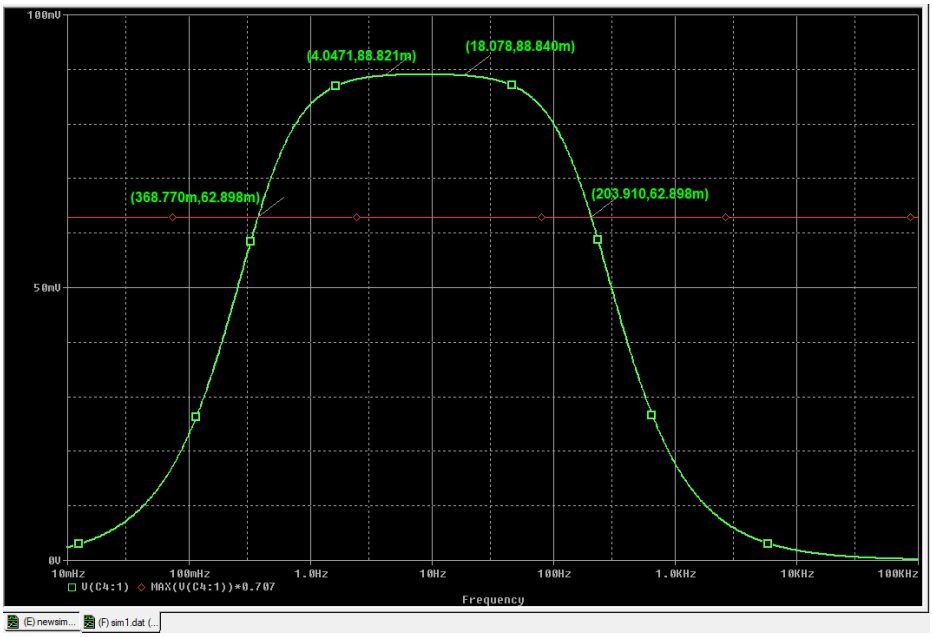


Figure 7. 8 Passive filter frequency response

The frequency response shows the calculated values in the passband. At -3db the frequencies are 0.3Hz to 203.9 Hz. It is also observed that the input used for simulation is 1V and the attenuation of 10 causes the passband amplitude to be less than 100mV.

- *Active Filter*

The measured values for resistors and capacitors in the active bandpass filter circuit are as follows

C1 = 2.2uf R1 = 174kohm

C2 = 0.1nf R2 = 20kohm

After calculation using the measured R-C values, the passband for active bandpass filter is 0.4Hz to 80kHz, with gain 0.1. Figure 7.9 shows the schematic drawn for simulation. The bandpass filter output is level shifted by 2.5V, hence, the schematic has the 2.5VDC input at noninverting input terminal of the Op-amp.

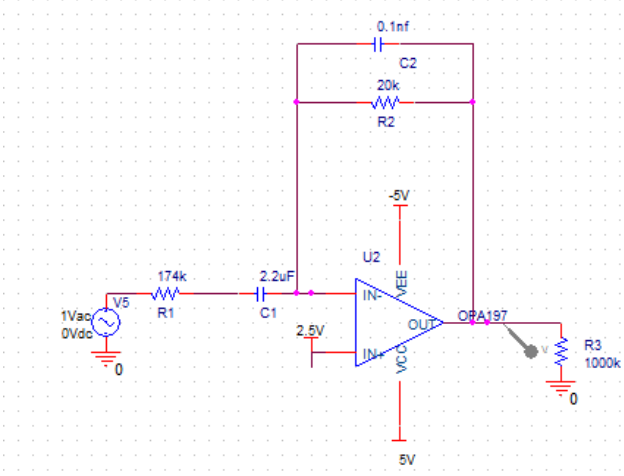


Figure 7. 9 Active Bandpass filter schematic for simulation

The filter design was simulated using a frequency sweep simulation from 0.001Hz to 10MHz. Figure 7.10 shows the frequency response achieved. The simulation shows the -3dB frequencies as 0.4Hz and 80kHz. Also, the amplitude shows the attenuation of a factor of 10.

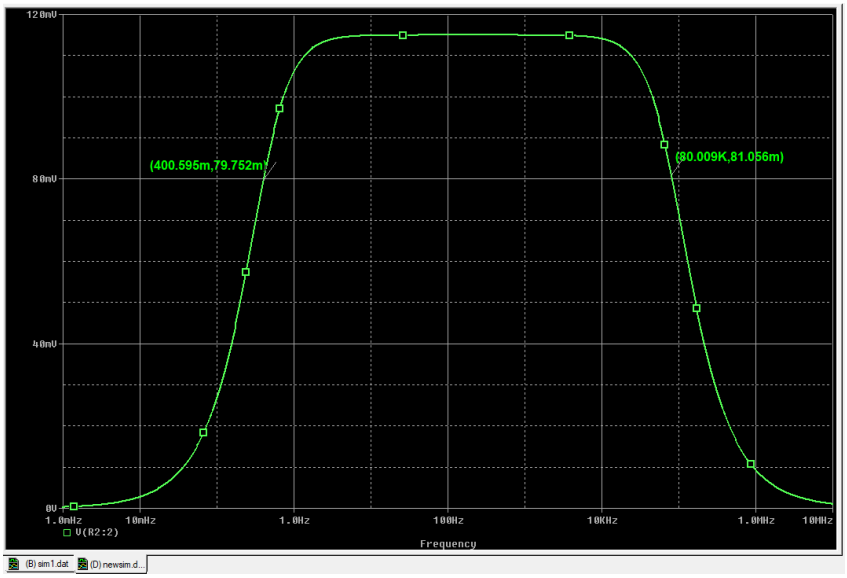


Figure 7. 10 Frequency response for active bandpass filter

- *Modifications*

To remove the attenuation at input stage that is passive filter stage, it was decided to bypass the high pass filter and keep the lowpass filter. This way, the low pass filter cutoff frequency would be 18Hz with no attenuation. Figure 7.11 shows the updated schematic for passive filter, drawn for simulation

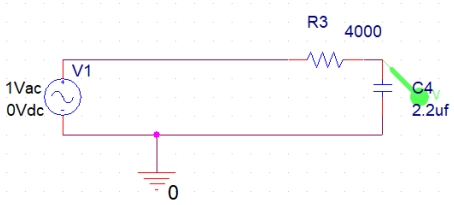


Figure 7. 11 Modified schematic for Passive filter

Figure 7.12 shows the simulation waveform for the passive lowpass filter.

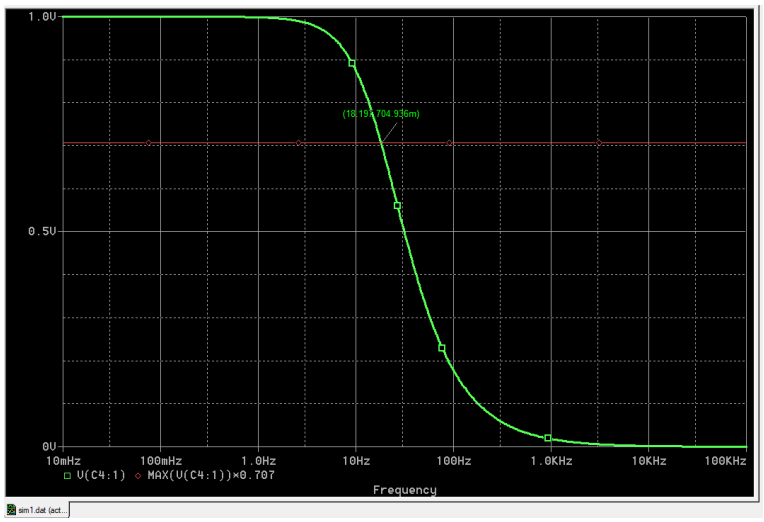


Figure 7. 12 Passive lowpass filter frequency response

The active bandpass filter was also modified to remove the attenuation and add the gain of two by changing the value of feedback resistor from 20kHz to 300kHz. This makes new active bandpass filter frequency range as 0.4Hz to 5.3kHz and gain of 1.72.

Figure 7.13 shows the schematic for the active bandpass filter

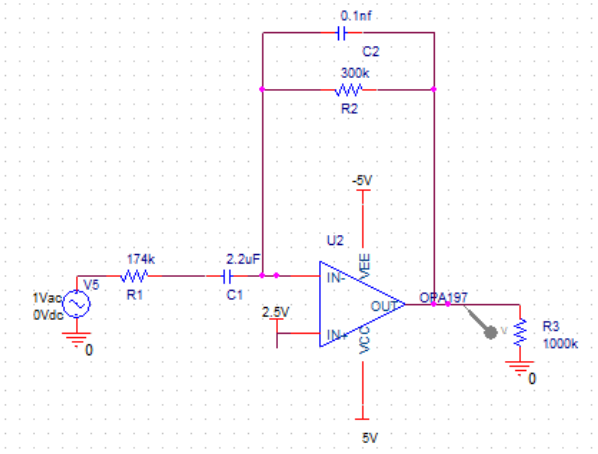


Figure 7. 13 Modified schematic for the active bandpass filter

Figure 7.14 shows the frequency response for above shown active bandpass filter. As can be observed the -3dB frequency pass band is form 0.4Hz to 5.3kHz. Also, the gain of 1.7 can be observed.

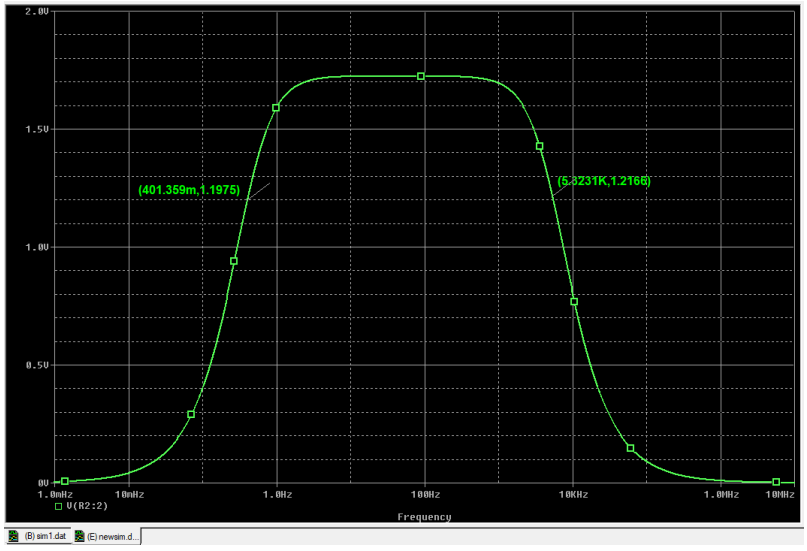


Figure 7. 14 Frequency response for modified active bandpass filter

The above described changes were implemented on AFE board and further testing was done. There is an active bandpass filter in the transmitter section of the AFE design. The changes done in the active bandpass filter of receiver section are replicated in the active bandpass of the transmitter section.

The updated AFE hardware was tested by connecting a phantom between the electrodes and signals were observed at multiple points in the design. The current injection signal was used as 10Hz. Table 7.2 shows the signal measurement at different points taken for channel 0. All the measurements are peak to peak.

Table 7. 2 Measurement for Channel 0

T+	**Signal_0**	**Passive Filter out (U21.1)**	**Signal_1**	**Passive Filter out (U21.4)**	**Instrumentation amplifier out (U21.7)**	**Active filter out (U23.7)**
1.22V	248mV	224mV	80mV	72mV	1.2V	1.96V

In this table, T+ is the transmitter output going to phantom. Signal_0 and Signal_1 are the electrode pair signals which will generate ADC channel 0 input. The instrumentation amplifier output and the active filter output shows the collective gain of around 7*1.72 = 12.

All the receiver channels were tested for the modified receiver design, on oscilloscope as well as the digitized data on serial port. Different frequencies from 2Hz to 30Hz were used for testing.

Chapter 8: FPGA Design Development

This chapter provides a detailed description for the FPGA design. As described in chapter 6, the data acquisition system includes a Zynq SoC based development board, Zedboard along with the AFE board. The Zynq SoC performs the signal generation, data acquisition and data processing tasks. The Zynq SoC constitutes of FPGA and processor. In this project, the processor and FPGA both are used to develop a complete EIT control system. This chapter covers the FPGA design as well as processor application developed in this project.

8.1 Zynq SoC Design Description

The Zynq SoC PS and PL can be programmed independently. However, a combination of PS and PL can be used to create more efficient designs. In this project, the Zynq SoC is used to control the flow of operation in the EIT system, generate the drive signals and acquire the measured data. The FPGA logic constitutes of the hardware for following functionalities

- Generate the digital sinusoidal signal
- Program the digital potentiometer through SPI
- Acquire the eight channel ADC data
- Control the sampling frequency of the acquisition
- Control the Number of samples of the acquired data
- DMA function for data read/write from PS DDR
- A Parallel multiplier for reconstruction computation

These functions are monitored and controlled by PS. The PS performs following functionalities

- Load the reconstruction matrices from SD card to PS DDR

- Communicate with GUI over serial port for control data such as operating frequency, number of samples, sampling frequency etc.
- Program the hardware in FPGA according to the control data
- Program the ADC through SPI communication
- Control the switching system
- Start the acquisition according to the commands received from GUI
- Read the ADC data stored in FPGA
- Process the ADC data through DFT
- Initialize the DMA transfer for reconstruction computation
- Perform part of the Total variation algorithm computation
- Transfer the image data or measurement data to the GUI

Xilinx provided VIVADO design suite 2018.3 is used to develop the Zynq SoC design. For generating the FPGA logic hardware, VIVADO IDE is used. For developing processor application, Xilinx SDK is used. Implementation for each of the above-mentioned PS and PL functionalities are described below. First a VIVADO design which includes the FPGA hardware design is described. Later, SDK code that is the processor code is described.

8.2 VIVADO Design

VIVADO IDE is used to generate the hardware platform including an FPGA design and an embedded processor. VIVADO IP Integrator is used to generate a block diagram for the required functionalities using processor IP, some of the Xilinx provided IPs such as DMA, FIFOs etc. and some of the custom designed IP cores. A custom logic for the required

functionalities is written in Verilog and converted to an IP core using Xilinx IP packager. Figure 8.1 shows the functional blocks in the VIVADO design. Most of the IP cores can be controlled by the processor through AXI lite ports.

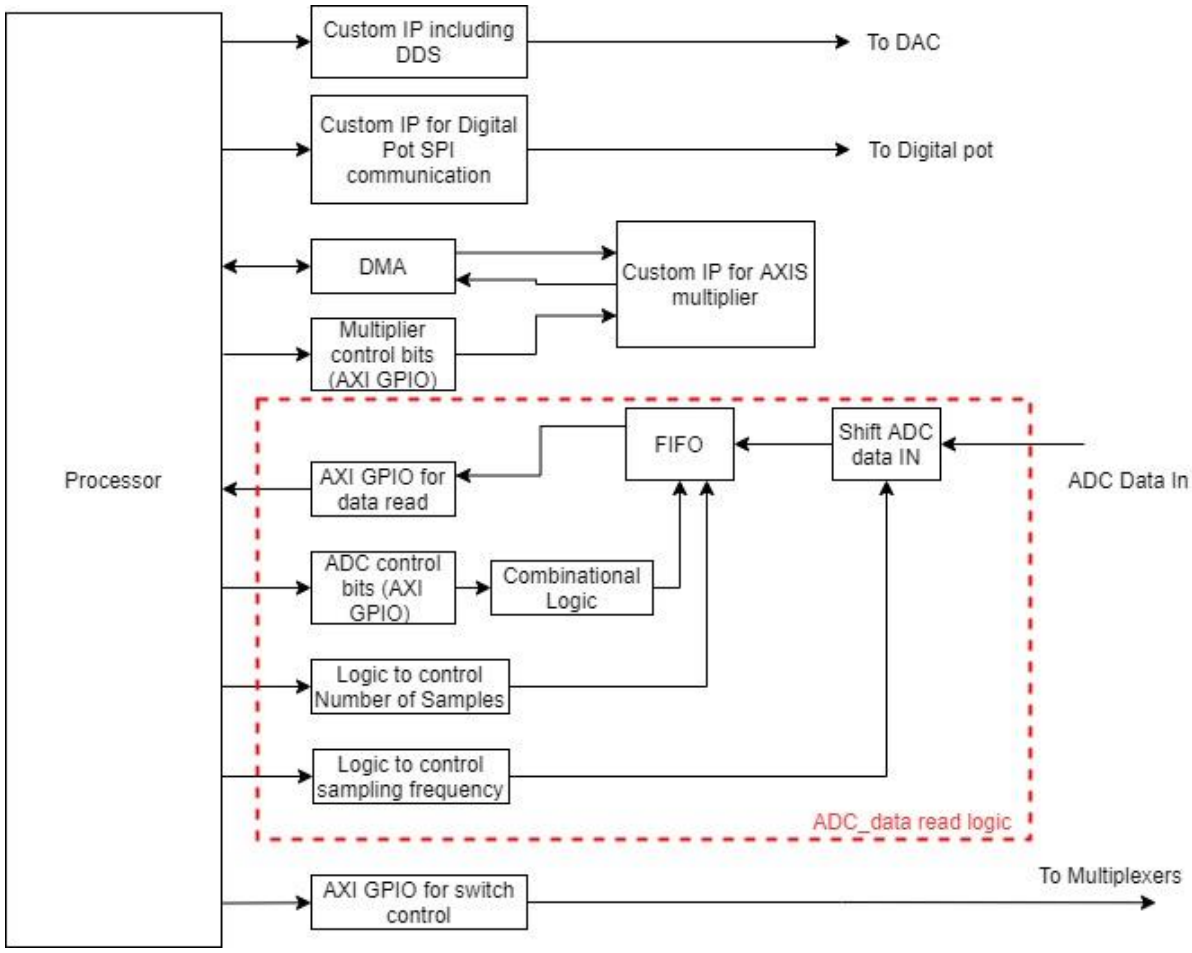


Figure 8. 1 VIVADO Functional block diagram

The complete VIVADO design can be divided into six parts, processor, signal generation, Digital pot controller, Data Acquisition, AXIS multiplier and switching system. The processor is the Zynq SoC ARM hardware processor. It is programmed using SDK and the design details are

given in next section of this chapter. Other parts' design and development is described in this section. Figure 8.2 shows the complete VIVADO block diagram built for this project.

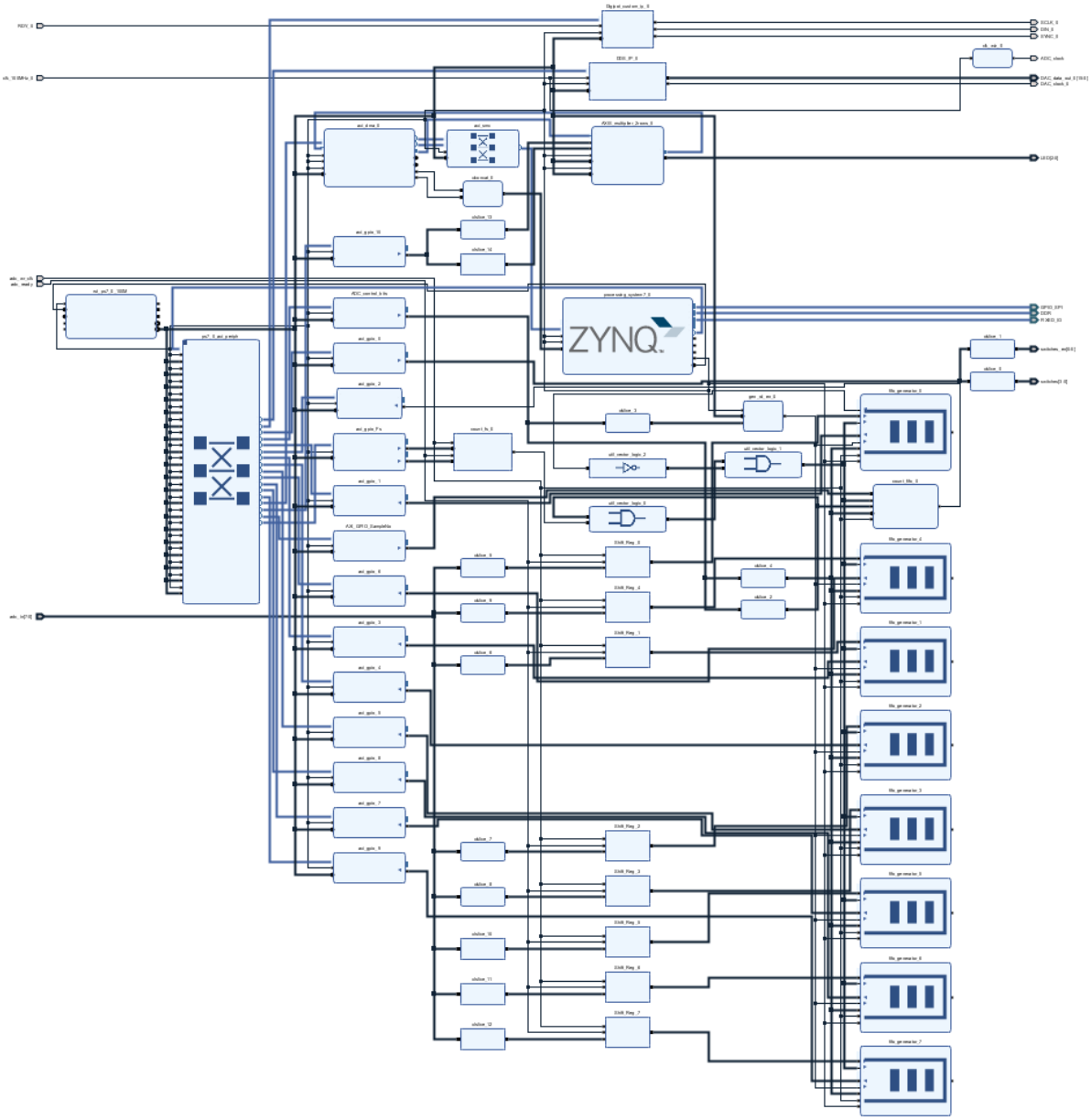


Figure 8. 2 Complete VIVADO Block Diagram

8.2.1 Processor

The Zynq 7 processing system is added to the block diagram as an IP core. Using the Run Block Automation option, the Zedboard settings are applied. There are few design specific options which needs to be enabled in the processor IP configuration. On the Clock Configuration page, under PL Fabric Clocks, FCLK_CLK0 is selected by default. The PL clock frequency is confirmed to be set at 100MHz. On the PS-PL Configuration page, under the HP Slave AXI Interface, S AXI HP0 Interface is selected. The HP interface is selected for DMA connections. The fabric interrupts are also enabled for further use in DMA transactions. To enable the fabric interrupts, on the interrupts page, under Fabric Interrupts, IRQ_F2P is selected. The EMIO GPIO is used for ADC SPI connections. A five-bit EMIO GPIO is enable from the page MIO Configurations. Figure 8.3 shows the processor IP after the configuration updates.

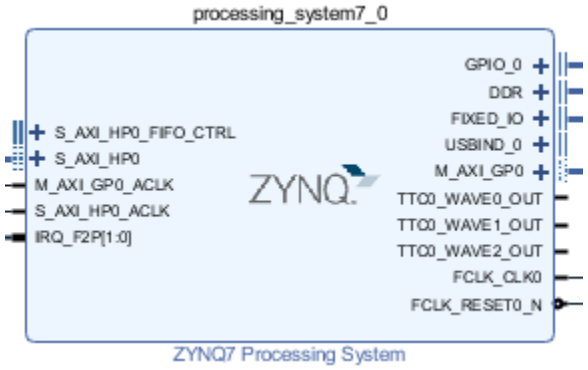


Figure 8. 3 Processor IP

8.2.2 Signal Generator

In this project a sinusoidal signal from frequency 1Hz to 10kHz is required to be generated. The Analog Front End board has a 16-bit DAC for analog signal transmission to the

electrodes. The DAC needs a 16-bit digital data clocked at 50MHz, along with the 50MHz clock. A custom AXI-lite Slave IP core is created to provide a 16-bit digital data and 50MHz clock for the DAC. The procedure to create and use custom IP using IP packager is given in [46]. Figure 8.4 shows the DAC data generation IP Core block, named as DDS_IP

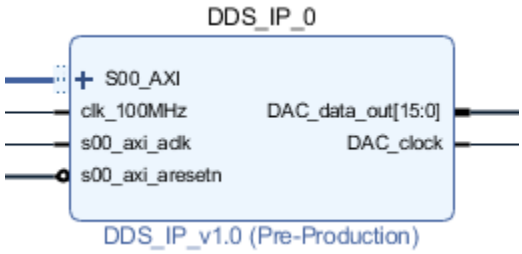


Figure 8. 4 DAC data generation IP core

It has an AXI Lite slave port to take the settings input from PS. The Zedboard has 100MHz onboard clock, it is used to generate a 50MHz clock required for DAC. The IP core gives 16-bit data output and 50MHz clock output. This IP includes a DDS IP core, a Clocking wizard IP core and an ODDR (output DDR primitive) for clock forwarding to output pin.

A separate design including the DDS, Clocking wizard and ODDR is created, simulated, and then incorporated with a custom AXI-lite Slave IP core. The phase data setting for DDS IP core is received through the AXI-Lite port from PS. Following is the DDS IP and Clocking wizard IP setting description. The simulated waveform is also given.

a) DDS IP Core:

The Xilinx LogiCORE IP DDS (Direct Digital Synthesizer) complier has the phase to sinusoidal waveform data generator logic. With the appropriate phase setting, the core generates

the sinusoidal waveform samples with a specific frequency [47]. An analog signal can be generated by taking the digital data generated by DDS and given to the DAC.

The DDS IP core needs a phase increment value to be set up to convert it to sinusoidal waveform. With the appropriate phase increment value, the output frequency is given by following equation

$$fout = fclk * \frac{\Delta\theta}{2^B} \tag{8.1}$$

where, $fout$ is the output frequency,

$fclk$ is the system clock frequency,

Δθ is the phase increment value and

B is the phase width that is number of bits used to set phase increment value

The frequency resolution that can be achieved is given as

$$\Delta f = \frac{fclk}{2^B} \tag{8.2}$$

In this project, the DAC used has 50MHz update rate. Hence, system clock is set as 50MHz. The frequency output required is 1Hz to 10kHz, hence, to achieve the frequency resolution less than 1Hz, the phase width is set as 32 bits

Hence, Resolution

$$\Delta f = 50MHz/2^{32} = 0.011Hz \tag{8.3}$$

To setup the required output frequency for the DDS, the phase increment value needs to be entered. The phase increment value can be calculated as follows

$$\Delta\theta = fout * \frac{2^B}{fclk} \tag{8.4}$$

For example, for 100Hz sinusoidal output waveform, the phase increment value to be entered is

$$\Delta\theta = 100Hz * \frac{2^{32}}{50MHz} = 8590 \quad (8.5)$$

The DDS compiler IP is set with following settings in graphical user interface

- The Configuration option is set *as Phase Generator and SIN/COS LUT*
- System Clock: 50MHz
- Number of Channels: 1
- Mode of Operation: Standard
- Parameter Selection: Hardware parameters
- Phase Width: 32 bits (can be set from 3 to 48 bits)
- Output width: 16 bits (can be set from 3 to 16 bits)
- Phase Increment Programmability: Streaming, it allows phase increment value to be taken in from the phase channel
- Phase offset programmability: None
- Output Selection: Sine
- Amplitude Mode: Full range
- Memory Type: Auto
- Optimization Goal: Auto
- DSP48 Use: Minimal
- DATA Has TLAST: Not Required
- No Output Ready
- Output form: Twos complement

- Latency Configuration: Auto
- Control Signals: ARESETn

b) Clocking wizard IP Core

The DAC requires the digital data clocked at 50MHz. Hence, a Clocking Wizard is used to convert an on board 100MHz clock to 50MHz. In the Clocking wizard IP settings, PLL primitive is used. Input clock is set as 100 MHz and the output clock, clk_out1 is set as 50MHz. The 50MHz clock is given as input to the DDS IP Core. For connecting the PLL IP core output to the FPGA output pin, an output DDR primitive is used.

Figure 8.5 shows the simulation for 100Hz output. The DDS core output is signed 16-bit. To get the continued sinusoidal data, the polarity needs to be reversed by adding 0x8000 in the 16-bit output data.

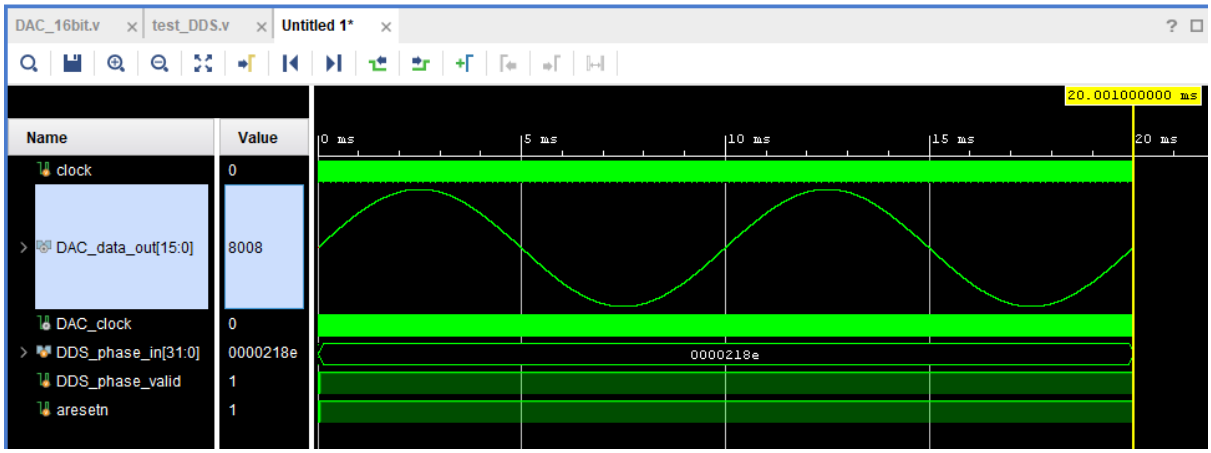


Figure 8. 5 Simulation waveform for 100Hz

c) Integration with AXI-lite custom IP

The above tested design is added to the design with AXI-lite slave interface and packaged as an IP. The DDS IP core inputs are 32-bit phase increment data input and a valid signal. In the AXI lite interface code, the 0^{th} bit of slave register0 is used as valid signal and slave register1 is used for phase data value.

8.2.3 Digital Pot controller

In the transmission section of the AFE board, a digital potentiometer is used to control the current. An AXI-lite custom IP core is generated to take the pot settings from PS and program into the digital pot through the SPI communication. Figure 8.6 shows the IP core block. It has a ready input coming from the potentiometer IC, AXI-lite port for input from PS, and SCLK, DIN and SYNC are the signals to transfer the data serially to the potentiometer.

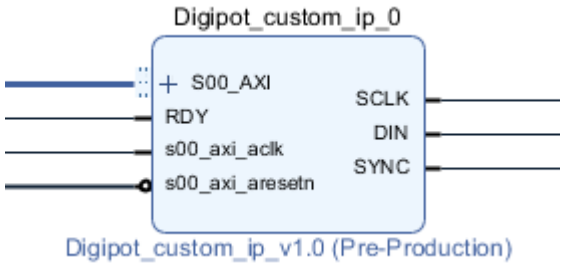


Figure 8. 6 Digital Pot control IP core

Again, a separate logic for digital pot communication is designed and then incorporated in a custom AXI-lite slave IP. The PS writes the digital pot value setting through AXI-lite slave port. Following is the logic description for the IP.

a) Digital potentiometer communication logic

The digital pot used in AFE is a 20kohm, 1024 position resistance. The resistance value can be controlled digitally through a SPI-compatible serial interface. The control includes

- SCLK: Serial data clock, can be set up to 50MHz
- DIN: Serial data input to Digital pot
- SDO: Serial data output from Digital pot
- SYNC: Frame synchronization signal for data input
- Ready: Ready output, identifies the completion of a write or read operation

The pot value can be written or read through this interface. In this project, the pot values are only written through the SPI interface.

The data transfer through the SPI interface is done through a 16-bit shift register. The shift register structure is as follows, bit 15-14 = 0, bit 13-10 = 4 control bits and bit 9 – 0 = 10 data bits.

The control bits control the pot operation and the data bits are written to the internal data register. To set the potentiometer wiper at some specific value, the wiper needs to be unlocked first, write the desired value and lock the wiper again. For detailed protocol please refer [48]. Following are the shift register words used in this project,

- Unlock wiper: 0x1802
- Data value: (0x0400 + wiper setting to be written)
- Lock Wiper: 0x1800

A state machine is developed to write above shown words through the serial interface.

Figure 8.7 shows the write timing diagram [48] for the SPI communication. As can be seen, if Ready signal is high, the Sync signal must be set low for starting to shift out the 16-bit data serially. Upon the shifting out of all 16 bits, the Sync needs to be set high.

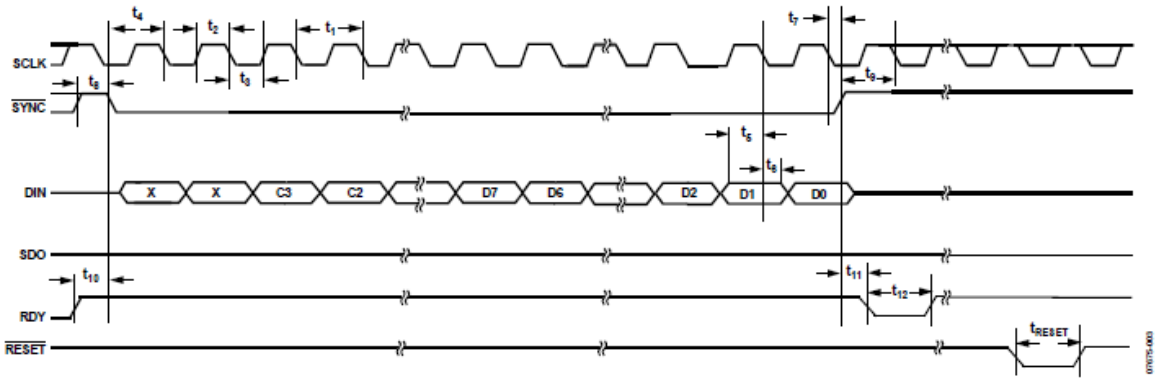


Figure 8. 7 Write Timing Diagram AD5293 *[48]*

As stated above, the SCLK can be set up to 50MHz, in this project the SCLK is set as 6.25MHz. A SYNC signal and Ready signals are used to start the data transfer and poll the completion respectively. To set the clock frequency and wait period, counters are used. All the data transfer logic is set up in five states such as IDLE, SYNC_LOW, LOAD, POLL_READY and WAIT_EN_LOW. As the digital pot setting value is written through PS, there are few handshaking signals used such as enable, done and error. If the enable signal is raised high by PS, the state machine moves from IDLE state to SYNC_LOW state. In SYNC_LOW state, SYNC signal is set low and the data to be transferred is copied in the shift register. The data load sequence is as shown above that is unlock, actual data word and lock data word. In next state that is LOAD, the data is shifted out from the MSB side. After the 16th bit shift, it moves into next state that is POLL_READY. To indicate the transfer has been completed, the digital pot IC

should lower the Ready signal. Hence the Ready signal is polled to be low for around 2us. If it does not go low, the error signal is raised high. In the next state that is WAIT_EN_LOW, the enable signal is polled to be lowered by PS to finish the data transfer. If the ready signal goes low without error, the state moves to SYNC_LOW to load the next data and the next data gets shifted out. At the end that is after transferring the lock word, a done signal is raised high to indicate the transfer is completed. The state machine flowchart for this code is shown in appendix in this document.

b) Integration with the AXI-lite custom IP

The above designed code is integrated with a custom AXI-lite slave IP and packaged as an IP. The digital pot SPI communication code requires 16-bit data for the potentiometer value and enable signal. Also, the code provides two status signals such as DONE and ERROR. For PS to be able to read and write to the digital pot logic, the AXI-lite slave communication is used. As shown in figure 8.8, the 0^{th} slave register is used to write the data and enable signal to the digital pot logic. The out_reg is defined as a 32-bit register which includes the DONE and ERROR signals. The out_reg is then connected to the output register 1of the AXI-lite protocol.

```
// Add user logic here
assign Data_load =  slv_reg0[15:0];         //the digital pot setting
assign Enable = slv_reg0[16];               //enable signal
assign out_reg[0] = Done;                   //done signal
assign out_reg[1] = Error;                  //error signal
assign out_reg[31:2] = 30'b000000000000000000000000000000;
```

Figure 8. 8 AXI-lite register usage for digital pot programming

8.2.4 Data Acquisition

The data acquisition logic is designed for reading and storing serial ADC input for 8 channels and transferring the stored data to PS for processing. It includes ADC interface, shift register logic for converting serial input to parallel, logic to control sampling frequency, logic to control number of samples, a combinational logic to control FIFO data read/write and AXI GPIOs to transfer FIFO data to PS. In this section, all sections of the data acquisition logic are described.

a) ADC Interface

The ADC used on AFE board is an 8 channel, 16-bit ADC. There is a SPI control interface for programming the sampling speed and power mode. The digital data is provided through 8 serial data output pins along with the clock and ready signal. In this project, the SPI programming is handled in PS. In PL the serial data input is converted to parallel and stored.

The ADC requires a master clock MCLK of 32.768MHz. This clock is provided through FPGA. The Zedboard has 100MHz input clock, it is used to generate 32.768MHz clock for ADC through a clocking wizard.

The ADC conversion data is clocked out on an output clock from ADC, that is DCLK. This clock is set by setting the ADC power mode. There are three modes, such as fast, median and low power mode. In this project, the median mode is set through SPI. This makes DCLK = MCLK/8 = 4.096MHz. There is a framing signal DRDY (Data Ready) which indicates the conversion data is ready. In the FPGA logic, DCLK and DRDY signals are used and serial ADC conversion data is shifted in.

Figure 8.9 shows the data output timing diagram for ADC. The ADC data word here is 24 bits, out of which first 8 bits are header which includes information like channel number and status. The other 16-bit is ADC output.

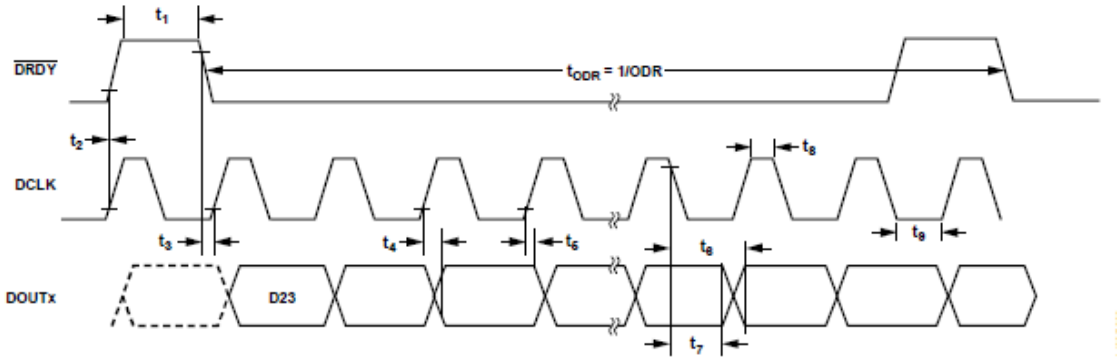


Figure 8. 9 ADC interface timing diagram*[49]*

b) Shift Register

There are eight ADC inputs which provide conversion data serially. The serial input needs to be parallelized to store in the memory. Hence, a custom logic is required. In this project, the shift register logic developed by Senior design team [9] is used as it is. In this section, the logic is described.

The eight inputs are connected as an 8-bit bus to the FPGA pins and separated using *Slice* IP core. Figure 8.10 shows the sliced serial input and the Shift register IP core

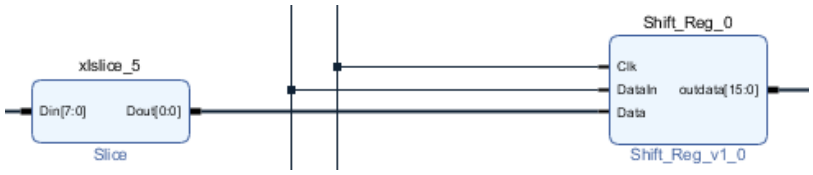


Figure 8. 10 ADC channel and shift register IP block

As described in previous section, the ADC data is clocked out on DCLK of ADC. The DCLK is also provided out for reading the digitized ADC data. Hence, the DCLK is used in the FPGA logic to shift in the serial data. In the above figure, the shift register IP has clock input, which is DCLK, DataIn is a Data Ready signal coming from ADC and Data is the serial data input. The output of the shift register is 16-bit ADC data. The logic is developed using Verilog and converted into a custom IP. In the Verilog code the data is clocked out on rising edge of DCLK. Hence, in FPGA, the data is shifted in on negative edge of the clock.

c) Sampling frequency control

In this project, the ADC operation is set at the median power mode. In this mode, the maximum sampling frequency can be set at 128kHz. However, at low frequency signals such as 2Hz to 100Hz, the sampling frequency needs to be lowered. This is achieved by setting the ADC sampling frequency at one fixed rate and while storing the converted data in the FPGA, dividing the sampling rate by skipping number of samples. A custom logic is designed using Verilog and then converted to an IP to integrate in the main design.

In figure 8.11, the count_fs is the IP core which divides the sampling frequency. The ADC Data ready signal is used to generate write enable for the FIFOs to write the acquired data. To divide the actual sampling frequency, the Data ready is forwarded only at the sampling rate which is required. Hence, the count_fs logic controls the forwarding of Data ready signal. The division counter value is loaded from PS, as the required sampling frequency is set through PS. Hence, the count_fs block has the inputs such as clock, which is DCLK, DataIn which is ADC Data ready, div_count, which is 12 bit division counter set from PS and a reset_count signal. The

output of count_fs is OutReady signal which is further used to generate write enable for FIFO. To load the signals coming from PS, AXI GPIO is used. As seen in the figure, two GPIO channels, one bit and 12 bit are set.

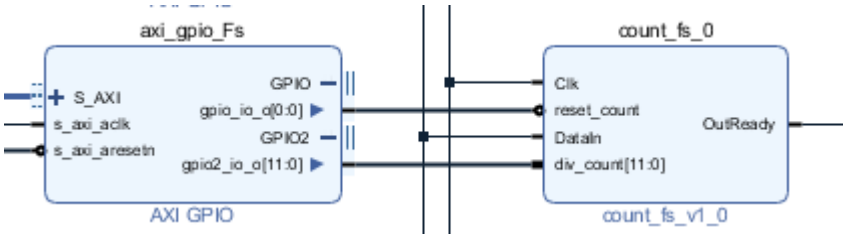


Figure 8. 11 Sampling frequency setting logic blocks

Initial part of the count_fs logic is taken from one of the IP cores designed by senior design team[9]. However, the sampling frequency division logic is developed in this project. Figure 8.12 shows the code snippet for the count_fs logic.

```
if(DataIn)                      //New data is ready
begin
    if(counter ==0)             //This counter is to make sure the data re
    begin
        counter = 1;            //Initialize the counter
        div_counter = div_counter + 1;  //start the division counter
    end
end

if(counter != 0)            //if counter was initialized, increment the co
begin
    counter = counter +1;
end

if((counter == 26) && (div_counter == div_count))       //if all the 24 bi
begin
    counter = 0;
    OutReady = 1;                       //set the data ready signal high f
    div_counter = 0;
end

else if (counter == 26)             //if all 24 bit data is shifted, reset
begin
counter = 0;
end
```

Figure 8. 12 sampling frequency division logic

The DataIn that is Data ready signal is sampled on the negative edge of the DCLK. Once the data ready signal is high, a counter is started. The division counter is incremented by one at every Data ready detection. Once the Data ready is detected, the ADC data starts shifting in shift_reg IP core, hence, to count if all the 24 bits are shifted in, the counter is incremented at negative edge of the DCLK. If the counter reaches 26 counts, it is set to zero until new data ready is detected. If the counter has reached 26 and division counter, which indicates number of data ready signals, has also reached to the expected count, then a OutReady signal is set high. This makes sure, the Data ready signal only gets forwarded after a specific number of samples. Once the OutReady is set high, all the counters are set to zero.

d) Number of Samples control

According to the signal frequency and sampling frequency, the number of samples which needs to be stored are variable. Depending upon the set number of samples, the ADC data writing to the FIFO is limited. This is achieved by a custom logic which monitors FIFO write enable signals and informs PS that the required number of samples are stored. The PS then disables the FIFO write enable signal. The logic is developed using Verilog and then converted to an IP for further use in main design. Figure 8.13 shows the count_fifo IP core.

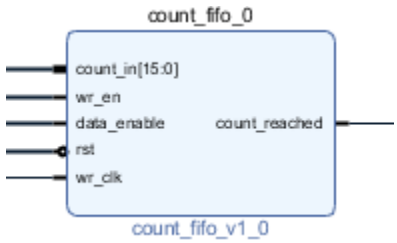


Figure 8. 13 Number of samples control IP block

The IP core has inputs count_in, which is required number of samples setting coming from PS, data_enable, which is enable signal coming from PS, rst, which is the reset signal coming from PS, the wr_en signal, which is the FIFO write enable and the wr_clk which is FIFO write clock. The output is count_reached which is connected to PS through AXI GPIO.

The count_in signal coming from PS is connected through an AXI GPIO named AXI_GPIO_SampleNo.It si configured for 16 bit output.

The control signals, data_enable and rst are connected from PS to count_fifo IP through an AXI GPIO IP core named ADC_control_bits. The AXI GPIO is configured for 4-bit output. Each of these outputs are used for separate function and separated by *Slice* IP cores. The 0^{th} bit of the GPIO output is data_enable and 2^{nd} bit is the reset signal.

In the count_fifo logic, if the data_enable signal is high, the FIFO write enable pulses are counted. If the count reaches to the set number of samples value, the count_reached signal is set high. The PS reads this signal and disables the data_enable signal which also stops writing data in FIFO.

e) FIFO storage and combinational logic

The shift register converts every ADC conversion data from serial to parallel data. The parallel data is stored in a FIFO. Eight 8192 depth, 16-bit FIFOs are used to store the eight-channel data. There is a combinational logic which combine PS control bits and ADC signals to control the data samples writing into FIFOs.

Figure 8.14 shows the FIFO IP core generated using Xilinx provided FIFO generator IP core.

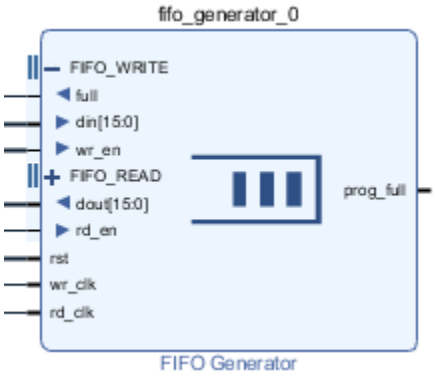


Figure 8. 14 FIFO IP Core

The FIFO IP is configured as native, independent clock built in FIFO. Other settings for the configuration are First word fall through, write width 16 bit, write depth 8192, Read clock frequency 100MHz, write clock frequency 4MHz and single programmable full threshold constant 8185. The programmable full threshold is set at its maximum.

As mentioned above, the FIFO is an independent clock FIFO. The ADC clock is used as write clock as the ADC data is clocked in at ADC clock. The read clock is set as the FPGA clock which is 100MHz. The write enable is generated using a combinational logic. Figure 8.15 shows the combinational logic used to control the write enables to FIFO. An AND gate is used to generate the wr_en signal. One of the inputs to the AND gate is inverted FIFO full signal. The FIFO full signal sets high only if the FIFO is full, hence, the write enable will automatically stop if the FIFO becomes full. As there are eight FIFOs and all the FIFOs have same sequence of write and read, the FIFO full signal used here belongs to the FIFO used to store ADC channel 1 data. Other input to the wr_en AND gate comes from another AND gate. This AND gate have

one of the inputs as data_enable. This signal is generated by PS and controlled over the required number of samples storage. Another input to this AND gate is OutReady signal which comes from count_fs IP core, shown in section c above. Hence, the OutReady will be used as write enable only if the FIFO is not full and PS has issued data_enable.

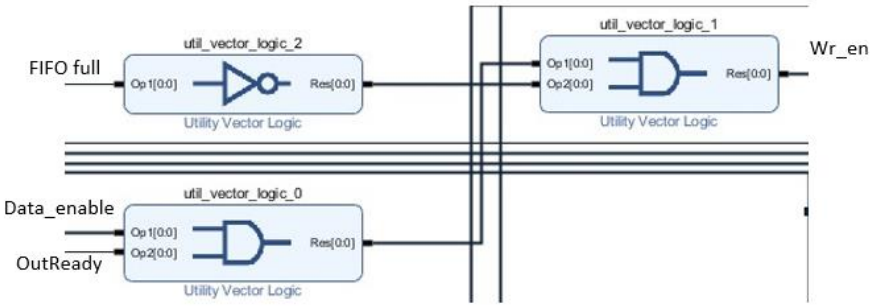


Figure 8. 15 Combinational Logic for FIFO write enable

Once the required number of ADC data samples are written in FIFO, the PS issues read enable signal and reads all the data. However, the read enable signal needs to be of only one clock pulse data width for only one data to come out on the output bus of FIFO. If the read signal is set high from PS and then set low for one data read, there is a possibility that due to the processor execution time the width of the read enable signal is more than one clock pulse. Hence, a custom logic is generated in Verilog and then converted to IP, to make sure the width of the read enable signal. Figure 8.16 shows the IP core to set the read enable for FIFO.

Figure 8. 16 FIFO Read enable signal IP core

The IP has the local clock and reset inputs. The gen_rd input is the read enable signal coming from PS. The read enable signal from PS is connected to the logic through ADC_control_bits AXI GPIO. The bit 1 from the ADC_control_bits AXI GPIO is the read enable signal. Figure 8.17 shows the logic used for sampling the PS read enable signal. The PS read enable is sampled on a negative edge of the clock so that the output read enable signal stays stable during the next positive edge of the clock. Once the PS read enable is detected, the output read enable is set high along with a flag. On the next negative clock edge as the flag is set high, the output read enable signal resets.

```
   always  @( negedge clock or negedge reset_n )      //the read enable is
begin
  if (reset_n == 1'b0)
  begin
  rd_en <= 0;
  flag <= 0;
  end
  else
  if((gen_rd == 1) && (flag == 0))        //sample the PS read enable
  begin
  rd_en <= 1;
  flag <= 1;
  end
  else if(gen_rd == 0)                //reset all signals if PS read enable
  begin
  flag <= 0;
  rd_en <= 0;
  end
  else
  rd_en <= 0;                     //reset output read enable if flag is 1
  end
```

Figure 8. 17 generate FIFO read enable logic

f) AXI GPIO Data transfer

The sixteen-bit ADC data stored in FIFOs is read through AXI GPIOs. For each ADC channel, there is one AXI GPIO IP core used, with 16-bit input port. The FIFO output is connected to PS through these AXI GPIOs. For each sample read, the PS issues read enable, corresponding data comes on the FIFO read port and PS reads the data through AXI GPIO.

8.2.5 *AXIS multiplier*

The AXI multiplier is designed to perform the reconstruction matrix and difference voltage vector multiplication. Due to the limitation of available DSP slices in the device, the computation needs to be divided as two rows and a vector multiplication at one time. The voltage difference vector and rows of reconstruction matrix are stored in PS DDR. This data is transferred to the PL through DMA (Direct Memory Access) IP core. In the following section, the details of DMA IP core and AXIS multiplier logic development are given.

a) Direct Memory Access IP Core

The Xilinx DMA IP core provides high-speed direct memory access between AXI memory mapped and AXI stream interfaces. In other words, it transfers data from one part to another without much involvement from PS. It works in two modes, simple and scatter gather. The scatter gather mode includes scatter gather engine which can manage the DMA data transfer itself, less involvement of PS, and can be used to transfer number of packets of data. In simple mode, the transfers are handled by reading source and destination address from PS. Simple mode involves less FPGA resources as it excludes scatter gather engine. In this design, simple mode is

used as it provides the required functionality. The IP is configured for data width as 32 bits and width of buffer length register as 14 bits. The width of buffer length register indicates the allowed length of the data transfer buffer and can be set from 8 to 25. The buffer length is specified as 2^width of buffer length. In this case, the maximum data that will be transferred from DDR to PL is two rows that is 96*2*4bytes = 768 bytes.

Figure 8.18 shows the IP core block configured in simple mode.

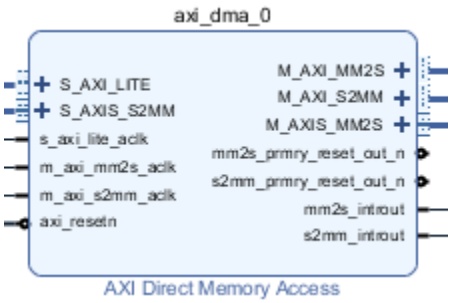


Figure 8. 18 AXI DMA IP Core block in the design

In simple mode, the IP core requires data transfer information such as source address, destination address and transfer length, configured. The information is configured by processor through AXI lite interface. The DMA IP also provides data transfer interrupts which can be used to identify if the transaction has completed. In this case, the interrupts are enabled in the IP core and connected to PS, however, are not used in final design. The ports shown in above figure are defined as follows

- S_AXI_LITE: AXI lite interface for communication between PS and IP core
- S_AXIS_S2MM: AXI stream interface from slave to memory mapped (PL to DDR)
- M_AXIS_MM2S: AXI stream interface from memory mapped to slave (DDR to PL)

- M_AXI_MM2S: AXI read channel
- M_AXI_S2MM: AXI Write channel
- mm2s_introut: memory mapped to slave data transfer Interrupt
- s2mm_introut: slave to memory mapped data transfer interrupt

For further information, please refer [50].

b) AXIS Multiplier Design

The matrix data for computation is transferred from DDR to PL through DMA. The DMA IP core uses AXI-stream protocol for data transfer on PL side. Hence, an AXI-stream custom IP core is generated with the functionality of two row multiplication. Apart from the AXI-stream communication ports, the custom IP also uses some handshaking signal to be controlled by PS. Figure 8.19 shows the AXIS multiplier IP core. It has AXI-stream slave input to take data input from DMA, AXI-stream master output to write the output to DMA. The handshaking signals that is load_diff and load_row, are coming from PS to indicate start storing the difference and row data respectively. The three-bit monitor port has some intermittent signals connected to it so that to debug the DMA transaction.

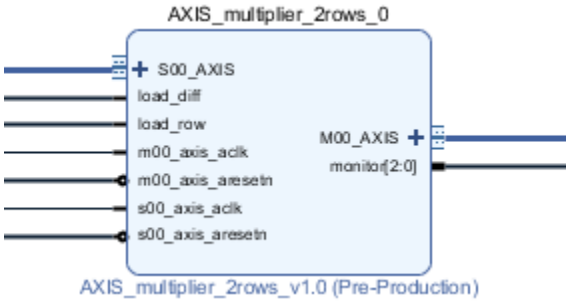


Figure 8. 19 AXIS multiplier IP core block

AXI stream is a type of AXI4 interface which is used for streaming data [51]. It does not include address for data transfer. It includes signals such as *Data, Valid, Ready and Last.*

The voltage difference data, which is 96 words and two rows data which is 96+96 = 192 words is loaded from DDR to PL using the handshaking signals load_row and load_diff. A multiplier is designed which takes 96 words voltage data and 96 words, single row data and performs multiply and accumulate function and produces the result. The parallel computation takes 96 DSP slices. Two such multipliers are configured for two row multiplication at a time. Hence, total 192 DSP slices are utilized.

The AXIS multiplier logic is designed using Verilog. In the logic flow, Verilog blocking and non-blocking both type of assignments are used. If blocking assignment (=) is used, the next statement in the always block gets executed only after the current statement is executed. If nonblocking assignment (<=) is used, all the statements in the same always block gets executed at the same time.

Figure 8.20 shows the logic block designed to store voltage difference data and matrix row data into local arrays.

```
if(t_valid_in && load_diff &&(index_diff <= 95))
    begin
     diff_data[index_diff] = t_data_in[19:0];              //using blocking assignment
       index_diff = index_diff +1;
    end
else if (index_diff > 95)
    index_diff = 8'h00;

if((t_valid_in && load_row) && (index_row <= 191))// && !(row_1))
begin
    row_data[index_row] = t_data_in[19:0];
    index_row = index_row + 1;
end
else if(index_row == 192)
begin
    row_load_done = 1'b1;                              //raise a flag if index is full
    index_row = 8'h00;
end
```

Figure 8. 20 AXIS multiplier data load logic

Here, blocking assignments are used in order to store the data in particular index and increment the index immediately after that. The AXI-stream signal *valid* and control signal from PS *load_diff* is used to load the difference data in the diff_data array. Similarly, *valid* and *load_row* signals are used to load the row data into the local array. As can be observed, two rows that is 96+96 = 192 data words are copied. The data width is kept as 20 bits for both difference data as well as reconstruction matrix row data. The reconstruction matrix, when computed in C# is stored as signed integer by multiplying the matrix data by 10000 and using only integer part.

There are two instances of multiplier and accumulator logic. The difference data and each row data are connected to each instance. Once the row data copy into local variable is done, the computation is performed. Here, we need the multiplication to be performed parallelly. However, the DSP slices in the device are spread column wise throughout the device. Also, to use DSP

slices efficiently for multiply and accumulate operation, the addition must be pipelined. In this code, the logic is pipelined in six stages. After loading the row data, it takes 70ns to output the multiply and accumulate computation. The output of computation for each row is 40 bits. The output needs to be transferred to the DDR through DMA and DMA data transfer is 32 bits wide. Hence, the 40-bit data is stored in a 64-bit word and the 64-bit data is transferred in two transactions.

Upon completion of computation for both rows, a counter is incremented and used to load the resulting data to DMA input. The logic developed for this loads the two rows computation result to the data output. For each row, there are two 32-bit words in the result. Hence, four words are transferred one after another. The valid and last signals are generated accordingly.

From the simulations during the development, it is found that, for the one-time difference data load it takes 960 ns. For two row data load, computation, and data output it takes 2us. Hence, for total 576 rows, that is total image computation, it should be 576us. Additionally, there are three signals which are taken out to monitor the flow of operation. The monitor signals are the difference data load done, rows load done, and data output done. These signals are connected to LEDs and were used to debug the DMA transaction at the design time.

c) Load Signals from PS

The load signals that is load_diff and load_row, are controlled by PS. An AXI GPIO IP is used to connect the control signals to AXIS multiplier IP block. The AXI GPIO. IP core named

axi_gpio_10 is configured to provide single channel two-bit GPIO output. The two bits are separated using Xilinx Slice IP core.

8.2.6 Switching Logic

The current injection and voltage measurement for the electrodes is performed in a pattern. The 16:1 multiplexer ICs are used to achieve this. The multiplexers have four select signals and one enable signal. The select signal switching and enable of the multiplexers is controlled through PS. An AXI GPIO IP core is used to connect the PS control signals to FPGA pins. The AXI GPIO is configured for 5-bit output channel. The 5-bits are separated as four for select signals and one for enable signal, using *Slice* IP cores.

8.2.7 Implementation and Bitstream Generation

Once the VIVADO block design is completed, the FPGA pin assignment is done by adding an XDC file. The pin configuration is added in the appendix in this document. After adding the XDC file, the design is synthesized, implemented and bitstream is generated. It also generates a hardware definition file (hdf), which includes the specification of the hardware design in PL for PS. Whenever an IP is added to the VIVADO design, which can be accessed through PS, it gets an address assigned. The hardware definition file includes the address information for these IP cores. In this VIVADO design there are multiple IPs which will be controlled by PS. The address assigned to each IP can be observed in address editor in VIVADO IDE. The AXI-lite IPs are assigned the address range from 0x4121_0000. Each AXI-lite IP is accessed through accessing its registers at the assigned address space.

The high-performance memory ports for AXI DMA connections are assigned the address space of DDR. The Zedboard provides 512Mbytes of DDR memory.

The hdf file then is exported along with the bitstream and SDK is launched.

8.3 SDK Design

Xilinx SDK is used to develop an application to run on the Zynq processor. In this section, the C application project design is described.

8.3.1 Design Description

The SDK design is mainly responsible for communicating with the GUI over serial port, configuring and controlling the FPGA hardware, DFT computation for voltage measurements, DMA communication for EIT computations in FPGA and part of Total Variation algorithm computation. There is a communication protocol designed to transfer data over serial port. The UART is configured in interrupt mode to receive different commands from GUI. The matrices required for EIT computation such as Gauss Newton One Step algorithm reconstruction matrix and Jacobian matrix are stored in SD-card. These matrices are read and copied to PS-DDR upon initialization. There is a 512MB DDR memory provided on Zedboard. For most of the computations, the DDR memory is used to store the data. The processor takes the system parameters such as signal frequency, sampling frequency and number of samples from GUI. These parameters are programmed accordingly into their respective PL IP cores, if necessary. The processor also takes in algorithm parameters such as type of algorithm, hyperparameter and number of iterations from GUI and store for further computation. For the EIT computation, two

types of measurements sets are taken, homogeneous and in homogeneous. The GUI sends the command to take homogeneous data input and inhomogeneous data input separately. For both measurement sets, the current injection and voltage measurement are taken in a pattern by changing the switch selectors. The acquired voltage measurement data is passed through a DFT computation to obtain the amplitude of the signal. Upon completion of inhomogeneous measurements, the set algorithm computation is performed. There are two algorithms, Gauss Newton One Step and Total Variation.

For Gauss Newton One Step computation, the reconstruction matrix and voltage measurement difference vector multiplication is performed. This multiplication is implemented in PL however the reconstruction matrix is stored in DDR. This matrix is transferred to PL two rows at a time through DMA. The result of the computation is also stored in DDR. After the computation completes the result vector is transferred to GUI over serial port.

For Total Variation computation, it requires Gauss Newton One step computation to get the initial solution, hence the it is calculated through PL implementation as mentioned above. Further TV computation needs to calculate a regularization parameter, a matrix inversion, and a matrix multiplication. These matrix computations are performed in PS. At the end, the updated reconstruction matrix is achieved and again the PL logic is used to calculate the reconstruction matrix and difference voltage computation. The result is stored in DDR and transferred to GUI for the display.

There is also an option, which can be set from GUI, to use the Zynq SoC only to acquire the voltage measurement data and perform the computation in GUI. If this option is set through GUI, the homogeneous and inhomogeneous measurement data is transferred to GUI over serial

port. For experiment purpose, an option of reference image computation is implemented. If this option is set from GUI, the homogeneous data is modified.

A functional flowchart of the SDK code is given in the appendix in this document. On initialization, all the peripherals such as UART, DMA, AXI GPIOs, ADC and DDS etc. are configured and initialized to the default value wherever necessary. As the UART is set up for the interrupt mode, the further functionality of the code depends upon the command received from GUI.

In the following section, every function implemented in the SDK design is described.

8.3.2 UART Configuration

The processor UART is configured to work in interrupt mode. The PS UART has the default setting of Baud rate 115200, data width 8bits, Stop bit 1 and no parity. In this project the UART default settings are kept unchanged. In Zynq FPGA, there is a Generic Interrupt Controller (GIC) which controls the interrupt request from the peripherals [52]. The GIC is configured for monitoring the UART interrupt. The UART can generate an interrupt on multiple events. It includes an interrupt mask register which can enable or disable a specific interrupt for the design [39]. A mask value is generated and loaded in the interrupt mask register which enables interrupt for receive FIFO full event, receive FIFO overflow event and Timeout error event. The Timeout error occurs when the receiver has remained idle for more than the time set in timeout register. A receive timeout is set to 8, so that it generates an interrupt if the receive channel has remained idle for 8*4*bit period time. Next, the PS UART can work in four modes, Normal, Local loopback, Remote loopback, and automatic echo. The UART is set to work in Normal mode.

The interrupt handler is set to read the received data and raise a flag once the data is read. There is a communication protocol set between GUI and Zynq processor. According to the protocol, the maximum data to be received is not more than 10 bytes at a time. Hence, the maximum data to be read from the serial port is set as 10 bytes. The received data is stored in the buffer INPUT.

8.3.3 Communication Protocol

The communication between GUI and processor includes transfer of various parameter settings from GUI to processor and the respective response or data transfer from processor to GUI. There is a communication protocol set for the data transaction between GUI and processor over serial port.

The command structure for transferring commands from GUI to processor, includes three fields as shown below

Command Identifier	Parameter Value	End of command

The *Value* and *End of command* field is there only in cases of transfer of some parameter value data along with the *Command Identifier.*

The End of command is always denoted by a semicolon.

Following are the commands used in this project.

- Read homogeneous data command

h	-	-

- Read inhomogeneous data command

i	-	-

- Number of Samples command

L	Parameter Value	;

- Signal Frequency command

F	Parameter Value	;

- Sampling frequency command

D	Parameter Value	;

- Switch command

s	-	-

- Gauss Newton One Step Algorithm set command

g	-	-

- Total Variation Algorithm select command

t	-	-

- Hyperparameter value set command

P	Parameter Value	;

- Number of iterations set command

N	Parameter Value	;

- Use reference image commands

R	-	-

- Do not use reference image command

r	-	-

- FPGA acquire and process command

A	-	-

- FPGA only acquire command

a	-	-

In response to the commands received from GUI, the processor responds with a response format. Also, the processor may send some other data to the GUI. Depending upon the data to be transferred from processor to the GUI there are four formats.

- For most of the parameter setting commands received from GUI, the processor response with a message that the parameter is received. The response structure is as follows

M	;	Color Char	:	Message	\n

The first filed in above response format indicates that this is just a message. The characters “;” and “:” are used as separators for different fields. The color character is introduced to display the messages in different colors on GUI. There can be two color characters “G” for

green and “R” for red. The end of the line character “\n” is for GUI to understand the end of response.

- At various places the execution time is measured and send to GUI for display. The format for sending measured time to the GUI is

T	;	Time	\n

The first field indicates that the data contains time measurement.

- If the FPGA is set in acquire only mode, upon the inhomogeneous measurement completion, the homogeneous and inhomogeneous data words are transferred to the GUI. The format is as follows

&	;	All Homogeneous data words separated by comma	#	All Inhomogeneous data words separated by comma	\n

- If the FPGA is set in acquire and process mode, upon the inhomogeneous measurement completion, the computation is performed as per the selected algorithm and the resulting image data is transferred to the GUI. The image data includes two words, higher and lower per element, that is 576. The format for transferring higher word and lower word for all 576 elements is as follows

$	;	All [Higher word, lower word] separated by colon	\n

8.3.4 Command Handling

In this section, most of the commands' execution implementation is described. For the commands which come with a value, the characters between command identifier and end of command are stored in a character array and depending upon the command identifier, the character array is converted to integer or float. For instance, if the command is for setting hyperparameter, the character array is converted to float. After retrieving the integer or float value, respective action is taken. In case of set number of iterations command, since the maximum number of iterations that can be set are less ten, the value is received as a character, hence the character is just subtracted from 48 to get the respective integer value. For the commands which include only one character, the respective action is taken.

For set hyperparameter value and set number of iterations value, the received characters are converted to float and integer respectively and stored in the respective variables for later use. In both cases, a message including the received value is sent back to GUI for display.

For set signal frequency command, the value is stored in a variable and the DDS IP core in FPGA is set accordingly. The details of DDS programming are given in DAC setting section. After programming the DDS, a message including the set frequency value is sent to GUI for display.

For set sampling frequency command, the value is stored in a variable and the sampling frequency IP core in FPGA is set accordingly. The details of sampling frequency programming are given in the data acquisition section. After programming the sampling frequency, a message including the set sampling frequency value is sent to GUI for display.

For set number of samples command, the value is stored in a variable and programmed in the AXI GPIO used to load the number of samples value to FPGA, using XGpio_DiscreteWrite() command. After programming the number of samples, a message including the set number of samples value is sent to GUI for display.

For the set switch command, the current switch value is written to the switches and the variable holding current switch value is incremented by 1. The default value at the initialization, for the switch variable is set at zero. At every command, the value is incremented by one up to eleven and then reset to zero again. This command is implemented for test purpose. In case, the switch rotation is required to be performed manually, this command can be used.

For the FPGA acquire and process or FPGA acquire only commands, a flag is set and reset respectively. A message is sent to GUI to acknowledge the setting. For reference image use or do not use commands, again the respective flag is set and reset respectively.

For read homogeneous data command, a flag is set indicating homogeneous data acquisition is performed. The data acquisition loop is executed which takes the voltage measurement with moving patterns and computes its DFT to record the measured voltage. This data acquisition loop is described in data acquisition section. In this project the computations are performed by following the EIT computations implemented in MATLAB EIDORS. For this purpose, the homogeneous and inhomogeneous measurements are required to be reordered. The reorder sequence is followed from EIDORS. An array with reorder sequence is declared as constant. Once the homogeneous measurements are completed, all 96 measurements are reordered using the reorder array as index.

For read inhomogeneous data command, first it is checked if homogeneous data acquisition is completed. If not, a message is sent to GUI saying acquire homogeneous data first. If the homogeneous data acquisition is performed already, the inhomogeneous data measurements are taken by execution of data acquisition loop. The details of this loop are given in data acquisition section. Upon completing the measurements for all twelve electrode pairs, the inhomogeneous measurement data is reordered as explained above. After reordering is done, the difference between inhomogeneous and homogeneous measurements is computed.

For performing the reconstruction computations, the FPGA mode is checked. If the acquire only mode is set, the homogeneous and inhomogeneous measurements are sent to the GUI for further computation, using the format shown in previous section. If the FPGA is set for acquire and process, the Gauss Newton computation is performed by execution the multiplication_loop() function. Then the algorithm setting is checked, if Gauss Newton One Step algorithm is set, the result of multiplication_loop() is sent to the GUI for display. If the Total Variation algorithm is set, the result of multiplication_loop() is used as the initial solution and the respective computation is performed. The implementation of Gauss Newton and Total Variation computation is given in following sections. After the TV computations are performed for set number of iterations, the resulting mage data is transferred to GUI for display.

8.3.5 Memory management

As previously mentioned, most of the computation data is stored in DDR memory. Hence, the DDR memory is mapped for different variables. The DDR memory address range is from 0x00000000 to 0x000FFFFF, that is 1MB is reserved [39]. Hence, the memory from

0x00100000 onwards can be used for processor code. The memory up to 0x0FFFFFFF, that is 256MB is kept as processor code region. The memory starting from address 0x10000000 is mapped for variables.

Table 8.1 gives the detail of the DDR memory allocation and its functionality.

Table 8. 1 DDR Mapping

Base Memory Name	**Base Address**	**Functionality**
MEM_BASE_ADDR	0x1000 0000	Used to store the reconstruction matrix
diff_base_addr	0x1006 0000	Used to store the difference matrix
RX_BUFFER_BASE	0x1007 0000	Used to store the image data result received from FPGA
RX_BUFFER_BASE_Wb	0x1007 2000	Used to store the TV regularization parameter Wb
Mat_base	0x1008 0000	Used to store the Jt*W Matrix read from SD Card
Mat_base_result	0x1010 0000	For storing matrix inversion result
Mat_base_JtWJ	0x1050 0000	For storing Jt*W*J matrix read from SD card
Mat_base_mult	0x1070 0000	To store the new reconstruction matrix
Mat_base_JtWJ_N	0x1080 0000	For storing Jt*W*J + Wb
For_char	0x10B0 0000	For storing SD card file data

The heap and stack sizes are also modified to 0x50000 each to be able to allocate the temporary data.

8.3.6 SD Card Read

The Zedboard provides SD-card interface. The SD-card storage can be used to store the data which can be loaded to DDR memory on initialization. The reconstruction matrix and Jacobian matrices are stored in an SD card in text file formats. A 4GB SD-card is used. On

initialization, the SD card is mounted, the text files are read, and the data is copied to the DDR memory on fixed locations.

To use SD-card, the generic fat file system library “xilffs” needs to be enabled from board support package settings [52]. Additionally, in application project C/C++ build settings, xilffs library needs to be added under ARM v7 gcc linker -> libraries.

A separate SD_card.h header file and SD_card.c files are created to handle the SD_card mount, unmount and read functions. The header file includes SD_Init(), SD_Eject() and ReadFile() function definitions. It also includes ff.h, generic fat file system module provided by Xilinx.

In the main code, after initialization and configuration of peripherals, SD card functions are executed, and the SD card data is read and copied to DDR memory. Before being able to access the contents of SD-card memory, the memory module needs to be mounted. The SD_init() function mounts the memory module using f_mount() command. The f_mount() function is given a fat file system type (fatfs) variable as input and 0:/ as path as in Zynq PS the SD-card0 is always mounted on 0:/ path.

After reading form the SD-card memory is completed, the memory module is unmounted. The SD_Eject() function unmounts the memory module using f_mount() command but with 0 instead of fatfs variable as one of the input parameters.

There are three text files stored in the SD-card, the reconstruction matrix, the Jt*W Matrix and Jt*W*J matrix. The reconstruction matrix text file includes integer data and the Jacobian matrices text files includes floating point data. A function ReadFile() is implemented which can read the data from a given text file and copy the contents on given DDR memory

location. The text file contents are read as characters. Hence, the file contents need to be read completely and then each data must be converted to its respective data type, integer, or float, before storing it to DDR location. The maximum text file content size is around 4MB which is far more than the local data cache of the Zynq PS, which is 32kB. Hence, the text file contents are also stored in DDR before converting the contents into integer or float. The Readfile() function requires name of the text file to be read, the DDR base address to copy the text file contents, the base address to copy the converted data and the data type in which the text file data needs to be converted. The text files are read one by one. Each file is opened in read mode and the contents are read. In all the text files the semicolon or "$" character is used to separate data words. Hence, using the separator characters each data word is converted to its respective data type and stored in the respective DDR memory location. After converting all the contents of the file, the file is closed. The flowchart for Readfile() function is added in the appendix in this document.

8.3.7 DAC Setting

As described in VIVADO design section, the digital data for a set sinusoidal signal frequency is generate using DDS IP core. To program the required DDS IP core, it is incorporated in an AXI-lite custom IP. The slave registers of this IP are used to set the DDS settings value. According to the design, slave register0 is used to enable or disable DDS and slave register1 is used to set the phase value.

In SDK code, a separate header file DAC.h is generated to include the AXI-lite DDS IP variables. The slave registers are accessed through a structure variable. The header file also

includes the function definitions such as enable_DDS(), disable_DDS and Phase_data(). In the DAC.c file, the AXI-lite DDS IP base address is assigned to a structure variable defined in DAC.h. This way the first variable in the structure is used to access slave register0 and second variable is used to access slave register1. In enable_DDS() function, the first variable in structure is set as one to enable the DDS. Similarly, in disable_DDS(), the first variable in structure is set to zero to disable the DDS. The phase value is loaded in the slave register1 through second structure variable.

The DDS is set at 30Hz frequency by loading the respective phase value calculated using equation 8.4. When the set DDS frequency command is received, the DDS phase value is calculated as follows

$$DDS\_setting = ((freq\_out/50x10^{6}) \times 2^{32}) + 0.5 \quad (8.6)$$

Where, freq_out is the required frequency setting received from GUI. The 0.5 value is added to round the float variable to nearest integer. This value is passed to Phase_data() function to load into DDS. Before setting the phase value, the DDS is disabled, and after updating the phase value the DDS is enabled again.

8.3.8 Digital Potentiometer setting

As described in VIVADO design section, the digital potentiometer is programmed through SPI protocol. The value setting to be transferred through SPI is programmed through AXI-lite registers. According to the IP, the slave register0 is used to enable/disable and load 16-bit data value, and the slave register1 is used to read the transfer status.

In SDK code, a separate header file digital_pot.h is generated to include the AXI-lite digital pot variables. The slave registers are accessed through a structure variable. The header file also includes the function definitions such as enable_digipot(), disable_digipot() and read_digipot(). In the digital_pot.c file, the AXI-lite digital pot IP base address is assigned to a structure variable defined in digipot.h. This way the first variable in the structure is used to access slave register0 and second variable is used to access slave register1. The functions enable_digipot() and disable_digipot() write the pot value at the slave register 0. The read_digipot() function reads the slave register1 value and stores it in a local variable.

In the main code, the digital pot value is updated at the initialization using enable_digipot() and disable_digipot() functions. The value programmed in the present code is 0xCD, which corresponds to 4kohm resistance. For enabling the digital pot IP core logic, the 16^{th} bit is set one as the enable bit in the slave register0.

8.3.9 ADC setting

The ADC requires to be programmed for sampling frequency and power mode through SPI communication. The SPI communication is achieved through EMIO GPIO. The EMIO pins are extended GPIOs for processor which can be routed through PL for pin assignment. In Zynq PS, the EMIO connections are separated in two 32-bit banks, bank 2 and bank 3 [52].To control each EMIO pin, the respective bank registers need to be used. In this project, the SPI signals are controlled by using the EMIO bank2 registers. The ADC SPI programming code is taken from the SDK code designed by senior design team [9].

a) ADC SPI Details:

The ADC SPI is a 16-bit 4-wire interface, which can be used for reading or writing the ADC configuration. In the 16-bit SPI frame, the 15th bit is read/write bit (1 for read and 0 for write). Bits 14 to 8 are address bits and bits 7 to 0 are data bits. Figure 8.21 shows the read write command format.

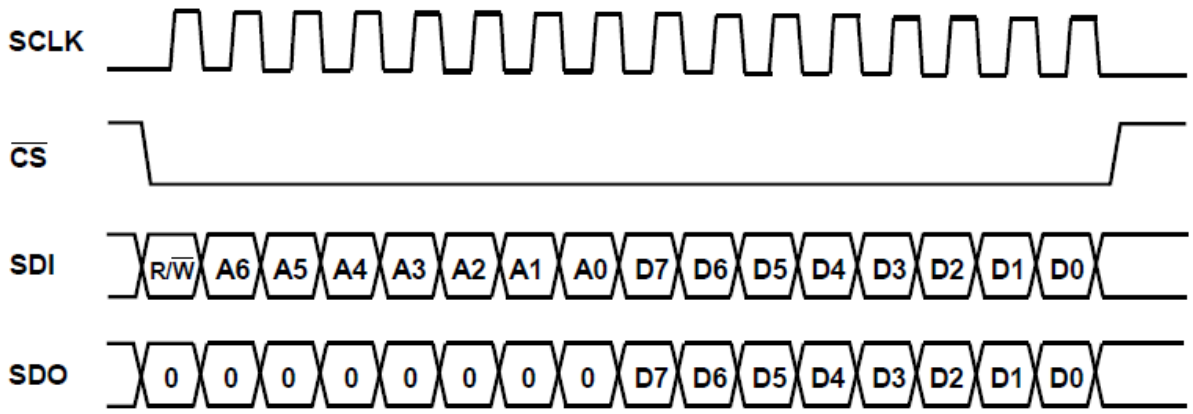


Figure 8. 21 ADC SPI communication format *[49]*

To set the sampling frequency, channel mode A register with address 0x01 is programmed. In this project, the ADC is set for median power mode and sync5 (low latency) filter with 128ksps sampling frequency. In the median mode, the MCLK is 4.096MHz. The desired sampling frequency that is output data rate can be achieved by setting the decimation rate. The ODR is given by

$$\text{ODR} = (\text{MCLK}/32)/\text{Decimation Ratio} \tag{8.7}$$

Hence, for 128ksps, the decimation rate needs to be set as 32. This makes the data word to be written in the channel mode A register as 0x0108. To set the median power mode, the power mode register, address 0x04, is loaded with the data 0x0422. Refer [49] for register bitwise details.

b) Processor Implementation:

During the PS configuration in VIVADO design, five EMIO pins are enabled. The five pins are used for ADC SPI connections and ADC Reset pin as shown in table 8.2.

Table 8. 2 ADC SPI EMIO Pin Assignment

EMIO Number	**SPI Signal Name**	**FPGA Pin**
0	SCLK	N18
1	SDO	G16
2	SDI	G15
3	CS	N17
4	Reset	J18

As mentioned above, the EMIO pins are accessed through GPIO bank 2 registers. In the SDK code, the init_ADC() function includes the ADC SPI logic. The GPIO bank registers are used to enable the EMIO pins and set them as output. The command data to set the power mode and the sampling frequency are programmed by toggling SCLK to generate clock and shifting the data word out on SDI. The functional flowchart of the SPI logic is added in the appendix in this document. The SPI master communication requires a two-frame communication. Therefore, the SPI write command is executed twice for each data. Also, the Reset pin is always kept high as the ADC reset signal is active low.

8.3.10 Data Acquisition

As described in VIVADO design, the data acquisition logic in PL includes sampling frequency logic, number of samples logic, ADC interface logic, FIFOs, ADC control logic and AXI GPIOs to read FIFO contents. In the SDK code, the GPIOs are initialized, number of samples and sampling frequency variables are set through respective GPIO channels, the ADC data acquisition is controlled, and the stored data is read and processed.

a) ADC Sampling Frequency setting:

The ADC sampling frequency setting is set through ADC SPI programming, which is set at 128ksps. However, the modified sampling rate or the divided sampling rate value is set in the FPGA through an AXI GPIO IP core. In the SDK code, during initialization, the AXI GPIO for sampling frequency divisor is initialized. As mentioned in VIVADO design, there two channels set in this GPIO, channel 1 for reset signal, which resets the count and channel 2 for setting the divisor value. The required divisor value for the required sampling frequency is calculated using following equation

$$\text{Divisor} = 128000/\text{Required Sampling frequency} \qquad (8.8)$$

For instance, for required sampling frequency 250Hz, the divisor needs to be set at 512. The divisor value is written in channel 2 of the GPIO. The reset signal is set high and low, so that the counter resets and starts a new cycle with modified sampling rate divisor. When the command to modify the sampling frequency is received, again a new divisor is calculated and programmed in the GPIO. The reset signal is set and reset to restart the counter.

b) Data Acquisition Loop

For the homogeneous and inhomogeneous measurements, the data acquisition loop is run. It rotates the current injection pattern by changing the switch settings from 0 to 11. The relation between switch pattern and ADC channels is given in a table in chapter 7. For every switch setting, the data acquisition is enabled, and the set number of samples are stored in FIFOs in PL. This stored sampled data is read and processed through Discrete Fourier Transform (DFT). The DFT magnitude for the set signal frequency is computed and stored as a measurement.

The DFT computation of a discretized signal is represented by following equations

$$\text{Real R(k)} = \sum_{n=0}^{N-1} x\,(n)\, cos\left(\frac{2\pi}{N} kn\right) \tag{8.9}$$

$$\text{Imaginary I(k)} = -\sum_{n=0}^{N-1} x\,(n)\, sin\left(\frac{2\Pi}{N} kn\right) \tag{8.10}$$

for k = 0, 1, …N-1

where, N = Total number of samples,

The magnitude of the DFT spectrum using real and imaginary values is given as

$$\text{Magnitude } |\text{X(k)}| = \sqrt{R^2(k) + I^2(k)} \tag{8.11}$$

In this project, the DFT computation is performed for specific frequency. The relation between k and the signal frequency *f* is given by following equation

$$f = k * \frac{Fs}{N} \tag{8.12}$$

where, Fs is the sampling frequency. In the SDK code, for the DFT computation, the k value calculated as

$$k = f * \frac{N}{Fs} \tag{8.13}$$

For the computation, the 16-bit digitized ADC data is converted to its corresponding voltage level by multiplying it by 0.0000625 which is ADC resolution 0.000125uV divided by 2.The division by 2 is done to reduce the amplitude of the voltage signal by a factor of 2. Also, to use the math library functions such as sine and cosine, in the C/C++ build settings, under ARM v7 gcc linker -> libraries *m* library needs to be added.

The functional flowchart of data acquisition loop is added in the appendix in this document.

8.3.11 Gauss Newton one step implementation

In this project, the Gauss Newton one step computation is reduced to a reconstruction matrix and voltage difference, as explained in chapter six. The reconstruction matrix (576x96) is stored in the DDR and used for computation once the voltage measurements are done. In SDK code, the multiplication_loop() function performs the required computation.

The actual multiplication for Gauss Newton One Step computation is implemented in PL. The data that is reconstruction matrix and difference voltage vector is transferred from PS-DDR to PL through DMA. The processor initializes the DMA transfers. First, the difference voltage vector is transferred and then the rows of reconstruction matrix are transferred. The rows are transferred two at a time. The PS polls for the reception through DMA and initiates for next two row transfer. Once all the rows are transferred and the computation result is received, the Gauss Newton One Step computation is completed.

As described in VIVADO design, the DMA is configured in simple mode. The simple transfer requires base DDR address and the size of the transfer. The code in figure 8.22 shows

the simple transfer from DMA to device that is PL. The TxBufferPtr is the base address of the voltage difference vector. The MAX_PKT_LEN is a length variable, which is 96 in this case. Once the transfer is initialized, a polling method is used to monitor if complete data is transferred.

Following figure 8.22 shows the DMA transfer in simple mode.

```
//Initialize DMA transfer for difference data
XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR) TxBufferPtr,MAX_PKT_LEN*sizeof(int), XAXIDMA_DMA_TO_DEVICE);

//Wait while AXI DMA transmit channel is busy, this is a polling method to wait for AXI DMA to complet
while(XAxiDma_Busy(&AxiDma,XAXIDMA_DMA_TO_DEVICE))
{
}
```

Figure 8. 22 DMA transfer

After transferring the difference data, reconstruction matrix rows are transferred. Since two rows are transferred in one DMA transfer, the for loop is repeated for 288 iterations for 576 rows transfer. The flowchart of multiplication_loop() function is shown in appendix in this document.

8.3.12 Total Variation Implementation

The Total Variation (TV) computations are performed in TV_iterations() function. As described in chapter 6, the TV computation includes following steps

- computation of initial solution using Gauss Newton One Step method
- TV regularization parameter computation
- Denominator computation
- Matrix inverse computation
- Matrix multiplication computation to achieve the new reconstruction matrix and

- computation of solution using newly calculated reconstruction matrix in Gauss Newton one step method

The steps after initial solution are repeated for the set number of iterations. The Gauss Newton computation is performed using multiplication_loop() function as described in previous section. The TV regularization parameter is computed using following equation

$$W_B = \frac{0.5}{\sqrt{(R * \Delta\sigma_0)^2 + \beta}} \tag{8.14}$$

where, R is the regularization matrix, which in this case is 576x576 identity matrix. However, the regularization matrix is always used as (hyper parameter * R) in the TV computation. The $\Delta\sigma_0$ is the initial solution calculated using Gauss Newton computation and β is smoothness parameter and kept constant throughout iterations as 0.0004. This value is taken similar to the TV computation developed by Rabi [33].

As R is 576x576 identity matrix and initial solution is 576x1, the ($R * \Delta\sigma_0$) computation is 576x1 and equals to initial solution. Figure 8.23 shows the W_B computation code snippet.

```
///Wb calculation
for(i=0;i<576;i++)
{
    mult_out = 0;
    mult_t1 = *(RxBufferPtr_o+((2*i)+1));
    mult_t2 = *(RxBufferPtr_o+(2*i));
    //for combining two 32 bit words
    mult_out = ((long long)( mult_t1 ) << 32);      //the initial sol
    mult_out |=  (long long)mult_t2;

    mult_out_f = mult_out/10000.0;          //divide by 10k as Gauss
    mult_out_f = mult_out_f*h_param;        //multiply by hyperparame

    Wb_ptr = (float*)(RX_BUFFER_BASE_Wb + i*sizeof(float));     //get
    temp_Wb = 0.5/(sqrt((mult_out_f * mult_out_f) + beta_param));
    *(Wb_ptr) = temp_Wb * (h_param*h_param);  //multiply by hyperpara
'/  printf("%0.5f \n ",*(Wb_ptr));              // test purpose
}
```

Figure 8. 23 Wb computation code (SDK)

The initial solution that is the output generated by multiplication_loop() is stored in DDR in two words, higher and lower. Each two-word data is read and combined. The reconstruction matrix is multiplied by 10000 while generation and storage in a text file, so that it can be used as integer. However, the Jacobian matrices are not scaled the same way. Hence, the initial solution words are divided by 10000. As mentioned above, the R matrix is always multiplied by hyperparameter, hence, the initial solution words are multiplied by hyperparameter. Next, the W_B computation is to be stored in DDR, hence the location for i^{th} W_B is computed using base address. Next, the W_B computation as per equation 8.14 is done. After W_B computation, the denominator in the following equation needs to be computed. In the above figure, that is W_B

loop, the last line computes ($R^T W_B R$) by multiplying by hyper parameter for R as R is the identity matrix. Here, the R is 576x576 and W_B is 576x1, hence for matrix computation the W_B matrix needs to be converted to a diagonal 576x576 matrix with W_B along the diagonal. However, here as R is identity matrix, minimum required computation is performed.

Next step is to compute denominator in the following equation.

$$\Delta\sigma = \frac{\boldsymbol{J}^T W \Delta V}{(\boldsymbol{J}^T W \boldsymbol{J} + R^T W_B R)} \tag{8.15}$$

For denominator, the ($\boldsymbol{J}^T W \boldsymbol{J}$) matrix is stored in DDR at the time of initialization. It is read and the W_B is added in the diagonal elements. Figure 8.24 shows the code snippet for the denominator computation.

```
for(i=0;i<576;i++)
{
    for(j=0;j<576;j++)
    {
        temp_JtWJ = *(JtWJ_ptr+((i*576)+j));        //read the JtWJ matrix el
        if(i==j)                                    //modify diagonal element
        {

            temp_Wb = *(Wb_ptr+j);                  //read the Wb value
            temp_JtWJ = temp_JtWJ + temp_Wb;                    //add the dia

        }
        *(JtWJ_N_ptr+((i*576)+j)) = temp_JtWJ;                      //upload

    }
    Xil_DCacheFlushRange((uint32_t)(JtWJ_N_ptr+(i*576)), sizeof(float)*576);
}
```

Figure 8. 24 TV Denominator computation code (SDK)

In this code, JtWJ_ptr shows the Jt*W*J matrix base address, Wb_ptr is the W_B base address and JtWJ_N_ptr shows the denominator base address. The denominator is also stored in the DDR after computation.

After denominator computation, the inverse of denominator is to be calculated. The inverse is calculated using row elimination method. Here, the denominator matrix is 576x576,

hence an identity matrix of size 576x576 is created and row elimination method is performed to get the inverse of the denominator.

As the data sizes of all the matrices are very large, the computation involves reading and writing rows of the matrices from DDR as per requirement.

Figure 8.25 shows the flowchart of the inverse computation logic. Here, D is used for denominator matrix and I is used for numerator matrix.

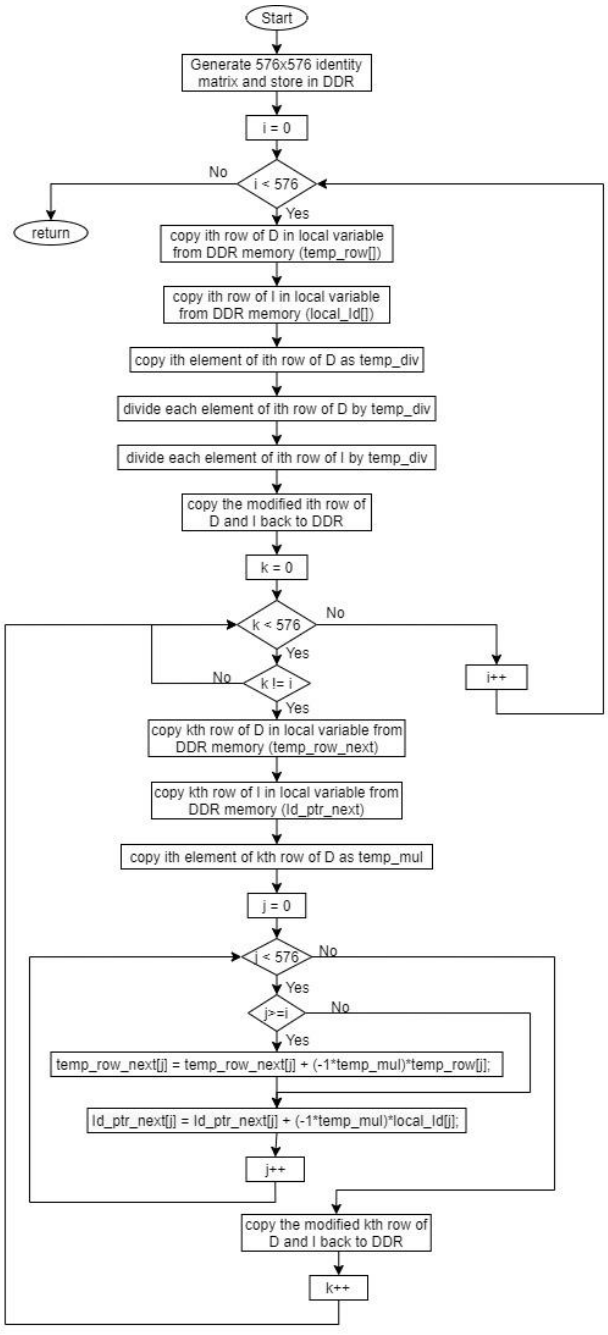


Figure 8. 25 Matrix Inversion logic flowchart (SDK)

The inverse matrix is multiplied with Jt*W matrix to get the new reconstruction matrix. The inverse matrix is stored in DDR. The Jt*W matrix is also stored in DDR. However, the Jt*W matrix is of 576x96, hence, to speed up the computation, the matrix is copied to local memory. A matrix multiplication loop is executed. Each element of the result is multiplied by 10000 and stored as an integer, as it is a new reconstruction matrix and will be used for Gauss Newton computation in PL.

After the computation of new reconstruction matrix, the multiplication_loop() is executed to perform reconstruction matrix and voltage difference vector multiplication. This way the solution for the TV algorithm for first iteration is achieved. If there are more than one iterations set, new computed result from multiplication_loop() is used as new solution and all the computations are repeated for each iteration. Once the TV iterations are completed, the last solution is transferred to the GUI over serial port for display.

8.3.13 Reference Image data logic

The reference image implementation was done for experimentation purpose. The assumption behind reference image is that the reference impedance distribution is uniform throughout the phantom. To achieve this, the homogeneous measurement is taken for all twelve electrode pairs. Before reordering the homogeneous measurements, the first eight measurements for the first current injection electrode pair are copied to all other pairs. Hence, making all the twelve measurements identical. This data is reordered and used as homogeneous measurement. The inhomogeneous data is measured as usual with salt water and the image is formed.

8.3.14 Time measurement

In this project, the execution time is measured at various places. A 64-bit global timer is used to measure the elapsed time.

The start time is measured using XTime_GetTime() function, before the start of the section for which the elapsed time is to be measured . After execution of the section, end time is measured. The elapsed time is calculated using following equation

Elapsed time = (End time – Start time) / COUNTS_PER_SECOND (8.16)

Here, COUNTS_PER_SECOND parameter is provided in xparameters.h, and is half of the CPU frequency, as the global timer is clocked at the half of the CPU frequency. The PS operating frequency is 667 MHz. This way the execution time in seconds can be calculated and sent to the GUI for display.

Chapter 9: Graphical User Interface Design and Development

This chapter provides the detailed description of the Graphical User Interface (GUI) design. The GUI is developed using C# windows form application in .net framework. The GUI includes various controls and displays to interact with EIT system to set the system parameters and observe the results. It also includes Gauss Newton One Step and Total Variation algorithm computations for EIT image reconstruction. In this chapter the C# GUI and code design are described.

9.1 Design Description

The graphical user interface is mainly responsible for communicating with EIT system hardware over serial port to transfer control data and image data. It provides control settings for EIT system operation such as operating frequency setting, sampling frequency setting and number of samples settings. It also provides controls to start the data acquisition for homogeneous as well as inhomogeneous measurement data. The acquisition can be performed one time or in a continuous mode. As described in chapter 6, the image reconstruction algorithms are developed in Zynq SoC as well as C#. The GUI provides the option to select where to perform the image reconstruction computation. The algorithm settings such as type of algorithm, hyperparameter and number of iterations can also be set from GUI. There are two features added for experimentation purpose, the usage of reference image in FPGA and manual switch settings from GUI. Depending upon the control settings, the Zynq SoC transfers the image data or voltage measurement data to GUI. If the received data is image data, it is displayed on the screen added in the GUI. If the received data is voltage measurements, the data is stored in text files for

later use as well as used to reconstruct the image using set algorithm computation developed in C#. The reconstructed image is then displayed. There is a color bar provided to display the data range spread with respect to the colors used for image display. The image data is normalized to for mapping to the color matrix. A color scale bar is provided to vary the data used for normalization. The GUI can also be operated in a demonstration mode. The frame data stored in text files during acquisition can be used to reconstruct and visualize the image offline. For the reconstruction computation in Zynq SoC as well as C#, the reconstruction matrices are stored in SD card and text files respectively. These reconstruction matrices computation is implemented in C#. The logic used to compute Jacobian matrices and 2D mesh is taken from the C# implementation done by Xin Chen [19]. Figure 9.1 shows the GUI designed in this project.

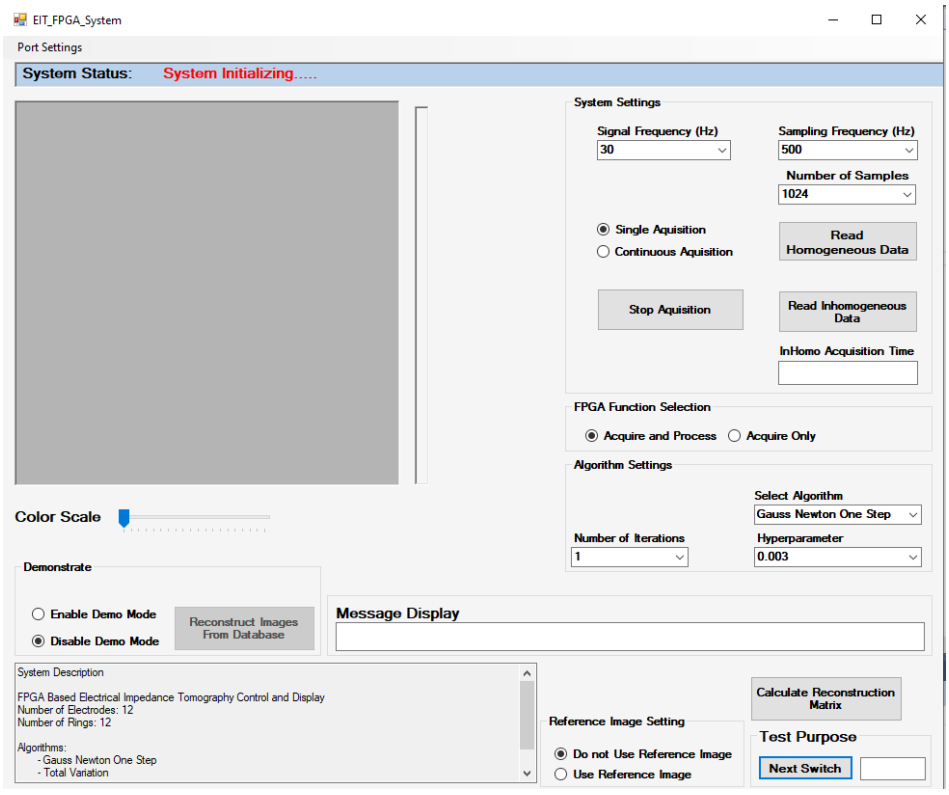


Figure 9. 1 GUI Design for EIT System

9.1.1 GUI Specifications

In this section, the detailed specifications and functions of each control and display feature provided in the GUI is given. Following are the EIT system GUI specifications

- Port Settings menu to set the serial port connection settings
- Signal Frequency setting combo box: available options - 2Hz, 4Hz, 10Hz, 30Hz, 100Hz, 500Hz, 1kHz, 5kHz and 10kHz.
- Sampling frequency setting combo box: available options - 250Hz, 500Hz, 1kHz, 2kHz and 128kHz
- Number of samples combo box: available options – 256, 512, 1024, 2048, 4096, 8192
- Read Homogeneous Data Button: To send read homogeneous command to the Zynq SoC
- Read Inhomogeneous Data Button: To send read Inhomogeneous command to the Zynq SoC
- Radio buttons for single or continuous acquisition setting
- Stop Acquisition button: to stop the acquisition in continuous mode
- Inhomo Acquisition Time textbox: to display the time taken during the inhomogeneous acquisition and image computation process
- Radio buttons for FPGA Acquire and Process or Acquire only options: To set the Zynq SoC in acquisition only mode
- Select Algorithm combo box: available options – Gauss Newton One Step and Total Variation
- Hyperparameter combo box: available options 0.3,0.03 and 0.003
- Number of Iterations combo box: available options – 1 to 6

- System Status pane: To display the system status
- Message Display textbox: To display miscellaneous data
- Calculate Reconstruction Matrix button: to calculate the Gauss Newton One step reconstruction matrix and Total variation Jacobian matrices
- Display Screen: to display reconstructed image
- Color bar: to display data range with respect to colors
- Reference image usage buttons: to enable or disable reference image usage in the Zynq SoC, experimentation purpose
- Switch setting button: to set the switches manually, test purpose
- Demonstration radio buttons: To enable or disable demonstration mode
- Reconstruct images from data base button

9.1.2 GUI Implementation

As mentioned in previous section, the GUI provides various functions to control the EIT system and reconstruct the image. The functional flowchart of the C# implementation is shown in appendix in this document. In this section, the implementation of each function is explained.

1) Serial Port Settings

The GUI communicates with the Zynq SoC over serial port. For serial port connection control, a menu option named *Port Settings* is provided on the left-top side corner on the GUI. A separate form is added in the design which gets opened when user clicks on the Port Settings menu. Also, upon initialization, the GUI is set to try to connect to a default serial port (COM 8,

in this case). If it could not connect, it shows the error message which asks to select another port. Figure 9.2 shows the port setting menu form. User needs to select the desired port settings and select Apply settings. The GUI starts to establish the serial port connection on the selected com port, however, if the connection is not established, it displays the error message asking to select another port.

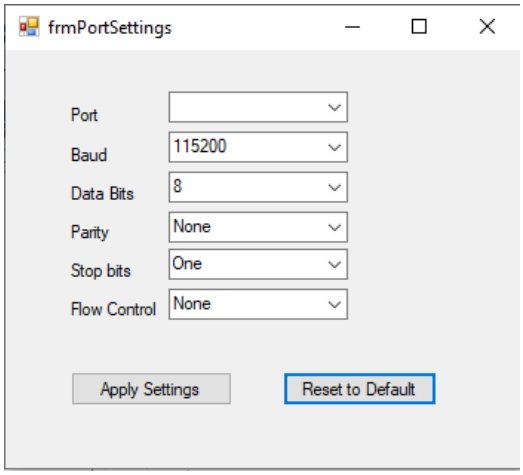


Figure 9. 2 Port settings form

2) System Controls

There are two types of controls on the GUI. One for which the selection is transferred to the Zynq SoC over serial port and other for which the selection is used to perform some function in C# code itself. In this section the function of each control is explained.

The GUI communicates with the Zynq SoC over serial port using a command structure. The command structure is described in chapter 8. For the GUI controls such as Signal Frequency, Sampling Frequency, Number of Samples, Algorithm type, Hyperparameter and

Number of Iterations, if the selected index is changed, the selected value is stored in a local variable. Also, the system mode is checked to be in demo mode, if not, the selected value is sent to the Zynq SoC using the command format to send a value from GUI to Zynq SoC.

If *Read Homogeneous Data* button is clicked, the system mode is checked to be in demo mode, if not, the read homogeneous data command is sent to the serial port. If *Read Inhomogeneous Data* button is clicked, again the system mode is checked to be in demo mode, if not, the read homogeneous data command is sent to the serial port. Also, the radio button for *Continue Acquisition* is checked, if selected, a flag is set to indicate the inhomogeneous data acquisition command is to be sent continuously. The time taken for inhomogeneous data measurement and respective image display is displayed on the GUI in a textbox. To achieve this, a *DateTime* structure is used to read the current time once the inhomogeneous data read command is sent to the serial port. A flag indicating the current time is read, is also set. There is a timer component added in the GUI design to generate a tick after every 1000ms. If this flag is set, the current time is measured again and subtracted from the time measured at the time of inhomogeneous data read command. This difference gives the elapsed time till the current time. It is displayed in a text box.

The Zynq SoC is set in Acquire and Process mode, by default. Hence, the *Acquire and Process* radio button is selected by default. If the *Acquire Only* radio button is selected and if the system is not in the demo mode, the respective command is sent to the serial port. Also, as with the FPGA *Acquire Only* setting, the voltage measurement data sent by FPGA is to be stored in text files. Hence, if *Acquire Only* is selected, a path to the folder where the text files are to be

stored is asked to be set. The folder browser dialogue is opened and if the user selects a folder, the folder path is stored in a variable for later use.

If a *Continue Acquisition* radio button is selected, a flag to indicate continuous acquisition is set. If the *Stop Acquisition* button is clicked, the continuous acquisition flag is reset.

If radio button *Use Reference Image* selection is changed, the system mode is checked to be in demo mode, if not, respective command is sent to the serial port. The *Next Switch* button is added to send the command over serial port to increment the switch position manually. If the system is not in the demo mode and the *Next Switch button* is clicked, the respective command is sent to the serial port. The switch position is also displayed in the text box next to *Next switch* button. To display the current position, a variable is set 0 at the initialization and once the *Next switch* button is clicked, the variable is incremented by one and displayed in the text box. As there are twelve switch position, if the variable reaches eleven, the variable is set to zero again.

The *Calculate Reconstruction Matrix* button click executes a function which takes number of rings and number of electrodes as input and generates the Jt*W matrix, Jt*W*J matrix and reconstruction matrix. These matrices are then stored in text files. This function *Generate_ReconMatrix()* is explained in later section.

The panel *Demonstrate* includes the controls for the demo mode. There are two radio buttons *Enable* and *Disable* demonstration mode and a button *Reconstruct Images from Database.* The demonstration mode is described in a separate section.

3) Data Reception on Serial Port

As explained in chapter 8, there are four types of data formats that Zynq SoC sends to GUI over serial port. The types are message, time, image data and voltage measurement data. In all types of data packets received from Zynq SoC, the first character is the packet identifier and it is separated from rest of the data using a semicolon. Hence, whenever a data packet is received over serial port, it is split using string split for ";" character. This way the packet identifier character and rest of the packet is separated. Next, the packet identifier character is checked, and respective action is taken. In this section, the functions performed in GUI for each packet identifier are explained.

a) Message packet:

If the packet identifier character is "M", the following data contains a message. This message also includes the color identifier character at the starting, R for red and G for green. The message is displayed in the system status label using *Display_label()* method. *Display_label()* method separated the color character from the message and displays the message in the respective color on a label.

b) Execution Time Packet

If the packet identifier character is "T", the following data in the packet is execution time. The data is displayed in a textbox next to *Messages* label. This packet can be used to transfer any miscellaneous information from Zynq SoC to GUI.

c) Image Data Packet

If the packet identifier character is “$”, the following data is image data. As explained in chapter 8, the image data is 576, 64-bit words. Hence, each word is sent as two 32-bit words, higher and lower. The image data words are separated by “:” character and the 32-bit data words are separated by “,”. When the image data packet is received, the 576 words are separated by splitting the received string for “:”. Next, the higher and lower data words are combined to form an image data word. Once all the 576 image data words are formed and stored as an integer, a *start_reconstruct_flag* is set.

d) Voltage Measurement data packet

If the packet identifier character is “&”, the following data is homogeneous and inhomogeneous voltage measurement data. If the voltage measurement packet is received, there are two functions performed. First, the received data is stored in text file in a specific format, second, the difference voltage is calculated and using the set algorithm, image reconstruction computation is performed.

There is a limit set of 100 such voltage measurement data files. When the system is set in to *Acquire Only* mode, a folder is selected to store the files created using data received in this packet. Whenever a new folder for file storage is selected, the file counter is set at zero. Every time a packet is received, a new text file is created. The name of the file is created as “testyyyyMMddHHmmssffff.txt”. Where, “yyyyMMddHHmmssffff” indicates the current time stamp from year to milliseconds. The current time is read using *DateTime* structure provided in C#. Next, the current system settings, such as signal frequency, sampling frequency, number of

samples, hyperparameter and number of iterations are read using a function *get_frame_details().* This system setting information is written in the file. In the packet, there is a homogeneous data and inhomogeneous data. All the data is stored in the text file. The system settings are stored in the following format

Setting Name	:	Value	;

Here, the setting names used are

- Frequency: for signal frequency
- Fs: for sampling frequency
- Samples: for number of samples
- hp: for hyper parameter
- Iterations: for number of iterations

The *value* parameter is the respective system setting.

The system settings, homogeneous and inhomogeneous data are separated by a character "$". The homogeneous data starts with a word Homogeneous, followed by a character "#" and the homogeneous data words separated by a character ",". Next, the inhomogeneous data starts with a word Inhomogeneous, followed by a character "#" and the inhomogeneous data words separated by a character ",". After all the data is stored in the file, the file counter is increased by one.

For image reconstruction computation, voltage difference data is calculated by subtracting homogeneous data from inhomogeneous data. A method *run_algorithm()* is run to perform the reconstruction computation. In this method, if the selected algorithm is Gauss

Newton one step, the reconstruction matrix and difference data array are passed to a Gauss Newton algorithm computation method. If the selected algorithm is Total Variation, the reconstruction matrix, difference data, number of iterations, hyperparameter, Jt*W matrix and Jt*W*J matrix are passed to a Total Variation algorithm computation method. The algorithm computation methods' implementation is explained in a separate section. Once the computation is completed, the *start_reconstruction_flag* is set.

4) Image Display

In this project, the image data is calculated using a twelve electrode, twelve ring forward model. After the image data computation or image data reception over the serial port, a *start_reconstruction_flag* is set. On each timer tick event of 1000ms timer, the *start_reconstruction_flag* is checked to be set. If it is, the available image data is passed to a *Start_reconstruction()* method to map the image data on the forward model mesh and draw the mesh on screen. The flag is reset after the image display is done. *Start_reconstruction()* method code is taken from the C# implementation developed by Chen [19]. In the *Start_reconstruction()* method, a 2D forward model mesh is computed. Next, a color map is generated, and the image data is normalized to map to the color map data. The mapped image data is then drawn as polygons on the screen. The actual image data and the mapped image data are then used to draw a color bar to show the data range with respect to colors used to show image.

Once the complete image data is plotted, the flag for continuous data acquisition is checked. If the flag is set and the system is not in demo mode, the *Read Inhomogeneous Data* command is sent to serial port.

a) Mesh Computation

The 2D mesh computation code is taken from the C# implementation developed by Chen [19]. This code is developed by following the 2D circular mesh computation implementation in EIDORS, which is open source software. In this section, a brief description of the functionality of the 2D mesh code is described.

A separate class *Painter* is there, which includes the definitions for 2D mesh components such as elements matrix, node matrix, electrodes structure etc. It also includes the method *Make2DModel()* which performs the computation. The *Make2DModel()* takes number of rings and number of electrodes as input. A separate *Matrix* class is there which includes the matrix definitions and matrix computation methods. The 2D mesh computation is done using matrices. To generate a 2D mesh element data, a unit circle is divided into twelve (number of rings) equidistant rings. The area between each ring is divided into triangles with equal area. This way for twelve rings, a 2D mesh generates 576 triangular elements. The output of 2D mesh computation is a matrix of nodal coordinates which includes coordinates for nodes on each ring for the triangles and a matrix of element node sequence which can be used to utilize the nodal coordinates and get the coordinates for each triangle.

b) Image Drawing

As mentioned above, most of the *Start_reconstruction()* method is adopted from Chen's C# implementation. Hence, in this section the overall function of the code and the additions done in this project are explained. After mesh generation, the nodal coordinates are modified with respect to the height and width of the screen panel on which the mesh will be displayed. An RGB

color model is generated. In the next step, the image data is normalized so that it can be mapped to the RGB color model matrix. To normalize the data, a value which corresponds to the respective maximum image data for that system setup, is used. This value is retrieved through a *get_color_scale()* method. This method is explained in next section. The *ImageMap()* method is defined in *Matrix* class. Next, for all the elements, nodal coordinate data is used to form a triangle coordinate. Then, Red, Green and Blue values corresponding to the image data for that element is read and a color is formed. The triangle is drawn on the screen and filled with the above formed color. Once all the elements are drawn, a color bar indicating the image data range with respect to the colors used is updated. This method is explained in a separate section.

c) Color Scale

The *ImageMap()* method was using maximum of the absolute values of image data available at the time, to normalize the data for mapping. However, in the cases where homogeneous and inhomogeneous measurements were taken one after another without making any changes for example not adding any anomaly, the slight measurement noise was reflecting in the image mapping. Hence, the feature of adjusting color scale is added, so that the image data is normalized with the same reference for all measurements for one system settings. A scroll bar is provided on the screen, which is set at 1 by default. Various measurements with the current phantom are observed. It is concluded that the maximum image data value changes with respect to the number of samples and sampling frequency. Hence, in *get_color_scale()* method, following equation is used to calculate the color scale value

$$Color\_scale = 8*10^6 * (C\_value)/(2^{N\_index} * 2^{Fs\_index}) \quad (9.1)$$

where, $8*10^6$ is set by observing the image data value for various settings.

C_value: color scale scroll bar value

N_index: Number of samples combo box index

Fs_index: Sampling frequency combo box index

This value is then used to normalize the image data. In case, this value is smaller, the image data plot logic stops and displays an error asking to increase the color scale value on scroll bar.

d) Color Bar Display

A color bar is displayed next to the image which indicates the image data range with respect to the colors used to display that image. To achieve this, a rectangle panel is created on the GUI and filled with a color blend which is retrieved using RGB map and image map. First, the image map data is sorted in an ascending order. Five values from the sorted image map data are read such as maximum, that is at index 575, index 431, index 287, index 143 and minimum that is at index 0. The index of these values in unsorted image data are retrieved and its corresponding actual image data values are displayed on the labels. These labels are placed besides the color bar, as maximum being on the top and minimum being at the bottom.

Next, the Red, Green and Blue values corresponding to each image map data index are read from RGB map and respective combined color is created. These five colors are blended with five different positions using *ColorBlend* class. The generated color blend variable interpolates the five colors at five positions. A rectangle panel is filled with the color array in color blend variable.

5) Reconstruction Algorithm Implementation

There are two algorithms implemented in C#, Gauss Newton One Step algorithm and Total Variation algorithm. As described before, as most of the system parameters in this project are constant, the Jacobian matrix and reconstruction matrix can be calculated once only. There is a provision in C# code to compute the Jacobian and reconstruction matrices. There is a button provided on GUI which computes the required matrices and stores each matrix in a separate text file. For Zynq SoC implementation, these text files are copied in a SD card. For C# implementation the text files are stored in the project directory. On initialization, these files are opened, and the matrices are stored in local variables. In this section the reconstruction matrices computation and both algorithm implementation is discussed.

a) Reconstruction matrices computation

In this project, the system parameters like number of electrodes, number of rings and stimulation pattern are constant. Hence, the Jacobian matrix computation can be performed only once. As described in chapter 6, for Gauss Newton computation, a reconstruction matrix is required to be calculated. For Total Variation implementation, as it is an iterative method, the reconstruction matrix needs to be calculated at each iteration, however, the Jt*W and Jt*W*J matrices can be calculated at once. The *Calculate Reconstruction Matrix* button on GUI executes the above described matrices computation through *Generate_ReconMatrix()* method. A separate class *Operations* is created to include this method.

The Jacobian matrix computation needs a 2D mesh and a stimulation pattern. The C# code Jacobian computation, 2D mesh computation and stimulation pattern computation is taken

from the C# implementation developed by Chen [19]. In the *Generate_ReconMatrix()* method, first, three text files are generated. Next a 2D mesh is computed using *Painter* class. A background conductivity is assumed in the Jacobian computation, which is set as one. Hence, a 576x1 array with value 1 is generated for Jacobian background. The parameters required for stimulation pattern computation are the current injection pattern, which is opposite in this case, the measurement pattern, which is adjacent in this case, there is no measurement taken on the current driving electrodes and the applied current is 0.25mA. These values are passed to a method *Mk_stimulation()* method. This method returns the injection and measurement pattern arrays. Next, the Jacobian matrix is calculated by passing the mesh, stimulation pattern and Jacobian background to a *Jacobian_adjoint()* method. It returns a 96x576 matrix.

For the reconstruction matrix computation, the matrices Jt*W and Jt*W*J are required. Here, J is the Jacobian matrix, Jt is Jacobian matrix transpose and W is the covariance matrix, which in this case is a 96x96 identity matrix. Hence, Jt*W is only the transpose of Jacobian matrix calculated above. This matrix is then written in the respective text file. Here, the data type is double, and it is written in text files only up to eight decimal places. Each data word is separated by a semicolon. Also, to speed up the data writing process a *StringBuilder* class is used. It allows to build a large string of all data and all data can be written in the file at once. Next, the Jt*W*J matrix is computed by implementing a matrix multiplication. The resulting 576x576 matrix is written in the respective text file.

Next, the denominator for the reconstruction matrix is calculated which is (Jt*W*J + λ^2*R). Here, R is the regularization matrix which is identity matrix as Tikhonov regularization is used. The hyperparameter λ is set as 0.003. Next, an inverse is calculated for the resulting

denominator matrix using row elimination method. The row elimination method implementation is done similar to the Zynq PS implementation, as described in chapter 8. The only difference is, in this case, all the matrices are stored in the local variables, which makes the computation faster as compared to that of Zynq PS implementation. The inverse matrix is multiplied with Jt*W matrix to get the reconstruction matrix of size 576x96. The reconstruction matrix is multiplied by 10000 and only integer part is stored in the text file. This is implemented so that the FPGA computation becomes easier to implement. The same reconstruction matrix is used in C#, so that the computation remains same throughout the project.

b) Algorithm Implementation

Once the reconstruction matrix is calculated, the Gauss Newton algorithm computation is a 576x96 reconstruction matrix and a 96x1 difference voltage vector multiplication. A Gauss Newton computation method that is *GNOS()* includes a *Mat_vec_mult()* method which is developed to implements the required matrix and vector multiplication.

The Total Variation implementation computation involves steps such as calculating initial solution using Gauss Newton method, calculating TV regularization parameter, calculating denominator, calculating inverse, calculating new reconstruction matrix, and calculating the solution for the iteration. First, the initial solution is calculated by passing the stored reconstruction matrix and the voltage difference data to *GNOS()* method. For the calculation of TV regularization parameter $W_{B,}$ the initial solution is divided by 10000, as the reconstruction matrix is multiplied by 10000 while storage. The calculation of $W_{B,}$ the denominator and inverse computation are implemented similar to the implementation of same variables' computation in

Zynq PS. The only difference is, in this case all the matrices are stored in local variables. The new reconstruction matrix data is also multiplied by 10000 and the solution for the iteration is achieved by multiplying new reconstruction matrix and the voltage difference data. If the number of iterations is set more than one, the resulting solution is used as the new solution and the steps from calculating W_B to next solution, are repeated for the number of iterations.

6) Demonstration Mode

The system can be run in a demonstration mode. This means that the voltage measurement data stored in a specific format can be used to reconstruct the images using the EIT GUI. In this project, the Zynq SoC can be used only for acquisition. In such case the voltage measurement data sent by Zynq SoC is stored in text files along with the system parameters set at the time of acquisition.

If the *Reconstruct Images from Database* button is clicked and the radio button *Enable Demo Mode* is selected, a folder selection dialog box is opened. The text files with frame data are supposed to be set with a name including “test”. If the folder is selected, the files in the folder containing “test” in the name are opened one by one. The file contents are read and the system setting information and homogeneous and inhomogeneous data are separated. The system information is updated at their specific fields on the GUI. The homogeneous and inhomogeneous data are used to calculate the voltage difference vector. Depending upon the algorithm type setting, the voltage difference data is passed to the respective algorithm computation function. The resulting image data is then displayed on the screen. This is repeated until all the text files in that folder, with the text “test” in their names are opened, read and the image computation is

performed. There is a delay of 1000ms added between any two file reads so that the image data displayed for all the files can be visualized properly.

7) Execution Time Measurements

In this project, the elapsed time measurement is done at various steps, for instance, reconstruction computation. A stopwatch class in C# is used to measure the elapsed time. A stopwatch is reset and then started immediate before the code section for which the elapsed time measurement is required to be done. After the code section execution, the stopwatch is stopped. The total elapsed time is then read using the *stopwatch.elapsed()* method and displayed in a message textbox.

Chapter 10: Experiments and Results

The EIT system designed in this project provides various controls for EIT image reconstruction. The EIT system is tested by changing various parameters available and the results are observed. This section includes the description and results of the experiments performed to test the system. The system performance is also discussed at the end.

10.1 Analog measurements

The Analog Front End board is tested for transmission and reception section signals at various places on board. Figure 10.1 shows the image of the EIT system hardware, including Zedboard and AFE board.

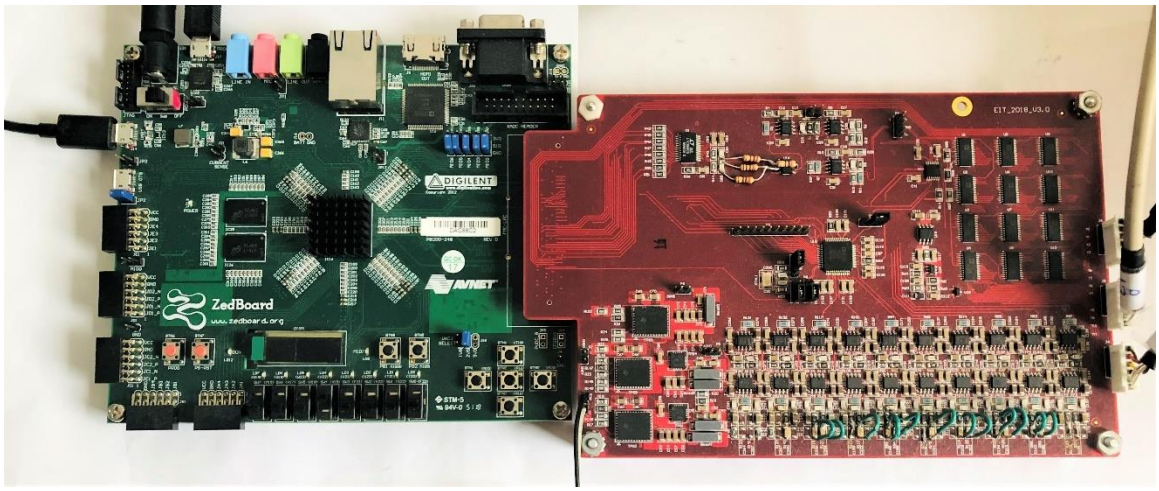


Figure 10. 1 EIT System Hardware Setup

Chapter 7 shows the results of the AFE testing. In this section, the phantom response at four different frequencies, 2Hz, 4Hz, 10Hz and 30Hz is observed by measuring the voltages at eight ADC channel inputs, for switch position 0.As mentioned in the system specifications, this

system works from 2Hz to 10kHz. However, the hardware that is AFE currently available only works up to 30Hz.

For these observations, the transmission signal, that is DAC frequency is set at specific frequency and the resulting voltage measurement at eight ADC channels is observed on oscilloscope. The current injection setting for all the signal frequencies is 0.25mA. Figure 10.2 shows the peak to peak voltage for each channel with respect to the frequency. It can be observed as the frequency increases the voltage level decreases. However, there is some attenuation expected for 30Hz signal frequency as the lowpass filter in the receiver section before ADC is designed for the cutoff of frequency of 18Hz.

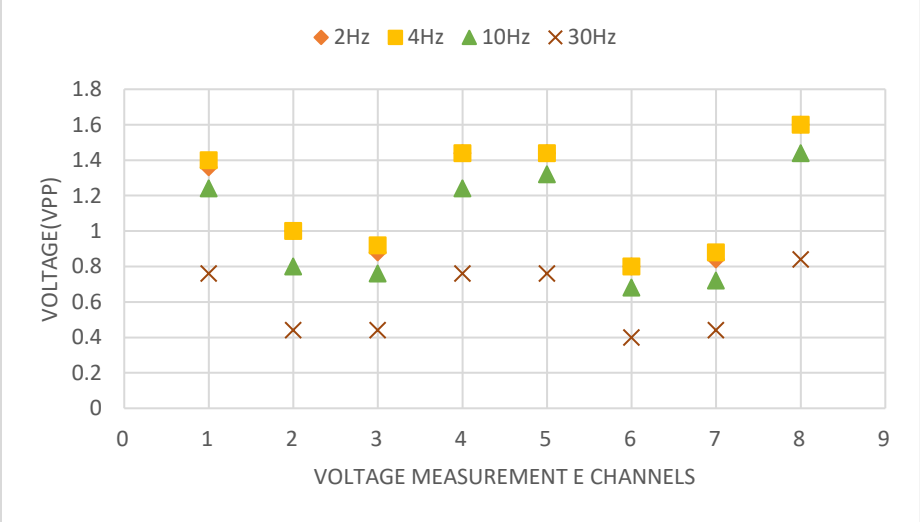


Figure 10. 2 Voltage measurement at 8 ADC channels for different frequencies

The change in voltage with respect to frequency can be more elaborately examined if the measurements are taken for broader range of frequency such as 2 Hz to 10kHz in the same setup.

10.2 Image Reconstruction results

The image reconstruction is verified by making holes in the phantom and adding salt water as a low impedance anomaly. Figure 10.3 shows the tissue phantom used for testing. As can be observed, there are two holes made in the phantom.

Figure 10. 3 Tissue phantom used for the testing

The Gauss Newton One Step and Total Variation algorithm image reconstruction results are observed at different system settings. To compare the image output between both algorithms, only single hole is used. Following are few of the results.

10.2.1 Image reconstruction results

For this observation, the homogeneous measurements are taken without adding any salt water in the phantom holes. For inhomogeneous measurements, the left side hole in the phantom shown in figure 10.3 is filled with salt water.

1) Gauss Newton One Step

a) Figure 10.4 shows the observations taken for Signal Frequency 30Hz, Sampling Frequency = 250Hz, Number of Samples = 2048

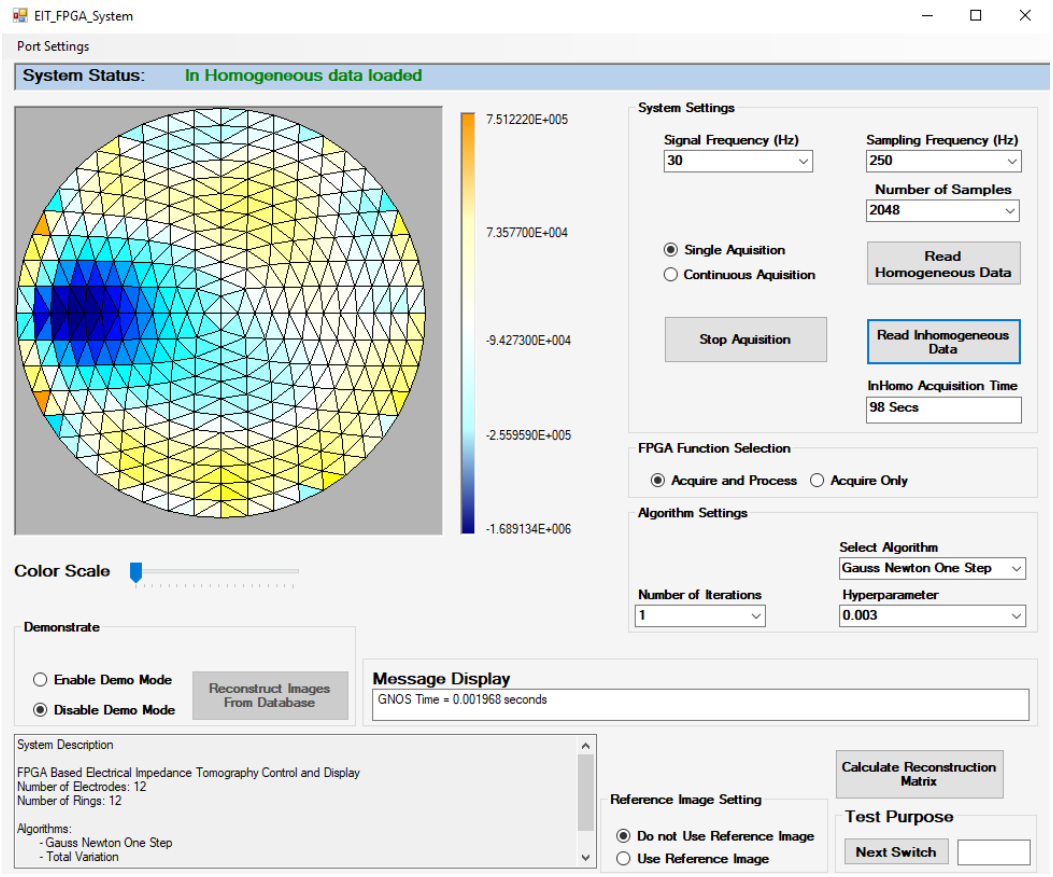


Figure 10. 4 Gauss Newton One Step Low Impedance Image at 30Hz

b) Figure 10.5 shows the observations taken for Signal Frequency 2Hz, Sampling Frequency = 250Hz, Number of Samples = 8192

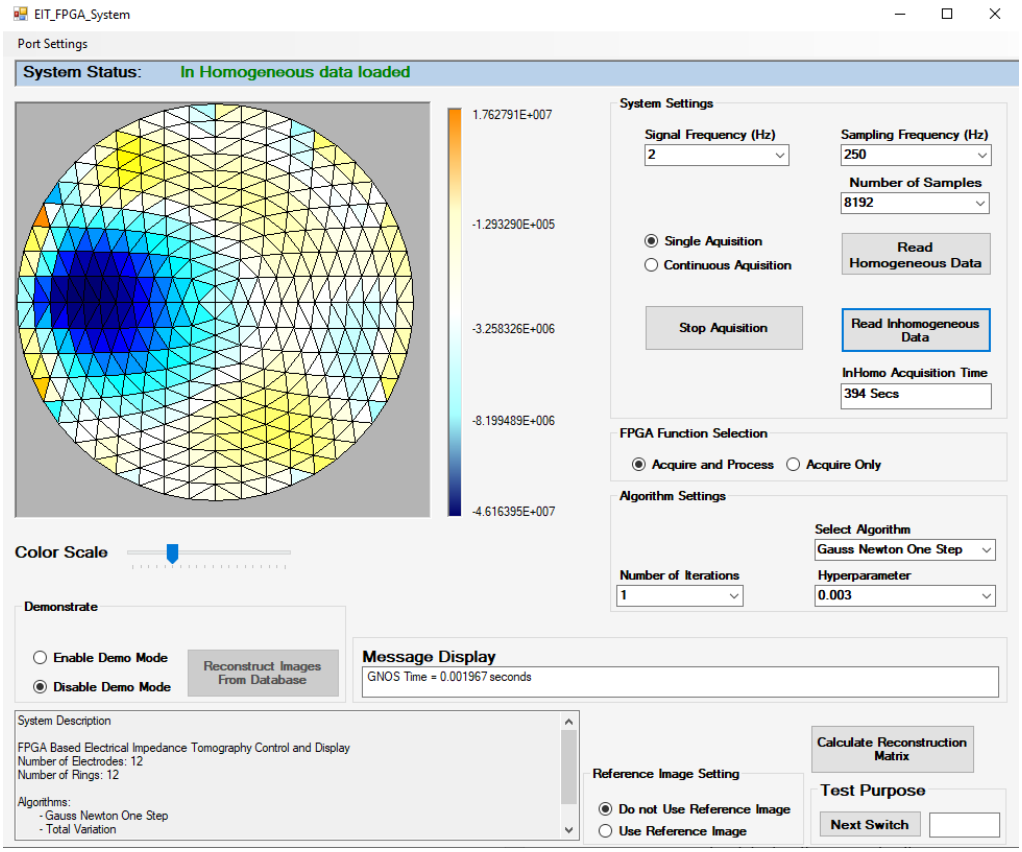


Figure 10. 5 Gauss Newton One Step Low Impedance Image at 2Hz

2) Total Variation image reconstruction result

a) Figure 10.6 shows the observations taken for Signal Frequency = 30Hz, Sampling Frequency = 250Hz, Number of Samples = 2048, hyper parameter = 0.003 and Iterations = 1

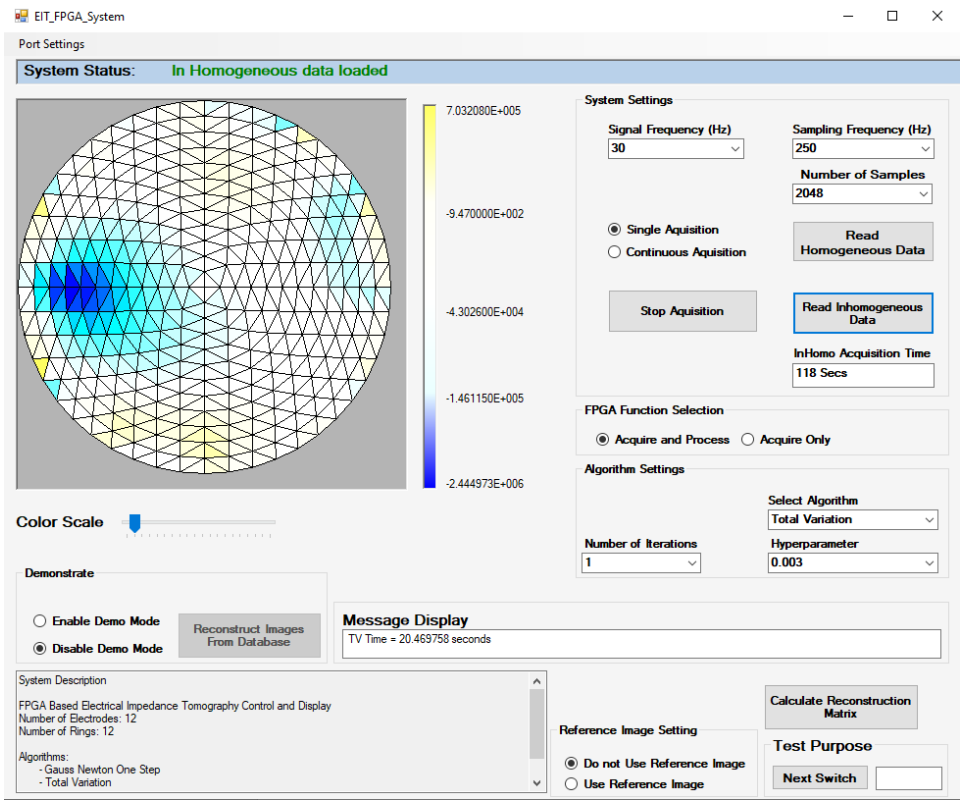


Figure 10. 6 Total Variation Low Impedance Image at 30Hz

b) Figure 10.7 shows the observations taken for Signal Frequency 2Hz, Sampling Frequency = 250Hz, Number of Samples = 8192, hyper parameter = 0.003 and Iterations = 1

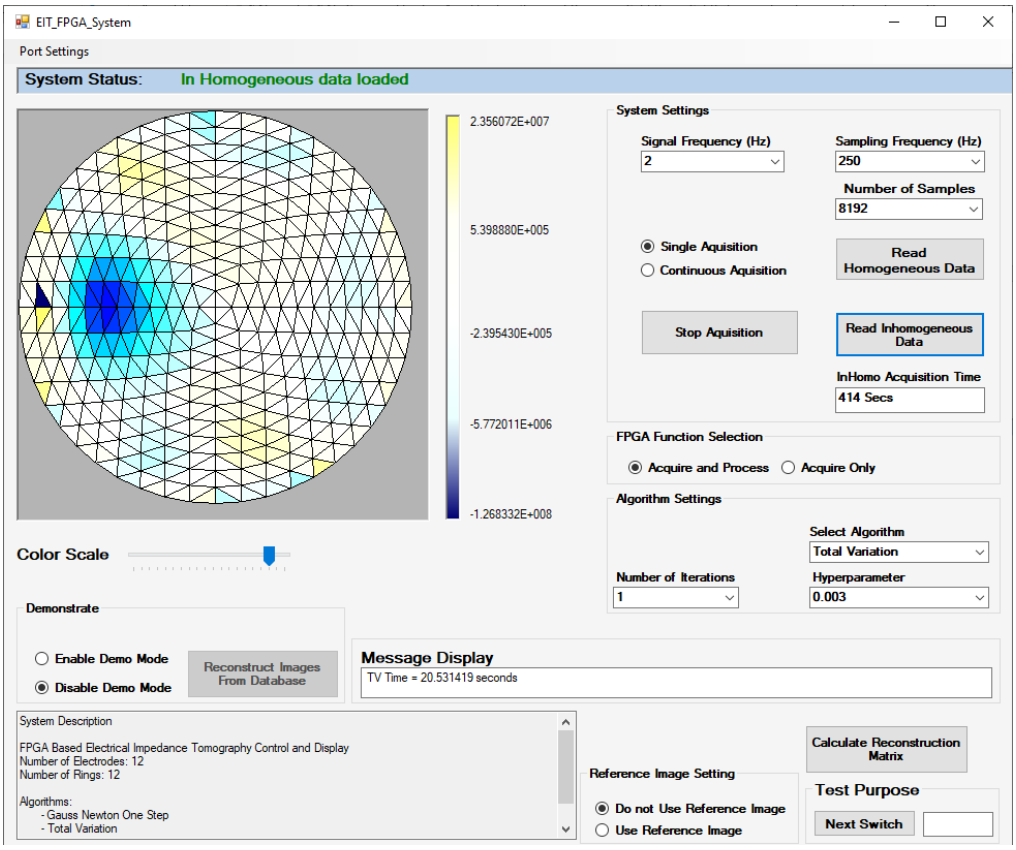


Figure 10. 7 Total Variation Low Impedance Image at 2Hz

As observed from figure 10.4 to 10.7, the Total Variation algorithm produces more precise result as compared to the Gauss Newton one step algorithm. Also, the image reconstruction result does not vary much between 2Hz and 30Hz.

10.2.2 Image Reconstruction with Two holes

To observe if impedance change at two different places gets detected, two holes are filled with salt water, alternatively. For homogeneous measurements, left side hole is filled with salt water. For inhomogeneous measurements, salt water from left side hole is cleared and right-side hole is filled with salt water. For reconstruction Gauss Newton One Step algorithm is used. The red color in the image shows high impedance change and the blue color shows the low impedance change. As for the left side hole, the salt water is removed, the effective impedance during inhomogeneous measurements was higher, hence it shows red color. Similarly, for the right-hand side hole, the salt water is added during inhomogeneous measurements, this makes the effective impedance at that area lower, hence it shows the blue color.

Figure 10.8 shows the resulting image for frequency 30Hz, sampling frequency 250Hz and number of samples 2048.

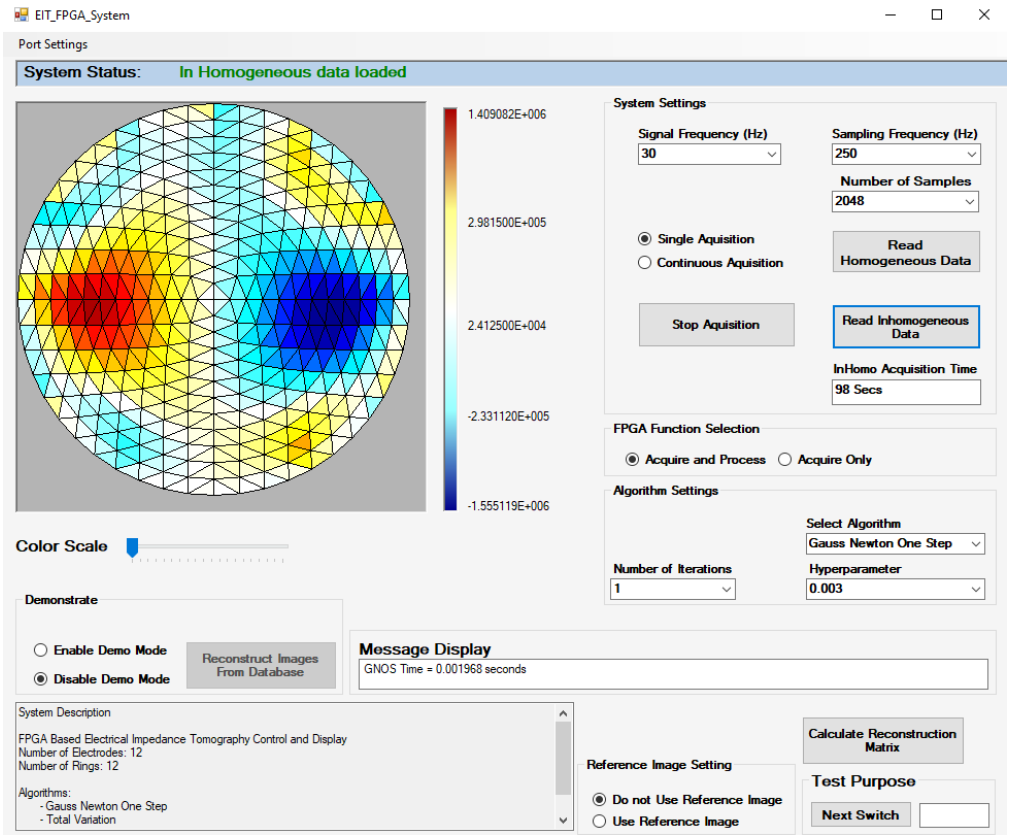


Figure 10. 8 Impedance change at two locations, with Gauss Newton One Step

Figure 10.9 shows the above image reconstructed using Total Variation algorithm.

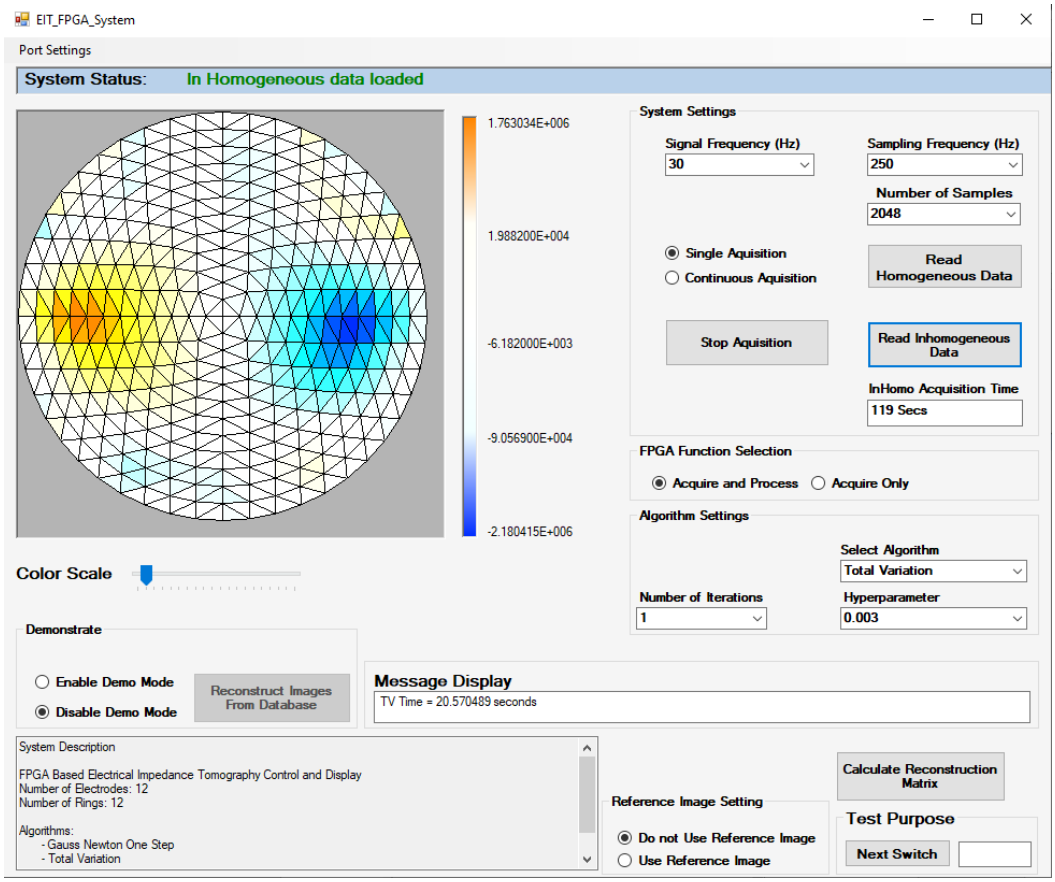


Figure 10. 9 Impedance change at two locations, with Total Variation

10.3 System Implementation and Performance Analysis

In this section the FPGA design implementation and overall system performance is discussed.

10.3.1 FPGA Design Implementation Report

The FPGA design, that is VIVADO design is explained in detail in chapter 8. The design includes DAC logic, Digital pot logic, 8-channel ADC data acquisition and storage logic and two 96-word depth multiplier and accumulator sections. Figure 10.10 shows the device utilization report for the FPGA design. It shows that the design uses 192 DSPs to implement the multiplier. Also, if compared to the FPGA design implemented by senior design team as shown in chapter 3, the LUT utilization is much less in this project.

Resource	Utilization	Available	Utilization %
LUT	28924	53200	54.37
LUTRAM	1030	17400	5.92
FF	21731	106400	20.42
BRAM	41.50	140	29.64
DSP	192	220	87.27
IO	46	200	23.00
MMCM	1	4	25.00
PLL	1	4	25.00

Figure 10. 10 FPGA Design Device Utilization Report

Figure 10.11 shows the timing summary report of the FPGA design. It shows that there are no timing violations in the design. The multiplier section of the design uses 192 DSPs. It may introduce timing violations after the implementation. However, while designing, the accumulator part is implemented as a pipeline in six stages. This helps the implemented design meet timing constraints.

Figure 10. 11 FPGA Design Timing Summary

Again, if compared to the FPGA design implemented by the senior design team, the timing performance is much improved.

10.3.2 Timing Analysis

The execution time and computation time measurement is done at various sections in this project. The respective analysis is presented. The execution time measurement methods utilized in SoC and C# are explained in chapter 8 and chapter 9 respectively.

First, since the AFE receiver section works with only low frequency range, the acquisition time for low frequencies with respect to number of samples is shown in table 10.1 below. This time is calculated using following equation

$$\text{Required time} = (1/\text{sampling frequency}) * \text{Number of samples} * 12 \quad (10.1)$$

The multiplier 12 is used as there will be twelve measurement cycles for twelve electrodes. Table 10.1 shows the time required for one homogeneous or inhomogeneous measurement, in seconds.

Table 10. 1 Acquisition time at low frequency

No. of Samples -> **Fs (Hz)**	256	512	1024	2048	4096	8192
250	12.288	24.576	49.152	98.304	196.608	393.216
500	6.144	12.288	24.576	49.152	98.304	196.608
1000	3.072	6.144	12.288	24.576	49.152	98.304
2000	1.536	3.072	6.144	12.288	24.576	49.152

The computation time required for Gauss Newton One Step calculation and Total variation calculation for one iteration is measured. Since the algorithms are implemented at both Zynq SoC and C#, the execution time is measured at both implementations. Table 10.2 shows the execution time for both algorithms' computation. The execution time for C# implementation was measured on the computer with specifications as Intel core i5-6200U CPU with 2.30GHz frequency and 16 GB RAM.

Table 10. 2 Algorithm computation time

Algorithm Name	**Zynq SoC Implementation**	**C# Implementation**
Gauss Newton One Step	1.96ms	0.6ms
Total Variation (1 Iteration)	20.49seconds	4.6seconds

It can be observed that the Gauss Newton One Step computation time in SoC is larger than the time required for the same computation in C#. The time required for one iteration computation of Total Variation algorithm in SoC is also very large as compared to the same computation time in C#. Moreover, for the computations performed in SoC, the resulting image data is required to be transferred to the GUI over serial port. In this project the serial port baud rate is set at 115200. According to the image data transfer format shown in chapter 8, the data packet contains, two starting characters, 576 x 2 image data words separated by a character and two end characters. Hence, serial data transfer time can be calculated as

$$Time\ (image\ data\ transfer)\ =\ \frac{(2+(576*5)X2+2)*8bits}{115200} \qquad (10.2)$$

This equals to 400ms. Hence, for every image data computation performed in SoC, there would be 400ms time overhead added for image data transfer from SoC to GUI. In case of image computation in C#, the voltage measurement data is required to be transferred from SoC to GUI. According to the voltage data transfer format shown in chapter 8, this data packet contains two starting characters, 96 homogeneous data words separated by a character, a separation character, 96 inhomogeneous data words separated by a character and two end characters. Hence, serial data transfer time can be calculated as

$$Time\ (measurement\ data\ transfer)\ =\ \frac{(2+(96*5)X2+1+2)*8bits}{115200} \qquad (10.3)$$

This equals to 67ms. This shows that there is a serial data transfer overhead in case of image computation in C# software also, however, it is far less than that of in case of image data computation in SoC.

In the GUI, the 2D mesh computation and the image data mapping to the mesh and display time is measured. It takes around 500ms.

10.3.3 Low Impedance Anomaly Detection:

The phantom used for testing is a circular phantom with diameter 19cm. As the FEM used for image reconstruction uses 12 rings, the diameter is divided into 24 equally spaced triangles. Hence, one triangle represents 19cm/24 = 0.79cm length.

The width of the left-hand hole is around 2.5cm and the height is 1.5 cm. Hence, it should have around 3.16 triangles as width and 1.89 triangles in height. Similarly, the width of the right-hand hole is 2cm and height is 1.5cm. Hence, it should show 2.53 triangles in width and 1.89 triangles in height. Figure 10.12 shows the measurements of the width of left-hand and right-hand holes on phantom.

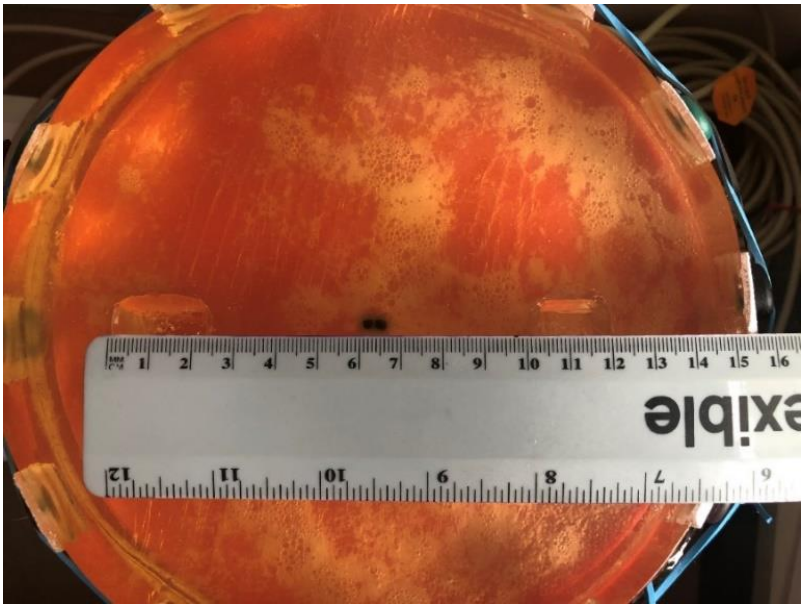


Figure 10. 12 Phantom holes measurements

Figure 10.13 shows image reconstructed after filling both the holes with salt water. For this observation, a homogeneous measurement is taken with both holes empty and inhomogeneous measurements are taken by filling both holes with salt water. The image is reconstructed using TV algorithm. As can be observed, the left side hole is showing 3 triangles in width and 2 triangles in height. Similarly, the right-hand side hole is showing 3 triangles in width and two triangles in height. Here depth of the holes is not considered. As can be observed, low impedance anomalies are detected very close to the actual dimensions in the physical phantom.

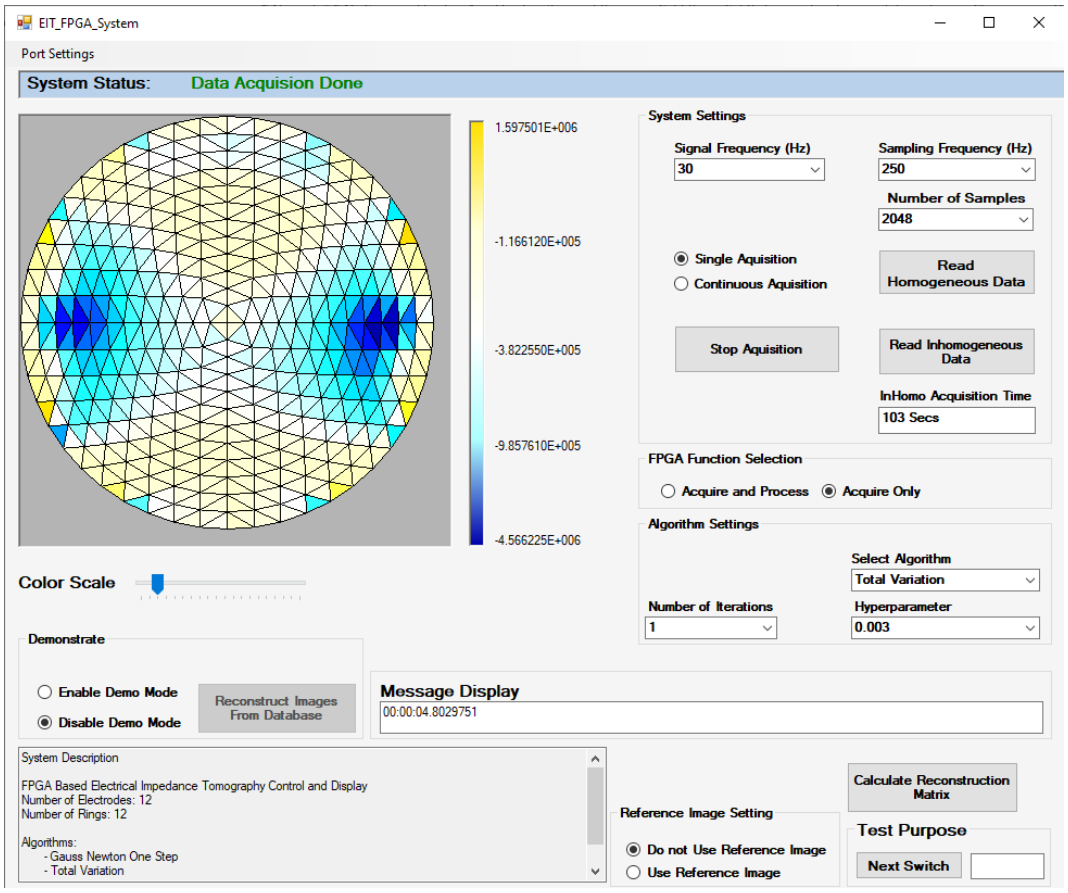


Figure 10. 13 Low Impedance Anomaly detection

Chapter 11: Discussion and Conclusion

Electrical Impedance Tomography imaging technique has a potential to be used in various biomedical applications. As mentioned previously in this document, there are various manufacturers who provide EIT systems for commercial clinical use. An EIT system includes a signal source and data acquisition hardware and a processing unit to reconstruct a conductivity image using the measurement data. As the reconstruction computation involves complex algorithm implementation, it is often implemented on computers. In this project, a Zynq SoC, along with some custom designed hardware, is used to build a complete EIT system, which includes signal source, data acquisition and image reconstruction capabilities.

A twelve electrode EIT system, with signal frequency options 2Hz to 10kHz and reconstruction algorithm options Gauss Newton One Step and Total Variation is developed. The system includes a Zedboard, Analog Front End (AFE) board and a C# GUI. The AFE board is designed by a senior design team and used in this project with some modifications. The AFE board includes required hardware to drive the current signal and digitize the acquired voltage measurement signals. It is modified so that it can work with lower frequency range 1Hz to 30Hz. The Zynq SoC design includes signal generation, control of signal application and voltage measurement pattern, data acquisition and processing logic. It constitutes of a VIVADO design for generating a SoC hardware and an SDK design. The GUI is designed to receive data from Zynq SoC over serial port, display the image data and process the measurement data if required. The EIT reconstruction algorithms are implemented in Zynq SoC as well as C#. This way the efficiency of Zynq SoC implementation is evaluated.

The system provides various controls over system parameters as well as image algorithms. The system is tested by varying the control parameters and observing the results. The image reconstruction at different frequencies is observed. As per the input frequency, different sampling frequency and number of samples are used for data acquisition. It is observed that when the number of cycles acquired for an input signal is more than sixty, the low impedance anomaly can be observed in the reconstructed image. The reconstruction result using the single step Gauss Newton One Step algorithm and iterative Total Variation algorithm is observed. The results observed for 2Hz and 30Hz are shown in chapter 10. It is observed that, the Total Variation algorithm detects the low impedance anomaly more precisely. The system is also tested for detecting impedance changes at two different locations in the phantom. It is observed that the system can detect, a high impedance and low impedance change at one time in the given setting. The reconstructed images are observed at 2Hz, 4Hz, 10Hz, and 30Hz using Gauss Newton One Step and Total Variation algorithms. It is observed that there is not much change in the low impedance anomaly detection for all the frequencies, for this tissue phantom. For low frequencies like 2Hz and 4Hz, the number of samples are required to be kept as 8192 and sampling frequency 250Hz. In case of Total Variation algorithm, the hyperparameter can be changed but in this testing process, it is not changed. This is because, in case of TV algorithm, the initial solution is calculated using the stored reconstruction matrix which uses 0.003 as hyperparameter.

The FPGA implementation reports, as shown in chapter 10, shows that the device utilization for LUTs is reduced and the timing constraints performance is much improved if compared to the previous FPGA based design implemented by senior design team. Also, in the

previous design, the Gauss Newton One Step image computation in FPGA takes around 100ms, however, in the current FPGA implementation, the same computation takes around 2ms time. The device utilization performance improvement is achieved by using SD-card, DDR and DMA for reconstruction matrix storage and transfer in FPGA only as per requirement. The timing constraints performance improvement is achieved by using pipelined implementation for the multiply and accumulate logic. Hence, it can be said that the timing performance of the Gauss Newton One Step FPGA implementation has largely improved in this project. However, if the timing performance of the image reconstruction algorithms' implementation in SoC is compared with the similar implementation in C# software, the later performs much faster. From the observations shown in chapter 10, it can be said that even if one step image reconstruction algorithm implementations perform much faster in FPGA, the overhead of image data transfer makes the process slower. Also, the iterative Total variation algorithm computation involves large matrices computations, because of which the SoC implementation becomes much slower that the similar implementation done in C#.

From overall computation performance observations of the system, it can be said that the software implementation of image reconstruction algorithms gives better timing performance as compared to SoC implementation. However, in future, to develop an SoC based standalone EIT system with only one step algorithm computation, some modifications can be implemented. One of the modifications which can be done is to use ethernet or USB for data transfer between SoC and GUI. This way, the serial data transfer overhead can be reduced. Another modification which can be tried is to use Linux OS on Zynq processor and display the image on a screen through HDMI out available on Zedboard. Additionally, from the point of view of AFE

hardware, in future, the AFE hardware can be modified to work with a wide range of frequencies from 2Hz to 10kHz.

To conclude, in this project, a Zynq SoC based, twelve electrode, twelve ring EIT system is designed. To understand the EIT technology, a background research is performed. To understand the implementation requirements, the EIT implementations developed by previous St. Cloud State University students are studied. A MATLAB code using EIDORS is developed to understand the flow of image reconstruction process. A data acquisition system is designed using Zynq SoC and a GUI is designed using C# windows form application. A serial link is established between Zynq SoC and GUI for data transfer. A one step Gauss Newton One Step and an iterative Total Variation algorithm are developed in Zynq SoC as well as C#. For algorithm implementations in Zynq SoC, software - hardware co-design is used. The system is tested for performance. It is observed that the Gauss Newton One Step FPGA implementation, if compared with the senior design team implementation, is improved. For image reconstruction performance, it is observed that the iterative Total Variation algorithm gives the better result as compared to one step Gauss Newton algorithm. The computation time performance between C# implementation and SoC implementation is compared and it is observed that the C# implementation for both algorithms is much faster as compared to the implementation in Zynq SoC.

Bibliography

[1] R. P. Henderson and J. G. Webster, “An Impedance Camera for Spatially Specific Measurements of the Thorax,” *IEEE Trans. Biomed. Eng.*, vol. BME-25, no. 3, pp. 250–254, May 1978, doi: 10.1109/TBME.1978.326329.

[2] T. K. Bera, “Applications of Electrical Impedance Tomography (EIT): A Short Review,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 331, p. 012004, Mar. 2018, doi: 10.1088/1757-899X/331/1/012004.

[3] C. Putensen, B. Hentze, S. Muenster, and T. Muders, “Electrical Impedance Tomography for Cardio-Pulmonary Monitoring,” *J. Clin. Med.*, vol. 8, no. 8, p. 1176, Aug. 2019, doi: 10.3390/jcm8081176.

[4] B. Brown, “Electrical impedance tomography (EIT): a review,” *J. Med. Eng. Technol.*, vol. 27, no. 3, pp. 97–108, Jan. 2003, doi: 10.1080/0309190021000059687.

[5] J. Latikka, T. Kuurne, and H. J. Eskola, “Conductivity of living intracranial tissues,” *Phys. Med. Biol.*, vol. 46, no. 6, 2001.

[6] A. Adler *et al.*, “GREIT: a unified approach to 2D linear EIT reconstruction of lung images,” *Physiol. Meas.*, vol. 30, no. 6, pp. S35–S55, Jun. 2009, doi: 10.1088/0967-3334/30/6/S03.

[7] M. Fernández-Corazza, N. von-Ellenrieder, and C. H. Muravchik, “Estimation of electrical conductivity of a layered spherical head model using electrical impedance tomography,” *J. Phys. Conf. Ser.*, vol. 332, p. 012022, Dec. 2011, doi: 10.1088/1742-6596/332/1/012022.

[8] Sunjoo Hong, Jaehyuk Lee, and Hoi-Jun Yoo, “Wearable lung-health monitoring system with electrical impedance tomography,” in *2015 37th Annual International Conference of the*

IEEE Engineering in Medicine and Biology Society (EMBC), Milan, Aug. 2015, pp. 1707–1710, doi: 10.1109/EMBC.2015.7318706.

[9] R. Dubois, C. Burg, L. Hengel, and J. Kelzenberg, "Real-Time Electrical Impedance Tomography," Department of Electrical Engineering, St. Cloud State University, MN, USA.

[10] O. Gilad, L. Horesh, and D. S. Holder, "Design of electrodes and current limits for low frequency electrical impedance tomography of the brain," *Med. Biol. Eng. Comput.*, vol. 45, no. 7, pp. 621–633, Aug. 2007, doi: 10.1007/s11517-007-0209-7.

[11] W. R. B. Lionheart, "EIT Reconstruction Algorithms: Pitfalls, Challenges and Recent Developments," *Physiol. Meas.*, vol. 25, no. 1, pp. 125–142, Feb. 2004, doi: 10.1088/0967-3334/25/1/021.

[12] M. G. Crabb, "Convergence study of 2 *D* forward problem of electrical impedance tomography with high-order finite elements," *Inverse Probl. Sci. Eng.*, vol. 25, no. 10, pp. 1397–1422, Oct. 2017, doi: 10.1080/17415977.2016.1255739.

[13] E. J. Woo, P. Hua, J. G. Webster, and W. J. Tompkins, "A robust image reconstruction algorithm and its parallel implementation in electrical impedance tomography," *IEEE Trans. Med. Imaging*, vol. 12, no. 2, pp. 137–146, Jun. 1993, doi: 10.1109/42.232242.

[14] T. A. Khan and S. H. Ling, "Review on Electrical Impedance Tomography: Artificial Intelligence Methods and its Applications," *Algorithms*, vol. 12, no. 5, p. 88, Apr. 2019, doi: 10.3390/a12050088.

[15] B. Grychtol, N. Lesparre, W. R. B. Lionheart, and A. Adler, "EIDORS: the past, the present and the future," p. 4.

[16] F. Zamora-Arellano *et al.*, “Development of a Portable, Reliable and Low-Cost Electrical Impedance Tomography System Using an Embedded System,” *Electronics*, vol. 10, no. 1, p. 15, Dec. 2020, doi: 10.3390/electronics10010015.

[17] S. Russo, S. Nefti-Meziani, N. Carbonaro, and A. Tognetti, “A Quantitative Evaluation of Drive Pattern Selection for Optimizing EIT-Based Stretchable Sensors,” *Sensors*, vol. 17, no. 9, p. 1999, Aug. 2017, doi: 10.3390/s17091999.

[18] N. K. Soni, K. D. Paulsen, H. Dehghani, and A. Hartov, “Finite element implementation of Maxwell’s equations for image reconstruction in electrical impedance tomography,” *IEEE Trans. Med. Imaging*, vol. 25, no. 1, pp. 55–61, Jan. 2006, doi: 10.1109/TMI.2005.861001.

[19] X. Chen, “STUDY ON HUMAN BRAIN ACTIVITY AND ELECTRICAL IMPEDANCE TOMOGRAPHY,” Department of Electrical Engineering, St. Cloud State University, MN, USA, Master’s Thesis, Dec. 2012.

[20] J. Padilha Leitzke and H. Zangl, “A Review on Electrical Impedance Tomography Spectroscopy,” *Sensors*, vol. 20, no. 18, p. 5160, Sep. 2020, doi: 10.3390/s20185160.

[21] W. Zhang and D. Li, “An instrumental electrode model for solving EIT forward problems,” *Physiol. Meas.*, vol. 35, no. 10, pp. 2001–2026, Oct. 2014, doi: 10.1088/0967-3334/35/10/2001.

[22] M. N. O. Sadiku, *Numerical Techniques in Electromagnetics with MATLAB*, Third Edition

[23] M. N. O. Sadiku, “A simple introduction to finite element analysis of electromagnetic problems,” *IEEE Trans. Educ.*, vol. 32, no. 2, pp. 85–93, May 1989, doi: 10.1109/13.28037.

[24] T. K. Bera and J. Nagaraju, “A FEM-Based Forward Solver for Studying the Forward Problem of Electrical Impedance Tomography (EIT) with A Practical Biological Phantom,” in *2009 IEEE International Advance Computing Conference*, Patiala, India, Mar. 2009, pp. 1375–1381, doi: 10.1109/IADCC.2009.4809217.

[25] T. J. Yorkey, J. G. Webster, and W. J. Tompkins, “Comparing Reconstruction Algorithms for Electrical Impedance Tomography,” *IEEE Trans. Biomed. Eng.*, vol. BME-34, no. 11, pp. 843–852, Nov. 1987, doi: 10.1109/TBME.1987.326032.

[26] C. Gómez-Laberge and A. Adler, “Direct EIT Jacobian calculations for conductivity change and electrode movement,” *Physiol. Meas.*, vol. 29, no. 6, pp. S89–S99, Jun. 2008, doi: 10.1088/0967-3334/29/6/S08.

[27] S. Liu, J. Jia, Y. D. Zhang, and Y. Yang, “Image Reconstruction in Electrical Impedance Tomography Based on Structure-Aware Sparse Bayesian Learning,” *IEEE Trans. Med. Imaging*, vol. 37, no. 9, pp. 2090–2102, Sep. 2018, doi: 10.1109/TMI.2018.2816739.

[28] Md. R. Islam and Md. A. Kiber, “Electrical Impedance Tomography imaging using Gauss-Newton algorithm,” in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, Dhaka, Bangladesh, May 2014, pp. 1–4, doi: 10.1109/ICIEV.2014.6850719.

[29] B. M. Graham and A. Adler, “Objective selection of hyperparameter for EIT,” *Physiol. Meas.*, vol. 27, no. 5, pp. S65–S79, May 2006, doi: 10.1088/0967-3334/27/5/S06.

[30] Z. Zhou *et al.*, “Comparison of total variation algorithms for electrical impedance tomography,” *Physiol. Meas.*, vol. 36, no. 6, pp. 1193–1209, Jun. 2015, doi: 10.1088/0967-3334/36/6/1193.

[31] A. Borsic, B. M. Graham, A. Adler, and W. R. B. Lionheart, “Total Variation Regularization in Electrical Impedance Tomography,” p. 35.

[32] T. Dai and A. Adler, “EIT Reconstruction Using Total Variation Norms for Data and Image Terms,” p. 5.

[33] R. Gartaula, “Image Reconstruction and EIT,” St. Cloud State University, Minnesota, Starred Paper, Sep. 2014.

[34] H. R. Ferreira, H. I. A. Bustos, and W. B. Figuerola, “Simulation inverse problems of reconstruction of image data using patterned electrical impedance tomography female breast,” in *2014 IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Natal, Oct. 2014, pp. 1–6, doi: 10.1109/HealthCom.2014.7123253.

[35] T. Dai and A. Adler, “Electrical Impedance Tomography reconstruction using ℓ1 norms for data and image terms,” in *2008 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vancouver, BC, Aug. 2008, pp. 2721–2724, doi: 10.1109/IEMBS.2008.4649764.

[36] V. Sarode, S. Patkar, and A. N Cheeran, “Comparison of 2-D Algorithms in ElT based Image Reconstruction,” *Int. J. Comput. Appl.*, vol. 69, no. 8, pp. 6–11, May 2013, doi: 10.5120/11860-7642.

[37] “Instrumentation Amplifier AD8429 Datasheet.” Analog Devices.

[38] “Vivado Design Suite User Guide UG906.” Xilinx, Jun. 19, 2012.

[39] “Zynq-7000 SoC Technical Reference Manual UG585.” Xilinx, Jul. 01, 2018.

[40] “Zynq-7000 SoC Datasheet: Overview.” Xilinx, Jul. 02, 2018.

[41] "Zedboard, Zynq Evaluation and Development Board Hardware User's Guide." AVNET, Jan. 27, 2014.

[42] "Embedded System Tools Reference Manual, UG1043." Xilinx, Dec. 05, 2018.

[43] "VIVADO Design Suite User Guide, Using the VIVADO IDE, UG893." Xilinx, Apr. 04, 2018.

[44] J. Song *et al.*, "Electrical Impedance Changes at Different Phases of Cerebral Edema in Rats with Ischemic Brain Injury," *BioMed Res. Int.*, vol. 2018, pp. 1–10, Jun. 2018, doi: 10.1155/2018/9765174.

[45] S. Khalil, M. Mohktar, and F. Ibrahim, "The Theory and Fundamentals of Bioimpedance Analysis in Clinical Status Monitoring and Diagnosis of Diseases," *Sensors*, vol. 14, no. 6, pp. 10895–10928, Jun. 2014, doi: 10.3390/s140610895.

[46] "UG1118, VIVADO Design Suite User Guide, Creating and Packaging Custom IP." Xilinx, Mar. 02, 2020.

[47] "DS794 Xilinx LogiCORE IP DDS Compiler v5.0." Xilinx, Mar. 2011.

[48] "AD5293, Analog Devices Single Channel 1024 position potentiometer." Analog Devices.

[49] "AD7761, 8-channel,16-bit, Simultaneous Sampling ADC Datasheet." Analog Devices.

[50] "PG021, AXI DMA v7.1 LogiCORE IP Product Guide." Xilinx, Jun. 14, 2019.

[51] "UG1037, VIVADO Design Suite, AXI Reference Guide." Xilinx, Jul. 15, 2017.

[52] L. Crockett, R. Elliot, M. Enderwitz, and R. Stewart, *The Zynq Book, Embedded Processing with the ARM Cortex -A9 on the Xilinx Zynq-7000 All Programmable SoC*, 1st

ed. Department of Electronic and Electrical Engineering, University of Strathclyde, Glasgow, Scotland, UK.

Appendix

A) MATALB EIDORS Code

```
%EIDORS Code for 8 Electrode data, with opposite current Injection
%and Adjucent Measurements
%Monali Sinare
%September - 2020
 %Load EIDOR
run X:\696_Matlab\EIDORS\eidors-v3.10\eidors-v3.10\eidors\startup.m
 %Read Data from text files
homo = fopen('homo.txt','r');
inhomo = fopen('inhomo.txt','r');
homo_data = fscanf(homo,'%f');
inhomo_data= fscanf(inhomo,'%f');
fclose('all');
 n_elec = 8;       %number of electrodes
 Rings = input('Enter Number of Rings (4,8,12,16)');
if Rings == 4
    n_rings = 'a2c';
elseif Rings == 8
    n_rings = 'b2c';
elseif Rings == 12
    n_rings = 'c2c';
```

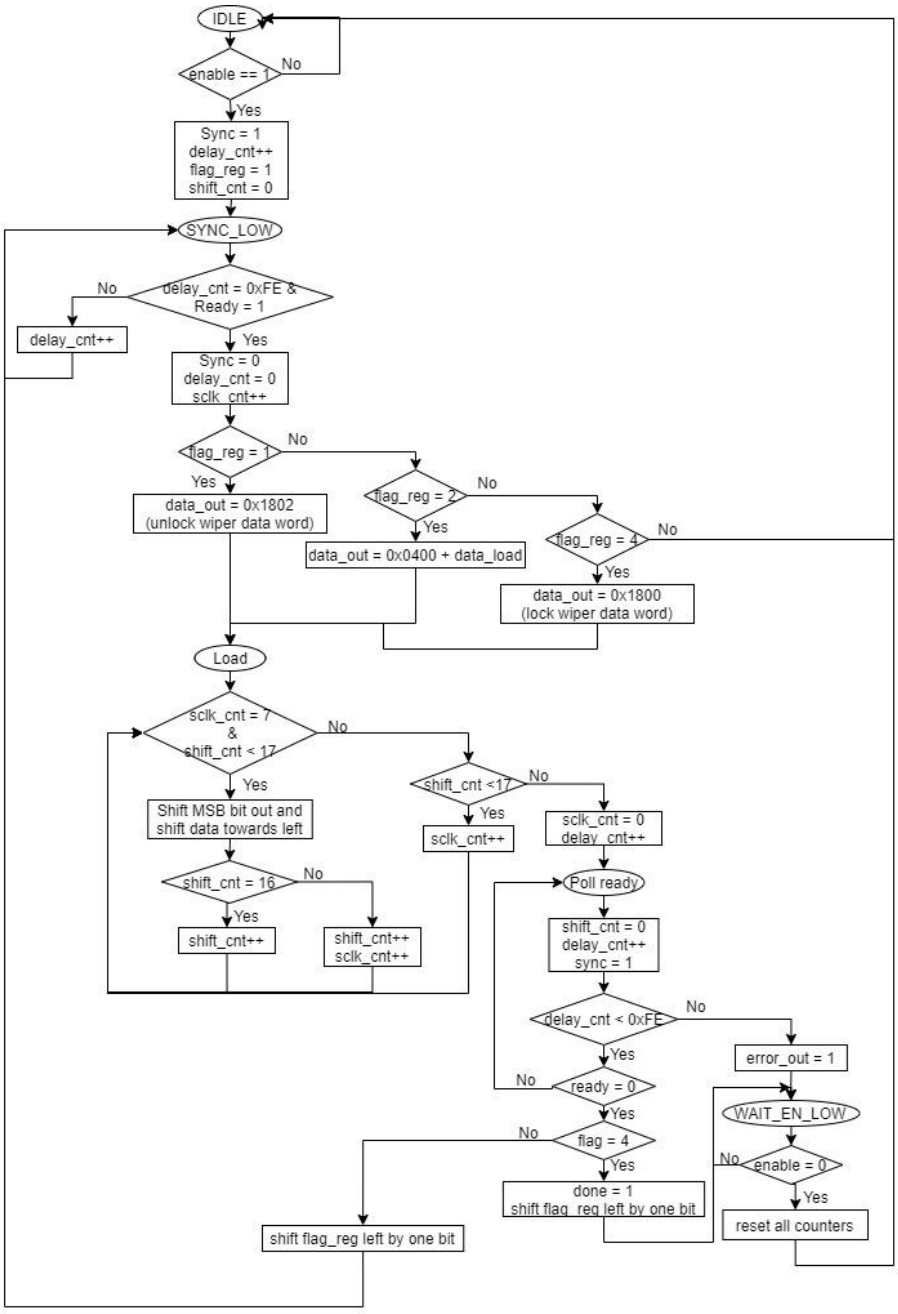
```
elseif Rings == 16
    n_rings = 'd2c';
else
    disp('Invalid Number Entered, Default set at : 8');
    n_rings = 'b2c';
end
 imdl = mk_common_model(n_rings,n_elec);   %create FEM model
%create stimulation pattern
options = {'no_meas_current'};  %No measurements taken on driving electrodes
ampl = 0.25; %current amplitude
[stim,meas_sel] = mk_stim_patterns(n_elec,1,'{op}','{ad}',options,ampl);
imdl.fwd_model.stimulation = stim;
imdl.fwd_model.meas_select = meas_sel;
 imdl.hyperparameter.value = 0.003;  % Assign Hyperparameter
 %Assign Algorithm
disp('Select one of the following Algorithms');
disp('1: Gauss Newton One Step');
disp('2: Total Variation');
 Algorithm = input('Enter Serial number of Algorithm : ');
 if Algorithm == 1
    imdl.solve = @inv_solve_diff_GN_one_step;
    disp_var_a = 'Gauss Newton One Step';
```

```
elseif Algorithm == 2
    imdl.solve = @inv_solve_TV_irls;
    disp_var_a = 'Total Variation';
    imdl.parameters.max_iterations = 3;
else
    disp('Invalid Number Entered, Default Set : Gauss Newton One Step');
    imdl.solve = @inv_solve_diff_GN_one_step;
    disp_var_a = 'Gauss Newton One Step';
end
disp('Select one of the following Priors');  %Assign Prior
disp('1: Laplace');
disp('2: Noser');
disp('3: Tikhonov');
Prior = input('Enter Serial number of Prior : ');
if Prior == 1
    imdl.RtR_prior = @prior_laplace;
    disp_var_p = 'Laplace';
elseif Prior == 2
    imdl.RtR_prior = @prior_noser;
     disp_var_p = 'Noser';
elseif Prior == 3
    imdl.RtR_prior = @prior_tikhonov;
```

```
    disp_var_p = 'Tikhonov';
else
   disp('Invalid Number Entered, Default Set : Laplace Prior');
   imdl.RtR_prior = @prior_laplace;
   disp_var_p = 'Laplace';
end
disp('calculating.....');
img = inv_solve(imdl, homo_data, inhomo_data);    %Reconstruct Image
show_fem(img);                  %Display the image
colorbar;
title("Output Image For " + disp_var_a + " Algorithm and " + disp_var_p + " Prior");
```

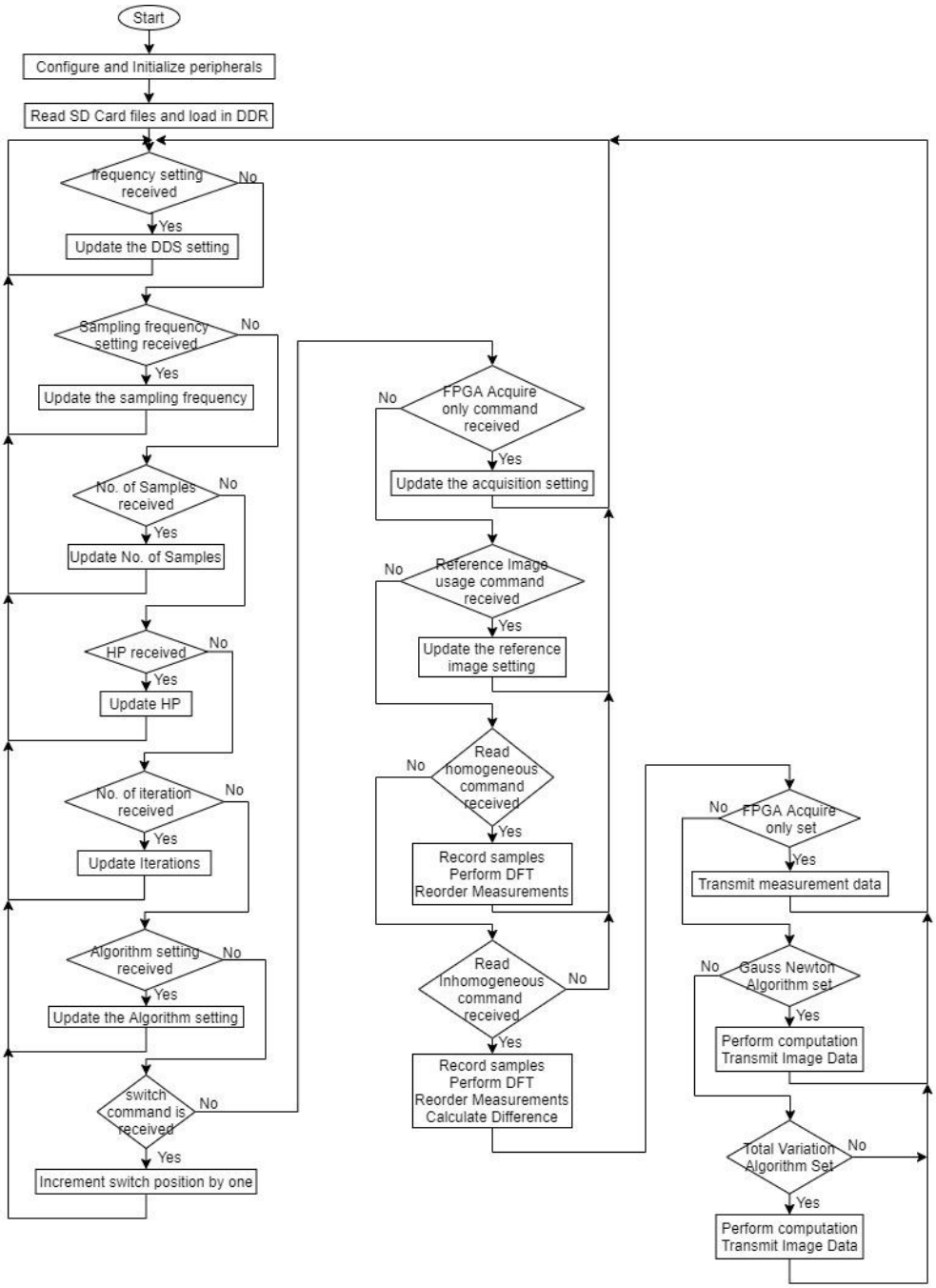
B) VIVADO Design Flowcharts

1) Digital pot state machine Verilog code flowchart

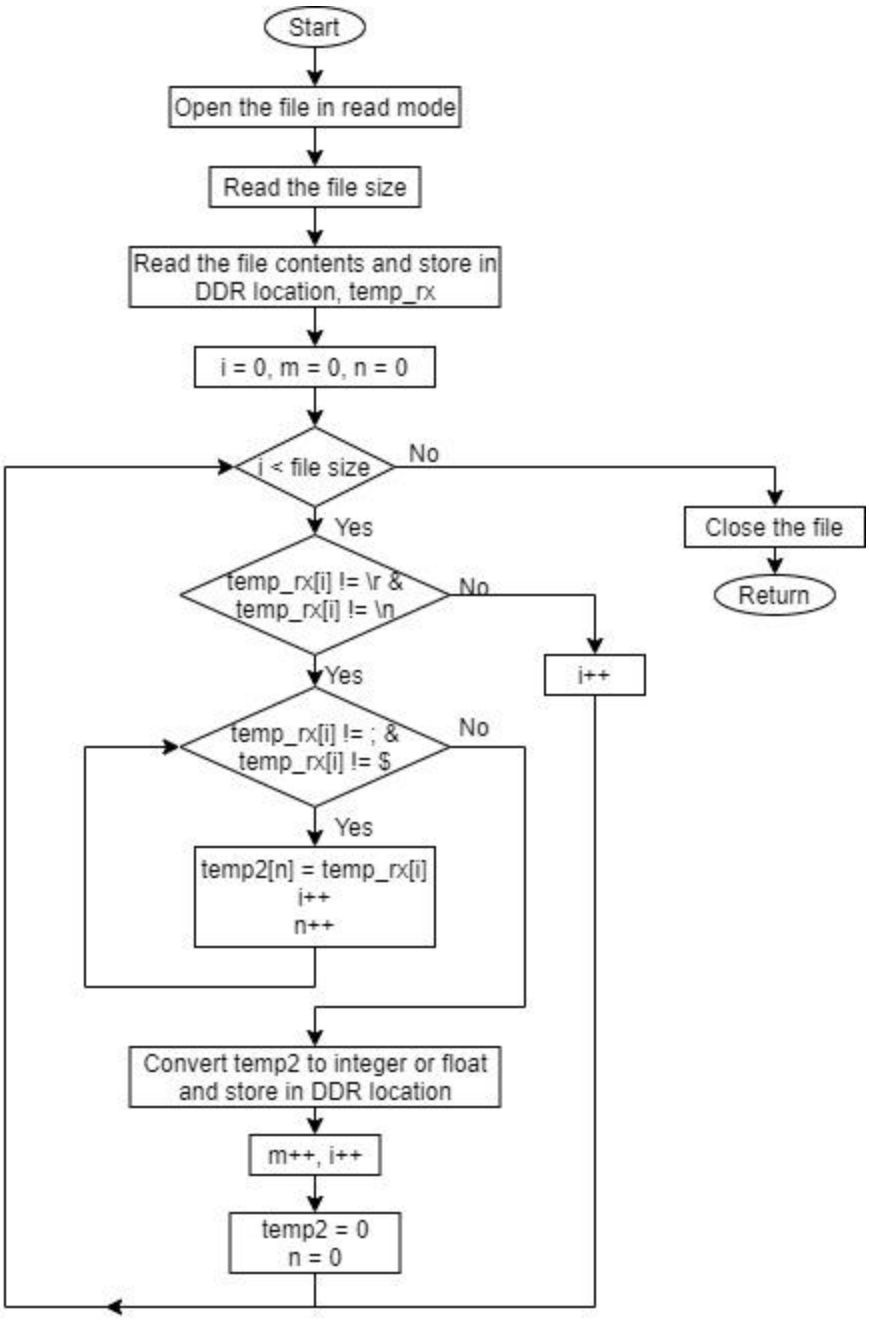


C) SDK code flowcharts

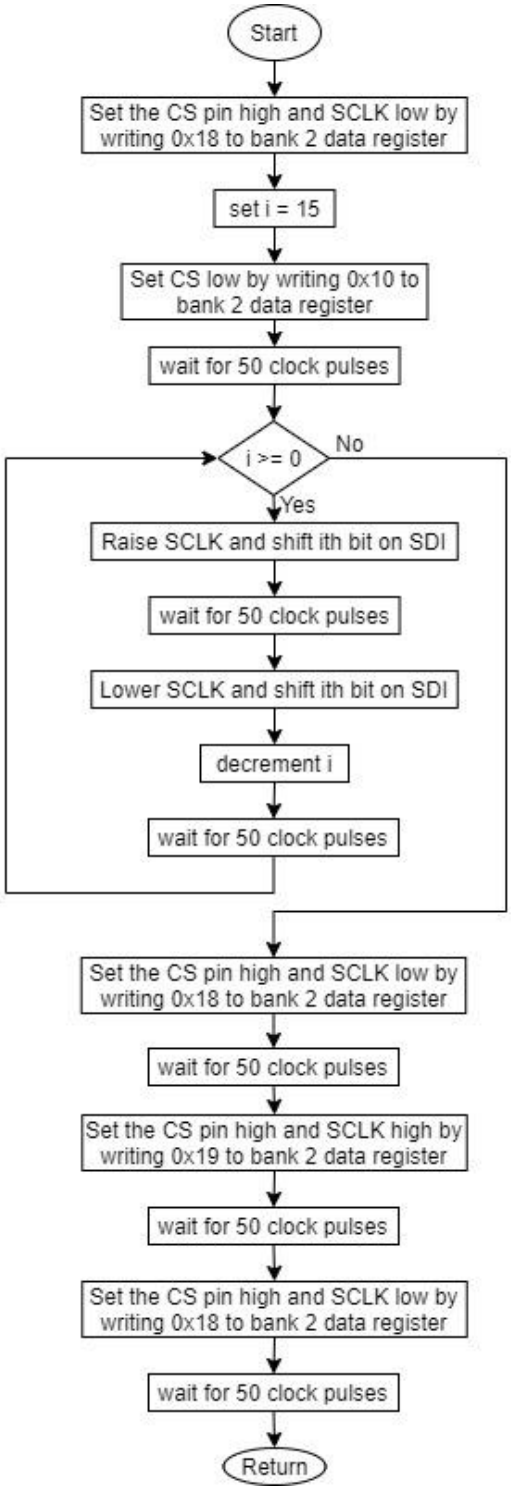
1) SDK main code functional flowchart



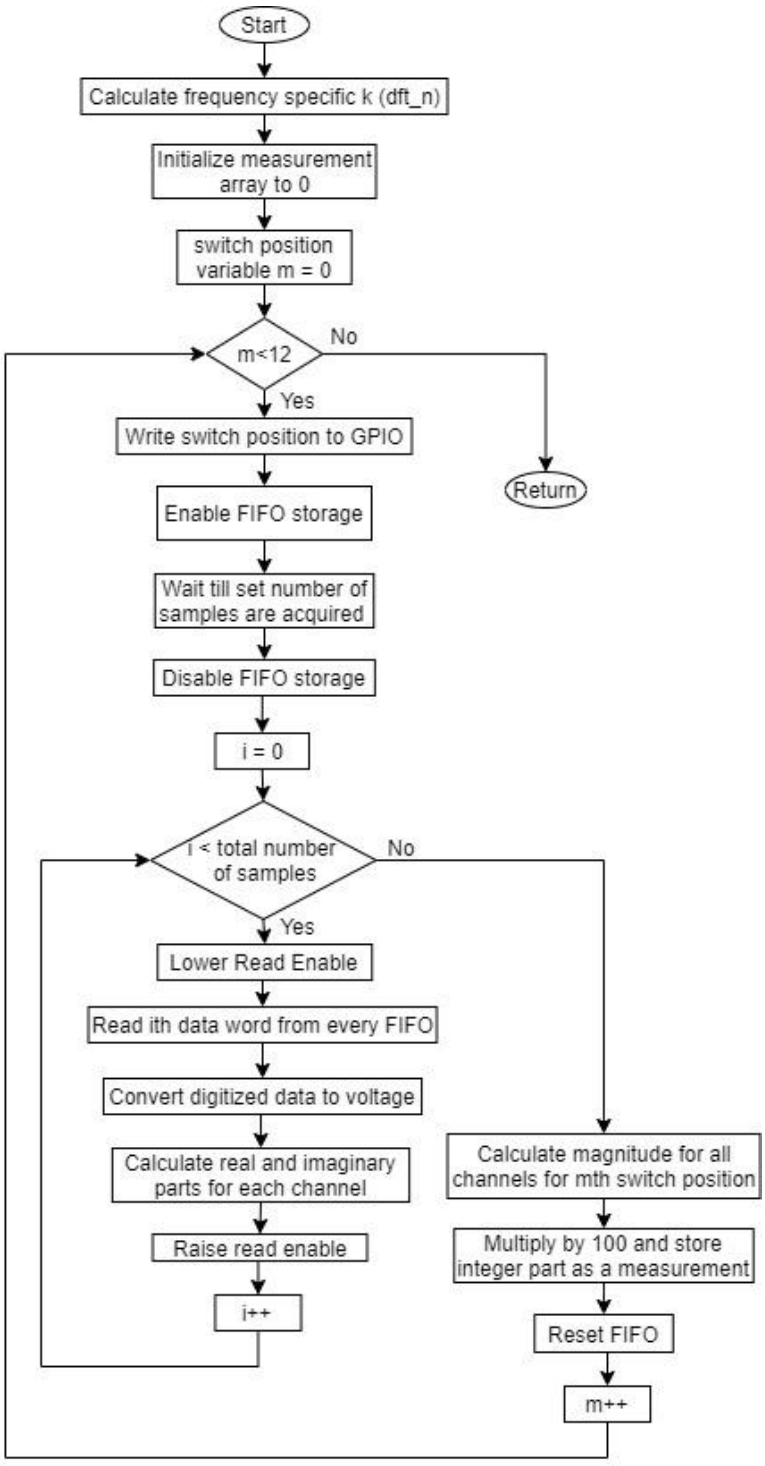
2) SD card *Readfile()* code flowchart



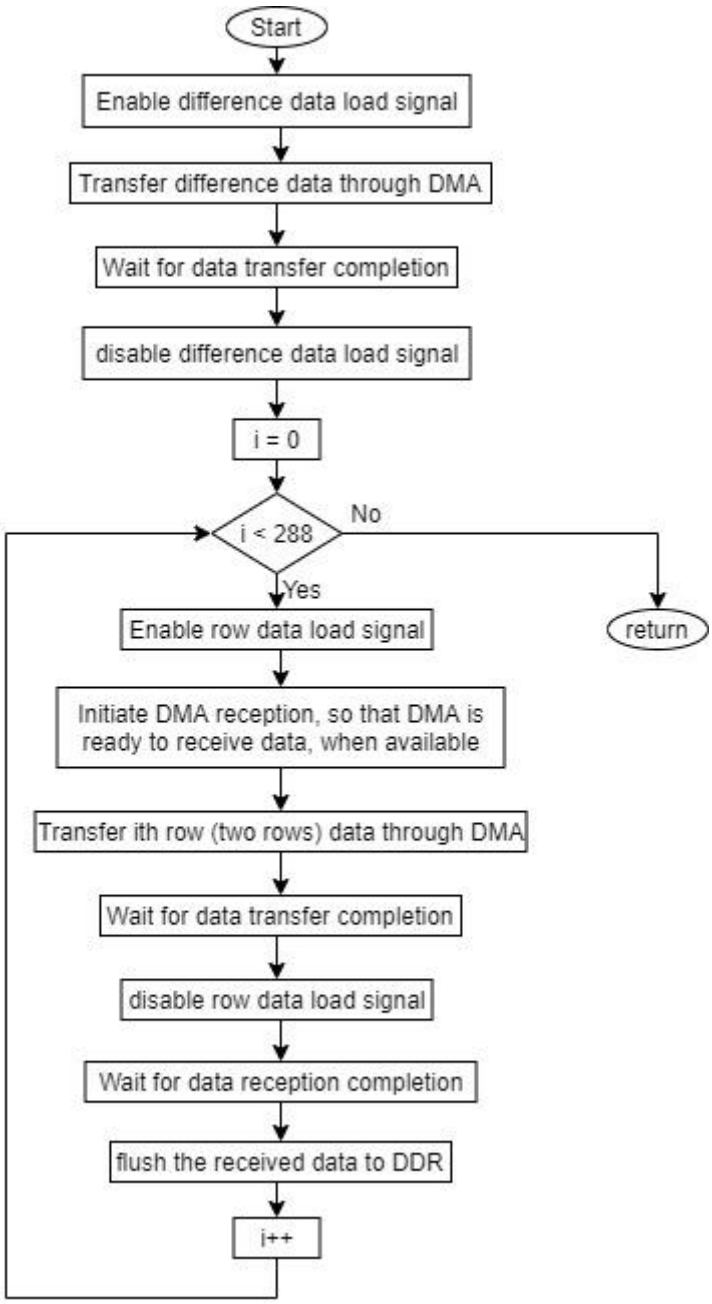
3) ADC SPI SDK implementation flowchart



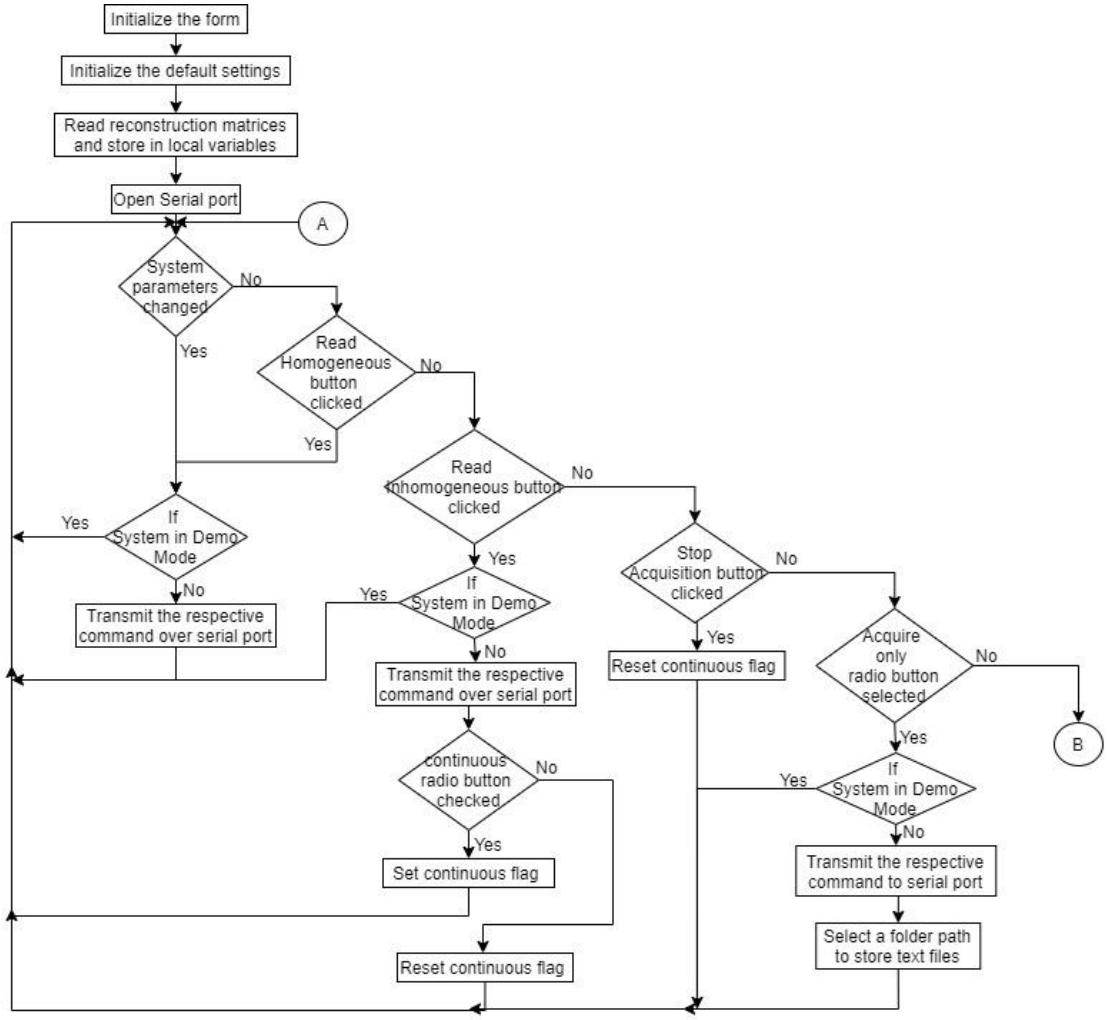
4) Data Acquisition loop flowchart

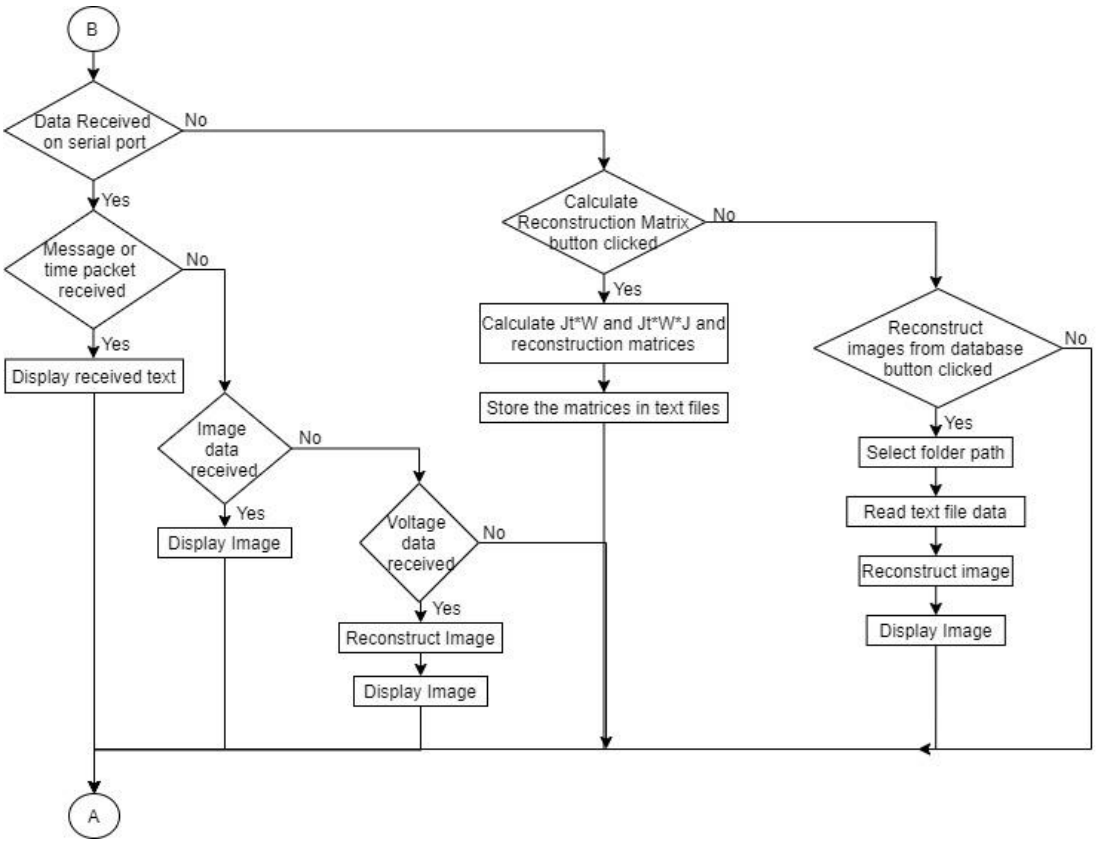


5) Multiplication loop flowchart



D) C# Implementation functional flowchart




B
Data Received on serial port
No
Yes
Message or time packet received
No
Yes
Display received text
Image data received
No
Yes
Display Image
Voltage data received
No
Yes
Reconstruct Image
Display Image
Calculate Reconstruction Matrix button clicked
No
Yes
Calculate Jt*W and Jt*W*J and reconstruction matrices
Store the matrices in text files
Reconstruct images from database button clicked
No
Yes
Select folder path
Read text file data
Reconstruct image
Display Image
A