Digital Hardware Implementation for Smart Portable Device for Water Quality Classification Using ANN-based Data Augmentation

THESIS

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

ABHEEK GUPTA

ID No. 2016PHXF0423P

Under the Supervision of

Prof. Anu Gupta

Under the Co-Supervision of

Sr. Prof. Rajiv Gupta

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

2024

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI – 333 031 (RAJASTHAN) INDIA

CERTIFICATE

This is to certify that the thesis entitled "**Digital Hardware Implementation for Smart Portable Device for Water Quality Classification using ANN based Data Augmentation**" submitted by **Abheek Gupta**, ID No. 2016PHXF0423P for the award of Ph.D. of the Institute, embodies original work done by him under my supervision.

Signature of the Supervisor	Signature of the Co-Supervisor
Name: Prof. Anu Gupta	Name: Prof. Rajiv Gupta
Designation: Professor	Designation: Senior Professor

Date:

Acknowledgement

I express sincere gratitude and indebtedness to my supervisor, Prof. Anu Gupta, Professor, Department of Electrical and Electronics Engineering, BITS Pilani, and co-supervisor Prof. Rajiv Gupta, Senior Professor, Department of Civil Engineering, BITS Pilani, Pilani Campus. They have been a constant source of inspiration throughout my work. The enthusiasm, moral support, and advice that they have given me will stimulate me to be the best in my endeavours. I sincerely acknowledge the help rendered by them at all times.

I express my profound sense of gratitude and indebtedness to the Principal Investigator of the WTI DST Project, Sr. Prof. Rajiv Gupta, Senior Professor, Department of Civil Engineering, BITS Pilani, Pilani Campus, for his valuable guidance and suggestions to do my research and the help he provided to me in the formulation of ideas for my thesis. I gratefully acknowledge the Department of Science and Technology, MHRD, Govt. of India, for the financial support in the form of a Junior Research Fellowship.

I express my heartfelt appreciation to BITS Pilani for providing all the necessary facilities and support to complete the research work. My special thanks to Prof. V. Ramgopal Rao, Vice-Chancellor of the University, and Prof. S. K. Barai, Director, BITS-Pilani, Pilani Campus, for allowing me to pursue my research work successfully. I also express my gratitude to Prof. S. K. Verma, Dean of Academic Research, and other faculty members of AGSRD for providing valuable support throughout the program.

I would like to take this opportunity to express my gratefulness to Dr Chandra Shekhar and Dr Arnab Hazra, who are members of the Doctoral Advisory Committee (DAC), for their kind suggestions, motivation, and moral support throughout the program, both technically and personally. I would like to thank Dr V. K. Chaubey, Ex-HoD, Department of Electrical and Electronics Engineering, Dr H. D. Mathur, Ex-HoD, Department of Electrical and Electronics Engineering, and Dr Navneet Gupta, HoD, Department of Electrical and Electronics Engineering, for their motivation during difficult times.

I am also thankful to all my professors for their support and motivation throughout my studies at BITS-Pilani, Pilani Campus. I would like to extend my thanks, to Dr. Navin Singh, Ex-Chief Warden, and Dr. Shibashish Choudhury, Ex-Chief Warden. Gratitudes are also due to all the non-teaching staff of the Department of Electrical and Electronics Engineering and Civil Engineering Mr. Tulsi Ram Sharma, Mr. Manoj Kumar, Mr. Mahesh Saini, Mr. Surrendar Kumar, Kamalji, Shivpalji, Sureshji, Shivratanji, Rameshji and Sultanji for their constant co-operation throughout my work. I also thank all my friends and colleagues, especially Dr. Dhananjay Kumar Mishra, Dr. Heema Dave, Dr. Puneet Khatri, Dr. Prateek Bindra, Dr. Ziaur Rahman, Dr. Gaurav Kumar, Dr. Farhan M Khan, Dr Soumya Kar, Dr Raya Raghavendra Kumar, Dr. Kanika, Mr. Girish Salaka, Dr Varun Jain, Dr Dhritabara Pal, Dr. Moyna Das, Dr. Pracheta Sengupta, Mrs. Bijoya Das, Dr Arghya Maiti, Ms. Soumana Joardar, Mr Utsav Jana and all others whose name are not here and have helped me during my Ph.D. Thanks are also due to all faculty and staff members of BITS-Pilani, Pilani Campus, for helping me out at various times.

I express my hearty gratitude and dedicate this thesis to my mother Late Mrs. Jayashree Gupta, who has been the sole motivation, influence, and reason for my Ph.D. My late Grandfather Dr. Nirendra Nath Sen, has been the motivation to pursue a Ph.D. My late aunt Mrs. Suparna Dasgupta stood as a pillar for me during the most difficult phase of the Ph.D. My father Mr Sisir Gupta, the most beautiful constant in my life. I would like to extend my thanks to all my friends in Pilani for their care and affection shown towards me in undertaking this journey. I owe thanks to my family for their love, encouragement and moral support, without which this work would have been an impossible task. My parents have always been supportive and their encouragement in all my endeavours from the beginning has always been a source of inspiration for me. I would cherish the happy moments spent at the BITS Pilani campus for my whole life.

Abheek Gupta

Abstract

The development of a classification device for a real-life application must meet severe challenges such as accuracy, cost economy, portability, and speed. Additionally, it must make it easier for both rural and urban public to make informed choices for a better quality of life, eliminating the need for human expertise and laboratory testing.

In this thesis, we have worked on developing a portable low-cost device taking Water Quality Classification as the application for real-time monitoring of ground and surface water resources. Water pollution is a major concern globally and needs immediate attention. The methodology developed in this work can also be applied to other natural resource monitoring applications.

Existing implementations of WQC are based on sensors for parameter measurement, which are expensive. They are also dependent on laboratory-based methods which are time-consuming and not suitable for application in regions with limited access to such well-equipped laboratories. Also, the majority of them are software-based implementations leading to high power consumption and low portability. Software-based approaches are also more complex to implement. Some hardware implementations also exist, but they are partly based on software approaches and involve conventional number representation systems for calculation. This increases the complexity of the design and reduces the accuracy of calculations. The conventional number system also increases hardware resources and power consumption in the case of ANN implementation.

We have addressed the above problems and proposed a device for water quality classification that is accurate, smart, portable, low cost, and power and resource efficient. The device is designed using two ANNs. To reduce the cost of the device, a novel ANN-based data augmentation method has been implemented which predicts the parameters whose measurement techniques are expensive. The Augmentation ANN takes two parameters – pH and Oxidation Reduction Potential (ORP) as inputs and predicts Dissolved Oxygen (DO) and Electrical Conductivity (EC) based on these parameters. The second ANN (Classification ANN) is used for the classification of water quality based on the four parameters. The Classification ANN takes four parameters – pH, ORP, DO, and EC as inputs and classifies the water sample into one of the three categories – Potable, Agricultural, and Wastewater. To

reduce the complexity of calculation, the Posit number system has been explored and used. For the hardware implementation of Posit, a novel parameterization method has been used to avoid unlimited hardware resource allocation. The results are compared with the standard IEEE 754 floating point representation system. The hardware resource requirement and power consumption of the proposed device were reduced by 50%, and the speed increased by 13.2%. The WQI device implemented achieves accuracy comparable to the standard Atlas Scientific lab kit with a 92% reduction in cost.

The complete design of the device has been implemented using two hardware design approaches – The embedded System design approach and the ASIC Design approach. The Embedded System approach gives faster and simpler design methodology and cheaper designs for low-volume production. The ASIC approach is more robust, power efficient, faster, and economical for mass production.

The ANNs have been trained using a data set that was made using the readings of standardized equipment from Atlas Scientific lab kit, Labtronics Multiparameter Water Analyser, and YSI Sonde. The prediction accuracy of Augmentation ANN was 98%, and the classification accuracy of the classification ANN was above 97% for each class averaging at 98% overall against the testing data set.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
WQI	Water Quality Indexing
WQC	Water Quality Classification
ANN	Artificial Neural Network
BOD	Biochemical Oxygen Demand
COD	Chemical Oxygen Demand
IOT	Internet of Things
LM	Levenberg Marquardt
MLP	Multi-Layer Perceptron
WSN	Wireless Sensor Networks
ALVINN	An Autonomous Land Vehicle in Neural Network
ASIC	Application-Specific Integrated Circuit
SoC	System-On-chip
BP	Backpropagation
CMOS	Complementary Metal Oxide Semiconductor
NaN	Not-a-Number
IEEE	Institution of Electrical and Electronics Engineers
IEEE 754	IEEE 754 Floating Point Number Representation
DO	Dissolved Oxygen

EC	Electrical Conductivity
Ah	Ampere hour
TDS	Total Dissolved Solids
TSS	Total Suspended Solids
pH	Potenz/Potential of Hydrogen
NARX	Nonlinear Autoregressive with Exogenous Input
MCPO	Multi Classification with Probabilistic Output
NN	Neural Network
Unum	Universal Number
ORP	Oxidation Reduction Potential
mV	milli Volts
DW	Drinking Water
SW	Surface Water
RBF	Radial Basis Function
SVM	Support Vector Machine
CoNN	Constructive Neural Networks
C-Mantec	Competetive Neural Network trained by Error Correction
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
LUT	Look-up Table

PPANN	Parameterised Posit ANN
ES	Exponent Size
Exp	Exponent
REM	Regime, Exponent, Mantissa
LSB	Least Significant Bit
MSB	Most Significant Bit
LOD	Leading One Detector
LZD	Leading Zero Detector
MUX	Multiplexer
RC	Regime Check
TSMC	Taiwan Semiconductor Manufacturing Company
RTL	Register Transfer Level
SWAT	Solid and Water Assessment Tools
HSPF	Hydrological Simulation Program – Fortran
SWMM	Stormwater Management Model
A-ANN	Augmentation ANN
C-ANN	Classification ANN
RMSE	Root Mean Square Error
MSE	Mean Square Error
VLSI	Very Large Scale Integration

Chapter 1

1. Introduction

1.1 Design Challenges for in-situ Monitoring of Natural Resources

In addressing real-world challenges, the development of a device aimed at serving the needs of diverse human populations, encompassing both rural and urban areas, is paramount. Such a device must confront a multitude of hurdles, including accuracy, cost-effectiveness, portability, and speed. Of particular significance is the challenge posed by portability, which requires careful consideration of power consumption and compactness. Additionally, the device must empower individuals in both rural and urban contexts to make informed decisions, thereby eliminating the reliance on specialized expertise and laboratory testing.

The pressing real-life challenges demanding urgent attention are intricately linked to the preservation of life-sustaining natural resources, namely air, water, and soil.

This thesis endeavours to address these challenges by focusing on the development of a device tailored for real-time monitoring of ground and surface water resources, with water quality indexing as the primary application. Such an undertaking is poised to facilitate the judicious utilization of water resources across various domains.

1.2 Water Quality Classification as a Case Study

Water is one of the most basic resources that is required to sustain life, along with food and air. Thus, the availability of safe drinking water is a major concern. Water pollution caused by industrial and municipal wastewater discharges, agricultural and urban runoff, and other human activities is a major concern on a global scale [1]. Infectious such as waterborne illnesses are the leading cause of death for children under five worldwide, and each year, more people die from contaminated water than from all other types of violence, including war [2]. Each year, unsafe water contributes to 2.2 million fatalities, mostly children under the age of five, and 4 billion instances of diarrhoea. This means that a child dies from diarrhoea every 15 seconds, or 15% of all child deaths each year. Diarrhoea, which claims the lives of about 500,000 kids annually in India alone, is the leading cause of childhood illness and mortality [3].

Water Quality parameters such as electrochemical, biological, etc, have varied methods of measurement, and the testing times can vary from instantly to up to 24 hours or even more in

some cases in laboratory environments. There is a need to reduce this time using a reliable device which does both in-situ parameter measurement as well as Water Quality Classification (WQC).

A Water Quality Index is a method for summarizing a description of the quality of drinking water from a water supply source [4] in both urban and rural scenarios. Water Quality Classification is a two-step process that involves the measurement of water quality parameters and then indexing the water based on their measured parameter values. The development of the WQC device has to meet severe challenges in terms of accuracy, cost economy, portability, and speed. Here portability poses further challenges to the power consumption and compactness of the device. Additionally, it is necessary to create a WQC device that makes it easier for both rural and urban publics to make decisions regarding water quality classification, eliminating the need for human expertise and laboratory testing.

1.3 Data Classification Methods and Their Challenges

1.3.1 Conventional Methods

Water quality classification is generally done using various mathematical indexing rules, such as the ones presented in [5] [6]. Tools such as Principal Component Analysis [7], Fuzzy Logic [8], etc. Mathematical and Fuzzy logic models for WQC have applications limited by the input parameters guidelines depending on geography, water source, intended usage, and sundry. Since the water quality indicators used to depict dynamic pollution sources are complex, multivariable, and connected nonlinearly, mathematical and fuzzy models may not be able to adequately represent the outcome.

The main drawback of traditional classification methods, like moving averages, is the pre-assumption of the linear relationship between input parameters and output classes. So, they fail to classify the data where the output classes depict non-linear and non-stationary characteristics. Hence, when applied for applications like air quality indexing, water quality monitoring and indexing, facial recognition, natural language processing, medical imaging analysis, etc., these methods lead to inaccurate classification results. Further, with an increase in data, classification using traditional methods becomes computationally tedious.

1.3.2 Artificial Intelligence-Based Methods

Recently, Artificial intelligence technologies that analyze multivariate water quality data through potent visualization capabilities have replaced traditional techniques [1]. In [9], it has been shown that Artificial Neural Networks (ANN) have more reliability than fuzzy logic.

ANN are a model of the Biological Neural Network which helps living beings perceive the patterns in their environment, classify them and learn from them to improve their response to subsequent stimuli. There are two methods of implementation of ANN – Software, and hardware. Both methods have their challenges and trade-offs.

The present work aims to implement a hardware architecture of a WQC device with an emphasis on low complexity and high speed of operation. It also aims to achieve reduced cost and low power consumption. In developing and developed nations, the previously mentioned features are crucial for monitoring and indexing essential life resources. Thus, the hardware architecture proposed in this thesis has been optimized for low-power and low-cost water quality classification enabling this smart technology to reach across economic boundaries of society.

1.4 Artificial Neural Networks - Design and implementation challenges

Most commonly, ANNs are used when the mapping between the inputs and the outputs is not linear, e.g.:

- Classification of input data sets into predefined classes.
- Prediction of future output based on a set of current input-output mappings.
- Clustering data into groups based on prior knowledge about the data.
- An ANN can be trained to remember particular data patterns and then associate input data patterns with the ones in the memory or discard the data pattern.

1.5 Applications of Artificial Neural Networks

1.5.1 Environmental Engineering

ANN has multiple uses in environmental engineering aspects. Since ANN help to model a relation between parameters that cannot be expressed through direct mathematical modelling, ANNs help in modelling and classifying environmental parameters that indicate the quality of natural resources but cannot be modelled by mathematical relations. Another usage of ANN is to predict the outcome of a future event based on existing data and knowledge of past events.

The prediction abilities of ANN are also used in Environmental engineering. Some examples of ANN being used as classification and indexing applications are - Water Quality Classification, Water Quality Monitoring, and Air Quality Indexing. The prediction abilities of ANN are used in applications such as Water/Air Quality Prediction, Weather forecasting.

1.5.2 Machine learning

ANN have found many uses through machine learning applications. With machine learning, ANNs impart machines the ability to learn and become more intelligent. These are machines that we interact with daily in our lives and provide more data to the systems with our interactions, making them better at understanding our behaviour and thus enabling us to help us in our day-to-day lives better. Some examples of ANN-based machine learning applications are social media which learn the type of content we interact with more and the type of content we do not and, over time refine our feeds to deliver to us the content that we like. Similarly, on e-commerce sites, the machine learning algorithm learns the products we look for more often, the products that we buy more often, and offers or other facilities related to such products are pushed to our devices just like our neighbourhood shopkeeper, who knows our buying habits. Some other recent examples of neural networks include image search on Google, Auto-tagging by Facebook or product recommendations on Amazon, and completely driverless automobiles from Tesla Motors.

1.5.3 Healthcare

ANN has found many uses in the healthcare sector. All the four functions of ANNs, vis. Classification, prediction, prediction, and associative memory have found many applications in the medical and healthcare sectors. Facial feature analysis has been used to develop pain management systems where patients' pain levels are monitored based on the facial expressions of the patients. The various imaging methods like CT scans, MRIs, X-ray imaging, etc., use ANN image analysis to identify abnormalities using object classification and feature extraction. Voice or speech recognition and improvement in hearing aids for the hearing impaired, Image/Video inference for visual aids for blind people, Learning enabled prosthetics, etc.

1.6 Challenges in the development of ANN Hardware

Major challenges faced are in reducing the cost, and lowering power consumption while maintaining accuracy, in a portable in-situ device implementation.

The Implementation of ANN on hardware with the goals mentioned above comes with some challenges.

1.6.1 Software Approach

The most common method of implementing ANNs is the Software approach. Most ANN based models are implemented on software using a computer. For applications based on environmental interactions, the data from various sensors are collected and sent to these computing centers. These computers then perform the ANN computations and generate the required output.

The software code must decode down to a hardware level to be able to interact with the real world, it must go through multiple abstractions of computer architecture. This renders the process to become slow and limits the portable application of the ANN though it can achieve higher classification accuracy.

1.6.2 Hardware Approach

Hardware-based implementations, such as Application Specific Integrated Circuit (ASIC) or System-on-Chip (SoC), of an ANN algorithm make the system much more resource and time-efficient because of the application-specific nature of the hardware. A direct hardware implementation also improves the speed of the system. Hardware implementation is also economical because of its specificity.

While implementing logic on hardware, there are two approaches: - *Analog and Digital.* Morgan, et al [11], present the key advantages that a digital approach has over analog.

a) Analog Implementation of ANN

Biological neural networks which function on inaccurate components serve as models for ANN algorithms. Ideally, analog circuits which have an infinite resolution (continuous sampling) should serve as the better implementation option. Practically though, there is a limit to the representation of the smallest resolution on a circuit. i.e., the sampling rate of a circuit must be finite, thereby limiting the accuracy. Moreover, a multitude of ANN algorithms models the biological neural network very poorly and thus frequently need a wide dynamic range to bring about convergence to useful solutions. Popular stochastic algorithms such as backpropagation (BP) require 12 – 16 bit of

range or roughly four orders of magnitude between the largest weight value and the smallest weight change. The resolution required to achieve convergence cannot be obtained from analog circuits.

Secondly, the calculation of functions can be done by device physics by clever analog designing, but these designs are more dependent on circuit size than scalable CMOS (Complementary Metal Oxide Semiconductor) digital designs. Particularly, thermal noise power can be assumed to be proportional to inverse of the length unit (λ). Thus, for very small circuits the noise levels can be very high and render the circuit unreliable. Hence, as circuit sizes decrease, the digital approach offers better prospects for performance improvement [11] [12].

Thirdly, connectivity between elements on a chip is a big limitation. Particularly, for ANNs, they require huge multiplexing. Multiplexing analog signals is a design and resource intensive process. Interconnecting analog circuits at the board level is complicated because of radiated noise, power supply isolation problems, crosstalk, etc [11] [12]. Digital connections, on the other hand, tend to be rather reliable, comparatively. Since a neural network generally consists of more than one neuron, crosstalk needs special consideration when implementing computational multiplexing. Digital circuitry demands more Silicon area. When implementing changes in the algorithm, the analog circuit requires major redesign efforts. Digital circuits can accommodate such changes by implementing a different logic array.

Interfacing with peripheral circuits is difficult for analog implementation since most peripheral computational units are digital, nowadays. ANNs are subject to Amdahl's Law, as per which, the speed improvements of an analog neuron will stay untapped if the remaining network is implemented by slower peripheral implementations. So, a digital implementation improves compatibility with other systems, as compared to an analog circuit.

b) Digital Implementation of ANN

However, since the implementation of ANN involves the implementation of complex mathematical structures in the neuron and learning phase, the digital hardware implementation of such complex mathematical structures complicates the design. In particular, the implementation of the activation function involves the implementation of two very complex algorithms – division and exponent calculation. Since, the exponent function involves infinite series and real numbers, encoding either using hardware becomes a challenge. Hence, we adopted mathematical approximation methods to implement these operations in the neurons. Simplifying the neuron helped us simplify the implementation of the ANN on the hardware. The methods adopted till this point rely on the representation of real numbers as floating-point

numbers in the digital domain. Most conventional systems utilise IEEE 754 Floating point representation as the standard representation system.

Cost Efficieny of Design

Another major challenge in keeping the cost of the design low for applications like water quality monitoring, air quality indexing, which have implications for all sections of society.

The design cost is majorly impacted by the cost of the sensors required in real-world applications where the data has to be collected through interaction with the physical world. In such cases, most sensing technologies that can provide reliable data are generally expensive e.g. for water quality applications, the sensors for Dissolved Oxygen (DO), and Electrical Conductivity (EC). Many attempts have been made to use mathematical data augmentation to augment expensive sensor data using available field data [13]. Alternatively, ANN can also be used for data augmentation where there is no apparent mathematical relationship between the input and output parameters. This saves from complex operations of mathematical data augmentation. In this work, ANN has been used to augment the data of DO and EC of water to reduce the cost of design.

1.6.3 Mixed Software-Hardware ANN Implementation

There have been some studies about the mixed approach to reap the benefits of both hardware and software approaches. [14] shows one such approach where hardware and software are both used to design an ANN on an FPGA. The shortcomings in this approach are –

- The FPGA-based application is not suitable for mass production since FPGA boards are very expensive.
- The presence of a software module makes the design resource-intensive and less power-efficient because it needs a full computation unit for execution.

The preceding discussion underscores the necessity for tailoring the design of Artificial Neural Networks (ANNs) to align with specific application requirements. Consequently, within the scope of this study, directed towards the classification of water quality, a digital hardware implementation integrating suitable number representation techniques and data augmentation methodologies has been devised. This development aims to meet the demands for a low-cost, low-power portable ANN solution while maintaining an acceptable level of accuracy.

1.7 Challenges in Number Representation System for ANN Implementation

Since 1985, IEEE 754 Floating point representation has been the only way to represent floating point numbers in digital hardware architecture design.

Despite all its benefits, IEEE 754 representation has many limitations as described below:

- IEEE 754 has a rigid arrangement. This causes huge bit streams even for smaller numbers.
- The partitions of sign, exponent, and mantissa are fixed in size, this limits precision.
- Limited precision leads to rounding errors in the representation of some numbers.
- Additional bits have to be reserved during computations to adjust for rounding errors.
- Many reserved patterns are reserved for Not-a-Number(NaNs), denormals, +/- infinity, and other specials.
- Arithmetic inconsistencies occur due to reserved patterns

  Examples: -

  - One such inconsistency is the possibility of +/- 0representation. IEEE 754 guidelines state +0 = -0, which implies +1/0 = -1/0 => + infinity = - infinity.
  - Another inconsistency is noted in case of bracketed operations – say, x = 1e30; y = -1e30; z = 1. Then, (x + y) + z = 1, while x + (y + z) = 0.
  - Further inconsistency appears in dot products – A = [3.2e7, 1, -1, 8.0e7], and B = [4.0e7, 1, -1, -1.6e7]. In IEEE 754 notation the dot product A.B = 1, while in normal mathematics, A.B = 2.
- The precise and computed results may deviate as a consequence of IEEE 754 arithmetic inadvertently. The majority of the time, this inaccuracy appears to be harmless and even acceptable (for example, energy efficiency and accuracy are trade-offs in approximate computing techniques. In contrast, a number of works[16, 17, 18] have produced crucial errors in arithmetic expressions that FP arithmetic evaluates to drastically inaccurate results. Moreover, neglecting rounding errors has resulted in severe errors in some real-world instances, such as the 1991 Patriot missile battery failure.

To overcome these challenges, many other representation methods, like Unum 1, Unum 2, and Unum3, have been proposed over the years, each having its advantages and drawbacks, to improve or replace the IEEE 754 Floating Point representation. Thus, an optimum choice of

number representation through their exploration is crucial for efficient and accurate hardware implementation for a given application.

1.8 Organization of Thesis

The Thesis has been organized as follows:

Chapter 2 presents the literature survey done for the study.

Chapter 3 presents a discussion on the collection of various water quality parameter data from different water samples. The various parameters used for Water Quality Classification in the study and how the data for each of these parameters were collected. This discussion is included prior to the discussion on system design because this data has been used as the training and validation data for the system design. This maintains the logical flow in the organization of this thesis.

Chapter 4 delves into the digital hardware implementation of Artificial Neural Networks (ANNs) utilizing the conventional number representation system. Within this chapter, mathematical approximation techniques for the sigmoid function, serving as the activation function for the implemented ANN, are elucidated. Herein, we identify the optimal implementation of an ANN employing the sigmoid activation function, leveraging the conventional IEEE 754 Floating Point representation system.

Chapter 5 explores the implementation of Artificial Neural Networks (ANNs) utilizing a Parametrized Posit number system. Within this chapter, we delve into an examination of diverse floating-point number representation systems, ultimately opting to employ the Posit representation system for the realization of an ANN tailored to classify water quality parameters.

Chapter 6 presents the comprehensive design of a Portable Smart Water Quality Classification device utilizing ANN-based Data Augmentation. Within this chapter, we expound upon the hardware implementation of two distinct ANNs integrated into the device. The first ANN serves the purpose of Data Augmentation, aimed at mitigating device cost. This segment examines the utilization of data augmentation as an alternative to costly and intricate Dissolved Oxygen and Electrical Conductivity sensors. Due to the absence of a direct correlation between these parameters and the unfeasibility of implementing mathematical

models, we opt for an ANN-based approach utilizing pH and Oxidation Reduction Potential as input parameters.

The Data Augmentation ANN extrapolates Dissolved Oxygen and Electrical Conductivity values based on pH and Oxidation Reduction Potential inputs, which are readily obtainable through simple potential measurements. Subsequently, the second ANN is employed to classify water into three distinct categories: Potable, Agricultural, and Wastewater. The design implementation encompasses two methodologies: the Embedded Systems approach and the Application Specific Integrated Circuit (ASIC) design approach.

Chapter 7 concludes the findings of the study and paves the path for future studies that can be done in this field.

Appendices A, B, and C contain the Verilog and Python codes for the ANNs designed and implemented.

1.9 References

[1] S. Wechmongkhonkon, N. Poomtong and S. Areerachakul, "Application of Artificial Neural Network to Classification Surface Water Quality," *World Academy of Science, Engineering and Technology,* vol. 6, no. 9, pp. 199 - 203, 2012.

[2] World Health Organisation, "The World Health Report 2002: Reducing Risks, Promoting Healthy Life," The World health Organisation, Geneva, 2002.

[3] World Health Organisation and United Nations Children's Fund, "Global water supply and sanitation assessment 2000 report," WHO and UNICEF Joint Monitoring Programme for Water Supply and Sanitation, 2000.

[4] G. o. N. a. Labrador, "Drinking Water Quality Index - Environment and Climate Change," Government of Newfoundland and Labrador, Canada, 2022. [Online]. Available: https://www.gov.nl.ca/ecc/waterres/quality/drinkingwater/dwqi/. [Accessed 13 February 2023].

[5] R. O. A. Adelagun, E. E. Etim and O. E. Godwin, "Application of Water Quality Index for the Assessment of Water from Different Sources in Nigeria," in *Promising Techniques for Wastewater Treatment and Water Quality Assessment*, IntechOpen, 2021.

[6] M. G. Uddin, S. Nash and A. I. Olbert, "A review of water quality index models and their use for assessing surface water quality," *Ecological Indicators,* vol. 122, no. 107218, 2021.

[7] M. Tripathi and S. K. Singal, "Use of Principal Component Analysis for parameter selection for development of a novel Water Quality Index: A case study of river Ganga India," *Ecological Indicators,* vol. 96, no. 1, pp. 430 - 436, 2019.

[8] B. V. Raman, R. Bouwmeester and S. Mohan, "Fuzzy Logic Water Quality Index and Importance of Water Quality Parameters," *Air, Soil and Water Research,* vol. 2, pp. 51 - 59, 2009.

[9] R. Trach, Y. Trach, A. Kiersnowska, A. Markiewicz, M. Lendo-Siwicka and K. Rusakov, "A Study of Assessment and Prediction of Water Quality Index Using Fuzzy Logic and ANN Models," *Sustainability,* vol. 14, no. 9, p. 5656, 2022.

[10] D. A. Pomerleau, "ALVINN: An Autonomous Land Vehicle in a Neural Network," in *Advances in Neural Information Processing Systems*, Morgan-Kaufmann, 1988, pp. 305 - 313.

[11] N. Morgan, Considerations for Electronic Implementation of Artificial Neural Networks, Berkley, California: International Computer Science Institute, 1990.

[12] N. Morgan, K. Asanovic, B. Kingsbury and J. Wawrzynek, "Developments in Digital VLSI Design for Artificial Neural Networks," University of California, Berkley, Berkley, California, 1990.

[13] J. Kim, D. Seo, M. Jang and J. Kim, "Augmentation of limited input data using an artificial neural network method to improve the accuracy of water quality modeling in a large lake," *Journal of Hydrology,* vol. 602, no. 126817, 2021.

[14] S. Nambi, S. Ullah, S. S. Sahoo, A. Lohana, F. Merchant and A. Kumar, "ExPAN(N)D: Exploring Posits for Efficient Artificial Neural Network Design in FPGA-Based Systems," *IEEE Access,* vol. 9, pp. 103691 - 103708, 2021.

[15] P. Gerald, "Water science. University of Washington.," *PMC [serial on the internet],* 2011.

[16] D. Molden, Water for Food Water for Life: A Comprehensive Assessment of Water Management in Agriculture, London: Routeledge, 2007.

[17] M. M. David and B. E. Haggard, "Development of Regression-Based Models to Predict Fecal Bacteria Numbers at Select Sites within the Illinois River Watershed, Arkansas and Oklahoma, USA," *Water, Air, and Soil Pollution,* vol. 215, pp. 525 - 547, 2011.

[18] World Health Organisation, "Guidelines for Drinking-water Quality (Fourth," World Health Organisation, Geneva, Switzerland, 2017.

[19] "Introduction to Artificial Neural Networks (ANN): Secret mincontrol in Sweden and worldwide," mindcontrolinsweden.wordpress.com, 30 01 2015. [Online]. Available: https://mindcontrolinsweden.wordpress.com/2015/01/30/introduction-to-artificial-neural-networks..

[20] E. Morancho, "Unum: Adaptive Floating-Point Arithmetic," in *2016 Euromicro Conference on Digital System Design (DSD)*, Limassol, Cyprus, 2016.

[21] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, Avignon, France, 2013.

[22] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," *IEEE Design & Test ,* vol. 33, no. 1, pp. 8 - 22, 2016.

[23] W. Kahan, "How futile are mindless assessments of roundoff in floating-point computation?," Preprint, University of California, Berkeley , Berkeley, 2006.

[24] U. W. Kulisch and W. L. Miranker, "The Arithmetic of the Digital Computer: A New Approach," *SIAM review,* vol. 28, no. 1, 1986.

[25] S. M. Rump , “Algebraic computation, numerical computation and verified inclusions,” in *Trends in Computer Algebra. Lecture Notes in Computer Science, Vol 296*, Heidelberg, Berlin, Springer, 2005, pp. 177 - 197.

[26] U. G. A. Office, “Patriot Missile Defense. Software Problem Led to System Failure at Dharan Saudi Arabia,” U. G. A. Office, 1992.

[27] J. L. Gustafon and I. . T. Yonemoto , “Beating Floating Point at its Own Game: Posit Arithmetic,” *Supercomputing Frontiers and Innovations,* vol. 4, no. 2, p. 71–86, 2017.

[28] K. J. Setshedi, N. Mutingwende and N. Ngqwala, “The Use of Artificial Neural Networks to Predict the Physicochemical Characteristics of Water Quality in Three District Municipalities, Eastern Cape Province, South Africa,” *International Journal of Environmental Research and Public Health,* vol. 18, no. 10, p. 5248, 2021.

Chapter 2

2. Literature Review

This chapter provides a comprehensive review of literature pertaining to diverse implementations of Artificial Neural Networks (ANNs) utilized for Water Quality measurement. Additionally, it examines literature focusing on various hardware implementations of ANNs. Furthermore, the article conducts a survey of literature concerning the implementation of the Posit representation system. Finally, the article concludes by identifying research gaps within the published literature that have motivated the progression of this work.

2.1 Introduction

Water quality models are used extensively in research and design to enforce water quality regulations (that is, to ensure that the maximum/minimum permissible concentration of a substance in each water body is not exceeded). Many models, on the other hand, are predicated on the assumption of linearity functions. Different deterministic models have been used in the past to predict water quality in a stream under various scenarios of interest. However, because natural systems are sometimes too complicated for state-of-the-art deterministic modelling methods, the statistical accuracy of the models is frequently low in practise. ANNs are a quick and versatile way to create models for water quality estimation. ANNs have demonstrated excellent performance as regression tools in recent years, particularly when employed for pattern recognition and function prediction. An Artificial Neural Network (ANN) is a computational approach inspired by biological organisms' brain and nervous systems. ANNs are mathematical models that are highly idealised representations of our current understanding of complex systems. The ability of neural networks to learn is one of its properties. A neural network is not designed like a traditional computer programme; instead, it is given instances of the patterns, observations, and concepts, or any other form of data that it must learn. The neural network organises itself to develop an internal collection of features that it uses to classify input or data through the learning (also known as training) process. There are numerous advantages to using an ANN approach to problem-solving, including: (1) the application of a neural network does not necessitate prior knowledge of the underlying process; (2) It is possible that one does not recognise all the complicated relationships that exist between various components of the process under consideration. (3) A traditional optimisation strategy or statistical model only delivers a solution when allowed to run to completion, whereas a neural

network always converges to an optimal (sub-optimal) solution condition, and (4) In the ANN development, neither constraints nor an a priori solution structure are expected or tightly enforced. These properties make ANNs ideal instruments for dealing with a variety of hydrological modelling challenges [1]

2.2 ANN for Water Quality Applications

In the early 1990s, applications of ANN in the areas of groundwater, ecology, and environmental engineering were reported. In recent years, however, ANN has been widely used for prediction and forecasting in a variety of engineering and water-related fields, including water resource analysis by Liong and Sivapragasam, 2002 [2]; Muttil and Chau, 2006 [3]; El-Shafie et al. 2008 [4]; El-Shafie et al. 2011 [5]; Noureldin et al. 2011 [6]; Najah et al. 2009 [7]; oceanography by Makarynska et al. 2008 [8]; and environmental engineering by Grubert, 2003 [9].

Rankovic et al. (2010) [10] used ANN to forecast the concentration of dissolved oxygen (DO). The study's shortcoming is that the parameters involved are chemical parameters that can only be observed in a laboratory environment. Thus it can't be utilised for real-time monitoring. Gazzaz et al. (2012) [11] employed ANN to estimate WQI using 23 water quality metrics. The concept cannot be utilised for real-time monitoring since the price of the sensors necessary makes it prohibitively expensive. For continuous and remote monitoring of water quality data, Menon et al. (2012) [12] developed a wireless sensor network-based river water quality monitoring system in India. The device's wireless sensor node was designed to monitor the pH of water. The technology was limited in that it could not be utilised to control regional water contamination. Meanwhile, Ali et al. (2013) [13] used an unsupervised machine learning algorithm to classify water quality into three categories. The study's shortcoming is that it did not consider the numerous parameters that are linked to the Water Quality Classification.

Sensor nodes employed an Arduino core, which was then used by sensor nodes to interpret measured data. Faustine et al. (2014) [14] created a solar-powered system for monitoring water quality in the Lake Victoria Basin utilising WSN. The data was then transmitted over ZigBee to the gateway. The gateway collected all the data and sent it to the application programme through GPRS. Based on field test findings, the authors showed the suggested system's proper functionality and deployment in the real world. Despite this, there was no capability for local data analysis on the device. As a result, it will be unavailable whenever a mobile network

outage occurs. These technologies usually operate in the 2.4 GHz ISM license's open band, which is frequently crowded and vulnerable to interference and security breaches. Using Internet of Things (IoT) technology, Vijayakumar et al. (2015) [15] created a low-cost, real-time water quality monitoring system. The node was controlled by a Raspberry Pi model B+ CPU, which was coupled to many water quality sensors. This device can detect water quality parameters like temperature, pH, turbidity, conductivity, and dissolved oxygen. As a central controller, the Raspberry Pi platform was used. From experimental data, the proposed gadget was able to demonstrate water quality characteristics on the Internet. Due to cyber-attacks, this approach had weaknesses that can impair the legitimacy, reliability, and secrecy of measurement data.

Kalpana et al., 2016 [16] created a water monitoring system that included conductivity, turbidity, and pH sensors. The Raspberry Pi3 Model B single-board computer can automatically detect the parameters. The data from the three sensors is received by the single-board computer, which then sends it to the webserver through the internet. This gadget is suitable for both business and household use. The system can be expanded to track hydrology, air pollution, industrial and agricultural product development, and so on. Amruta et al. (2013) [17] proposed using a board aligned with the sun to create a regulated water supply system. The gadget consists of a centre and a base station, with the centre point connected to the base station by a Zigbee advance controlled by the board and controlled by daylight. The system would stop working if the panel in the sun could not be charged for any reason. Previous research employed fundamental water quality measures as a reference, such as pH, temperature, turbidity, and TDS, because differences in their values reflect the level of water pollution. As a result of overcoming this limitation, we are developing a new system that will require minimal work, improve, and be user-friendly.

Gopavanitha et al., 2017 [18] used IoT to design a system for real-time monitoring and control of water quality. The gadget is made up of sensors that can measure temperature, turbidity, conductivity, pH, and flow, among other physical and chemical properties of water. The Raspberry Pi's output value is sent to the cloud by the sensor. The sensed data is eventually viewable on the cloud, thanks to cloud computing, and IoT controls the water flow in the pipeline. Puneeth et al., (2018) [19] suggested an application that used the WSN and IoT concepts to monitor metrics such as pH, turbidity, and temperature via each node, which were then recorded and made available on the cloud. Solar energy is used to power the system. Lin

et al. (2008) [20] used wireless sensor network technology and a solar panel to create a water quality monitoring system. The prototype device was created and deployed utilising WSN technology and a single node powered by a solar cell. Data from node-side sensors such as pH, turbidity, and oxygen density were collected and transferred to the base station through WSN.

In an IoT setting, Kumar et al., 2019 presented a smart sensor interface device for water quality monitoring. Sensors such as a CO2 sensor, a temperature sensor, a pH sensor, a water level sensor, and a turbidity sensor were employed by the inventors. This sensor system manages the entire process and is managed by cloud-based wireless communication devices. The water level sensor detects and displays the water level in the tank. The sensors can monitor the water quality automatically. Amareshwar et al. (2019) [21] developed a sensor-based water quality monitoring system that assesses physical factors of water quality such as temperature, pH, and water humidity using MEMS sensors. The Raspberry Pi variant can be used as the controller of the central node. Finally, via API, the sensor data may be seen on the web. Demetillo et al. [22] created a low-cost, real-time water quality monitoring system (2019). It's suitable for isolated rivers, reservoirs, coastal locations, and other bodies of water. The device's nodes were powered by a 6 V/3.5 amp-hour (Ah) lead-acid battery. Minu et al. (2019) [23] created an IoT-based sensor that monitors pH, temperature, conductivity, dissolved oxygen, turbidity, bacteria, and other parameters in a water sample. The sensors collected data and relayed it across a network. The information would subsequently be uploaded to the cloud by the server. The data will be read, and the water quality will be assessed at the remote water station.

Using several Machine Learning methods, Ahmed et al. (2019) [24] predicted and classified the Water Quality of Rawal Lake, Pakistan. Alkalinity, Appearance, Calcium, Chlorides, Conductance, Faecal Coliforms, and Hardness as CaCO3, Nitrate as NO2-, pH, Temperature, Total Dissolved Solids, and Turbidity are the 12 parameters they used. In their analysis, the highest level of accuracy achieved by any algorithm was around 85 percent. They also didn't offer any suggestions about how to put the algorithms into practise in the field. ANN and multivariate linear regression were used by Abyaneh (2006) [25] to estimate Biological Oxygen Demand (BOD) and Chemical Oxygen Demand (COD) using four parameters: pH, temperature, Total Suspended Solids (TSS), and Total Suspended Solids (TSS) (TS).

On-site sampling and subsequent laboratory-based tests are both labour- and cost-intensive operations in traditional water quality measurement systems. The data isn't updated in real-time. As a result, real-time monitoring of water quality for drinking applications is required to

reduce labour costs and time consumption. In the old approach, captured data is uploaded to remote data storage via Zigbee boards. It costs a lot of money to set up this technology because it requires additional gear. When parameters are abnormal, there is no alert indicator in such a system. The progress of the water sensing network is controlled by the sun board in the Solar Powered Water Quality Monitoring System with Remote Sensor Network. The system will not turn on if the sun board is not charged, which is the limitation connected with this technology. The technology is unable to achieve the goal of real-time water quality parameter monitoring. This research seeks to build and develop a low-cost Raspberry Pi and Arduino Uno-based water quality monitoring system with artificial intelligence for real-time monitoring. Unlike a solar-powered water quality network monitoring system, this system is portable, the output is legible for those with poor or no literacy, and it will work in any location.

Sarkar and Pandey in [1], have tested three ANNs with varying data sets to predict the values of Dissolved Oxygen (DO) in the waters of the Yamuna River downstream of Mathura city in Uttar Pradesh, India. The work shows how both underrepresented data and overrepresented data are detrimental to the accuracy of the ANN. In [26] the authors have shown a comparison between static and dynamic ANNs in predicting the concentration values of Ammonium-Nitrogen (NH_3-N) in the waters of the Dahan River in Taiwan. The output of the study shows the benefits of dynamic ANN, Nonlinear autoregressive with exogenous input (NARX) network over some other static ANNs. On the other hand, Amanollahi, et al, in [27] evaluate the accuracy of an ANN in predicting chlorine concentration in the Wetland Areas of Iran. The work considers the water quality parameters such as Turbidity, TDS, and Hardness of the water to make the predictions. Some other methods to predict chlorine concentration based on the available data is also discussed in the work mentioned. Huang, et al [28] have discussed the implementation of a multi-classification support vector machine for the classification of pollutants in the water. The work discusses how the implementation of a multi-classification Probabilistic Output (MCPO) Support Vector Machine reduces the dependency on the concentration of contaminants in the online classification application.

Table 2.1 presents a summary of the reviewed literature on ANN implementation in Water Quality measurement.

Table2. 1: A Summary of ANN Implementations for Water Quality Measurement

Study	**Methodology/Novelty**	**Findings**	**Performance Metric**
Rankovic, et al [10](2010)	Implemented MLP architecture with Lavenberg-Marquadt learning. Software Approach	Parameters involved are chemical and cannot be tracked in real time in the field	$R^2 = 0.96$
Gazzaz et al [11](2012)	Showed the benefits of using ANN for WQI compared to mathematical methods for WQI. Software Approach	Utilises twenty-three parameters for indexing. Measurement method for these parameters render the device very costly and unsuited for real-time monitoring	$R^2 = 0.954$
Ali, et al. [13](2013)	Shows that MLP is the most suited architecture for supervised learning ANN for WQI. It is a Software based approach.	The study takes pH as the only parameter for Indexing Water Quality	Not mentioned in the paper
Fausstine, et al. [14] (2014)	Implemented a Sensor network based WQI model. *A Hardware Approach*	Solar Powered WSN based Water Quality Measurement. Dependence on solar reduces use cases. Uses zigbee and Mobile data network to transmit data to a server for analysis. Thus, real time monitoring is not possible.	NA

Vijaykumar, Ramya [15] (2015)	Used Raspberry Pi to implement ANN for Water Quality Monitoring. *Hardware Approach*	Low-cost Raspberry Pi based device measures pH, DO, Turbidity and Conductivity in real time. Does not measure ORP. No implementation of ANN makes the device require frequent recalibrations.	NA
Gopavanitha et al. [18] (2017)	Present a low-cost approach to measure water quality using Raspberry Pi. *Hardware Approach*	IoT based solar powered water quality monitoring and control system. Heavily dependent on Solar energy, thus weather conditions.	NA
Ahmed et al. [24] (2019)	Shows that MLP gives maximum accuracy for classification of Water Quality. *Software Approach*	Applied multiple network topologies with various learning algorithms. They achieved the best accuracy with Multilayer Perceptron topology.	Accuracy = 0.85
Abyaneh [25](2016)	Shows that ANN is better than Multi Linear Regression. *Software Approach*	ANN and multivariate linear regression to estimate Biological Oxygen Demand (BOD) and Chemical Oxygen Demand (COD) using four parameters: pH, temperature, Total Suspended Solids (TSS), and Total Suspended Solids (TSS) (TS).	r = 0.75
Sarkar and Pandey [1] (2015)	Prediction of DO based on temperature, pH, and flow discharge using ANN. *Software Approach.*	evaluated three ANN s with varying data sets to predict the values of Dissolved Oxygen (DO) in the waters of the Yamuna River downstream Mathura city in Uttar Pradesh, India. The work shows how both underrepresented data and overrepresented data are detrimental to the accuracy of the ANN	R = 0.9

Amanollahi, et al, in [27]	Predicted Turbidity, TDS, and TSS using ANN on Remote Sensing data. Also shows that ANN is better at prediction of WQI parameters compared to Linear Regression. *Software Approach*	evaluate the accuracy of an ANN in predicting chlorine concentration in the Wetland Areas of Iran	Predicted various parameters with differing R^2 values

From Table 2.1 we observe :

i) Many studies have used ANN for various water quality applications with differing degrees of success. However, as compared to other methods used for indexing of water quality, ANNs have proven to be more accurate.

ii) Data augmentation has been used for various applications to supplement the data of parameters not easily measurable. Most of the methods have used mathematical augmentation methods for supplementing the parameters that have a mathematical relationship with another measurable parameter.

Hardware implementation of ANN is a core design element for this study. The major components of ANN are the algorithm, its architecture, the activation function, and the number representation. The performance of an ANN implementation, in terms of power and resource efficiency, speed, and accuracy, depends highly on the aforementioned parameters. Thus, a detailed exploration of methods used for their implementation in published literature is essential for making an appropriate decision during the design.

In the following section we present the review of ANN Algorithms and architectures section 2.3. This is followed by a review of the literature present regarding the Hardware implementation in section 2.4. Section 2.4 delves into various methods of hardware implementation of ANN. In section 2.4.1 presents the review of the methods to optimise the activation function implementation to make it suitable for hardware implementation, at minimal loss of accuracy of ANN. Section 2.4.2 reviews the literature on more optimisation of

ANN implementation by exploring the various number representation systems that have been proposed in the literature for hardware implementation of ANN.

2.3 Artificial Neural Networks (ANN)

ANNs are most commonly expected to perform one of the four tasks:

- *Classification* of input data sets into predefined classes.
- *Prediction* of future output based on a set of current input-output mappings.
- *Clustering* data into groups based on prior knowledge about the data.
- An ANN can be trained to remember particular data patterns and then *associate* input data pattern with the ones in the memory or discard the data pattern.

An ANN neuron is modelled as: -

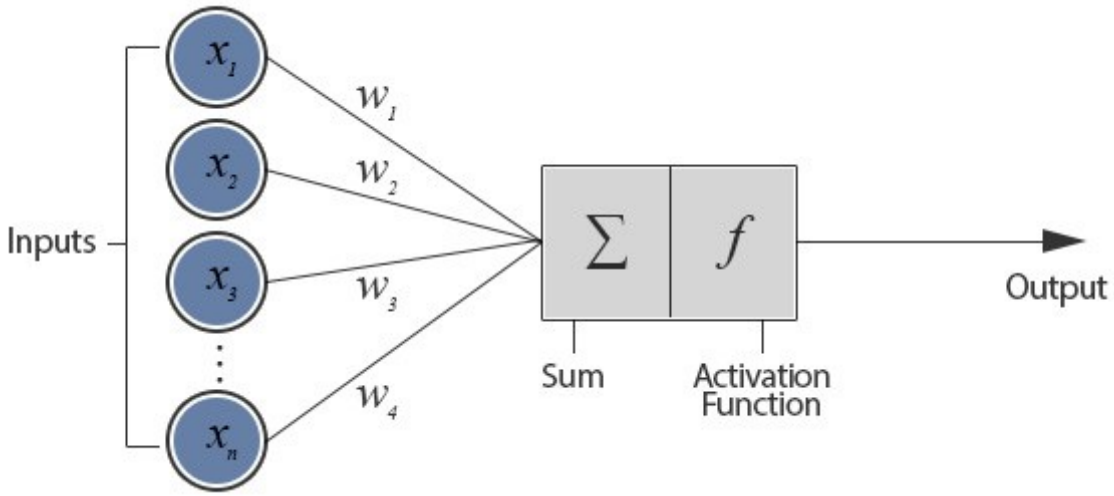


Figure 2. 1: Model of a neuron of an ANN. Image Source: [5]

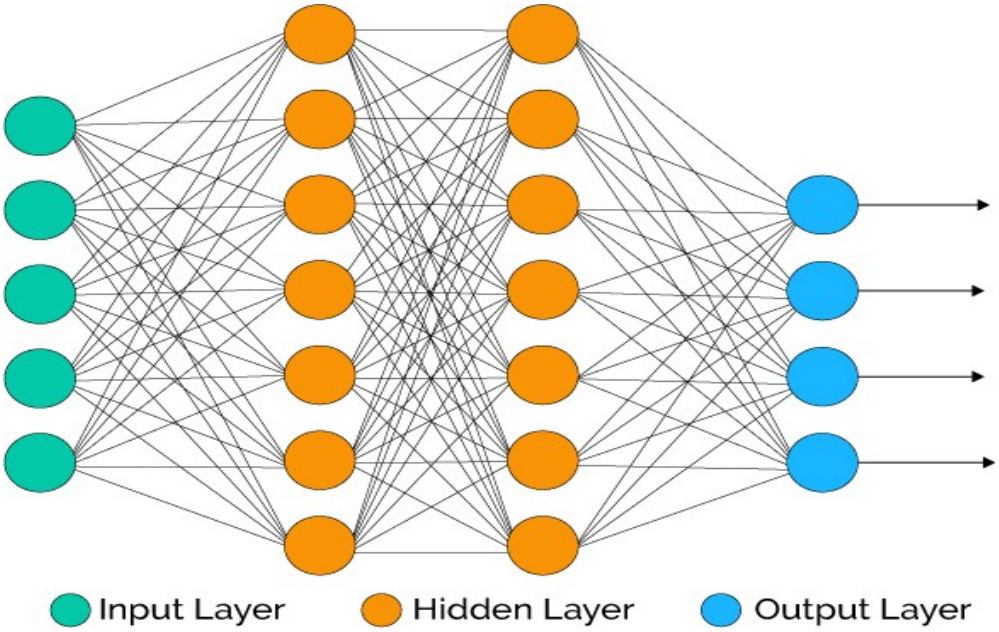


Figure 2. 2 A typical ANN Architecture; Image Source [35]

The input points on Figure 2.1 are analogous to synaptic connections on a nerve cell. For an n-dimension input vector [x_1, x_2, x_n] each input is multiplied by a synaptic weight [w_1,w_2, …….w_n]. These products are hereafter summed up in the nerve centre and the final sum is passed through an activation function (a threshold function, stepwise linear function, or sigmoid function) that defines the final output of the neuron. A multitude of such neurons form a layer of parallel processing centres which can work on a huge range of inputs. The outputs of one such layer of neurons serves as the input to many such subsequent layers of neurons, thus implementing a huge parallel processing capability to the system. The outputs are then compared with the expected outputs and the errors are measured. The weights of the synapses are thus altered in accordance with the error. This is the basic learning process of a neuron. Figure 2.2 shows a typical architecture for an ANN: -

- **Input Layer** – It contains those neurons that receive the input data that is to be processed by the ANN.
- **Hidden Layer(s)** – These are the layers of units between the input and the output layers. The hidden layers take the data from the input layer and perform the computations such that it can give useful information to the other hidden or output layer.
- **Output Layer** – This consists of the neurons or units that give output depending on the learning that has taken place inside the ANN.

While implementing logic on hardware, there are two approaches: - *Analog and Digital.* Morgan, et al. [42] present the key advantages that a digital approach has over analog.

Biological mechanisms which function on inaccurate components serve as models for ANN algorithms. Thus, analog circuits having infinite resolution (continuous sampling) should serve as the better implementation option. Practically though, there is a limit to the representation of the smallest resolution on a circuit. i.e., the sampling rate of a circuit must be finite. Moreover, a multitude of ANN algorithms model biology very poorly and thus frequently need a wide dynamic range to bring about convergence to useful solutions. Popular stochastic algorithms such as backpropagation (BP) require a 12 – 16-bit range or roughly four orders of magnitude between the largest weight value and the smallest weight change [30] [31]. The resolution required to achieve convergence cannot be obtained from analog circuits. [42]

When implementing changes in an algorithm, an analog circuit requires major redesign efforts. Digital circuits can accommodate such changes by implementing a different logic array.

Since most other computational units are digital, nowadays. A digital circuit also improves compatibility with other systems, as compared to an analog circuit. Also, ANNs are subject to Amdahl's Law, as per which, the speed improvements of an analog neuron will stay untapped if the remaining network is implemented by slower digital implementations.

2.3.1 ANN Algorithms:

Traditionally, ANNs have been implemented on software which requires a complete processing system. That means a lot of unnecessary hardware is engaged but not utilised. Also, when the software code must be decoded down to a hardware level to be able to interact with the real world, it must go through multiple abstractions of computer architecture. This renders the process to become time-consuming and limits the usability of the ANN. Thus, an Application Specific Integrated Circuit (ASIC) or System-on-Chip (SoC) based implementation of an ANN algorithm makes the system much more resource and time efficient because of the application specific nature of the hardware. A direct hardware implementation also improves the speed of the system. An application specific IC is also economical because of its specificity.

Hardware implementation of ANNs offer a simpler development cycle for powerful machine intelligence. Application-specific nature of these hardware implementations offer better performance but at the cost of programmability. Moreover, physical resource (Silicon area)

utilisation is also minimal in this approach. The trade-off between application specificity and program flexibility is a part and parcel of VLSI design.

On most neural networks, each neuron in a hidden layer is connected to each unit in the previous (input) layer and the subsequent (output) layers. ANNs can be implemented in several architectures. In 1943, McCulloch and Pitts presented the first ever model of an artificial neuron, called the perceptron [32]. A layer of perceptrons can perform some tasks. Thus, a single layer of perceptrons can form a network. We term such a network as a single-layer perceptron. An arrangement of a series of a Single Layer Perceptron is called a Multi-Layer Perceptron (MLP). MLPs are also called feedforward networks. Another architecture, where the activation function of the perceptron of the MLP is the Radial Basis Function, such a network is called the Radial Basis Function Neural Network. While the aforementioned networks have a structure where each perceptron sends the information only to the next perceptron, a network where there are self-loops on each perceptron is called the Recurrent Neural Network. Recurrent Neural Networks have memories and can be activated by both, activation functions from a lower-level perceptron and previous activation value. There are some other types of architectures as well, such as Long/Short Term Memory Networks, Hopfield Networks, Boltzmann Machine, Convolutional Neural Networks, etc. Generally, the architectures of these types of networks are derivatives of the four mentioned above.

a) Multi-Layer Perceptron (MLP) Neural Network (NN) Architecture

It has feedforward architecture within the Input layer, hidden layers, and output layer. The input layer has 'N' units representing the N-dimensional input vector. The input units are fully connected to 'I' hidden layer units, which are further connected to 'J' output layer units. 'J' represents the number of output classes. If our training data contains 'l' pairs (x_i,y_i) where x_i is the pattern vector and y_i is the class of the corresponding pattern. The activity of a neuron J in a hidden layer is given by: -

$$S_j = \sum w_{ji} x_i \tag{1}$$

$x_i = f(s_j)$; f is a sigmoid function

Where $w_{1i} =$ set of weights of neuron$i$; $b_1(i) =$ threshold; $x_i =$ input of neuron

Output layer activity is: -

$$S_j = \sum_{i \in input} w_{ji} x_i \tag{2}$$

b) Training Methodology

A neural network relies mostly on its training methodology to learn. The better the training methodology, the better the outputs of the network.

Backpropagation is the most common training methodology and is simpler to implement which reduces the time to market and is also much more capable when it comes to supervised pattern matching. But backpropagation has its limitations, such as problems with convergence and the time cost of backpropagation hardware is hugely variable.

The Rapid Restart method [33] has been demonstrated to be prominently suppressive of the heavy-tailed nature of training instances. Computational efficiency also is improved with the Rapid Restart Method.

2.4 Hardware Implementation of ANN

P Skoda, et al in [34] present an implementation framework for implementing ANN onto an FPGA. They make use of an LUT to implement the activation function and a ROM memory that serves up the weights to each neuron input. However, this approach becomes tedious to implement when the number of inputs increases. Also, because of the high usage of memory cells, the process output will become slow and resource intensive.

Kim and Jung in [35] present a 32-bit processor with special instructions and hardware units to perform single precision floating point units. The processor is specifically designed to implement an RBF Neural Network with a Backpropagation algorithm for online learning. However, the implementation does not give a complete hardware-based approach and the ANN tasks are performed by the ALU based on the instruction set. However, the implementation being a processor and the operation being carried out by instruction sets renders more functional customisability to the hardware.

In the article [36], the author presents an algorithm to be implemented on ANN Hardware. It is a feedforward architecture that treats each neuron as a special case of Boolean functions. The Boolean function properties can be used to achieve compactness.

2.4.1 Optimisations for Hardware Implementation

There are many algorithms like the Gradient Descent Algorithm, Backpropagation, Hebb Rule, Kohonen Self Organising maps, etc. [37], which are used to facilitate learning for an ANN. Most of the activation functions and learning algorithms are very abstract mathematical functions, generally, nonlinear. This makes the implementation of the activation function and learning algorithm on ASIC very complicated. Some approximation methods like the ones discussed in [38] give a close enough approximation for the calculation of the output of activation functions.

MLP training is dependent on the repeated presentation of sample input and desired targets, whence outputs and targets are compared, and errors measured. Finally, weights are adjusted as the error is minimised. The most crucial, resource intensive and difficult to implement part of any hardware implementation of ANNs is the non-linear activation function [39].

- ***Sigmoid activation function:***

Backpropagation may be applied to any number of layers, but it has been proven that a single layer of hidden units suffices for the approximation of any function [40]. Hence, MLP NNs with a single layer of hidden units with a sigmoid activation function is used most commonly.

$$f(a) = \frac{1}{1 + e^{-a}} \tag{3}$$

It has an easy-to-calculate derivative.

$$f'(a) = f(a)[1 - f(a)] \tag{4}$$

Implementation of activation function - Zbigniew Hajduk proposed [38] a direct implementation of the functions with accuracy of the method higher compared to other published solutions. Here, the difficulty of implementation of activation function is transferred

to the approximation of the exponent function. The hyperbolic tangent $(T(x))$ and sigmoid $\big(S(x)\big)$ functions are represented as -

$$T(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{5}$$

$$S(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

For the implementation of hyperbolic tangent function with high accuracy, McLaurin series approximation is used with exploiting the symmetry feature (i.e., tan hyperbolic of only the negative arguments are calculated and then properly adjusted by changing sign to obtain the result of the positive arguments).

For sigmoid functions, however, symmetry does not result in improved accuracy. Here, the McLaurin interpolation is done by equation (7): -

$$e^x = 1 + x\left(1 + \frac{x}{2}\left(1 + \frac{x}{3}\left(1 + \frac{x}{4}\left(1 + \frac{x}{5}\left(1 + \frac{x}{6}\left(1 + \frac{x}{7}\left(1 + \frac{x}{8}\left(1 + \frac{x}{9}\left(1 + \frac{x}{10}\right)\right)\right)\right)\right)\right)\right)\right)\right) \tag{7}$$

For sigmoid functions, another approximation, Padé approximation is as given below: -

$$e^x = \frac{1680 + 840x + 180x^2 + 20x^3 + x^4}{1680 - 840x + 180x^2 - 20x^3 + x^4} \tag{8}$$

Padé approximation does compromise the accuracy by a small margin but reduces the number of floating-point operations. Taking a higher degree of McLaurin or Padé approximation does not improve accuracy of the activation function. The above expressions are valid only for$x \in (-1,1)$. For a wider range the value e^x can be broken down as follows: -

$$e^x = e^{p+f} = e^p . e^f \tag{9}$$

Where $p + f = x$ and$f \in (-1,1)$.

We can thus calculate e^f using any of the above two approximations and e^p can be calculated using an LUT. We can exploit the fact that the activation function's values become constant beyond a threshold value. Experimentally the number of LUTs can be limited to 35 (or 17 in cases where symmetry is exploited) [38].

2.5 Number Representation Systems

The conventional number system used to represent floating point numbers in binary logic is the IEEE 754 Floating Point Representation (IEEE 754). Hardware implementation of ANN involves the use of floating point calculation and thus makes it imperative to use floating point representations. In this section, we present the literature review regarding the IEEE 754 and the Posit number system in brief. A detailed review of both and their comparison are presented in Chapter 5.

IEEE 754 Floating point representation (IEEE 754) has been the conventional method of representing floating point numbers in digital computation. IEEE 754 has made the computation of real numbers possible in digital computing fairly accurately. It represents the real number akin to the scientific notation. However, IEEE 754 has a rigid representation of floating point numbers. This leads to many problems like rounding errors, inconsistencies in representation, errors in dot product calculation, the existence of two zeroes, etc. These problems have led to some costly mistakes like the Patriot missile misfire [42].

Some researchers proposed a new type of number system called the universal number (Unum) system in 2015. So far, Unum has developed three revisions. Type-1 [43] [44] [45], Type-2 [46] [47], and Type-3 [48] [49] are the three types. Type-3, also known as Posit, was the most recent revision. Unum was allegedly utilised to replace the IEEE 754-2008 floating-point standard [50] with greater efficiency and precision, according to its creators. Unum and Posit both feature several advantages over IEEE 754-2008, such as a greater dynamic range, higher coding space use, tapering accuracy, parameterized precision, and so on [48].

In 2013, John L. Gustaffson proposed a novel method called Universal Numbers (Unums). Gustaffson defined 2 types of Unums. Type 1 was developed as a superset to floating point numbers to accommodate greater range and accuracy. However, the hardware cost made it impractical. Type 2 was based on a positional bit pattern instead of actual data conversion. This

conversion was based on look-up tables. This allowed extremely fast computations but at the cost of operations that could be performed [48].

In their 2017 paper, John L. Gustaffson proposed the posit representation of floating-point numbers. The Oxford dictionary defines posit as "a statement that is made on the assumption that it will prove to be true." Posits are a hardware-friendly version of Unum2 with relaxations in 2 rules: -

i) Reciprocals only follow the perfect reflection rule for 0, +/- infinity and integer powers of 2.
ii) There are no open intervals

The first relaxation enables one to populate the u-lattice such that finite numbers are all represented in the form of IEEE 754 representation of $m.2^k$.

The structure of the Posit Representation of Floating Point numbers is as shown in Figure 2.3.

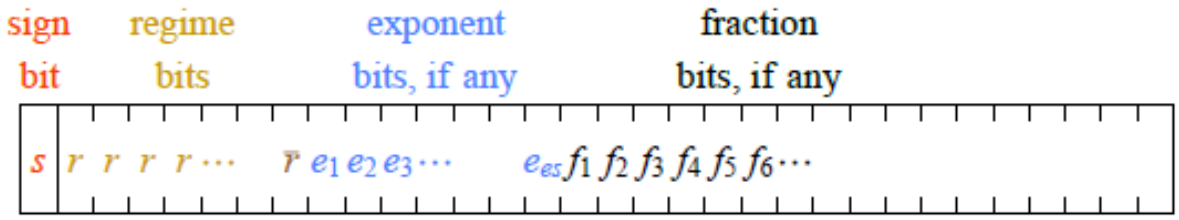


Figure 2. 3: Format of Posit Representation [25]

The sign bit is the same as the IEEE 754 Floating point representation: 0 for positive numbers and 1 for negative numbers. If the sign bit is 1, the rest of the number should be in 2's complement.

Table2. 2: Run length meaning k of the regime [25]

Binary	0000	0001	001x	01xx	10xx	110x	1110	1111
Numerical Meaning, k	-4	-3	-2	-1	0	1	2	3

Consider the binary strings shown in Table 2.2 to make sense of the regime bits. The run length of the bits is denoted by numerical meaning, k. These are strings of either all 0 or all 1bits. The

bits are terminated either by the opposite bit or end of the string is reached. If the bits are 0 and there are m bits then $k = -m$, if the bits are 1 then $k = m - 1$. The regime gives us the scale factor for useed^k, $useed = 2^{2^{es}}$. $useed$ values examples are shown in Table 2.3

Table2. 3: The useed as a function of es

es	0	1	2	3	4
useed	2	$2^2 = 4$	$4^2 = 16$	$16^2 = 256$	$256^2 = 65536$

The next part is the exponent, e, taken as an unsigned integer. Unlike IEEE 754 Floating points, there is no bias in the exponent and represents scaling by 2^e. If enough bits are remaining after the regime, the highest number of bits the exponent can occupy is es. This is how the tapered accuracy of Posits is expressed. Numbers near 1 need to be presented in more accuracy than very large or very small numbers which are not so common in calculation.

If more bits remain in the bit stream after regime and exponent, they are used to represent the fraction part of the number. The fraction part of a posit is just like that of IEEE 754 floating point in the format of 1.f with a hidden bit that represents the whole number part, 1. Posits have no subnormal numbers with a hidden bit 0 for numbers less than 1.

There are only 2 exceptions in the posit representation, i.e., 0(all 0's) and $\pm\infty$ (1 followed by all 0 bits).

Table 2.4 shows the dynamic range offered by both posits and IEEE 754 Floating Point representation for some bit lengths [48].

Table2. 4: IEEE 754 Float and Posit dynamic ranges for the same no. of bits.

Size, Bits	**IEEE Float Exp. Size**	**Approx. IEEE Float Dynamic Range**	**Posit es value**	**Approx. Posit Dynamic Range**
16	5	6×10^{-8} to 7×10^{4}	1	4×10^{-9} to 3×10^{8}
32	8	1×10^{-45} to 3×10^{38}	3	6×10^{-73} to 2×10^{72}

64	11	5×10^{-324} to 2×10^{308}	4	2×10^{-299} to 4×10^{298}
128	15	6×10^{-4966} to 1×10^{4932}	7	1×10^{-4855} to 1×10^{4855}
256	19	2×10^{-78984} to 2×10^{78913}	10	2×10^{-78927} to 5×10^{78296}

We make use of the Posit number representation system to implement a constructive Neural Network architecture on Digital Hardware for water classification applications. With the aim of water quality study, the proposed hardware for the ANN needs to implement pattern recognition of input data set and comparing it with a set of prescribed data patterns and thus classify the water sample accordingly. Pattern localization and classification are CPU time intensive when normally implemented in software. They also have lower performance than custom implementations. With custom hardware implementation we can reap the benefits of real-time processing but at a higher cost and time-to-market than software implementations [42].

2.6 Conclusions and Gaps in Research

In the proposed work, the aim is to develop an ANN Classification algorithm and implement it onto hardware that would classify the water quality parameters based on a pre-decided classification parameter that would be in accordance with WHO Drinking Water Quality Guidelines.

The above literature survey shows a few important areas that have not been explored properly in the published literature: -

1. Most implementations in Water Quality management are for predicting certain water quality parameter based on its correlation with other parameters. The Literature survey shows very limited work where the ANN is used to classify the water based on the parameters input by the sensors. Implementation of such a Network has been presented in this thesis.
2. All the hardware implementations mentioned in the literature survey involve a software part either in learning or in activation function implementation. None of the approaches are completely ASIC design example. The reason being the algorithms for learning and

activation function are heavily nonlinear and involve tedious real number calculations. With a good mathematical approximation method, the algorithms can be approximated to simpler reduced floating-point calculations making the hardware implementation more resource economic.

3. Most neural network implementations use binary functions like tan hyperbolic or sigmoid function as activation function for classification of data. These functions require calculations in the floating-point number domain. The current floating point number representation system, i.e., IEEE 754 Floating point representation is not ideal for executing calculations such as exponential function. Thus, a novel number representation system – Posit number representation has been used for the implementation of activation function so that better power and area efficiency can be achieved.

We have worked upon these research gaps to enhance the performance of ANN Hardware Implementation for Water Quality Classification applications. Work has been carried out in accordance research objectives as mentioned in following section.

2.7 Research Objectives

- Literature Survey of published ANN architectures and hardware implementations
- Exploration of existing ANN algorithms/architectures and number representation systems
- Hardware Implementations of Existing Algorithms and their optimization for meeting required design challenges
- Development of novel Hardware Implementation of ANN Architecture with applications in Water Quality

2.8 References

[1] A. Sarkar and P. Pandey, "River Water Quality Modelling using Artificial Neural Network Technique," *Aquatic Procedia,* vol. 4, pp. 1070 - 1077, 2015.

[2] S.-Y. Liong and C. Sivapragasam, "FLOOD STAGE FORECASTING WITH SUPPORT VECTOR MACHINES," *Journal of American Water Resource Association,* vol. 38, no. 1, pp. 173 - 186, 2007.

[3] N. Muttil and K.-W. Chau, “Neural network and genetic programming for modelling coastal algal blooms,” *International Journal of Environment and Pollution,* vol. 28, no. 3 - 4, pp. 223-238, 2006.

[4] A. El-Shafie, A. Noureldin, M. Taha and H. Basri, “Neural Network Model for Nile River Inflow Forecasting Based on Correlation Analysis of Historical Inflow Data,” *Journal of Applied Sciences,* vol. 8, pp. 4487-4499, 2008.

[5] A. El-shafie, M. Mukhlisin, A. A. Najah and M. R. Taha, “Performance of artificial neural network and regression techniques for rainfall-runoff prediction,” *International Journal of the Physical Sciences,* vol. 6, no. 8, pp. 1997-2003, 2011.

[6] A. Noureldin, A. El-Shafie and M. Bayoumi, “GPS/INS integration utilizing dynamic neural networks for vehicular navigation,” *Information Fusion,* vol. 12, no. 1, pp. 48 -57, 2011.

[7] A. Najah, A. Elshafie, O. . A. Karim and O. Jaffar, “Prediction of Johor River Water Quality Parameters Using Artificial Neural Networks,” *European Journal of Scientific Research,* vol. 28, no. 3, pp. 422-435, 2009.

[8] D. Makarynska and O. Makarynskyy, “Predicting sea-level variations at the Cocos (Keeling) Islands with artificial neural networks,” *Computers & Geosciences,* vol. 34, no. 12, pp. 1910-1917, 2008.

[9] J. . P. Grubert, “Acid deposition in the eastern United States and neural network predictions for the future,” *Journal of Environmental Engineering and Science,* vol. 2, no. 2, pp. 99-109, 2003.

[10] V. Ranković, J. Radulović , I. Radojević, A. Ostojić and L. Čomić , “Neural network modeling of dissolved oxygen in the Gruža reservoir, Serbia,” *Ecological Modelling,* vol. 221, no. 8, pp. 1239 - 1244, 2010.

[11] N. M. Gazzaz, . M. K. Yusoff, A. Z. Aris , H. Juahir and M. F. Ramli, “Artificial neural network modeling of the water quality index for Kinta River (Malaysia) using water quality variables as predictors,” *Marine Pollution Bulletin,* vol. 64, no. 11, pp. 2409 - 2420, 2012.

[12] K. U. Menon, P. Divya and M. V. Ramesh, “Wireless sensor network for river water quality monitoring in India,” in *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*, Coimbatore, India, 2012.

[13] M. Ali and A. M. Qamar, “Data analysis, quality indexing and prediction of water quality for the management of rawal watershed in Pakistan,” in *Eighth International Conference on Digital Information Management (ICDIM 2013)*, Islamabad, Pakistan, 2014.

[14] A. Faustine, A. M. Mvuma, H. J. Mongi, M. C. Gabriel, A. J. Tenge and S. B. Kucel, “Wireless Sensor Networks for Water Quality Monitoring and Control within Lake Victoria Basin: Prototype Development,” *Wireless Sensor Network,* vol. 6, no. 12, pp. 281-290, 2014.

[15] N. N. Vijayakumar and R. Ramya, “The real time monitoring of water quality in IoT environment,” in *2015 International Conference on Circuit, Power and Computing Technologies [ICCPCT]*, Coimbatore, India, 2015.

[16] M. B. KALPANA, “Online Monitoring Of Water Quality Using Raspberry Pi3 Model B,” *(IJITR) INTERNATIONAL JOURNAL OF INNOVATIVE TECHNOLOGY AND RESEARCH,* vol. 4, no. 6, pp. 4790-4795, 2016.

[17] M. K. Amruta and M. T. Satish, “Solar powered water quality monitoring system using wireless sensor network,” in *2013 International Mutli-Conference on Automation,*

Computing, Communication, Control and Compressed Sensing (iMac4s), Kottayam, India, 2013.

[18] K. Gopavanitha and S. Nagaraju, "A low cost system for real time water quality monitoring and controlling using IoT," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Chennai, India, 2017.

[19] K. M. Puneeth, S. Bipin, C. Prasad, R. J. Kumar and M. K. Urs, "Real-time Water Quality Monitoring using WSN," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, India, 2018.

[20] J.-S. Lin and C.-Z. Liu, "A monitoring system based on wireless sensor network and an SoC platform in precision agriculture," in *2008 11th IEEE International Conference on Communication Technology*, Hangzhou, 2008.

[21] S. Jahan, E. Amareshwar, S. Prasad and A. T. S, "Raspberry Pi Based Water Quality Monitoring and Flood Alerting System Using Iot," *International Journal of Recent Technology and Engineering (IJRTE),* vol. 7, no. 6S4, pp. 640 - 643, 2019.

[22] . A. T. Demetillo, M. V. Japitana and E. B. Taboada, "A system for monitoring water quality in a large aquatic area using wireless sensor network technology," *Sustainable Environment Research,* vol. 29, no. 1, pp. 1 - 9, 2019.

[23] M. Minu, P. Kumari, A. K. Singh and S. Avinash , "Wired Sensor Systems for Water Quality Monitoring," *International Journal of Recent Technology and Engineering (IJRTE),* vol. 8, no. 4, pp. 847 - 852, 2019.

[24] U. Ahmed, R. Mumtaz, H. Anwar, A. A. Shah, R. Irfan and J. García-Nieto, "Efficient Water Quality Prediction Using Supervised Machine Learning," *Water,* vol. 11, no. 11, p. 2210, 2019.

[25] H. Z. Abyaneh, "Evaluation of multivariate linear regression and artificial neural networks in prediction of water quality parameters," *Journal of Health Science and Engineering,* vol. 12, no. 1, p. 40, 2014.

[26] F.-J. Chang, Y.-H. Tsai, P.-A. Chen, A. Coynel and G. Vachaud, "Modelling water quality in an Urban River using Hydrological Factors - Data driven approaches," *Journal of Environmental Management,* vol. 151, pp. 87 - 96, 2015.

[27] J. Amanollahi, S. Kaboodvandpour and H. Majidi, "Evaluating the accuracy of ANN and LR Models to Estimate the Water Quality in Zarivar International Wetland, Iran," *Natural Hazards,* vol. 85, no. 3, pp. 1511 - 1527, 2017.

[28] P. Huang, Y. Jin, D. Hou, D. Tu, Y. Cao and G. Zhang, "Online Classification of Contaminants Based on Multi-Classification Support Vector Machine Using Conventional Water Quality Sensors," *Sensors,* vol. 17, no. 3, p. 581, 2017.

[29] N. Morgan, K. Asanovic, B. Kingsbury and J. Wawrzynek, "Developmets in Digital VLSI Design for Artificial Neural Networks," University of California at Berkeley, Berkeley, California, 1990.

[30] N. Morgan, Artificial Neural Networks: Electronic Implementations, Washington, D.C.: Computer Society Press of the IEEE, 1990.

[31] T. Baker and D. Hammerstrom, "Modifications to Artificial Neural Networks Models for Digitial Hardware Implementation," Department of Computer Science and Engineering, Oregon Graduate Centre, Washington County, Oregon, USA, 1988.

[32] W. S. McCulloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *The Bulletin of Mathematical Biolphysics,* vol. 5, no. 4, pp. 115 - 133, 1943.

[33] F. Smach, M. Atri, J. Miteran and M. Abid, “Design of a Neural Network Classifier for Face Detection,” *Matrix,* vol. 15, pp. 124 - 127, 2006.

[34] P. Skoda, T. Lipic, A. Srp, R. B. Medved, K. Skala and F. Vajda, “Implementation Framework for Artificial Neural Networks on FPGA,” in *MIPRO, 2011 Proceedings of the 34th International Convention*, Opatija, Croatia, 2011.

[35] J. S. Kim and S. Jung, “Implementation of the RBF neural chip with the back-propagation algorithm for on-line learning,” *Applied Soft Computing ,* vol. 29, pp. 233 - 244, 2015.

[36] A. Dinu, M. N. Cirstea and S. E. Cirstea, “Direct Neural-Network Hardware-Implementation Algorithm,” *IEEE Transactions on Industrial Electronics,* vol. 57, no. 7, pp. 1845 - 1848, 2010.

[37] L. Fausett, Fundamentals of Neural Networks : Architectures, Algorithms and Applications, Prentice-Hall, 1994.

[38] Z. Hajduk, “High accuracy FPGA activation function implementation for neural networks,” *Neurocomputing,* vol. 247, pp. 59 - 61, 2017.

[39] V. Tiwari and N. Khare, “Hardware implementation of neural network with Sigmoidal activation functions using CORDIC,” *Microprocessors and Microsystems,* vol. 39, no. 6, pp. 373 - 381, 2015.

[40] F. Yang and M. Paindavoine, “Prefiltering for pattern recognition using Wavelet Transform and Neural Networks,” *Advances in Imaging and Electron Physics,* vol. 127, pp. 125 - 206, 2003.

[41] H. V. H. Ayala, D. M. Munoz, C. H. Llanos and L. d. S. Coelho, “Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations,” *Control Engineering Practice,* vol. 60, pp. 124 - 132, 2017.

[42] “Some disasters attributable to bad numerical computing,” University of Montreal, [Online]. Available: https://www.iro.umontreal.ca/~mignotte/IFT2425/Disasters.html. [Accessed December 2022].

[43] J. L. Gustafson, The End of Error Unum Computing, New York: Chapman and Hall/CRC, 2015.

[44] W. Tichy, “The end of (numeric) error: An interview with John L. Gustafson.,” *Ubiquity,* pp. 1 - 14, 2016.

[45] R. Brueckner, “Slidecast: John Gustafson Explains Energy Efficient Unum Computing.,” Inside HPC, 2015. [Online]. Available: https://insidehpc.com/2015/03/slidecast-john-gustafson-explains-energy-efficient-unum-computing/.

[46] J. L. Gustafson, “A radical approach to computation with real numbers.,” *Supercomputing frontiers and innovations,* vol. 3, no. 2, pp. 38-53, 2016.

[47] W. Tichy, “ Unums 2.0: An interview with John L. Gustafson,” *Ubiquity,* vol. 1, 2016.

[48] J. L. Gustafson and I. T. Yonemoto, “Beating Floating Point at its Own Game: Posit Arithmetic,” 2017. [Online]. Available: http://www.johngustafson.net/pdfs/BeatingFloatingPoint.pdf.

[49] J. L. Gustafson, “Beyond Floating Point: Next Generation Computer Arithmetic,” in *Stanford EE Computer Systems Colloquium.*, 2017.

[50] IEEE, ““IEEE Standard for Floating-Point Arithmetic,,” IEEE Std 754-200829, August 2008. [Online]. Available: https://ieeexplore.ieee.org/document/4610935.

[51] mindcontrolinsweden, “Introduction to Artificial Neural Network (ANN) | secret mind control in sweden and wordwide,” wordpress.com, 30 January 2015. [Online]. Available:

https://mindcontrolinsweden.wordpress.com/2015/01/30/introduction-to-artificial-neural-networks/. [Accessed 13 October 2017].

[52] J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress," *Neurocomputing,* vol. 74, no. 1 - 3, pp. 239 - 255, 2010.

[53] D. M. Munoz, D. F. Sanchez, C. H. Llanos and M. Ayala-Rincon, "FPGA based floating point library for CORDIC algorithms," in *IEEE Proceedings of the 2010 VI Southern Programmable Logic Conference*, Ipojuca Port de Galinhas, Brazil, 2010.

[54] D. M. Munoz, D. F. Sanchez, C. H. Llanos and M. Ayala-Rincon, "Tradeoff of FPGA design of a floating-point library for arithematic operators," *International Journal of Integrated Circuits and Systems,* vol. 5, no. 1, pp. 42 - 52, 2010.

[55] J. Gomez-Ortega and E. F. Camacho, "Neural Network MBPC for mobile robot path tracking," *Robotics and Computer - Integrated Manufacturing,* vol. 11, no. 4, pp. 271 - 278, 1994.

[56] A. Alessandri, M. Baglietto and G. Battistelli, "Moving-horizon state estimation for nonlinear discrete-time systems," *Automatica,* vol. 44, no. 7, pp. 1753 - 1765, 2008.

[57] J. V. Frances-Villora, A. Rosado-Munoz, J. M. Martinez-Villena, M. Bataller-Mompean, J. F. Guerrero and M. Wegrzyn, "Hardware implementation of real-time Extreme Learning Machine in FPGA: Analysis of precision, resource occupation and performance," *Computers and Electrical Engineering,* vol. 51, pp. 139 - 156, 2016.

[58] M. T. Mitchell, "Machine Learning," in *Machine Learning*, Chennai, McGraw Hill Education (India), 2013, pp. 81 - 124.

[59] D. A. Pomerleau, "ALVINN : An Autonomous Land Vehicle In a Neural Network," Carnegie Mellon University, Pittsburg, USA, 1989.

[60] Jagreet, "Overview of Artificial Neural Networks and its Applications," Xenonstack: A Stack Innovator, 5 May 2017. [Online]. Available: https://www.xenonstack.com/blog/overview-of-artificial-neural-networks-and-its-applications. [Accessed 1 October 2017].

[61] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford: Clarendon Press, 1995.

[62] W. H. Organisation, "Guidlines for drinking-water quality Surveilleance and Control of community supplies," Word Health Organisation, Geneva, 1997.

[63] M. Nielsen, "Neural Networks and Deep Learning," Michael Nielsen, August 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/. [Accessed 29 09 2017].

[64] A. Deshpande, "A Beginner's Guide to Understanding Convolutional Neural Networks," Adit Deshpande, 20 July 2016. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/. [Accessed 29 September 2017].

[65] D. A. Patterson and J. L. Hennessey, Computer Architecture A Quantitative Approach, Waltham, MA: Morgan Kaufmann, 2011.

[66] World Health Organization, "Guidlines for Drinking-water Quality | Incorporating the First Addendum," World Health Organization, Geneva, Switzerland, 2017.

[67] Z.-H. Zhou, N. V. Chawla, Y. Jin and G. J. Williams, "Big Data Opportunities and Challenges:Discussions from Data Analytics Perspectives," *IEEE Computational Intelligence Magazine,* vol. 9, no. 4, pp. 62-74, Nov. 2014.

[68] J. Shawe-Taylor and N. Cristianini, Kernel Methods for Pattern Analysis, Cambridge: Springer, 2004.

[69] F. Bach, “Sharp analysis of low-rank kernel matrix approximations,” *ArXiv,* vol. 1208.2015, 2013.

[70] S. Fine and K. Scheinberg, “Efficient SVM Training Using Low-Rank Kernel Representations”.

[71] S.-B. Lin and D.-X. Zhou, “Distributed Kernel-Based Gradient Descent Algorithms,” *Constructive Approximation,* vol. 47, pp. 249 - 276, 2018.

[72] M. T. Hagan, H. B. Demuth, M. H. Beale and O. De Jesus, Neural Network Design, Boston, MA, USA: PWS, 1996.

[73] M. Frean, “The Upstart Algorithm: A Method for Constructing and Training Feedforward Neural Networks,” *Neural Computation,* vol. 2, no. 2, pp. 198 - 209, 1990.

[74] M. Frean, “A "Thermal" Perceptron Learning Rule,” *Neural Computation,* vol. 4, no. 6, pp. 946 - 957, 1992.

[75] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms, Washington DC: Spartan Books, 1962.

[76] J. L. Gustafson, Interviewee, *The end of (numeric) error: An interview with John L. Gustafson..* [Interview]. April 2016.

[77] J. L. Gustafon and I. . T. Yonemoto , “Beating Floating Point at its Own Game: Posit Arithmetic,” *Supercomputing Frontiers and Innovations,* vol. 4, no. 2, p. 71–86, 2017.

Chapter 3

3. Data Collection

In this chapter, we explore the various parameters that have been chosen to measure using our electrodes to decide the water quality classification. An explanation of the importance of each parameter to water quality, and methods to measure their values, has been presented. The data collection presented in this chapter forms the ground truth for the training and validation of the ANNs that are discussed later in the thesis. The validation of ANN-based prediction of DO and EC has also been presented.

3.1 Introduction

Data assumes a pivotal role in the training process of an artificial neural network (ANN). The efficacy of an ANN in discerning the relationship between inputs and outputs is directly correlated with the size and diversity of the training dataset. Consequently, the collection of a comprehensive and dependable dataset for training purposes becomes paramount, particularly when the ANN is deployed in a critical application such as water quality monitoring.

The World Health Organization (WHO) stipulates five parameters - pH, Oxidation Reduction Potential (ORP), Dissolved Oxygen (DO), Electrical Conductivity (EC), and Turbidity - as fundamental indicators for assessing water potability [1]. To measure and classify water quality, we focussed on four key parameters - pH, Oxidation Reduction Potential (ORP), Dissolved Oxygen (DO), and Electrical Conductivity (EC) - all of which are measurable using electrodes. These parameters exhibit correlations with several other critical variables pivotal for the classification of water quality into the desired three categories, namely Potable, Irrigation, and Wastewater. Additionally, these parameters represent the least complex measurements to conduct in a water sample yet provide valuable insights for various other essential parameters such as Total Dissolved Solids (TDS) and biological activity.

For the present study, 1806 Ground and surface water samples have been collected from various locations in and around Pilani, Rajasthan, India. Out of these, 551 samples are from Wells, 752 from tap water, 53 samples from washroom commodes, and 451 samples from surface water open tanks. Testing has been carried out using these samples to measure the four water quality parameters that form the ground truth dataset. This dataset is used for training and validation of the Augmentation and Classification ANNs discussed in Chapter 6.

Table 3.1 gives the prescribed values of each of the four parameters associated with each of the three categories as per the WHO [1], Bureau of Indian Standards for Drinking [2] Water, and Indian Standard Guidelines for Quality of Irrigation Water [3]

Table 3.1: Parameter values for each category

Category	pH	ORP (mV)	DO (mg/dl)	EC (µS/cm)
Potable	6.5 – 8.0	-225 – 50	5 - 15	< 250
Agricultural	5.5 – 8.5	-250 – 250	Any other value	150 - 3000
Wastewater	<5.5 and >8.5	< -250 and > 250	Any other value	Any other value

3.2 Potential of Hydrogen (pH)

3.2.1 Theory

pH is one of the primary water quality indicators. The pH of a solution, product, or material is one of the most commonly computed and measured qualities. The pH of a solution is the measure of hydrogen ion concentration in the solution, which is the measurement of the acidity of the solution. Pure water has an equal amount of hydrogen and hydroxyl (OH^-) ion concentration. Thus, pure water dissociates slightly into the component ions as per the chemical equation shown in equation 1 below –

$$H_2O \Leftrightarrow [H]^+[OH]^- \quad (1)$$

A solution becomes acidic when it has an excess of hydrogen ions, while it becomes basic when it contains a dearth of H^+ ions or an excess of hydroxyl ions. The equilibrium constant for this reaction, K_w, is equal to 10^{-14} and is the product of H^+ and OH^-concentrations. This relationship can be described as follows:

$$[H]^+[OH]^- = K_w = 10^{-14} \quad (2)$$

where $[H]^+$ and $[OH]^-$are the concentrations of hydrogen and hydroxyl ions, respectively, in moles per litre. Considering Equation (1) and solving Equation (2), in pure water, where the concentrations of both ions are equal –

$$[H]^+ = [OH]^- = 10^{-7} \quad (3)$$

Instead of as moles as per litre, we define a quantity pH as the negative logarithm of $[H]^+$, so that: -

$$pH = -\log_{10}[H]^+ = \log_{10}\frac{1}{[H]^+} \quad (4)$$

$[H]^+$ equals 10^{-7} in a neutral solution, or pH = 7. The pH of the solution is then 7 with higher hydrogen ion concentrations. If the hydrogen ion concentration is 10^{-4}, the pH will be 4 and the solution will be acidic. The concentration of hydroxyl ions in this solution is $10^{-14}/10^{-4}$ = 10^{-10}. The presence of a substantial surplus of $[H]^+$ ions in the solution since $10^{-4} \gg 10^{-10}$ confirms that it is acidic. A basic solution, with a low concentration of $[H]^+$ ions have $[H]^+$, or a pH greater than 7. Dilute solutions have a pH range of 0 (extremely acidic; 1 mole $[H]^+$ ions per litre) to 14 (neutral) (very alkaline). Solutions having greater than 1 mole of H+ ions per litre have negative pH [2].

pH is a fundamental parameter for assessing water quality, reflecting the concentration of hydrogen ions (H+) and, consequently, the solution's acidity or alkalinity. In conjunction with EC, which measures the ability of water to conduct electricity due to dissolved ionic species, a qualitative estimation of water hardness can be obtained. High pH readings (above 8.5) coupled with elevated EC values (exceeding 200 mS/cm) are indicative of hard water, likely containing a high concentration of carbonate and bicarbonate ions. The synergistic analysis of pH and dissolved oxygen (DO) concentration can provide valuable insights into potential biological activity within a water sample. A low pH (below 5.5) measured alongside a high DO concentration (greater than 50 mg/dL) can be suggestive of biological decomposition processes. During decomposition, organic matter acidifies the water by releasing acidic byproducts, while microbial respiration consumes dissolved oxygen, leading to a temporary increase in DO [1].

The pH of drinking water must be monitored and stringently controlled. Changes in the pH values of drinking water can indicate the presence of toxic chemicals. These can then be found out with further testing of the water sample. However, for drinking purposes, it can be confidently stated that drastic divergence from prescribed limits of pH value renders the water

inconsumable. One of the primary reasons for water pH changing is the presence of ionic impurities such as undesirable amounts of nitrates, hydroxides, et al, which may render the water acidic. On the other hand, another class of impurities is heavy metals, which may make the water unnecessarily alkaline. While water alkalinity beyond the permissible range has its disadvantages, the presence of heavy metals could also make such water a potential carcinogen. Previous studies have established a correlation between the presence of As, Pb, U, No^{3-}, F^{-}, Cu, Zn, and other heavy metals and ions and carcinogenicity [3] [4].

pH measurement is also critical for agricultural water quality assessment because pH can affect the equilibrium state and rate of many reactions substantially. Many plants can only withstand a small pH range in the soil. A small change in blood pH can kill any animal. In many industrial processes, precise pH regulation is critical for high yield.

3.2.2 pH Measurement

pH is the measure of the concentration of Hydrogen ions in the liquid. pH indicates the acidity or alkalinity of the solution. To measure the pH of a there are many methods, starting with the simple litmus paper test. Also, we have titration-based methods, where we find out the concentration of the Hydrogen ion or hydroxide contributing salts, and then from there derive the concentration of hydrogen ion. Once we have the concentration of hydrogen ions in the solution, we calculate the pH of the solution using the Nernst Equation. The method we have used here is by using electrodes. These electrodes consist of two elements, a reference and a measuring element. The reference element is held inside a standardized alkaline solution, and the Hydrogen ion exchange across a glass bulb causes a potential difference between the two elements. This potential difference is directly proportional to the difference in Hydrogen ion concentration in the standard solution inside the electrode and the sample solution. This voltage then is used to calculate the pH of the sample solution.

The most basic measurement technique for pH is the litmus paper test. In the litmus paper test, the various values of pH are colour-coded based on how the litmus paper changes colour when a strip of the litmus paper is dipped in the solution. Since this is not a very accurate measure of the pH of a solution, more accurate methods have been proposed that involve measurement of the concentration of each ion and a mathematical relationship is established to give the pH of a solution. This mathematical relationship between the concentrations of the said ions is called

the Nernst Equation. Nernst Equation is a relation between the concentration of the oxidation and reduction half-cell concentrations of a redox reaction. In the dissociation of a solution into $[H]^+$ and $[OH]^-$ ions, the atom losing $[H]^+$ ion is said to be oxidized and the atom losing $[OH]^-$ ion is said to be reduced. Equations 1 through 4 show the mathematical relationship between the concentration of $[H]^+$ion and pH value. This relationship is thus used to measure the value of pH in electrode-based measurement.

Fritz Haber and Zygmunt Klemensiewicz [5] [6] created the first glass pH electrode in 1908. It is commonly considered that the electrode was constructed in 1909 because the paper describing it was published a year later [5]. The original electrode is made of glass, filled with a strong electrolyte, and has an Ag/AgCl half-cell with Ag wire as a contact. The difference in H+ activity on both sides builds up a potential difference on the sides of thin glass in the bubble, which is measured with the use of reference electrodes and is known to be proportional to the pH on the outside of the bubble. Even today the general principle of a pH electrode hasn't changed much.

Glass electrodes are often used as pH electrodes. A typical model is a glass tube with a little glass bubble at the end. The electrode's interior is normally filled with a buffered chloride solution in which silver wire coated in silver chloride is immersed. The pH of the internal solution can range from 1.0 (0.1M HCl) to 7.0 (7.0M HCl) (different buffers used by different producers). Figure 3.1 shows the schematic of a general glass bulb pH electrode.

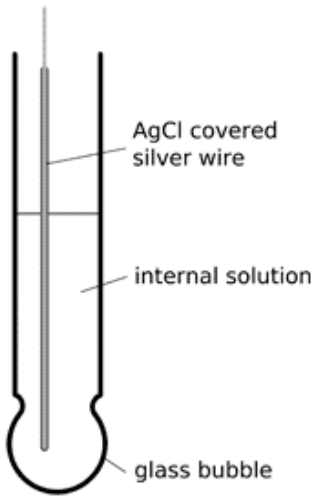


Figure 3. 1: A generic Glass bulb pH Electrode [5]

The glass bubble is the electrode's active component. While the tube walls are thick and strong, the bubble walls are as thin as possible. Both internal and exterior solutions protonate the glass surface until equilibrium is reached. The adsorbed protons charge both sides of the glass, and this charge is what causes the potential difference. The Nernst equation describes this potential, which is proportional to the pH difference between the solutions on both sides of the glass. The majority of the commercially available pH electrodes are combination electrodes, which combine a glass $[H]^{+}$ion-sensitive electrodes with an extra reference electrode in one housing. Figure 3.2 shows a combination pH electrode.

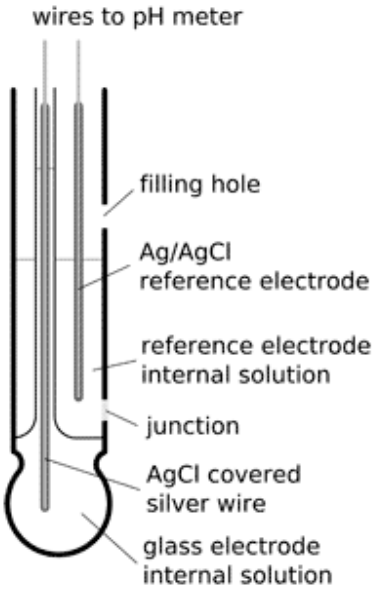


Figure 3. 2: Combination pH Electrode [5]

The processes that must take place when measuring pH define the construction of a combination electrode. In the glass electrode, we need to measure the difference in potentials between the two sides of the glass. We'll need a closed circuit for this. The pH meter and the internal and external solutions complete the circuit. However, for accurate and steady measurements the reference electrode must be isolated from the solution so that it does not cross-contaminate and connecting and isolating two solutions at the same time is not an easy process. The electrode body has a tiny hole via which the connection is made. Porous membrane or ceramic (asbestos in previous models) wicks are used to close this hole. The internal solution flows slowly via the junction, therefore these electrodes are referred to as "flowing electrodes." The internal solution is gelled in gel electrodes to reduce leaks.

Figures 3.3 and 3.4 show pictures of the pH electrode used for this study. In Figure 3.4 the detailed view of the tip of the electrode is shown, where the glass bulb can be seen. The potential across this glass bulb is measured as the measurement of the pH of the solution.

Figure 3. 3: pH electrode dipped in standard solution

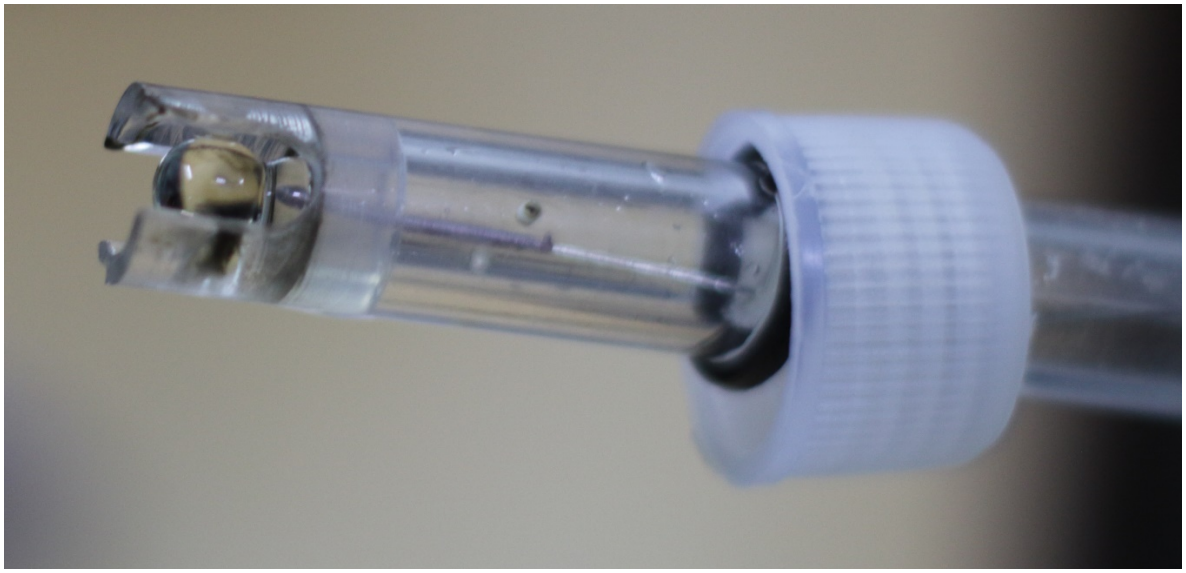


Figure 3. 4: pH electrode - detailed view of the interacting glass bulb

- *The Nernst Equation*

The Nernst equation defines the potential of an electrochemical cell as a function of the concentration of the ions taking part in the chemical reaction. The Nernst Equation is given by Equation 5: -

$$E = E_0 + \frac{RT}{nF}\ln(Q) \tag{5}$$

Where Q is the reaction quotient; R represents the Universal Gas Constant; T is the temperature in Kelvin; n is the number of moles of ions or electrons exchanged in the reaction; F is the Faraday Constant. E_0 represents the standard potential of the reaction involved in the solution in question. E represents the reduction potential of the reaction in question.

The pH electrode used in the project is an Ag (Silver) electrode coated with AgCl (Silver Chloride) dipped in a KCl (Potassium Chloride) solution.

For the said electrode, being used at standard room temperatures, Equation 5 becomes [7]: -

$$E = E_0 - 2.303 * \frac{0.0256}{n} * pH \tag{6}$$

At room temperature, for 1 Molar KCl solution, n = 1, E_0, $E_0 = 0.0235$; E is the voltage received on the electrode [8].

Thus, equation 6 becomes that:

$$pH = 0.0235 - 0.0256 * E \quad (7)$$

Since the pH of neutral solution is 7, an offset of 7 is given to equation 7, making the equation.

$$pH = 0.0235 - 0.0256 * E + 7 \quad (8)$$

Thus Equation 8 establishes a relationship between the reference electrode potential and pH. This relation is then used in Arduino programming to calculate the pH of the solution. To achieve that, we connect the pH electrode to the analog port of the Arduino board. In the program, we convert the analog readings to digital voltage. This digital voltage reading is then used in the equation to convert the reading into pH readings.

There are several pH measuring kits available in the market. We have in particular, used the Labtronics Water Quality kit, YSI Sonde, and Atlas Scientific pH kit (paired with Arduino Uno) as standards against which the pH readings of the proposed device have been validated.

3.2.3 pH Data

Table 3.2 shows the pH measurement data of only 100 samples (out of 1806 samples) collected from in and around the BITS Pilani, Pilani campus. The water has been collected from various groundwater sources, and surface water examples have been collected from manmade water receptacles on the campus. Some water samples were also collected from drinking water sources around the campus. To remove conformity bias, the water samples were blinded and shuffled but not mixed.

Table 3.2: pH measurement comparison against standard devices

S. No.	pH - Atlas Scientific electrode	pH - Labtornics LT-59
1	8.6	8.6
2	7.8	7.8
3	7.4	7.4
4	7.0	7.0
5	6.8	6.9
6	7.3	7.3

7	6.5	6.5
8	7	7
9	6.8	6.8
10	6.6	6.5
11	5.7	5.7
12	8	8
13	6.2	6.2
14	6.1	6.1
15	7	7
16	6.3	6.3
17	6.5	6.5
18	6.3	6.3
19	7	7
20	6.8	6.8
21	8	8
22	7	7
23	7.4	7.4
24	8	8
25	6	6
26	6.3	6.3
27	8.5	8.5
28	8	8
29	5.8	5.8
30	7	7.1
31	5.5	5.5
32	7	7
33	8.5	8.5
34	7.5	7.5
35	6.8	6.8
36	7	7
37	8.2	8.2
38	6.5	6.5

39	8	8
40	6.5	6.5
41	8.5	8.5
42	8	8
43	8.5	8.5
44	8	8
45	7.7	7.8
46	6	6
47	8	8
48	7	7
49	6.6	6.6
50	8	8
51	8.5	8.5
52	8.1	8.1
53	8.3	8.3
54	7	7
55	7.3	7.3
56	8	8
57	7.7	7.7
58	8	8
59	6.6	6.6
60	7	7
61	7	7.2
62	8	8
63	7	7
64	6.8	6.8
65	7.1	7.1
66	7.5	7.5
67	7.8	7.8
68	7	7
69	7	7
70	7.2	7.2

71	8.6	8.6
72	8	8
73	6.3	6.3
74	7.5	7.5
75	6.7	6.7
76	6.2	6.2
77	6.9	6.9
78	7	7
79	6.5	6.5
80	7	7
81	8.5	8.5
82	6.3	6.3
83	7.3	7.3
84	7.5	7.5
85	6.6	6.6
86	7	7
87	7.2	7.2
88	6.2	6.2
89	7.5	7.5
90	6.5	6.5
91	7	7
92	7.6	7.6
93	7.1	7.1
94	8.2	8.2
95	6.8	6.8
96	7.4	7.4
97	7.6	7.6
98	7	7
99	7	7
100	8	8

3.3 Oxidation-Reduction Potential (ORP)

3.3.1 Theory

The Oxidation Reduction Potential (ORP or Oxidation Reduction Potential) is a measurement of an aqueous system's ability to release or gain electrons as a result of chemical processes. In the oxidation process, electrons are lost, but in the reduction process, electrons are gained. ORP is commonly used in water treatment to manage chlorine and chlorine dioxide disinfection in cooling towers, swimming pools, potable water sources, and other water applications. ORP has a considerable influence on the life span of bacteria in water studies. ORP values can also help in estimating the presence of heavy metals in the water sample. Heavy metals are present in ionic salt forms in water and form positive ions inside the water body, increasing the Oxidation potential of the water and also the Electrical Conductivity of the water. Thus, for a high value of ORP (ORP > 250mV) and high EC values (EC > 300µS), a heavy metal presence in the water can be estimated qualitatively.

An ORP electrode, which has been used in the study, has been shown in Figure 3.5. In Figure 3.6, a detailed close-up shot of the ORP electrode is shown. The Platinum (Pt) electrode which measures the potential of the Redox reaction can be seen in this close-up shot of the electrode.

Figure 3. 5: ORP Electrode used in the study

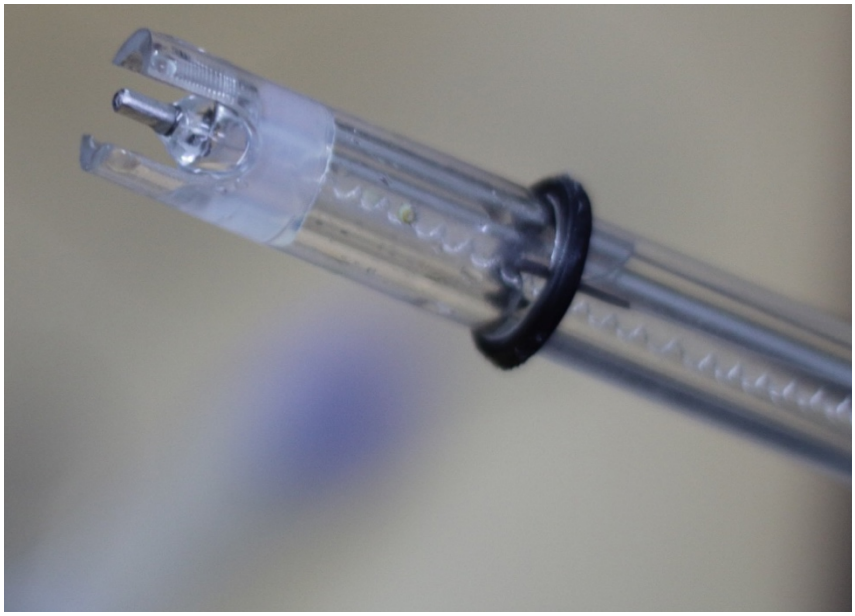


Figure 3. 6 : ORP electrode - detailed view of the interacting Pt Electrode

3.3.2 ORP Measurement

The Oxidation Reduction Potential (ORP) tells whether the sampled solution is oxidising in nature or reducing. For an oxidising solution, we get a positive oxidation potential, meanwhile, for a reducing solution, we get a negative oxidation potential. The method of measurement is similar to that of pH as ORP is also calculated based on the concentration of oxidising ions to reducing ions. Neutral water with pH =7 has an ORP of 0 millivolts.

The ORP sensor works quite similarly to a pH sensor, but it employs an inert metal (typically platinum) half-cell instead of a pH-sensitive glass membrane half-cell. A potentiometric measurement is performed using a two-electrode setup. Depending on the test solution, the platinum electrode acts as an electron donor or acceptor. For comparison, a reference electrode is utilised to provide a continuous, reliable output. A restrictive diaphragm makes electrical contact with an electrolyte solution (e.g., saturated potassium chloride KCl solution) from the reference half-cell. The Nernst equation describes the electrode behaviour [8] [9]:

$$E = E_0 - \frac{RT}{nF} \ln \frac{C_{ox}}{C_{red}} \tag{9}$$

Where,

E = Measured potential (mV) between the platinum (Pt) and the reference electrode

E_0 = Measure potential (mV) between the Pt and the reference electrode at a concentration $C_{ox} = C_{red}$

R = Universal gas constant

T = temperature in Kelvin

F = Faraday Constant

n = Electrical charge of the ion

C_{ox} = Concentration of oxidizing agent in moles/L

C_{red} = Concentration of reducing agent in moles/L

The potential is measured against a reference electrode, commonly Ag/AgCl, using platinum as the indicating sensor. Other noble metals, such as gold or silver, can also be employed.

When compared to pH measurements, ORP readings are slow. While a pH value can be produced in a matter of seconds, achieving a steady ORP value might take several minutes, if not hours. The platinum surface state has a significant impact on ORP behaviour. ORP probes "in use" will display different values than a new unconditioned ORP electrode. [8]

The probe was calibrated before being used to measure a sample.

Calculate the offset by:

$$E_{offset} = E_{standard} - E_{measured} \quad (10)$$

Drinking water (DW) has a low ionic strength (e.g., 80 to 200 S), which might cause issues with stabilisation time and final reading. After calibrating the ORP probe, rinse it with drinking water before transferring it to a fresh beaker containing the DW sample to be analyzed. Wait at least 15 minutes for the first reading, then check for stability every 5 minutes. It could take many hours to achieve a final reading depending on the temperature (low takes longer) and conductivity (low takes longer) [8].

Surface water (SW) has a conductivity of greater than 600 S/cm in most cases. The ORP measurement can be performed immediately following the calibration. Because sufficient ORP active species are present in rivers, reservoirs, and wells, the measurement should be steady within 6 minutes [8].

3.3.3 ORP Data

Table 3.3 shows the ORP measurement data of 100 samples collected from in and around the BITS Pilani, Pilani campus.

Table 3. 3: ORP measurement comparison against standard devices

S. No.	ORP - Atlas Scientific Electrode	ORP - Labtornics LT-59
1	3.01	3.01
2	0.92	0.92
3	1.73	1.73
4	3.28	3.28
5	3.32	3.32
6	2.74	2.74
7	0.03	0.03
8	4.22	4.22
9	3.08	3.08
10	2.53	2.53
11	0.01	0.01
12	3.39	3.39
13	0.04	0.04
14	0.02	0.02
15	3.4	3.4
16	0.15	0.15
17	0.01	0.01
18	0.01	0.01
19	1.46	1.46
20	1.58	1.58
21	4.63	4.63
22	0.43	0.43
23	0.44	0.44
24	2.83	2.83
25	0.01	0.01
26	0.01	0.01

27	2.89	2.89
28	0.07	0.07
29	0.01	0.01
30	7.11	7.11
31	0.09	0.09
32	0.06	0.06
33	3.33	3.33
34	3.28	3.28
35	0.04	0.04
36	0.22	0.22
37	3.22	3.22
38	0.03	0.03
39	3.43	3.43
40	0.13	0.13
41	2.45	2.45
42	4.95	4.95
43	4.95	4.95
44	7.9	7.9
45	2.48	2.48
46	0.05	0.05
47	4.93	4.93
48	0.08	0.08
49	0.03	0.03
50	3.11	3.11
51	2.82	2.82
52	3.12	3.12
53	3.11	3.11
54	0.85	0.85
55	0.91	0.91
56	0.91	0.91
57	0.64	0.64
58	1.22	1.22

59	0.23	0.23
60	0.24	0.24
61	0.23	0.23
62	0.59	0.59
63	0.23	0.23
64	0.24	0.24
65	0.28	0.28
66	0.93	0.93
67	1.55	1.55
68	0.33	0.33
69	0.49	0.49
70	0.24	0.24
71	1.19	1.19
72	0.85	0.85
73	0.24	0.24
74	0.59	0.59
75	0.23	0.23
76	0.23	0.23
77	0.23	0.23
78	0.25	0.25
79	0.23	0.23
80	0.25	0.25
81	4.34	4.34
82	0.25	0.25
83	0.36	0.36
84	0.64	0.64
85	0.25	0.25
86	0.25	0.25
87	0.23	0.23
88	0.26	0.26
89	0.9	0.9
90	0.42	0.42

91	0.25	0.25
92	1.21	1.21
93	0.24	0.24
94	1.22	1.22
95	0.25	0.25
96	1.08	1.08
97	0.49	0.49
98	0.51	0.51
99	0.27	0.27
100	0.64	0.64

3.4 Dissolved Oxygen (DO)

3.4.1 Theory

The amount of oxygen dissolved in water is referred to as dissolved oxygen (DO). The atmosphere and aquatic vegetation both provide oxygen to water bodies. Running water, such as a fast-moving stream, dissolves more oxygen than motionless water, such as that found in a pond or lake. When it comes to drinking water sources, the overall taste of the water is determined by the amount of dissolved oxygen present in the water. When DO levels are high, the drinking water has a superior taste to it. It's crucial to note, however, that higher DO levels are harmful to numerous components and systems utilised in the distribution and treatment of drinking water. For example, excessively high DO levels aid the corrosion of water pipelines. The amount of dissolved oxygen in the water is significant for various reasons. Any dissolved oxygen in the water will take up a certain amount of space [10]. When dissolved oxygen levels are high, there isn't much room in the water for other dissolved chemicals to exist. If the dissolved oxygen levels in the water are extremely low, minerals in the lake's bed will begin to dissolve in the water at a faster rate. Although a high mineral content in water may not cause health problems, it might alter the water's scent and taste [11].

3.4.2 Measurement of DO

Dissolved oxygen levels can be measured by a basic chemical analysis method (titration method), an electrochemical analysis method (diaphragm electrode method), and a

photochemical analysis method (fluorescence method). The diaphragm electrode method is the most widely used method [8].

The diaphragm electrode technology uses electrodes to monitor the amount of oxygen passing through an highly oxygen-permeable diaphragm. The galvanic electrode method and the polarographic electrode method are two ways of determining dissolved oxygen levels using electrodes. These methods each have their own set of benefits and drawbacks, thus the method that best suits the situation is chosen.

A PTFE membrane, an anode in an electrolyte, and a cathode make up a galvanic dissolved oxygen probe. At a consistent rate, oxygen molecules diffuse through the probe's membrane (without the membrane the reaction happens rapidly). When oxygen molecules enter the membrane, they are reduced at the cathode, resulting in a tiny voltage. The probe will produce 0 mV if no oxygen molecules are present. The mV output from the probe increases as the oxygen level rises. In the presence of oxygen, each probe will produce a distinct voltage. Only one thing remains constant: 0mV Equals 0 Oxygen [12]. Figure 3.7 shows a schematic of the galvanic DO probe. Figures 3.8 and 3.9 show the DO electrode. Figure 3.8 shows a detailed view of the interacting membrane of the electrode across which the DO concentration is measured.

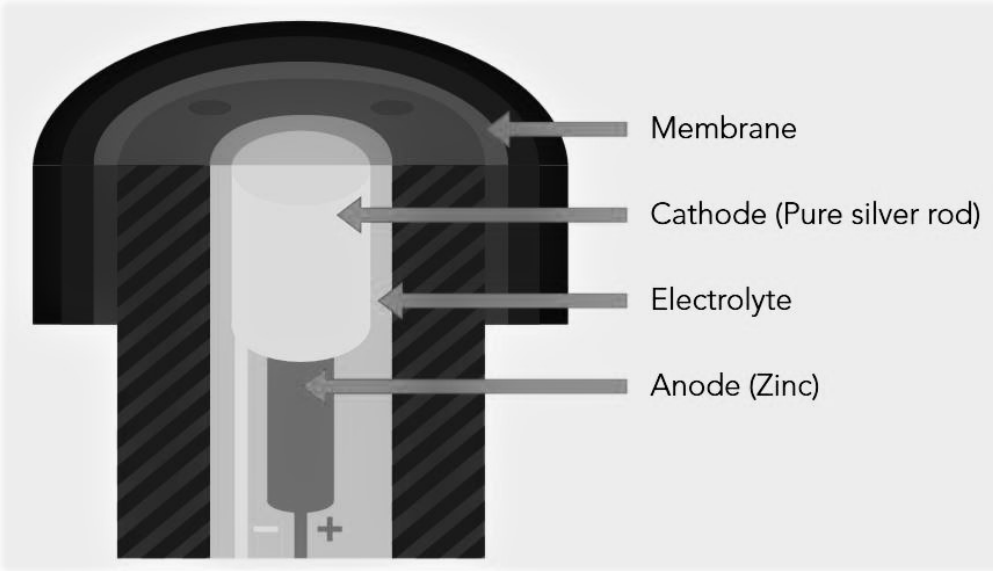


Figure 3. 7: A schematic of Galvanic DO Probe [11]

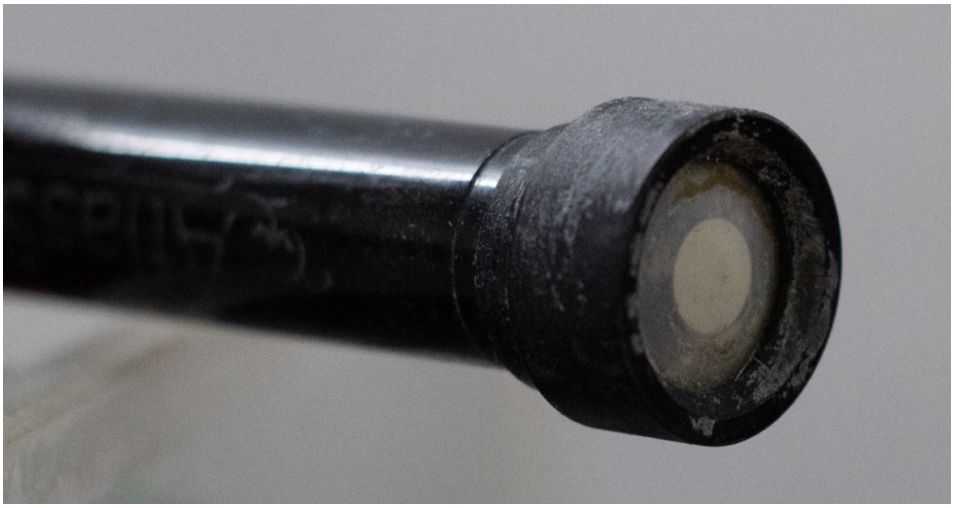


Figure 3. 8: DO electrode - detailed view of the interacting membrane

Figure 3. 9: DO Electrode

3.4.3 DO Data

Table 3.4 shows the DO measurement data of 100 samples collected from in and around the BITS Pilani, Pilani campus.

Table 3.4: DO measurement comparison against standard devices

S. No.	DO - Atlas Scientific Electrode	DO - Labtornics LT-59
1	10	10
2	11.5	12
3	14.5	14.5
4	12.5	12
5	12	12
6	16	16
7	15	14.5
8	14	14.3
9	10.3	10.3
10	10.4	10.4
11	13	13
12	16	16
13	15.5	15.5
14	13	13
15	12.5	12.5
16	13	13
17	11.5	11.5
18	10	10
19	1.7	1.7
20	10.8	10.8
21	12	12
22	12	12
23	9.5	9.5
24	11	11
25	9	9
26	12	12
27	10	10
28	8.5	8.5
29	7.5	7.5
30	8	8
31	9.5	9.5
32	10	10
33	11	10.8

34	10.5	10.5
35	9.5	9.5
36	9.7	9.7
37	9.8	10
38	9	9
39	7	7
40	9	9
41	10	10
42	10.2	10
43	8.8	8.8
44	4.5	4.5
45	9.8	9.8
46	8.5	8.5
47	5.5	5.5
48	8.2	8.2
49	7.5	7.5
50	8.2	8.2
51	7.5	7.5
52	8.5	8.5
53	7	7
54	8.5	8.5
55	9	9
56	9	9
57	8.5	8.5
58	9	9
59	8.5	8.5
60	9.5	9.5
61	9	8.5
62	6	6
63	10.2	10.2
64	9	9
65	7.8	7.8
66	7.5	7.5
67	6	6
68	7	7

69	9	9
70	9	9
71	7.3	7.3
72	6.5	6.5
73	8.5	8.5
74	8	8
75	7.8	7.8
76	8.6	8.6
77	7.4	7.4
78	8	8
79	7.5	7.5
80	8	8
81	7	7
82	7.6	7.6
83	7.4	7.4
84	9.5	9.5
85	9	9
86	7.5	7.5
87	9.2	9.2
88	9.5	9.5
89	8.3	8.3
90	8.7	8.7
91	9.6	9.6
92	6.4	6.4
93	10	10
94	7	7
95	10.5	10.5
96	8	8
97	9.5	9.5
98	8	8
99	8	8
100	7.5	7.5

3.5 Electrical Conductivity (EC)

3.5.1 Theory

The ability of a solution, a metal, or a gas - in other words, all materials - to pass an electric current is known as conductivity. Current is carried by cations and anions in solutions, but electrons carry it in metals. It serves as an indicator of dissolved ionic solid concentration and salinity in water. Compounds like calcium, magnesium, and sodium salts, which can affect the hardness and alkalinity of a water supply, are known as dissolved ionisable solids. High conductivity water does not inherently endanger human health, but it can cause corrosion in industrial equipment and plumbing systems, scale build-up, a mineral-like taste in drinking water, and dissolved solid concentration problems in agriculture. The conductivity limit for drinking water is 2500 micro-Siemens per centimetre (S/cm). Conductivity monitoring can offer information about the source and suggest whether geological conditions or pollutants are affecting water quality [13].

Moreover, the Electrical Conductivity also has almost linear relations with many other parameters, such as Total Dissolved Solids (TDS), Salinity, and Specific gravity. Amongst these, TDS is of particular interest as the TDS is one of the primary water quality parameters mentioned in the drinking water quality guidelines such as the WHO guidelines for Drinking Water Quality and the Water Quality standards published by the Bureau of Indian Standards, and various other local and national government guidelines. A water sample having TDS > 50 PPM and TDS < 150 PPM is considered excellent for drinking purposes [14] [15]. Electrical Conductivity has a linear relationship with TDS as shown below [16]:

$$TDS\left(\frac{mg}{l}\right) = 0.65 \times EC\left(\frac{\mu S}{cm}\right) \tag{11}$$

3.5.2 Measurement of EC

A steady, alternating electrical current (I) is applied to two electrodes immersed in a solution, and the resulting voltage is measured (V). Cations migrate to the negative electrode, anions to the positive electrode, and the solution acts as an electrical conductor during this process [8].

Conductivity is usually tested in electrolyte aqueous solutions. Electrolytes are ions containing substances, such as ionic salt solutions or chemicals that produce ions in the solution. The ions in the solution are in charge of transporting the electric current. Acids, bases, and salts are

examples of electrolytes, which can be strong or weak. Water has the ability to stabilize the ions generated through a process called solvation, hence the majority of conductive solutions studied are aqueous solutions [17].

Inside the conductivity probe, two electrodes are positioned opposite each other, and an AC voltage is given to the electrodes, causing cations to move to the negatively charged electrode and anions to move to the positively charged electrode. The electrical conductivity of liquid increases as the amount of free electrolyte in the liquid increases.

A conductivity probe is a very simple instrument. It consists of two conductors separated by a fixed distance and having a fixed surface area. The conductivity cell is the measurement of distance and surface area. The cells' distance and surface area are quantified as the conductivity cell K constant.

The resistance of the solution (R) can be calculated using Ohm's law

$$V = R * I \; or \; R = V / I \tag{12}$$

Where: V = voltage (volts), I = current (amperes), R = resistance of the solution (ohms)

Conductance (G) is defined as the reciprocal of the electrical resistance (R) of a solution between two electrodes.

$$G = 1/R \; (S) \tag{13}$$

[S = Siemens]

The conductivity meter measures the conductance and displays the reading converted into conductivity.

Conductivity is the ability of a solution to pass current.

$$\kappa = G * K \tag{14}$$

Where κ = conductivity (S/cm); G = conductance (S); K = cell constant (cm–1) [8].

Figure 3.10 shows a schematic view of the process of measuring Electrical conductivity. Figure 3.11 shows a close-up view of the eye hole where the two plates of the electrode are placed. An analog voltage is applied across the two plates and the conductivity of the water between the two plates gives us the conductivity of the water sample. Figure 3.12 shows a snapshot of the EC electrode.

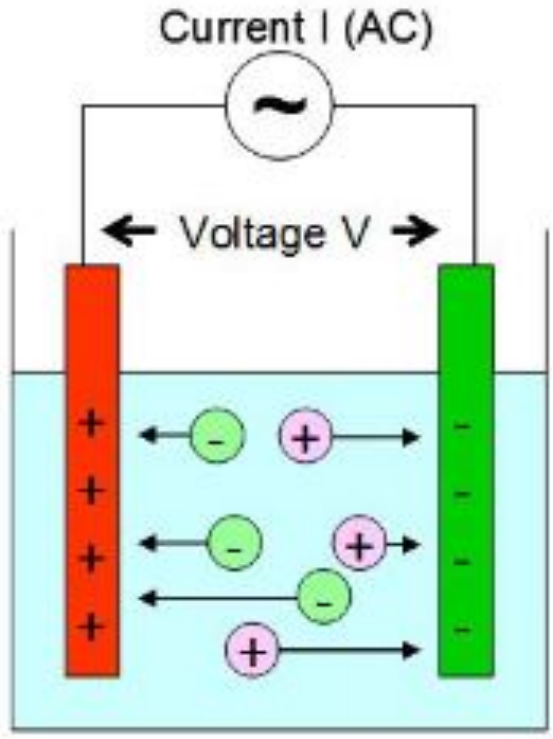


Figure 3. 10: Electrical Conductivity measurement schematic [7]

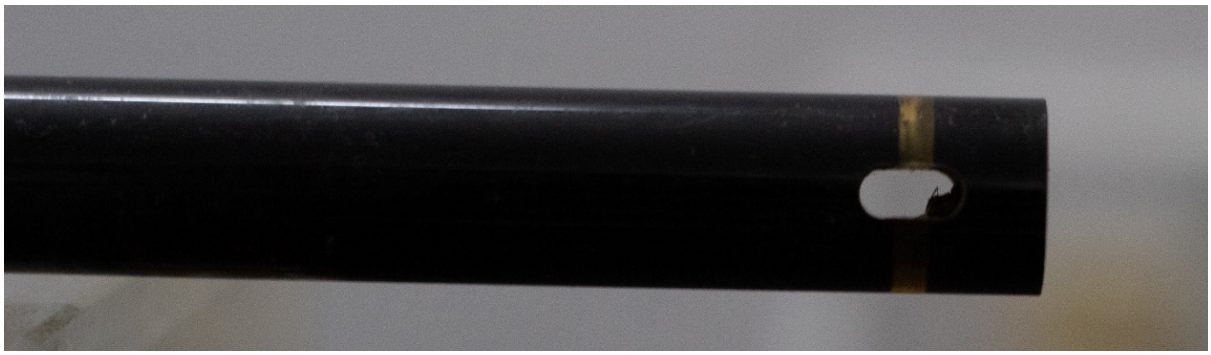


Figure 3. 11: EC electrode detailed view. The two plates between which the conductivity is measured are placed inside the eye-hole

Figure 3. 12: The EC Electrode

3.5.3 EC Data

Table 3.5 shows the EC measurement data of 100 samples collected from in and around the BITS Pilani, Pilani campus.

Table 3.5: EC measurement comparison against standard devices

S. No.	EC - Atlas Scientific Electrode	EC - Labtornics LT-59
1	281	281
2	294	294
3	277	277
4	292	292
5	302	302
6	294	294
7	298	298
8	308	308
9	277	277
10	278	278
11	270	270
12	254	254
13	293	293
14	299	299
15	283	283

16	298	298
17	291	291
18	295	295
19	288	288
20	257	257
21	265	265
22	271	271
23	284	284
24	259	259
25	211	211
26	235	235
27	215	215
28	245	245
29	264	264
30	273	273
31	262	262
32	263	263
33	252	252
34	266	266
35	269	269
36	273	273
37	279	279
38	275	275
39	267	267
40	277	277
41	264	264
42	263	263
43	268	268
44	254	254
45	259	259
46	275	275
47	273	273
48	280	280
49	284	284
50	267	267

51	263	263
52	269	269
53	248	248
54	255	255
55	247	247
56	257	257
57	260	260
58	273	273
59	265	265
60	266	266
61	271	271
62	253	253
63	264	264
64	281	281
65	258	258
66	253	253
67	258	258
68	265	265
69	255	255
70	270	270
71	247	247
72	238	238
73	278	278
74	269	269
75	270	270
76	252	252
77	271	271
78	269	269
79	270	270
80	279	279
81	238	238
82	265	265
83	263	263
84	245	245
85	267	267

86	278	278
87	274	274
88	269	269
89	265	265
90	282	282
91	281	281
92	274	274
93	272	272
94	256	256
95	263	263
96	276	276
97	258	258
98	266	266
99	271	271
100	297	297

3.6 Validation of Data

The data collected and presented in this chapter serves as the training set for the ANN designed in chapters 4, 5, and 7. A set of data values has been collected for each sensor using standard methods of measuring those parameters. This set has been used to validate the data predicted by the ANN in Chapter 7. Validation of data is necessary to ensure the proper functioning of the device designed to prevent any health issues for the users.

To validate the data, the standard data has been made using 1806 samples of ground and surface water samples collected from various locations near the BITS Pilani campus, Vidya Vihar, Pilani, Rajasthan, India. These samples were measured using standardised methods and instruments. 100 data points from the whole data set have been presented for each parameter in Appendix A. An ANN has been designed to predict the values of some of the parameters, which are detailed in Chapter 6.

The prediction has been made to bring down the cost of Water Quality Classification. The predicted parameter values have been validated against the data measured and presented in this chapter and the comparison tables are given in Appendix A. The results of the validation are presented and discussed in detail in Chapter 6.

3.7 Conclusion

This chapter discusses the collection of various parameter data using standard laboratory methods. The data collected in this chapter is used as the ground truth for training and validation of Augmentation ANN and Classification ANN presented in Chapter 6.

3.8 References

[1] "Water on the Web | Understanding | Water Quality | Parameters | pH," Water On The Web, 17 January 2008. [Online]. Available: https://waterontheweb.org/under/waterquality/ph.html. [Accessed April 2018].

[2] J. J. Pierce, R. F. Weiner and P. A. Vesilind, "Measurement of Water Quality," in *Environmental Pollution and Control*, 1998, pp. 57 - 76.

[3] G. Kaur, R. Kumar, S. Mittal, P. K. Sahoo and U. Vaid, "Ground/drinking water contaminants and cancer incidence: A case study of rural areas of South West Punjab, India," *Human and Ecological Risk Assessment: An International Journal,* vol. 27, no. 1, pp. 205-226, 2019.

[4] B. S. Bajwa, S. Kumar, S. Singh, S. K. Sahoo and R. M. Tripathi, "Uranium and other heavy toxic elements distribution in the drinking water samples of SW-Punjab, India," *Journal of Radiation Research and Applied Sciences,* vol. 10, no. 1, pp. 13 - 19, 2017.

[5] F. Haber and Z. Hlemensiewicz , "Über elektrische Phasengrenzkräfte," *Zeitschrift für Physikalische Chemie,* vol. 67U, no. 1, pp. 385 - 431, 1909.

[6] ph-meter.info, "ph-meter.info," ph meter, [Online]. Available: http://www.ph-meter.info. [Accessed 10 May 2022].

[7] Drinking Water Sectional Committee, Food and Agricultural Division, "Indian Standard Drinking Water - Specification (Second Revision)," Bureau of Indian Standards, New Delhi, 2012.

[8] D. A. Bier, Electrochemistry: Theory and Practice, 2018: HACH .

[9] M. Chaplin, "Water Redox," Water Structure and Science, 5 November 2021. [Online]. Available: https://water.lsbu.ac.uk/water/water_redox.html. [Accessed 12 May 2022].

[10] Sensorex, "The Importance of Dissolved Oxygen in Water and Water Systems - Sensorex," Sensorex - A Halma Company, 17 November 2020. [Online]. Available: https://sensorex.com/2020/11/17/importance-of-dissolved-oxygen/. [Accessed 15 May 2022].

[11] United States Environmental Protection Agency, "Indicators: Dissolved Oxygen | US EPA," United States Environmental Protection Agency, 7 July 2021. [Online]. Available: https://www.epa.gov/national-aquatic-resource-surveys/indicators-dissolved-oxygen. [Accessed 15 May 2022].

[12] Atlas Scientific, "Atlas Scientific Lab Grade D. O. Probe," Atlas Scientific, 2021.

[13] S. Jones, "Consuctivity in Drinking Water - Water Library | Acorn Water - H2OLabCheck," Acorn Water - H2OLabCheck, 2 January 2020. [Online]. Available: https://www.h2olabcheck.com/blog/view/conductivity. [Accessed 17 May 2022].

[14] World Health Organisation, “Guidelines for Drinking-water Quality (Fourth,” World Health Organisation, Geneva, Switzerland, 2017.

[15] Central Bureau of Health Intelligence, “NationalHealth Profile 2018,” Ministry of Health and Family Welfare, Government of India, New Delhi, 2018.

[16] A. . F. Rusydi, “Correlation between conductivity and total dissolved solid in various type of water: A review,” in *IOP Conference Series: Earth and Environmental Science, Volume 118, Global Colloquium on GeoSciences and Engineering 201718–19 October 2017*, Bandung, Indonesia, 2017.

[17] A. Scientific, “Conductivity K1.0 Kit | Atlas Scientific,” September 2021. [Online]. Available: https://files.atlas-scientific.com/EC_K_1.0_probe.pdf. [Accessed 17 May 2022].

[18] Organisation, World Health, “Diarrhoeal Disease,” World Health Organisation, 2017.

Chapter 4

4. Efficient ANN Hardware Implementation through Mathematical Approximation in IEEE 754 Representation

This chapter explores ANN architectures and mathematical approximations for activation functions suitable for digital hardware implementation.

4.1 Introduction

Artificial Neural Networks (ANN) have many uses in the modern world. An artificial neural network is very effective for certain problems, such as learning to interpret complex real-world sensor data. ANN learning is well-suited to problems in which the training data corresponds to complex sensor data, such as inputs from various sensors, which, when taken together, do not have an apparent relationship with output. Classification of data into classes and predicting future data based on system behaviour where the system behaviour cannot be modelled mathematically can be modelled by ANN accurately. ANN is also applicable to problems for which symbolic representations are often used, such as decision tree learning. [1]

With varying applications, various architectures have been developed for ANN. A large variety of architectures for ANN makes it challenging to determine the architecture that suits the desired application. To find out which architecture suits the chosen application of Water Quality Classification necessitates exploration of architectures as mentioned below: -

4.1.1 Multilayer Perceptron Feedforward Network with Backpropagation

The most basic ANN model is the Multilayer Perceptron (MLP) model with a Feedforward network and Backpropagation learning algorithm. It is based on the McCulloch-Pitts model of Perceptron. MLP consists of many perceptrons called a neuron. MLP model can be applied to almost all four tasks expected from ANN. It is a supervised learning model, and the learning algorithms are the Backpropagation algorithms. Due to their structural and mathematical simplicity, MLPs are best suited for hardware implementation for supervised classification applications such as Water/Air Quality Indexing, Facial Recognition, and Natural Language Processing.

4.1.2 Radial Basis Function

Radial Basis Function (RBF) Neural Networks are structurally similar to MLP for supervised learning algorithms, but for RBF, there is only one hidden layer between the input and output

layers. This hidden layer is the feature layer. The number of neurons involved in the classification decides the number of neurons in the hidden layer. So structurally, RBF architecture is more predictable than ANN. However, due to the limited number of hidden neurons, the classification process is slow on RBF and power consuming.

4.1.3 Support Vector Machines

Support Vector Machines (SVM) are another type of machine learning algorithm widely used for classification. SVMs are widely used for classification purposes like handwriting recognition, face detection, gene classification, etc. But to ensure convergence, SVMs increase the dimensionality of the vector space, thus making it unsuitable for digital hardware applications.

4.1.4 Constructive C-Mantec

Constructive Neural Networks (CoNN) are also called second-generation ANNs. Constructive neural networks start with only one hidden neuron but increase the number of hidden neurons by generating new neurons whenever specific learning parameters are not met. One such Constructive neural network model is the C-Mantec (Competitive majority network trained by error correction) algorithm. The C-Mantec algorithm uses the thermal perceptron learning method to decide the number of iterations, after which training of existing neurons is to be stopped and a new neuron is to be added to the network. CoNN algorithms can be very power and resource-efficient because they generate just the optimum number of neurons required for the given application. However, in the hardware space, the neurons can only be generated if they are already coded. This feature of CoNN introduces the problem of hardware resources occupied by non-functional units. Also, the neuron generation process makes the operation slower. Such implementation can be done using a hardware-software mixed design approach like FPGA or Processor supported IC design but CoNNs are particularly unsuitable for application ASICs.

4.1.5 Spiking Neural Networks

Spiking neural networks are third-generational neural network architectures. The spiking neural networks have a muscle memory approach to learning the data. In a spiking neural network, the weights of the synaptic connections that fired more often due to minimum error are incremented, while those of the others that generate more errors are decremented. Thus, the network grows a path memory of the input-output path taken more often than others, while it forgets the path that is not used frequently. It is much the same as human muscle memory

related to practising one particular task and not practising others. While this closely replicates the biological process, the learning algorithm here is not very well suited for applications like classification, where all the paths must be traversed.

4.1.6 K-means clustering

K-means clustering is a popular and one of the simplest unsupervised algorithms for classification. K-means clustering is used for all classification applications where the classes and/or the classification rule are not pre-defined. Due to the unsupervised learning algorithm, k-means cannot be used for applications such as Water/Air quality classification, where the output classes must be supervised.

Observation - From the above discussion, it can be observed that MLP with backpropagation is the suitable architecture for digital hardware implementation of supervised classification algorithms due to their structural and mathematical simplicity.

4.2 Modelling of MLP Architecture

An MLP ANN neuron is modelled as shown in Figure 2.1. Synaptic connections on a nerve cell are equivalent to the input points in the diagram. Each input is multiplied by a synaptic weight [w_1, w_2, ... w_n] for an n-dimensional input vector [x_1, x_2, ... x_n]. The sum of these products is then processed via an activation function (a threshold function, stepwise linear function, or sigmoid function) in the nerve centre, which determines the neuron's ultimate output. A layer of parallel processing centres formed by a slew of these neurons can handle a wide range of inputs. The outputs of one layer of neurons are used as the inputs to subsequent layers of numerous neurons, giving the system a massively parallel processing capability. After that, the outputs are compared to the expected outputs, and the errors are calculated. As a result, the synaptic weights are adjusted consistently with the error. This is the basic learning process of a neuron. Figure 4.1 shows a typical architecture for an ANN.

- Input Layer – It contains those neurons that receive the input data to be processed by the ANN.
- Hidden Layer(s) – These are the layers of units between the input and the output layers. The hidden layers take the data from the input layer and perform the computations to give useful information to the other hidden or output layers.
- Output Layer – This consists of the neurons or units that give output depending on the learning that has taken place inside the ANN.

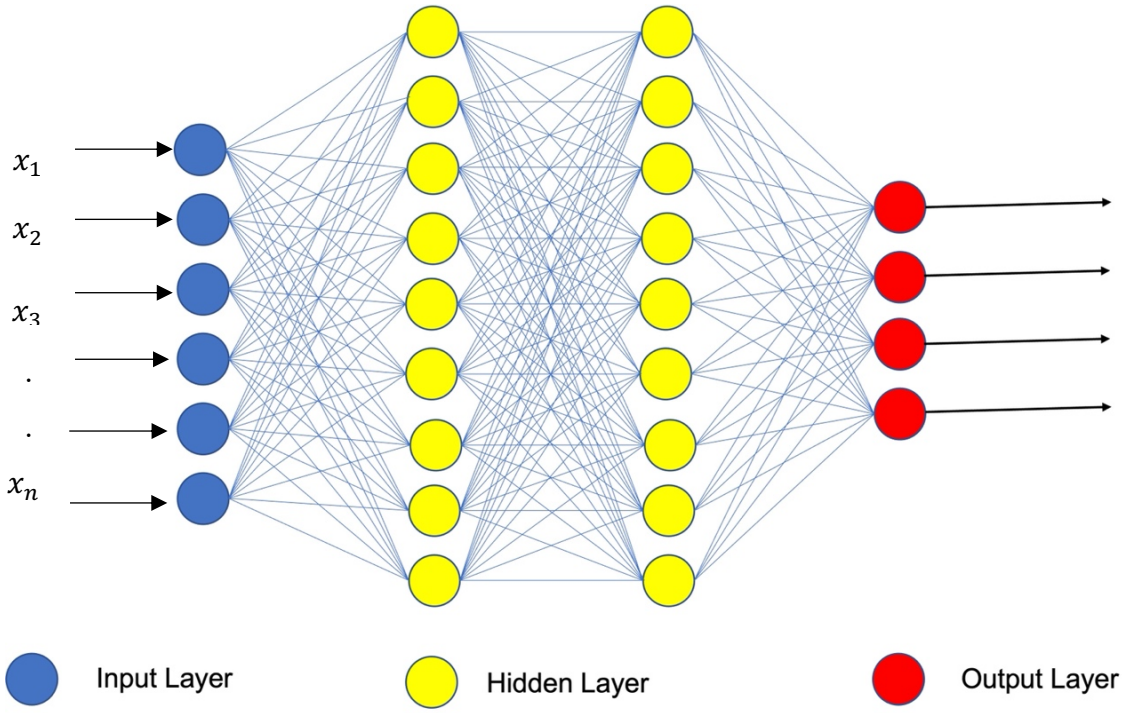


Figure 4.1: A typical ANN Architecture [5]

4.3 Methodology of Hardware Implementation

4.3.1 Choice of Hidden Layers and Number of Neurons

The choice of the structure of a neural network in terms of the number of hidden layers and the number of neurons per layer can affect the accuracy of the neural network. The model will overfit the data or experience the under-fitting problem as its size increases. Both problems converge toward poor generalization and get trapped in local solutions if the ANN architecture needs to be more complex. An economical and effective ANN model is required to handle this issue [2]. Thus, making the right decision concerning the structure of the neural network becomes crucial. In our work, since the ANN must be suited to the data for water quality samples acquired, divided into four input and three output parameters. Many studies have tried to propose a system to find out the network size and structure.

In [2], the authors have explored various techniques to select the optimum ANN structure. Their study found that amongst the two non-nature optimization techniques proposed in the literature, the Model selection algorithm had the best outcome in terms of performance metrics. Thus we applied the Model selection method based on classification accuracy to find the correct number of hidden layers and neurons per layer for the ANN. The process has been

detailed in Chapter 4. Through our experiments, we found that the most optimized structure for Water Quality classification is three hidden layers with four neurons.

4.3.2 The MLP Architecture

An Artificial Neural Network primarily consists of two major computational units –

a) The Neuron
b) Learning algorithms

a) The neuron is the basic computational unit of the ANN. Three types of neurons make up a neural network – Input neurons, output neurons, and hidden neurons. When there is more than one neuron of each type, we call it a layer – input layer, hidden layer, and output layer. Barring the output layer neurons, all the neurons have two major computational units – adder and activation unit.
   While much research has been done on the architecture of adders, the activation units, which are the major computational blocks, need more exploration and better implementation strategies. Exploration of suitable activation functions and their implementation for chosen Water Quality Classification is described in Section 4.3.3
   Observation – From the above discussion, the Sigmoid activation function is most suitable for supervised classification applications like Water Quality Classification.

b) Another significant component in any ANN architecture is the learning algorithm. The hardware architecture for learning algorithms is also complex and needs to be simplified. Backpropagation (BP)-based learning algorithms are primarily used in the training of MLP. An online neural network learning algorithm for handling time variable inputs [3], fast learning methods based on gradient descent of neuron space [4], and the Levenberg–Marquardt algorithm [5], [6] are only a few examples of BP learning algorithms that have been developed [7].
   - The Levenberg-Marquardt (LM) learning algorithm is an adaptive learning algorithm. The LM algorithm switches between the Gauss-Newton and Gradient Descent learning algorithms based on a “damping factor.” This makes the LM method unnecessarily complex for hardware implementation. Also, because of the LM learning algorithm's Gauss-Newton phase, the algorithm generates better accuracy curve-fitting tasks [8].

- The Gradient Descent Backpropagation learning algorithm is most commonly used to determine the weights in supervised learning Multi-layer FNN.

  The learning algorithm for this network is chosen to be backpropagation gradient descent. As propounded by Rumelhart et al. [9], backpropagation aims to obtain a set of weights to minimise the difference between the desired output and the actual output of each neuron, given a particular input vector. The total error E is given by:

$$E = \frac{1}{2}\sum_c \sum_j (y_{j,c} - d_{j,c})^2 \quad (1)$$

  Where c indicates the case (input-output pair), j indicates the output unit, y is the actual state of the output unit, and d is the desired state of that output unit. The partial derivative of E with respect to each weight in the network is computed to minimise the error using the gradient descent algorithm. The partial derivative is simply the sum of the partial derivatives of each case. Thus, we calculate $\partial E/\partial y$ for each case:

$$\frac{\partial E}{\partial y_j} = y_j - d_j \quad (2)$$

  Applying chain rule to compute $\partial E/\partial x_j$:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} . \frac{\partial y_j}{\partial x_j} \quad (3)$$

  Differentiating Eq. (10) for $\partial y_j/\partial x_j$ we get:

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} . y_j(1 - y_j) \quad (4)$$

  The total input is a linear function of the states of the previous levels and also a linear function of the weights on the connections. Hence, it becomes easy to compute how the error is affected by a change in the state and the weights. Given a weight, $w_{j,i}$, from i to j the derivative is given as:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} . \frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j} . y_j \quad (5)$$

  And the contribution of $\partial E/\partial y_j$ for the i^{th} output unit because of the effect of i on j is:

$$\frac{\partial E}{\partial x_j}.\frac{\partial x_j}{\partial y_i} = \frac{\partial E}{x_j}.w_{ji} \tag{6}$$

Considering all the synapses emanating from i, we get:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{x_j}.w_{ji} \tag{7}$$

So now we have the $\partial E / \partial y$ for a unit in the penultimate layer, given the $\partial E / \partial y$ for all the output neurons. To compute this term for previous layers, we use the same principle successively to compute $\partial E / \partial w$ for the weights. $\partial E / \partial w$ is used to change the weights in every epoch. The $\partial E / \partial w$ over the network are accumulated and then at the end of the epoch, we change the weights of each neuron by an amount proportional to the accumulated $\partial E / \partial w$:

$$\Delta w = -\varepsilon \frac{\partial E}{\partial w} \tag{8}$$

As stated by Rumelhart [9], the method has a low convergence rate as compared to methods that make use of second derivative but it is easy to implement on parallel hardware. The method is improvised by accelerating method where the current gradient modifies the velocity of the point in weight space instead of its position:

$$\Delta w(t) = -\frac{\varepsilon \partial E}{\partial w(t)} + \alpha \Delta w(t-1) \tag{9}$$

where t denotes the time-step or epoch and α denotes an exponential decay factor between 0 and 1 that controls contribution of current and previous gradients to the change in weights. Thus, α is also called the learning index.

The online neural network learning algorithm is particularly suited for time-series-based applications such as weather prediction, stock market analysis, behaviour prediction of natural solar energy, etc. These applications are more suited to online learning algorithms because of the ability of online learning algorithms to learn from sequentially arriving data [10]. Since online learning is dependent on time-series data, it is not suitable for classification applications.

Observation – The gradient descent algorithm is best suited For hardware implementations because of its mathematical simplicity leading to less resource requirement.

4.3.3 Sigmoid Activation Function Design for MLP Neuron

The most computationally complex part of the hardware implementation of an ANN is the implementation of the activation function, as it involves complex non-linear mathematical calculations. The development of an artificial neuron and its nonlinear activation functions is one of the challenges of neural network hardware implementation.

Many different functions have been used as the activation function of the neurons in MLP. The most notable are – the threshold function, the Sigmoid logistic function, and the hyperbolic tangent function.

The threshold function is a step function that changes from 0 to 1 at a given threshold. While it is a very simple implementation, it cannot be used in classification applications where there are more than two output classes. The threshold function is also non-differentiable. Thus, its learning algorithm applies to classification with a learning algorithm.

The hyperbolic tangent function is also a sigmoid function, but it has a range between -1 and 1. Thus, it is not suited for applications where the output is probabilistic since probabilities lie between 0 and 1.

Since the classification is a probability prediction indicating which of the output classes the input vector corresponds to, Sigmoid is apt as it is used in models when the output is a probability prediction with the output range 0 to 1. Also, this function has a smooth gradient and is differentiable, which is evident from its S-shape curve. Hence it prevents output value jumps during the learning process.

Eq. (10) represents the sigmoid function:

$$S(x) = \frac{1}{1+e^{-x}} \tag{10}$$

Calculation of the exponent function shown in Eq. (10), in digital is not physically feasible because of its infinite nature. Hence, there is a need for an approximation of the function.

1. The basic methods of approximation for exponent functions, such as the tabular method and the Taylor series, are used in published works on the digital implementation of nonlinear functions. However, the Taylor series requires a significant number of multiplications because the multiplication block takes up a lot of space [11]. Hence, it

is unsuitable for implementation on digital hardware. Inaccuracy introduced by the approximation is traded by the learning cycle of the network.

2. In [12], it has been shown that piecewise linear approximation of the exponent function could offer better accuracy in the approximation of sigmoid functions. Bajger, et al. [13] present a Low-error, high-speed approximation method for the sigmoid function for FPGAs. However, the approximation proposed is particularly designed for FPGA that have functional blocks such as multipliers and adders available on-board. The design proposed in their work makes heavy use of the pre-existing blocks to increase the computational speed. This approach is not suitable for a device where area and power minimisation are two of the primary aims. FPGAs are comparatively large blocks that draw a lot of power as compared to ASICs, which has been shown in the Results section of this chapter. Although, the approach proposed by Bajger et al. improves considerably on accuracy as compared to Faiedh, et al. [12].
3. Padé Approximation is another method for the approximation of the exponent function, which is shown to have less computational complexity [14]. Here the implementation complexity of the logistic function has been reduced at a mathematical level.

   Padé approximation Eq. (11), as proposed in [14] has reported fairly accurate network outputs despite compromising marginally on mathematical accuracy as compared to other expansions such as the Taylor series or McLaurin series. However, the Padé approximation for the exponential function is valid only for the input values lying in the interval $0 \leq x \leq 1$.

$$e^x = \frac{1680+840x+180x^2+20x^3+x^4}{1680-840x+180x^2-20x^3+x^4} \tag{11}$$

   This limits the application of the approximation for our project.

   Because of its simplicity of computation, this approximation method has been explored in this work in section 4.2. Figure 4.2 shows the schematic diagram for Padé approximation using Lookup table (LUT) blocks of Xilinx Zynq700 board.
4. [15] describes a Nonlinear approximation method to approximate the entire Eq (1). Here, a Lookup table (LUT) based approach is followed, which is particularly suited for FPGA implementation of the sigmoid function. The total domain of the sigmoid function is broken up into shorter intervals and the curve in those intervals is approximated by the curve fitting method to simpler polynomials.

For exploration in this work, the input values have been normalised in the range of [-1, 1]. Hence, we take up the intervals [-2, -1], (-1, 1) and [1, 2). The polynomials for the said intervals are given in Eq. (12), Eq. (13) and Eq. (14), respectively:

$$y = 0.0467x^2 + 0.1239x + 0.2969 \quad (12)$$

$$y = 0.2383\ x + 0.5 \quad (13)$$

$$y = -0.0467x^2 + 0.2896\,x + 0.4882 \quad (14)$$

This method has also been explored in this present work in Section 3 because of its simplicity and more accurate approximation of the sigmoid function. Figure 4.3 shows the schematic diagram for nonlinear approximation using the LUT blocks of a Xilinx Zynq 7000 series board.

It is required to cut down on computational complexity to achieve less power consumption of activation units of neurons. Hence, a suitable approximation method for the sigmoid logistic function is found after thorough exploration, as described in Section 4.4.

4.4 Results of Hardware MLP implementation with IEEE 754 Representation using Padé and Nonlinear Approximation of Sigmoid Function

The implementation of Artificial Neural Networks (ANNs) utilizes the IEEE 754 floating point representation format. IEEE 754 stands as the sole floating-point representation system universally embraced by major manufacturers as the standard for their mainframe and minicomputers. In the design presented in this chapter, all the peripherals (sensor electrodes and circuits) are designed for IEEE 754 format as well.

ANN implementation has been designed for the Water Quality Classification application. Here we have taken four parameters as input to the ANN, and the ANN classifies the water sample into one of the three categories – Potable, Agricultural, and Wastewater. The four parameters taken to measure the Water Quality are – pH, Oxidation Reduction Potential (ORP), Dissolved Oxygen (DO), and Electrical Conductivity (EC).

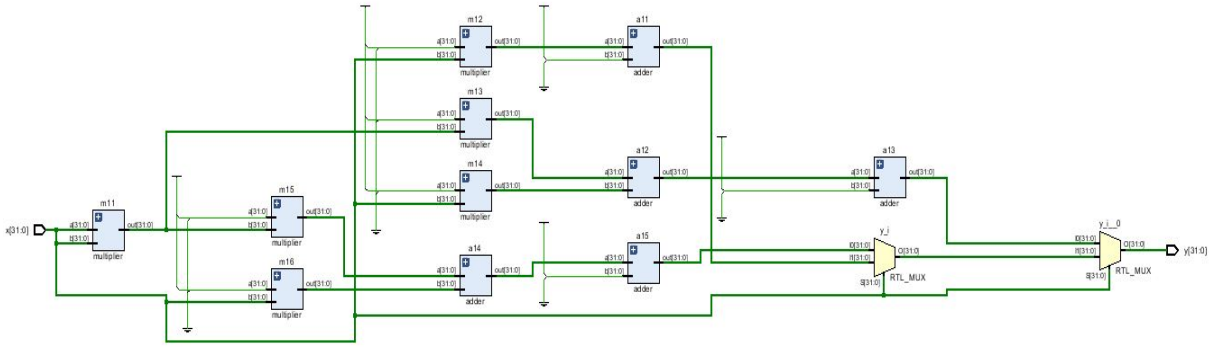


Figure 4. 2: Schematic Diagram of a neuron using Padé Approximation

Alkaline solutions have a reducing nature and, hence, have a negative ORP. Acidic solutions are oxidising in nature and, thus, have a positive ORP. Many ionic molecules can pollute drinking water which cannot be detected by pH alone. For such ions, we need to measure the ORP of the sampled solution.

The Dissolved Oxygen (DO) is a measure of the amount of molecular oxygen that is trapped inside the body of water. DO is very important for aquatic life. Furthermore, DO is necessary to maintain the taste of drinking water. Since DO is necessary for aquatic life, it is also an indicator of water quality in biological terms. Water with very low DO concentrations can be an indication that the water body is infested with some biological pathogen or some biological waste whose decomposition is hosting bacteria that are consuming the Oxygen in the water body. DO is measured using a membrane that is permeable to oxygen molecules only. When oxygen is permeated across this membrane, a potential difference is set across the membrane. This potential difference gives us the DO measurement in milligrams per decilitre or parts per million of the sampled solution. Because of the involvement of a complex chemical membrane, the DO electrode and the circuits accompanying the electrode are of very high sensitivity. Hence, the whole sensor system used for Dissolved Oxygen measurement is expensive.

Electrical Conductivity (EC) is an important parameter because EC has a direct linear relationship with 3 other parameters - Total Dissolved Solids (TDS), Specific Gravity, and Salinity of the sampled solution. Taken together, these parameters cover a wide range of conditions required to ensure water quality. EC is measured using two electrodes dipped in the sample solution, with one of them acting as the reference electrode and the other as the

measurement electrode. Measurement is taken by passing a voltage across these electrodes and measuring the resistance between the two electrodes. Resistance is converted to conductance and conductivity.

The detailed apparatus of each of these parameters is discussed in Chapter 6.

The complete hardware implementation of MLP consists of 2main parts: -

- Sigmoid Neuron Implementation
- Backpropagation learning

The Padé approximation method is used to approximate the exponent function. However, it only employs the calculation of up to the 4^{th} power of the variable. The four powers of the input variable are calculated and stored in registers. The coefficients of these powers being

4.4.1 Sigmoid Neuron Implementation Description

Two approximation methods – Padé and Non-linear, as described in Section 4.3.3, have been implemented using both FPGA and ASIC methodology and their results are compared.

a) Procedure for Implementation of Padé Approximation

the same for the numerator and denominator, as can be seen in Eq (2), are taken as constants. Thus implementing Eq. (11) gives us the approximation of the exponent function of the input variable. The sigmoid function, as shown in Eq (1), requires the reciprocal of the exponent function. Thus, a division algorithm is used to find the reciprocal of the exponent function. Then the reciprocal is added with 1 to form the denominator of Eq. (10), and Eq. (10) is implemented using another division module. Thus, Padé approximation is implemented and helps us reduce the number of exponent calculations to just the 4^{th} power of the input variable. The implementation is done using the IEEE 754 Floating Point representation method. Figure 4.2 shows the schematic of a neuron which is hereafter implemented on both FPGA and ASIC platforms. The schematic contains blocks that show 4 multiplier blocks which have the neuron inputs and their corresponding weights as the input to the multipliers. The output of these multipliers is then routed to adders which add the weighted inputs. The final summation of the weighted inputs acts as the input to the activation block. This block implements the mathematical Padé approximation function to approximate the sigmoid function output, which is the final output of the neuron.

b) Procedure for Implementation of Non-Linear Approximation Method

The Non-linear approximation method approximates the sigmoid function, unlike other approximation methods. In this method, we break up the domain of the function into smaller windows and then approximate the curve in that window using non-linear functions. The sigmoid function in the desired range has been approximated by the three equations as shown in Eqs (3 – 5). The maximum power we need to calculate for this implementation is the second power of the input variable. Thus mathematically, this method proves to be the most efficient. Further, this method removes the division algorithm from the implementation. The Schematic diagram of for nonlinear approximation is shown in Figure 4.3 and its FPGA and ASIC results are discussed in section 4.4.2.

The implementation results are compared to the Padé approximation in Figures 4.4 and 4.5 and Tables 4.1 and 4.2.

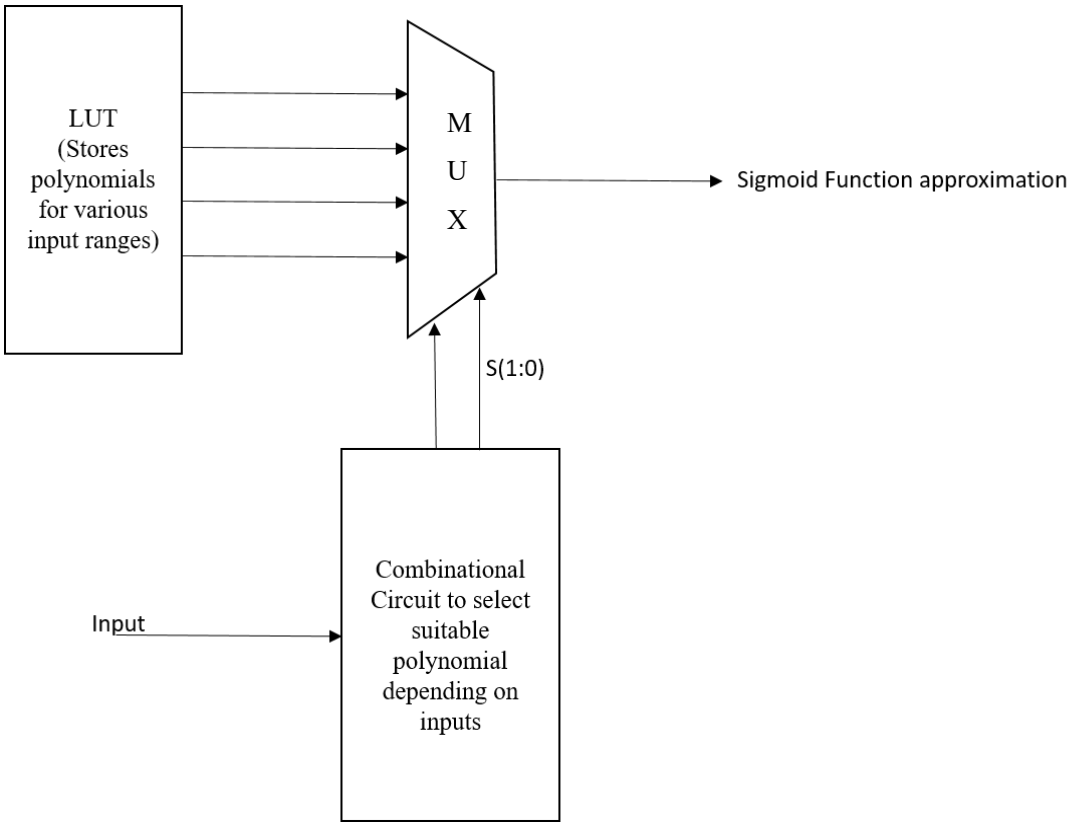


Figure 4. 3: Schematic diagram of nonlinear approximation

4.5 Sigmoid Neuron Implementation Results

4.5.1 FPGA Implementation

The algorithms have been coded in Verilog and implemented on ZynQ7000 FPGA using Xilinx Vivado. The results of the implementation are shown as a comparative bar chart for the various parameters between the two implemented architectures in Figure 4.5. The power consumption of the FPGA implementation was very high and can only be reduced to a certain limit as an FPGA is limited in terms of customizability. Thus, we synthesized the design using the ASIC design methodology.

4.5.2 ASIC Implementation

The ASIC was synthesized using the Cadence Encounter RTL Compiler tool with UMC 90 nm standard cell library. The power consumption of a neuron using the Padé approximation dropped from 5.95 Watts on FPGA to 3.75×10^{-4} Watts in ASIC synthesis. Similarly for the Non-linear approximation method, the power consumption of a single neuron drops from 5.3 Watts on FPGA to 2.47×10^{-4} Watts for ASIC synthesis. Padé approximation method makes use of 25538 Cells covering 241897 nm^2 of the library as compared to 15709 cells covering 130575 nm^2 for the nonlinear approximation method. Figure 4.5 shows the results of the ASIC implementation of the Activation function using the two approximation methods.

The Verilog codes for ANN using both Padé and Non-linear approximation are given in Appendix B.

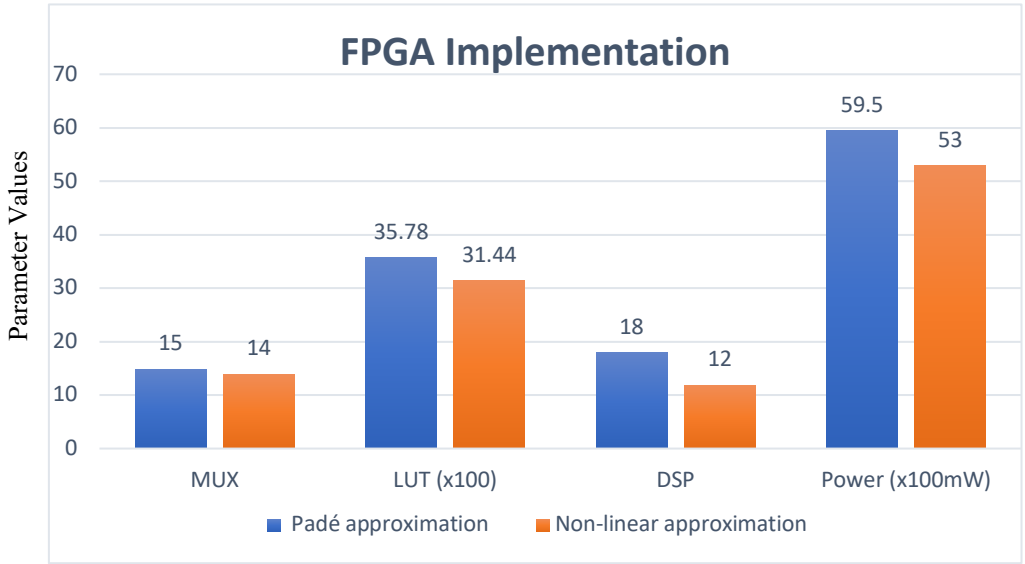


Figure 4. 4: Comparison of the two implementations of Activation Functions in FPGA-based design using IEEE 754

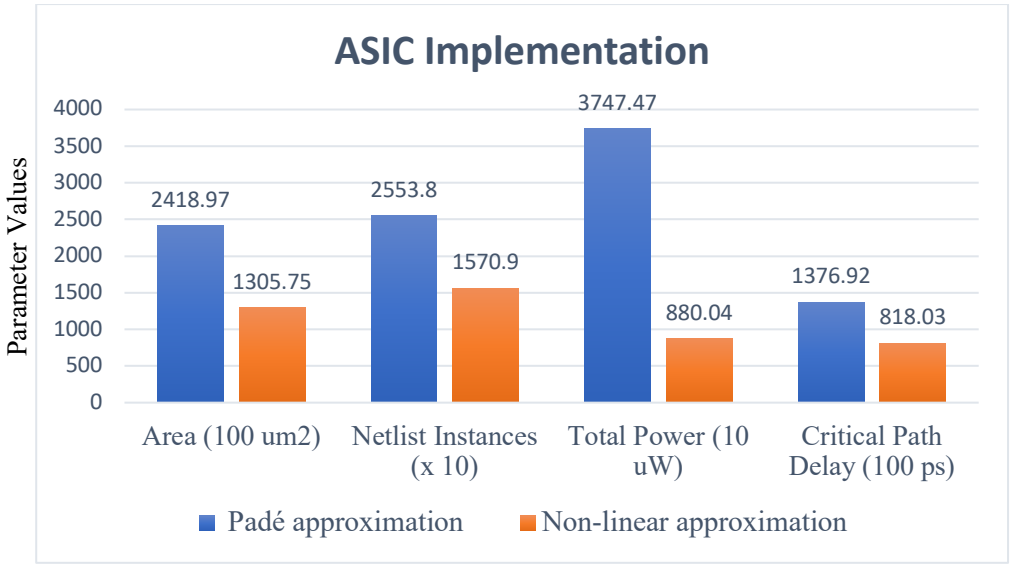


Figure 4. 5: Comparison of two implementations of the Activation Function in ASIC Implementation using IEEE 754

Observations

From Figures 4.4 and 4.5, it is observed that Non-linear approximation is:

- the most efficient implementation of the Sigmoid function using IEEE 754 Floating Point Representation.
- It uses a lesser number of resources and consumes less power in both FPGA and ASIC implementation.
- Also, It occupies 46% less Si area.
- It is also faster by 68.3%.

4.5.3 Backpropagation Learning Implementation Methodology

The backpropagation learning algorithm helps the network to improve the accuracy of the output. In this work, an FSM is designed to generate a signal to start a backpropagation algorithm as the final output of one epoch is generated [16]. This algorithm measures the difference between the desired output and the actual output. The error in the final output is back propagated to all the neurons in the preceding layer while the weights of the current layer are updated. The control of the learning mechanism is synchronized using an FSM, which gives a time-multiplexed learning mechanism to reduce the switching power consumption of the complete network.

4.6 Conclusions

From the discussions in this chapter, it is concluded for hardware implementation of ANN for Water Quality Classification using IEEE 754: -

- MLP is the suitable architecture.
- Sigmoid is suitable as the output range is between 0 and 1 which suits the probability prediction.
- A non-linear approximation of the Sigmoid function is 77% more power efficient and utilizes 38.5% lesser hardware resources with a 68.3 % faster Critical Delay path than Padé.
- ASIC implementation of Nonlinear approximated Sigmoid function consumes lesser power by at least 3 orders of magnitude than FPGA implementation.

Thus, the MLP architecture with sigmoid activation function using a non-linear approximation method is more suited for the proposed design for water quality application that requires low power, low cost, high speed, and portable design.

On another note, ASIC implementation is more cost-effective than FPGA for mass production scenarios. Further, FPGAs provide the flexibility of re-programmability to the user in case the design has to be improved in the near future. FPGAs are much more readily available and more straightforward to design on the user end but at a higher cost and power consumption.

Further, IEEE 754 has a rigid representation and several reserved bit patterns that lead to calculation errors. Thus, it becomes necessary to explore other representation systems to achieve more efficient and accurate design. This has been discussed in detail in Chapter 4.

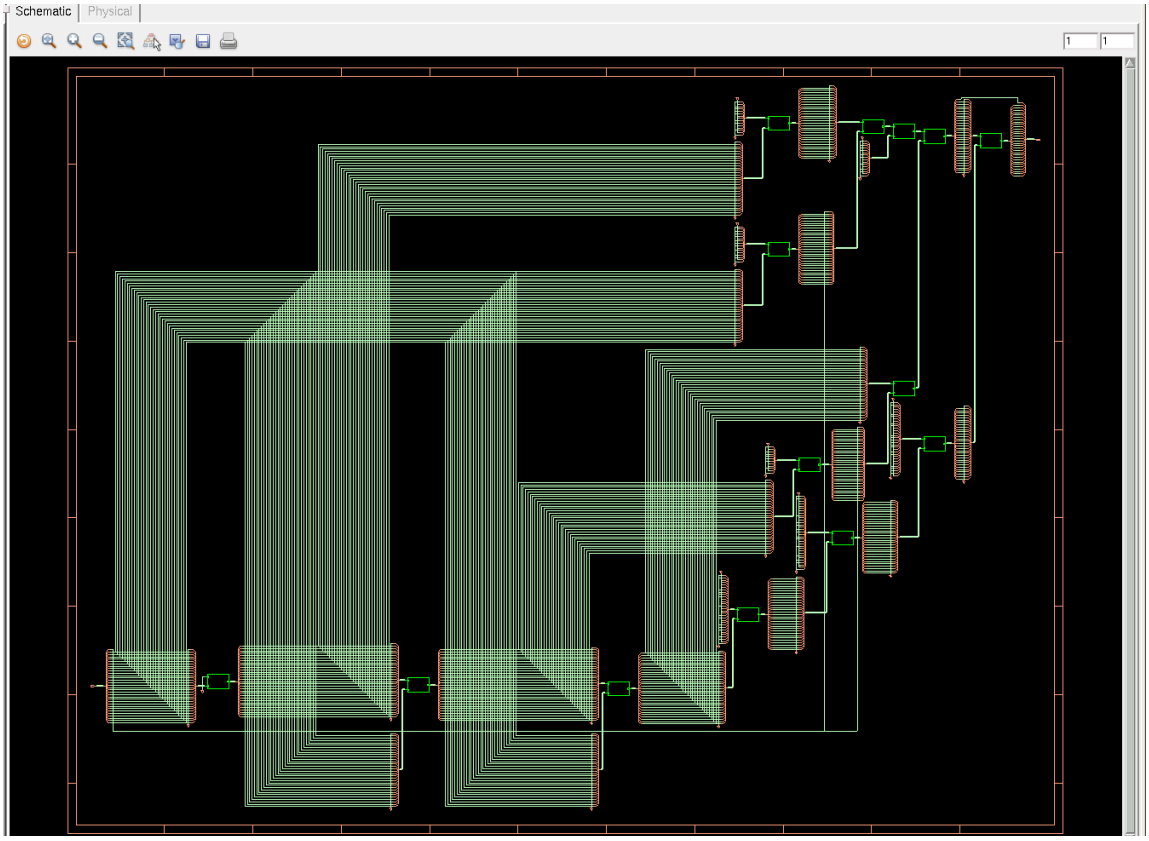


Figure 4.6: ASIC implementation of Padé approximation

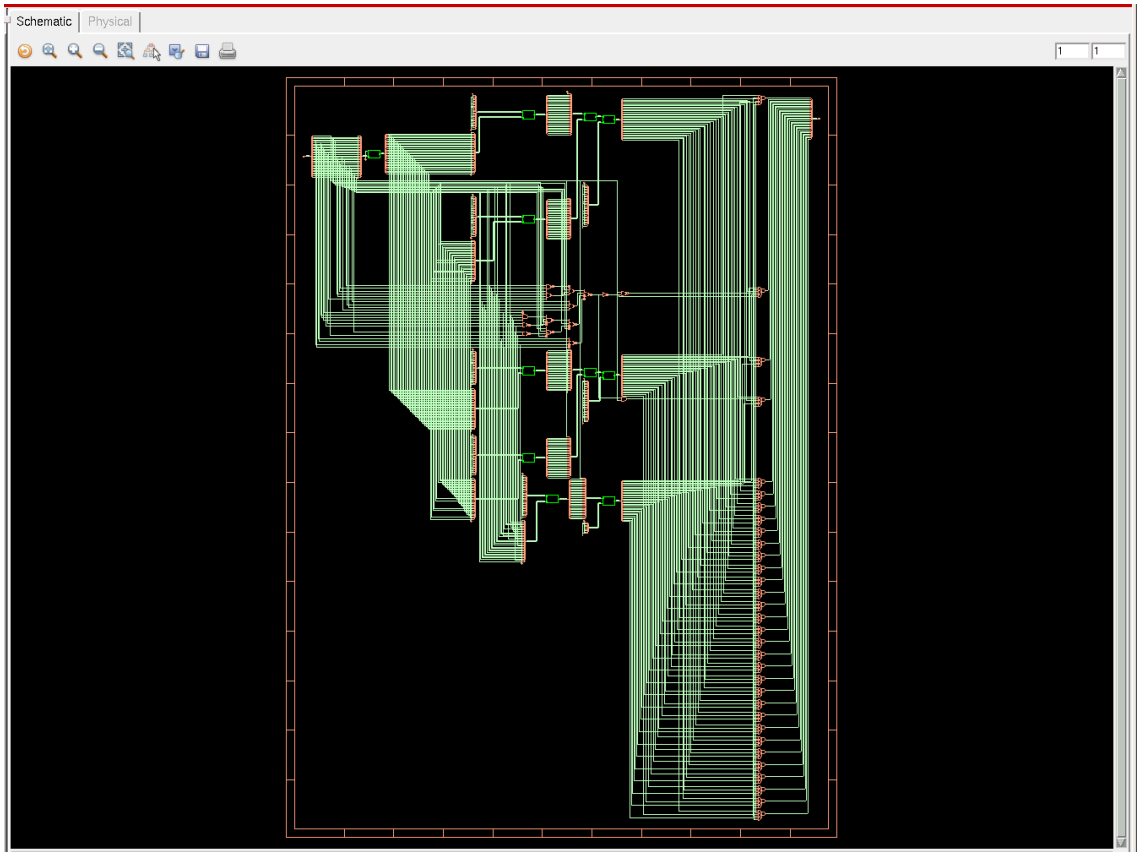


Figure 4. 7: ASIC Implementation of a Nonlinear Approximation of Sigmoid function

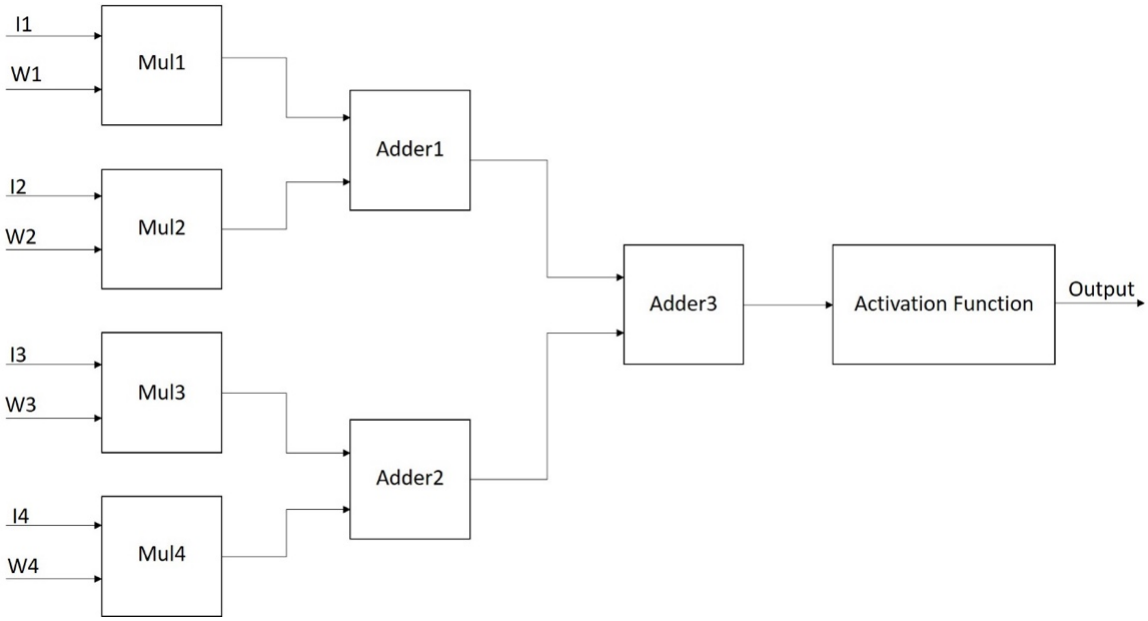


Figure 4.8: Basic Structure of a Neuron

4.7 References

[1] T. M. Mitchell, Machine Learning, McGraw-Hill Science Sciece/Engineering/Math, 1997.

[2] T. K. Gupta and K. Raza, "Chapter 7 - Optimization of ANN Architecture: A Review on Nature-Inspired Techniques," in *Machine Learning in Bio-Signal Analysis and Diagnostic Imaging*, Academic Press, 2019, pp. 159 - 182.

[3] X. Yu, B. Wang, B. Batbayar, L. Wang and Z. Man, "An improved training algorithm for feedforward neural network learning based on terminal attractors," *Journal of Global Optimization ,* vol. 51, pp. 271 - 284, 2010.

[4] G. Zhou and J. Si, "Advanced neural-network training algorithm with reduced complexity based on Jacobian deficiency," *IEEE Transactions on Neural Networks,* vol. 9, no. 3, pp. 448 - 453, 1998.

[5] R. Parisi, E. D. Di Claudio, G. Orlandi and B. D. Rao, "A generalized learning paradigm exploiting the structure of feedforward neural networks," *IEEE Transactions on Neural Networks,* vol. 7, no. 6, pp. 1450 - 1460, 1996.

[6] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Transactions on Neural Networks ,* vol. 5, no. 6, pp. 989 - 993, 1994.

[7] X. Yu, M. O. Efe and O. Kaynal, "A general backpropagation algorithm for feedforward neural networks learning," *IEEE Transactions on Neural Networks,* vol. 13, no. 1, pp. 251 - 254, 2002.

[8] H. P. Gavin, The Levenberg-Marquardt algorithm for, Duke University: Department of Civil and Environmental Engineering, 2019.

[9] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning representations by back-propagating errors," *Nature,* vol. 323, pp. 533 - 536, 1986.

[10] H. T. Huynh and Y. Won, "Regularized online sequential learning algorithm for single-hidden layer feedforward neural networks," *Pattern Recognition Letters,* vol. 32, no. 14, pp. 1930 - 1935, 2011.

[11] V. Shymkovych, . S. Telenyk and P. Kravets, "Hardware implementation of radial-basis neural networks with Gaussian activation functions on FPGA," *Neural Computing and Applications volume,* vol. 33, pp. 9467 - 9479, 2021.

[12] A. Armato, L. Fanucci, E. P. Sciligno and D. De Rossi, "Low-error digital hardware implementation of artificial neuron activation functions and their derivative," *Microprocessors and Microsystems,* vol. 35, pp. 557 - 567, 2011.

[13] M. Bajger and A. Omondi, "Low-error, High-speed Approximation of the Sigmoid Function for Large FPGA Implementations," *Journal of Signal Processing Systems ,* vol. 52, pp. 137 - 151, 2007.

[14] Z. Hajduk, "High accuracy FPGA activation function implementation for neural networks," *Neurocomputing,* vol. 247, pp. 59-61, 2017.

[15] X. Zhen-zhen and Z. Su-yu, "A Non-linear Approximation of the Sigmoid Function Based FPGA," in *Cybernetics, and Computer Engineering (ICCE2011) November 19–20, 2011, Melbourne, Australia*, Melbourne, 2011.

[16] A. Savich, M. Moussa and S. Akreibi, "A scalable pipelined architecture for real-time computation of MLP-BP neural networks," *Microprocessors and Microsystems,* vol. 36, no. 2, pp. 138 - 150, 2012.

[17] P. Ferreira, P. Ribeiro, A. Antunes and F. M. Dias, "Artificial Neural Networks Processor - A Hardware Implementation Using a FPGA," in *Field Programmable Logic and Application. FPL 2004. Lecture Notes in Computer Science*, Berlin, 2004.

[18] H. Faiedh, C. Souani, K. Torki and K. Besbes, "Digital Hardware Implementation of a Neural System Used for Nonlinear Adaptive Prediction," *Journal of Computer Science,* vol. 2, no. 4, pp. 355 - 362, 2006.

[19] Central Bureau of Health Intelligence, "NationalHealth Profile 2018," Ministry of Health and Family Welfare, Government of India, New Delhi, 2018.

[20] Organisation, World Health, "Diarrhoeal Disease," World Health Organisation, 2017.

[21] World Health Organisation, "Guidelines for Drinking-water Quality (Fourth," World Health Organisation, Geneva, Switzerland, 2017.

[22] "Introduction to Artificial Neural Network (ANN)," secret mind control in Sweden and worldwide mindcontrolinsweden.wordpress.com, 30 01 2015. [Online]. Available: https://mindcontrolinsweden.wordpress.com/2015/01/30/introduction-to-artificial-neural-networks. [Accessed 02 2017].

[23] N. Morgan, K. Asanovic, B. Kingsbury and J. Wawrzynek, "Developments in Digital VLSI Design for Artificial Neural Networks," International Computer Science Institute, 1990.

[24] N. Morgan, Artificial Neural Networks: Electronic Implementations, Washington DC: IEEE Computer Society Press, 1990.

[25] T. Baker and D. Hammerstrom, "Modifications to Artificial Neural Networks Models for Digital Hardware Implementation," Department of Computer Science Engineering Oregon Graduate Center, Oregon, 1988.

[26] W. S. McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics ,* vol. 5, pp. 115 - 133, 1943.

[27] N. S. Gill, “Overview of Artificial Neural,” Xenonstack: A Stack Innovator, 2017. [Online]. Available: https://www.xenonstack.com/blog/artificial-neural-network-applications. [Accessed 2017].

[28] J. M. Zurada, Introduction to Artificial Neural Systems, St Paul, MN: West Publishing Company, 1992.

[29] P. Mehra and B. W. Wah, Artificial neural networks: Concepts and Theory, IEEE Computer Society Press, 1992.

[30] C. Alippui and G. Storti-Gajani, “Simple approximation of sigmoidal functions: realistic design of digital neural networks capable of learning,” in *1991 IEEE International Sympoisum on Circuits and Systems*, Singapore, 1991.

[31] S. Oh, Y. Shi, J. del Valle, P. Salev, Y. Lu, Z. Huang, Y. Kalchiem, I. K. Schuller and D. Kuzum, “Energy-efficient Mott activation neuron for full-hardware implementation of neural networks,” *Nature Nanotechnology,* vol. 16, pp. 680 - 687, 2021.

[32] Y. van de Burgt, J. F. liot, S. T. Keene, G. C. Faria, S. Agarwal, M. J. Marinella, A. A. Talin and A. Salleo, “A non-volatile organic electrochemical device as a low-voltage artificial synapse for neuromorphic computing,” *Nature Materials,* vol. 16, pp. 414 - 418, 2017.

[33] T. Yokota, P. Zalar, M. Kaltenbrunner, H. Jinno, N. Matsuhisa, H. Kitanosako, Y. Tachibana, W. Yukita, M. Koizumi and T. Someya, “Ultraflexible organic photonic skin,” *Science Advances,* vol. 2, no. 4, 2016.

[34] G. W. Burr, R. M. Shelby, S. Sidler, C. di Nolfo, J. Jang, I. Boybat, R. S. Shenoy, P. Narayanan, K. Virwani, E. U. Giqacometti, B. N. Kurdi and H. Hwang, “Experimental Demonstration and Tolerancing of a Large-Scale Neural Network (165 000 Synapses) Using Phase-Change Memory as the Synaptic Weight Element,” *IEEE Transactions on Electron Devices,* vol. 62, no. 11, pp. 3498 - 3507, 2015.

[35] S. Oh, Z. Huang, Y. Shi and D. Kuzum, “The Impact of Resistance Drift of Phase Change Memory (PCM) Synaptic Devices on Artificial Neural Network Performance,” *IEEE Electron Device Letters,* vol. 40, no. 8, pp. 1325 - 1328, 2019.

Chapter 5

5. Digital Hardware Implementation of Artificial Neural Network with Posit Representation of Floating-Point Numbers

In this chapter, we discuss the implementation of an Artificial Neural Network on an ASIC using the Posit Floating point representation system proposed by John Gustafson. We study the implementation of ANN in comparison [1] *to the ANN Implemented using the Nonlinear Approximation function on the IEEE 754 representation system and compare the results to the Posit implementation.*

5.1 Introduction

Implementation of complex mathematical functions, such as sigmoid functions, involves calculations of real numbers that cannot be represented using the binary number format. Thus, there is a need for a format to represent fractions in the binary domain.

Representing real numbers in digital hardware architectures is a challenge. Real numbers constitute an important part of the number system, as most real-life calculations can only be represented by real numbers. Real numbers can be represented using either fixed point representation or floating point representation.

The fixed point is a simple and highly effective method for representing fractional values in computing. Fixed point arithmetic is many orders of magnitude faster than floating point arithmetic because it reuses all integer arithmetic circuits. This is why it is utilized in numerous game and DSP applications. It has a limited range of number representation, and also the accuracy for larger numbers is limited [2].

For ANN applications, the accuracy of number representation plays an important role since the primary processing unit of the ANN, the activation unit, relies on this calculation. In applications such as Water Quality Classification, it becomes even more important to have accuracy in the number representation system. Thus, it is necessary to use floating point representation for real numbers due to its higher range and resolution leading to more accurate representation.

The different existing formats of floating-point representation system are described below:

5.1.1 IEEE 754 Floating Point Representation

Floating point numbers have been represented in the IEEE 754 floating point representation format since 1985.

- IEEE 754 can accommodate both floating point numbers and integers as well.
- IEEE 754 supports the Not-a-Numbers (NaNs), and some special bit streams were reserved for some special cases.
- IEEE 754 represents the two infinities, + infinity and – infinity, separately.
- IEEE 754 supports the cohort representation of numbers since it was inspired by the scientific notation of decimal numbers.
- It is suitable for both software and hardware implementation owing to its fixed representation and simple encoding and decoding principles.

As referred to in Chapter 3, one of the prominent problems of IEEE 754 Floating Point Representation (referred to as Floating Point here onwards) has a rigid arrangement.

This results in very large bit patterns to represent even small numbers. There are two majorly used formats of representation – single precision (32 bits) and double precision (64 bits). Thus, to represent small numbers, this representation occupies large amounts of resources and are counter-intuitive to be used for energy-efficient operation such as ASIC design. The rigid representation has predefined fixed-size partitions for exponent and mantissa. It limits precision, which is the other end of the spectrum as compared to energy efficiency. The limited precision may lead to rounding errors in the representation of real numbers; therefore, some floating-point numbers are not represented precisely. An additional 80 intermediate bits must be reserved to obtain the correct result for an operation to yield correct results in double precision format [3].

IEEE 754 also has different bit patterns reserved for NaNs, denormals, +/- infinity, and other special cases. The reservation of these patterns for special cases also leads to arithmetic inconsistencies. One such inconsistency is that the representation is also flawed in the representation of zeros as it has the possibility to represent +/- 0. Now the IEEE 754 representation treats +0 = -0. This implies that +1/0 = -1/0, which further implies +infinity = infinity. More such cases are noted where the floating-point representation is inconsistent with algebraic rules of computation. E.g., for the values x = 1e30, y = -1e30, and z = 1; (x + y) +z = 1, while x + (y + z) = 0. Another inconsistency is noted in the dot product calculation of vectors.

Let us Assume vectors A = [3.2e7, 1, -1, 8.0e7] and B = [4.0e7, 1, -1, -1.6e7]. The dot product A.B is calculated to be 1 while the right answer should be 2.

Thus, designing a system for IEEE 754 Floating point representation is very cumbersome as it requires special considerations to be made for handling rounding of numbers, NaNs, denormals, etc. Secondly, verifying that design is also difficult because of the corner cases involved.

Thus, IEEE 754 is the standard accepted representation of floating point numbers but it has its own disadvantages. To overcome the challenges of IEEE 754, Universal Number format representation methods have been proposed over the years to improve or replace the IEEE 754 Floating Point representation.

5.1.2 Universal numbers Format

Unum (universal number) representation has been proposed as a superset of floating-point representations. Unum is a variable-length representation that adapts the bit-size of the representation to the actual numbers being represented, and it also associates and propagates accurate information via arithmetic operations [4].

The first version of unums, technically known as Type I unum, was introduced as a superset of the IEEE-754 floating-point format in Gustafson's book The End of Error [5]. These characteristics define the Type I unum format:

- a storage format with variable width for both the significand and exponent
- a u-bit that indicates whether the unum represents an exact number (u = 0) or an interval between consecutive exact unums (u = 1) Thus, the unums encompass the complete extended real number line [-infinity,+infinity].

The "Type II" unum [6] abandons compatibility with IEEE floats, allowing for a mathematically clean design based on projective reals. Type II unums have many ideal mathematical properties, but most operations require table lookups. For 2-argument functions with n bits of precision, there are (in the worst case) 22n table entries, though symmetries and other tricks typically reduce this to a more manageable size.

a) Challenges of Unum Format

But these number systems introduced new flaws into the system, and the trade-off was not worth changing a standardized system.

Unum Type 1 was basically a superset of the IEEE 754 format with variable length of the significand and the exponent. This meant without much increase in accuracy, Unum1 introduced computational complexity for hardware implementation of the representation system.

Unum Type 2 though very robust and fast, relies on a Look-up Table-based approach, which severely limits the range and resolution of numbers that this format can represent. Unum Type 2 is severely limited by the memory size available on the architecture for the range and resolution that it can represent. Hence, it is unsuitable for devices that need to minimize resource utilization and achieve maximum accuracy.

5.1.3 Posit

Posit representation was proposed in 2017, John L Gustafson proposed the Posit representation of floating numbers. The Posit representation system has proven more accurate [refer] for ANN implementations while also overcoming the shortcomings of IEEE 754 without introducing many trade-offs. Moreover, in the field of ANN, Posits are particularly useful for classification applications since they introduce a tapered accuracy. We have discussed the Posit number system in detail in Section 5.2

5.2 Posit Representation

In 2013, John L Gustaffson proposed a novel method called Universal Numbers (Unum). Gustaffson defined 2 types of Unum. Type 1 was developed as a superset to floating point numbers to accommodate greater range and accuracy. However, the hardware cost made it impractical. Type 2 was based on a positional bit pattern instead of actual data conversion. This conversion was based on lookup tables. This allowed extremely fast computations, but at the cost of operations that could be performed [3].

In their 2017 paper, John L Gustaffson proposed the posit representation of floating-point numbers. The Oxford dictionary defines posit as “a statement that is made on the assumption

that it will prove to e true." Posits are a hardware-friendly version of Unum2 with relaxations in 2 rules: -

iii) Reciprocals only follow perfect reflection rule for 0, +/- infinity, and integer powers of 2.

iv)There are no open intervals

The first relaxation enables one to populate the u-lattice such that finite numbers are all represented in the form of IEEE 754 representation of $m.2^k$.

The structure of a posit is shown in Figure 5.1.

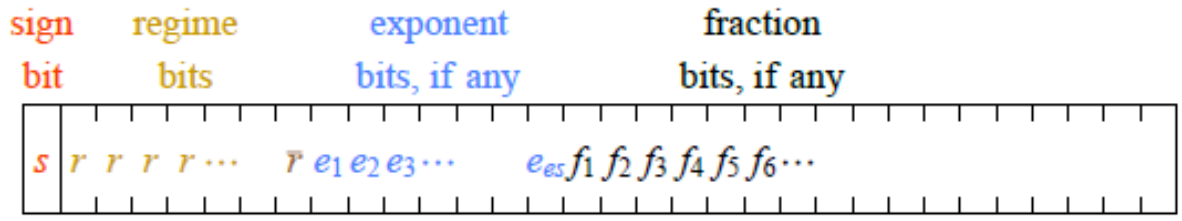


Figure 5. 1: Format of Posit Representation [1]

The sign bit is the same as IEEE 754 Floating point representation: 0 for positive numbers and 1 for negative numbers. If the sign bit is 1, the rest of the number should be in 2's complement.

Table 5. 1: Run-length meaning k of the regime.

Binary	0000	0001	001x	01xx	10xx	110x	1110	1111
Numerical Meaning, k	-4	-3	-2	-1	0	1	2	3

Consider the binary strings shown in Table 5.1 to make sense of the regime bits. The run length of the bits is denoted by numerical meaning, k. These are strings of either all 0 or all 1bits. The bits are terminated either by the opposite bit or the end of the string is reached. If the bits are 0 and there are m bits, then $k = -m$, if the bits are 1, then $k = m - 1$. The regime gives us the scale factor for useed^k, $useed = 2^{2^{es}}$. $useed$ values examples are shown in Table 5.2

Table 5. 2: The useed as a function of es

es	0	1	2	3	4
useed	2	$2^2 = 4$	$4^2 = 16$	$16^2 = 256$	$256^2 = 65536$

The next part is the exponent, e, taken as an unsigned integer. Unlike IEEE 754 Floating points, there is no bias in the exponent, and represents scaling by 2^e. If there are enough bits remaining after the regime, the highest number of bits the exponent can occupy is es. This is how the tapered accuracy of Posits is expressed. Numbers near 1 need to be presented with more accuracy than very large or very small numbers, which are not so common in the calculation.

If more bits remain in the bit stream after the regime and exponent, they are used to represent the fraction part of the number. The fraction part of a posit is just like that of IEEE 754 floating point in the format of 1.f with a hidden bit that represents the whole number part, 1. Posits have no subnormal numbers with a hidden bit 0 for numbers less than 1.

There are only 2 exceptions in the posit representation, i.e., 0(all 0's) and $\pm\infty$ (1 followed by all 0 bits).

Table 5.3 shows the dynamic range offered by both posits and IEEE 754 Floating Point representation for some bit lengths [3].

Table 5. 3: IEEE 754 Float and Posit dynamic ranges for the same no. of bits [1]

Size, Bits	**IEEE Float Exp. Size**	**Approx. IEEE Float Dynamic Range**	**Posit es value**	**Approx. Posit Dynamic Range**
16	5	6×10^{-8} to 7×10^{4}	1	4×10^{-9} to 3×10^{8}
32	8	1×10^{-45} to 3×10^{38}	3	6×10^{-73} to 2×10^{72}
64	11	5×10^{-324} to 2×10^{308}	4	2×10^{-299} to 4×10^{298}
128	15	6×10^{-4966} to 1×10^{4932}	7	1×10^{-4855} to 1×10^{4855}
256	19	2×10^{-78984} to 2×10^{78913}	10	2×10^{-78927} to 5×10^{78296}

5.2.1 Advantages of Posit

- Posits have only two reserved patterns for zero and +/- infinity. This reduces the arithmetic inconsistencies that occur due to a large number of reserved patterns in IEEE 754.
- Posits have tapered accuracy for numbers with very large or small exponents. This enables Posits to be able to represent comparable representation accuracy at a lesser number of bits.
- Since the bit widths for Regime, Exponent, and Mantissa are not rigid, many rounding-off errors in computations can be avoided.
- As can be observed in Table 5.3, for a 32-bit number, the dynamic range of Posits is much greater than that of IEEE 754 [3].

Since Posits offer a tapered accuracy, they are not suitable for applications where the accuracy needs to be consistent across the range of represented numbers. Some applications where Posit are not suited due to the tapered accuracy are [7]:

- Interfacing circuits, where the circuit has to communicate with legacy hardware
- Physical and astronomical circuitry, where the accuracy must remain constant for all numbers
- Because of trade-off, posit for processors executing general purpose applications has been debated. The variable bit format of posit, with changeable regime, exponent, and fraction bits, has prevented its use in general-purpose processors [8].

Posits offer a better use case where processing on the data is involved once the data has been normalized between a certain range. Examples of such applications include Machine Learning, Monte Carlo Simulations, graphics rendering, etc [7].

5.3 Posit ANN Implementation for Water Quality Classification

This section focuses on a detailed explanation of the design of ANN for Water Quality Classification using Posit representation. The input and output parameters of the ANN for Water Quality Classification is already presented in Section 3.4 of Chapter 3.

5.3.1 Parameterised Posit ANN (PPANN)

Posit representation is a flexible representation of floating point numbers. However, when implementing on hardware, the flexibility has to be bounded due to the limitations of hardware

implementation. The Posit has multiple parameters such as exponent size (Es), useed, etc. which govern the sizes of the sub-sections of the Posit bit stream – Sign bit, Regime, Exponent, and Mantissa. These parameters change during runtime to allow Posit to have variable bit-length for maximum accuracy. However, for hardware implementation, we have imposed some limits on all the Posit parameter to achieve desirable accuracy at a much lower bit width.

5 design steps of hardware implementation are as follows.

- Step – 1: Floating Point to Parameterised Posit Conversion
- Step – 2: Leading One/Zero Detector
- Step – 3: Parameterised Posit to Floating Point Converter
- Step – 4: Parameterised Posit Addition Unit
- Step – 5: Sigmoid implementation using Parameterised Posit

The details of each step are discussed below.

a) Step – 1: Floating point – to – Parameterised Posit Converter

The converter has been designed in parameterised manner to accommodate for hardware limitations. The converter has been divided into two major parts – Floating Point decoding and Posit Encoding.

In the Floating-point decoding part, we extract the sign bit, the exponent bits and the mantissa bits and store them in three registers. The floating-point decoding section also checks for special cases such as ZERO and INFINITY. NaN's are not checked for as posits do not have any encoding for NaN's. for sub-normal cases, we normalise the floating-point numbers by detecting the leading true bit of the mantissa and then left shifting the mantissa by as many bits. We then make changes in the exponent according to the position of the leading true bit of the mantissa. Now we add the BIAS to the exponent and thus we get all three parts of our floating-point number [9].

Next step is to encode the Posit. Encoding the posit brings in a new challenge in the form of variable positioning of the exponent and mantissa bits, which is decided based on the regime bits. The signed exponent (Exp) is used value to determine the Regime, R_0 and unsigned Exponent, E_0 (ES bits wide) for posit. So E_0 is obtained from the lowest ES bits of absolute Exponent, Exp_N, and the remaining higher bits denote the regime R_0. Here we check if $Exp > 0$ and ES lowest bits of Exp_N are non-zero, then the exponent E_0 of posit is the 2's complement

of the ES lowest bits of Exp_N . For negative exponent representation, as E_0 is an unsigned integer, the above procedure with an can be performed with an increment in corresponding negative R_0, otherwise E_0 is obtained from the lowest ES bits of absolute Exponent, Exp_N. Remaining MSBs $Exp_N[E-1:ES]$ denote R_0 if $Exp < 0$ and $Exp_N[ES-1:0] = 0$. Else, R_0 is the incremented value of the remaining MSBs.

So, the Posit representation is constructed from S_{FP}, R_0, E_0 and M_{FP}as follows:

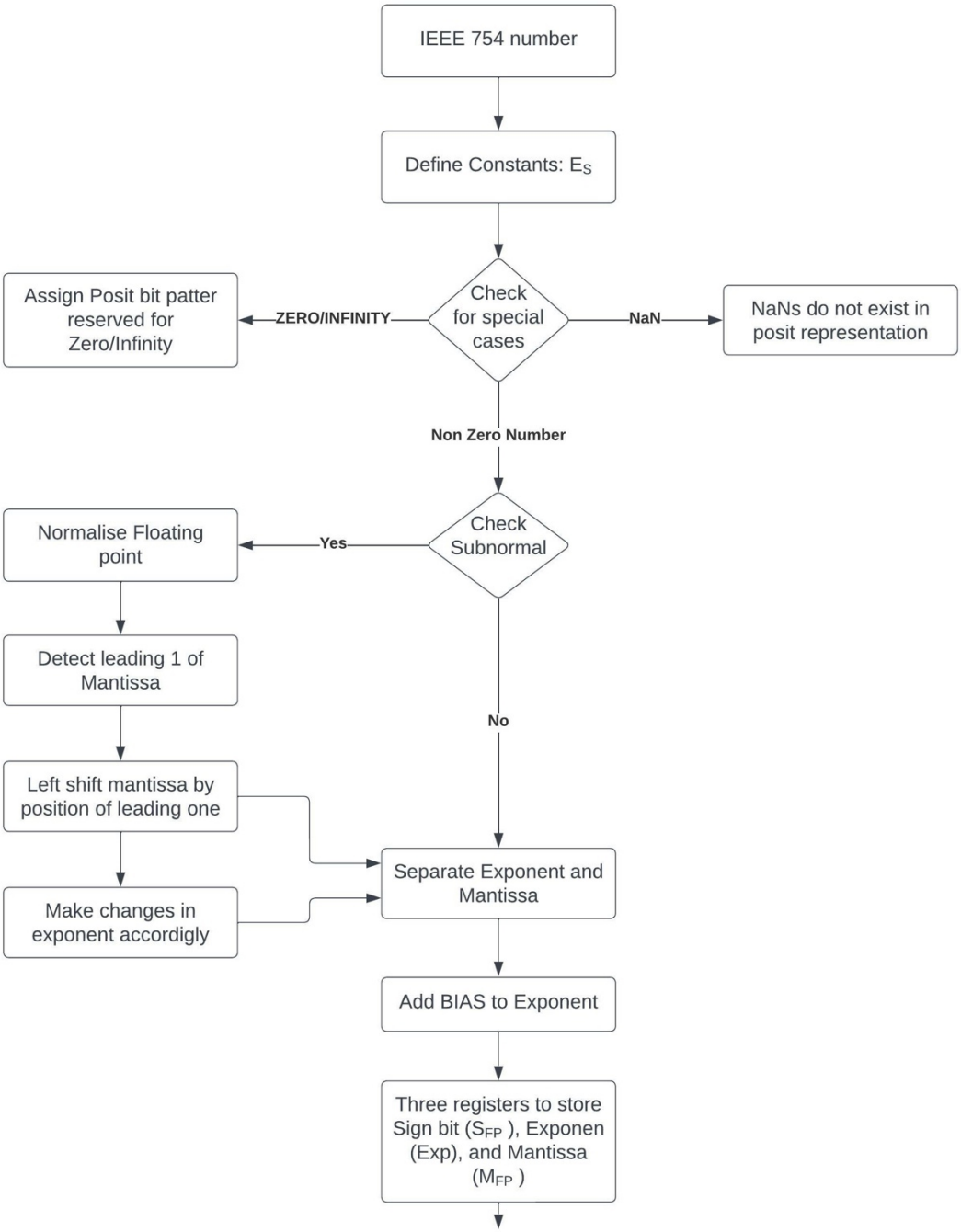
i. S_{FP} denotes the sign bit
ii. An N-bit sequence, $N\{!\,Exp[E]\}$, a repetitive sequence of $!\,Exp[E]$, gives the regime bits (repetitive 0's for negative exponent, and repetitive 1's for positive exponent)
iii. E^{th}bit of Exp terminates the regime.
iv. A second N-bit word $\{!\,Exp[E], E_0, M_{FP}\}$ with necessary 0-bits padded to the LSBs is combined with previous N-bit word to form a 2N – bit word {Regime, Exponent, Mantissa} (REM)
v. To desired regime sequence in LSB N-bit of REM, REM is dynamic right shifted by R_0bits are $S_{FP} = 1$, 2's complement of REM is taken. Thus, $REM[N-1:1]$ is the final {Regime, Exponent, Mantissa}. Finally, we combine this with the sign bit, keeping the special cases of zero and infinity in mind, to get our Posit number.

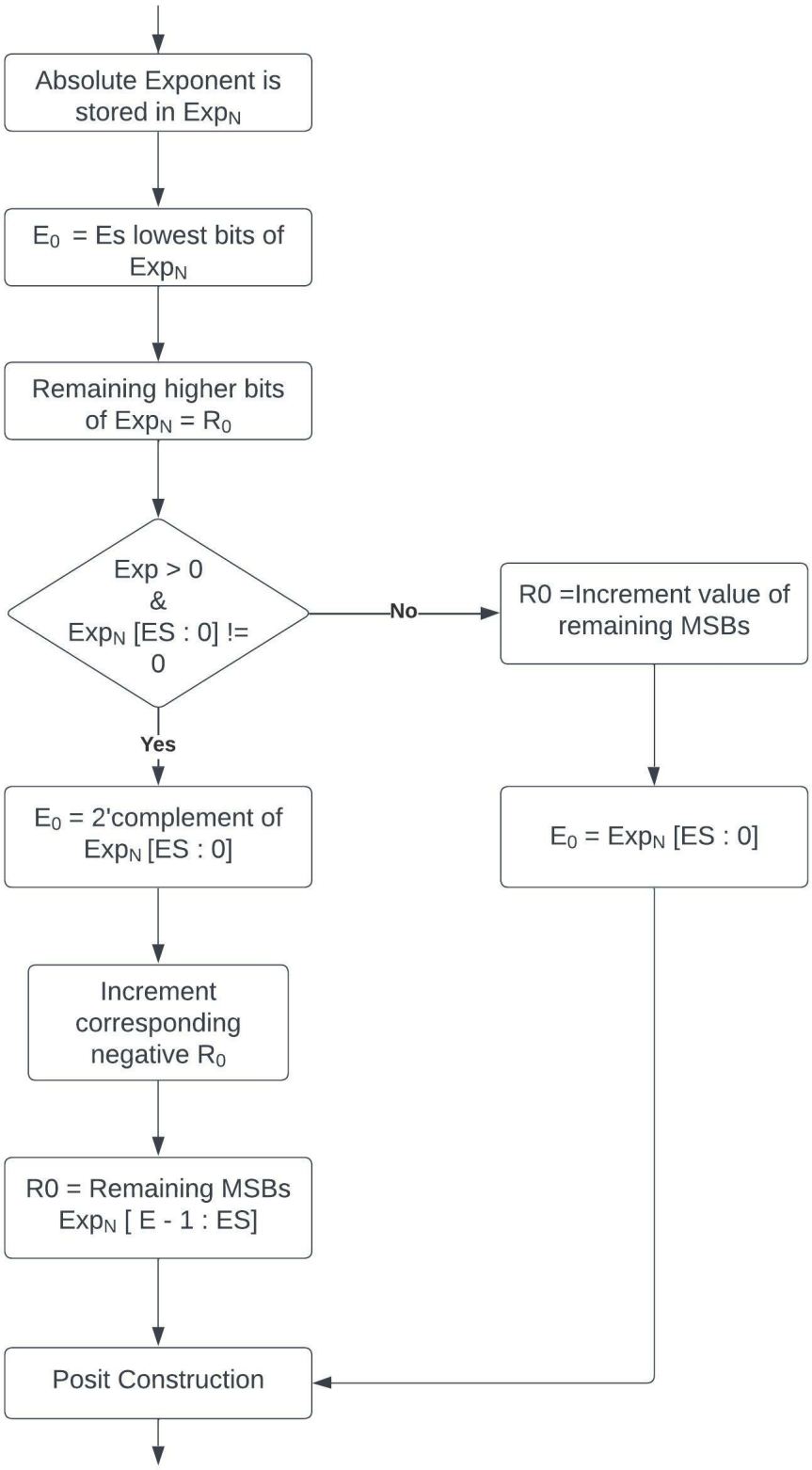
The process is presented as a pseudo-code algorithm in Algorithm 1 and the flow diagram is shown in Figure 5.2.

Algorithm 5. 1: IEEE 754 to Posit Conversion

IEEE 754 (FP) to Posit Conversion

1. **Constraints**
2. N: FP/Posit Word Size
3. E: FP Exponent Field Size
4. BIAS = (2**(E – 1)) – 1 : FP Exponent Bias
5. ES: Posit Exponent Field Size
6. ***Input bit string :*** *IN*
7. **FP component separation: Sign-bit** (S_F)**, Exponent** (E_F)**, Mantissa** (M_F)**, Expectations (Infinity** (INF_F)**, Zero** (Z_F) **)**
8. S_F <= In [N – 1]
9. E_F <= In [N – 2 : N – 1 – E]
10. M_F <= {|E_F, IN[N – 2 : N – 1 – E] }
11. Z_F <= !IN[N – 2 : 0]
12. INF_F <= &E_F
13. Pre-Normalisation of FP:
14. Lshift <= L1D of M_F
15. M_F [N − 1 : 0] <= Dynamic Left Shift of { M_F, E′ b0} by Lshift
16. Exp[E : 0] ← { E_F [E − 1 : 1], E_F [0]|(!(| E_F))} - BIAS - Lshift
17. **Posit Component Construction: Exponent (**E_0 **), Regime Value (**R_0 **), Mantissa(**M_0**) and their Packing (POS)**
18. Exp_N [E − 1 : 0] <= Exp[E] ? − Exp[E − 1 : 0] : Exp[E − 1 : 0]
19. **IF** (Exp[E]&(|Exp_N [ES − 1 : 0]))
20. E_0 [ES − 1 : 0] <= 2's complement of Exp_N [ES − 1 : 0]
21. **ELSE**
22. E_0 [ES − 1 : 0] <= Exp_N [ES − 1 : 0]
23. **IF** (!Exp[E]||(Exp[E]&(|Exp_N [ES − 1 : 0])))
24. R_0 [E − ES − 1 : 0] <= Exp_N [E − 1 : ES] + 1
25. **ELSE**
26. R_0 [E − ES − 1 : 0] <= ExpN [E − 1 : ES]
27. POS[2 ∗ N − 1 : 0] <= {N{!Exp[E]}, Exp[E], EO , M_F[N − 2 : ES]}
28. POS <= Dynamic Right Shifted by R_0 bits
29. **If** (S_F == 1): POS <= (2's complement of POS)
30. **Final Output <=** {S_F, LSB (N-1) of POS}
    Give Output considering INF_F and Z_F


IEEE 754 number
Define Constants: E_S
Check for special cases
ZERO/INFINITY
Assign Posit bit patter reserved for Zero/Infinity
NaN
NaNs do not exist in posit representation
Non Zero Number
Check Subnormal
Yes
Normalise Floating point
Detect leading 1 of Mantissa
Left shift mantissa by position of leading one
Make changes in exponent accordigly
No
Separate Exponent and Mantissa
Add BIAS to Exponent
Three registers to store Sign bit (S_{FP}), Exponen (Exp), and Mantissa (M_{FP})


Absolute Exponent is stored in Exp_N
E_0 = Es lowest bits of Exp_N
Remaining higher bits of Exp_N = R_0
Exp > 0 & Exp_N [ES : 0] != 0
No
R0 =Increment value of remaining MSBs
Yes
E_0 = 2'complement of Exp_N [ES : 0]
E_0 = Exp_N [ES : 0]
Increment corresponding negative R_0
R0 = Remaining MSBs Exp_N [E - 1 : ES]
Posit Construction

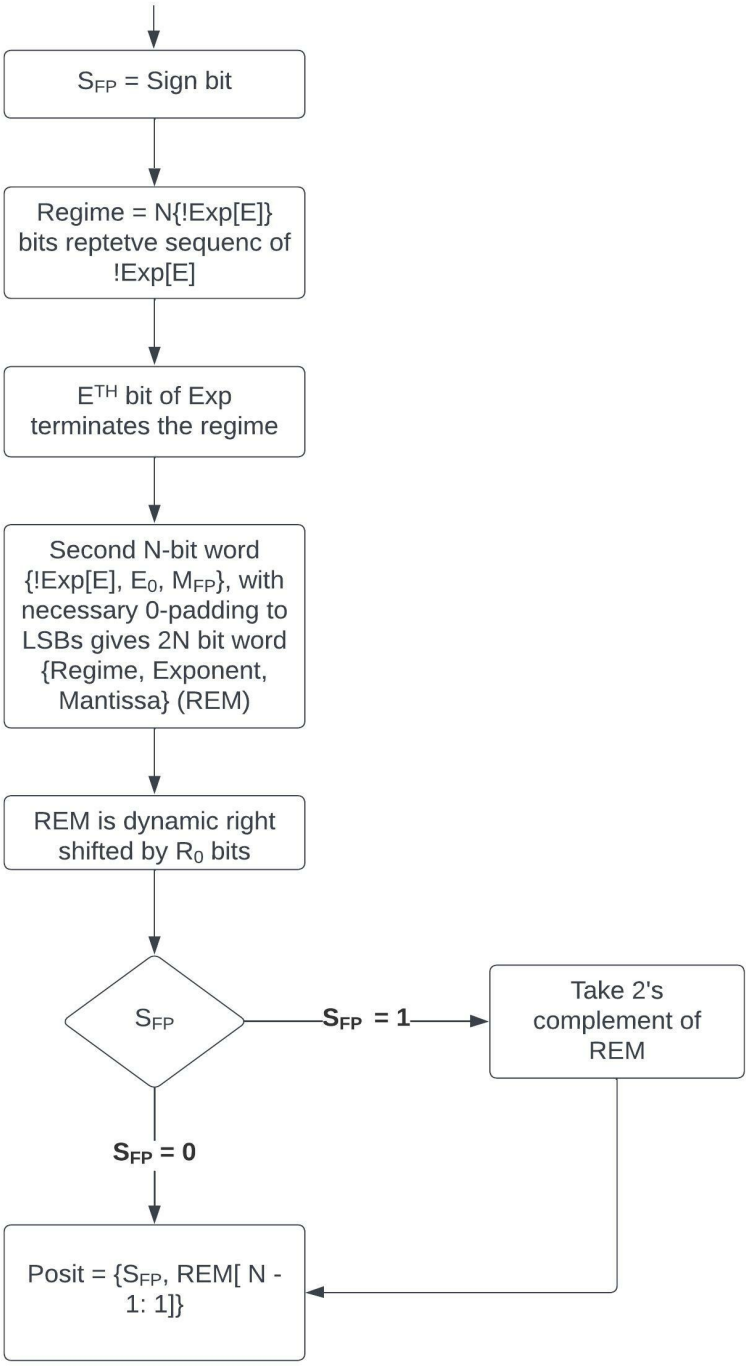


Figure 5. 2: Flow diagram of IEEE 754 to Parameterised Posit Conversion

b) Step – 2: Leading One/Zero Detector

For a 2N – bit wide posit, the LOD (Leading One Detector) consists of two parameterized LODs (h and l) that take MSB half N-bits and LSB half N - bits of the 2N- bit wide input and a MUX and OR gate. Each of these parameterized LODs consists further of LODs which take in inputs in the order – N, N/2, N/4, and so on until the leaf cell takes in a 2-bit input. The one of the outputs of these LOD's is $log_2 N$ bits wide, the other output is "vld". At the leaf cell level, the output is generated by performing AND operation on the inverse of the higher bit with the lower bit, while "vld" is obtained by the Reduction OR of the 2 bits. Thus, the output is true when there is a leading one. As we move to higher parameters, each parameterised LOD has 2 sub LODs for the upper half and lower half. Only the higher half consisting of the leading one will produce a true "vld" bit, else the lower half LOD will produce a "vl" bit. The LOD outputs are then given to a 2:1 MUX where the lower "vld" is concatenated with the output of the Lower LOD and zero is appended to the higher output. The Higher "vld" acts as the Select line for the MUX. So, these LODs can perform the function of both Leading One Detector and Leading Zero Detector in a parameterised dynamic manner.

For dynamic left/right shifting, a parameterized barrel shifter is constructed with word width (N) and shifting amount (S) as parameters. A barrel shifter requires one N-bit 2:1 MUX for each bit of S. So, here, it requires S numbers of 2:1 MUXs each of N-bit size [9].

The L1D/L0D pseudo-code is presented in Algorithm 5.2 and the flow diagram is shown in Figure 5.3.

Algorithm 5. 2: Algorithm for L1D/L0D and Dynamic Left Shift operator

Algorithm for L1D/L0D and Dynamic Left Shift operator

```
1.  L1D/L0D #(N) (in[N-1:0], K[S-1:0], vd):
2.  N: Word Size, S: Log2 (N)
3.  GENERATE
4.  IF (N == 2)
5.  For L1D: vd = |in, K = (!in[1]) & in[0]
6.  For L0D: vd = !(&in), K = in[1] & (!in[0])
7.  ELSIF (N & (N-1))
8.  LOD/LZD #(1<<S) (1<<S 1'b0 | in, K, vd)
9.  ELSE
10. K_L[S-2:0], K_H[S-2:0], vd_L, vd_H
11. L1D/L0D #(N>>1) (in[(N>>1)-1:0], K_L, vd_L)
12. L1D/L0D #(N>>1) (in[N-1:N>>1], K_H, vd_H)
13. vd = vd_L | vd_L
14. K = vd_H ? {1'b0,K_H} : {vd_L,K_L}
15. ENDGENERATE
16. Left Shift Operation
17. DLS #(N) (in[N-1:0], b[S-1:0], L1D_OUT):
18. N: Word Size, S: Log2 (N), TMP[S-1:0][N-1:0]
19. TMP[0] = b[0] ? in << 1 : in;
20. GENVAR i
21. GENERATE
22. for (i=1; i<S; i=i+1)
23. TMP[i] = b[i] ? (TMP[i-1] << 2**i) : TMP[i-1]
24. end
25. ENDGENERATE
26. L1D_OUT = TMP[S-1]
```

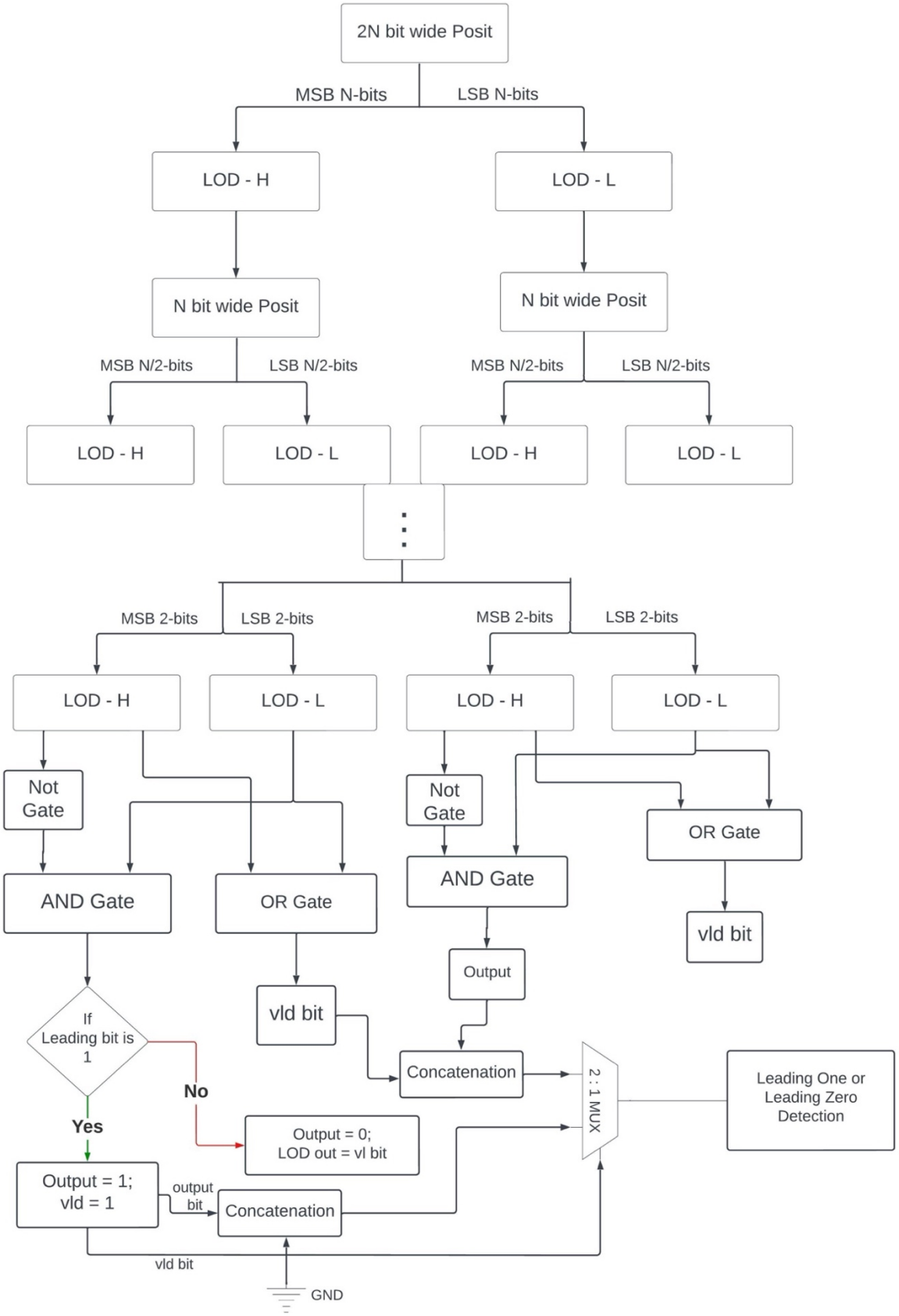


Figure 5. 3: Leading One/Zero Detector

c) Step – 3: Parameterised Posit–to–Floating Converter

The first step in this conversion is to check for zero and infinity special cases. The MSB is sign bit of the posit. If the MSB is 1, we take a 2's complement of the input posit number into a variable XIN.

The challenge of extracting the components of a posit number, considering it's run time variation, is handled by Leading One Detector, Leading Zero Detector and the Dynamic Left Shifter components as explained below:

i. The $(N - 2)^{th}$ bit of XIN is used to check whether regime is positive or negative. This bit is denoted RC (Regime Check bit)
ii. We use the LOD for (N – 1) LSB of XIN to count the number 0's when the regime is a sequence of 0's ending in 1, i.e., negative regime, and store that count value in temporary register K0. We use LZD for (N – 2) LSB to count the number of 1's ending in 0, i.e., positive regime, and store that count value in temporary register K1. In the case of positive regime, 1-bit lesser is used since regime is 1 less than the actual number of repeating 1's.
iii. The absolute value of Regime (K0 or K1) is decided as per the value of RC. RC is also used as a select signal to determine the left shift amount of for regime.
iv. XIN is then left shifted by (regime + 1) bits to align the exponent with the MSB of XIN.
v. Now the most significant ES bits denote the exponent (E_P), remaining bits are mantissa.
vi. The final floating-point exponent is constructed usingE_0, RC, Regime,E_P and BIAS values.

The pseudo-code for Posit-to-IEEE 754 conversion is shown in Algorithm 5.3 and the flow diagram is given in Figure 5.4.

Algorithm 5. 3: Posit to IEEE 754 conversion

Posit to IEEE 754 Converter

1. **Constrains**: N, ES, E, BIAS (Similar to the Algorithm-1 definition)
2. **Input**: IN
3. **Posit Component breakup**:**Sign** (S_{Pos}), **Regime** (R_{Pos}), **Exponent** (E_{Pos}), **Mantissa** (M_{Pos}), **Exceptions (Infinity** (INF_{Pos}), Zero (Z_{Pos})**)**
4. Z_{Pos} <= !IN, (All bits of IN are 0)
5. INF_{Pos} <= IN[N − 1]&(!IN[N − 2 : 0]), (all except MSB are 0)
6. S_{Pos} <= IN[N − 1]
7. $IN_X \leftarrow S_{Pos}$? − IN : IN, (2's complement for -ve posit)
8. Regime Check (R_C): R_C <= XIN[N − 2], (0 for -ve regime, 1 for +ve regime)
9. K_1 <= L1D of XIN[N-2:0], (For -ve regime sequence)
10. K_1 L0D of XIN[N-3:0], (For +ve regime sequence)
11. Absolute Regime Value: R <= Rc ? K_1 : K_0
12. Regime Left Shift : Lshift <= RC ? K_1 + 1 : K_0
14. IN_tmp[N − 1 : 2] <= IN_X[N − 3 : 0] << Lshift, (Dynamic left shifting)
15. E_{Pos} [E − 1 : 0] <= XIN_tmp[N-1:N-ES]
16. M_{Pos} [N − 1 : ES − 1] <= {|IN[N-2:0], XIN_tmp[N-ES-1:0]}
17. **FP Construction:**
18. E_0 [E : 0] ← R_C ? {R_{Pos}, E_{Pos} } + BIAS : {−R_{Poa}, E_{Pos} } + BIAS
19. **IF** (INF_{Pos} | E_0 [E] | &E0 [E − 1 : 0]): FP_0 <= Infinity
20. **ELSIF** (Z_{Pos} | (M_{Pos} [N − 1]): FP_0 <= {S_{Pos} , E − 1{1′ b0}, M_{Pos} [N − 2 : E]}
21. **ELSE** FP_0 ← {S_{Pos} , E_0 [E − 1 : 0], M_{Pos} [N − 2 : E]}

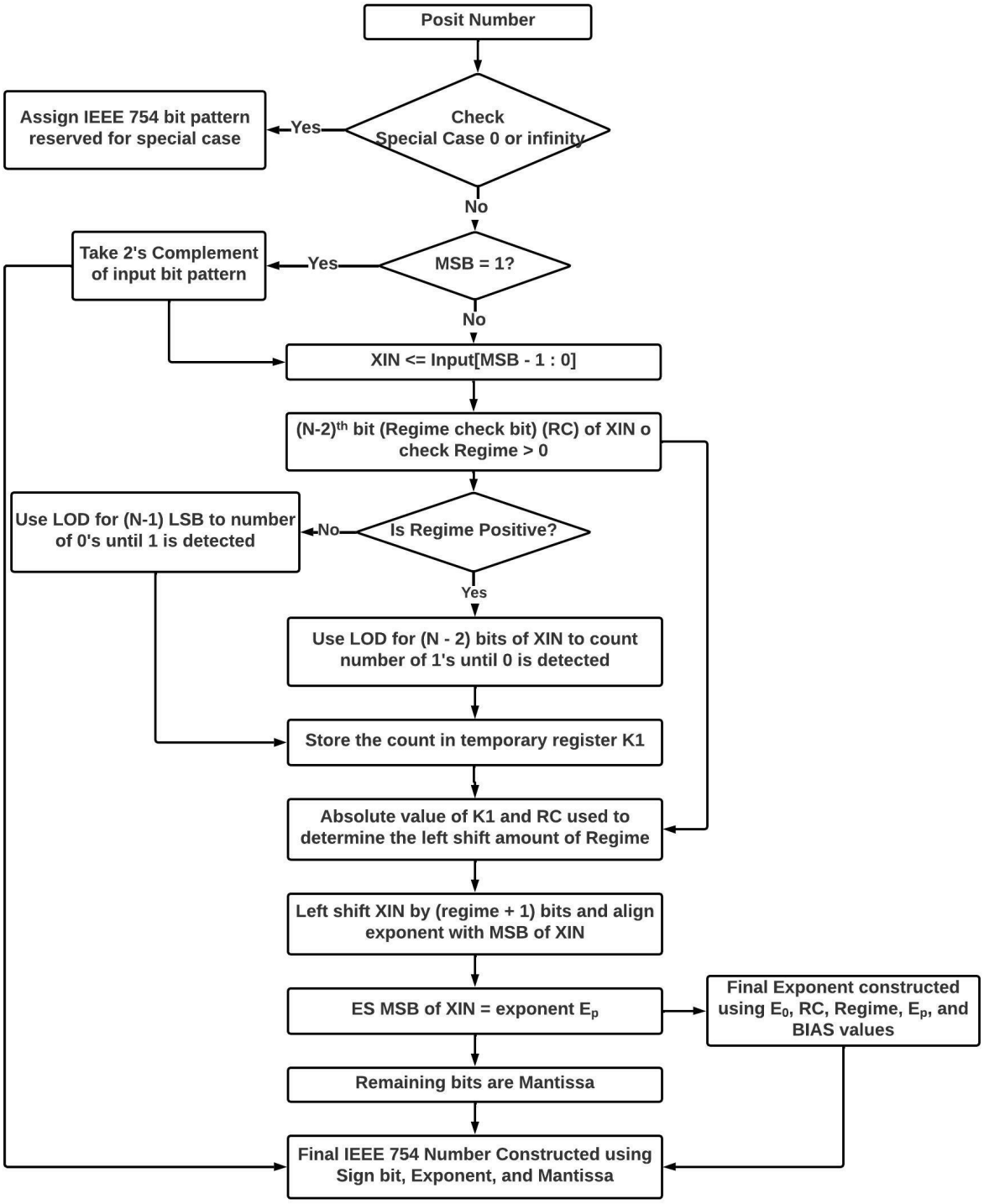


Figure 5. 4: Flow diagram of Posit to IEEE 754 Converter

d) Step – 4: Parameterised Posit Addition

For Posit Addition, first, the posits are decoded into the REM components of both the operands are extracted. This is followed by the arithmetic addition of the two posit numbers after equating the exponents and accordingly adjusting the mantissae. Here we get the sign, Exponent and Mantissa of the final sum. These are then recomposed to form the Sign, Regime, Exponent and Mantissa of the sum. Finally, the post-processing is done to round-off to the required accuracy, etc.

To perform the addition operation after the data extraction step, first, the total exponents of both the operands are calculated using the Regime and exponent bits of respective operands. These exponents are then compared to delineate the greater operand and the smaller operand. The difference between these two operands is used to shift the mantissa of the larger operand dynamically right. The difference is also added to the larger exponent. Hereafter the mantissa is checked for overflow or underflow and added together to get the sum of the addition. Now we have the Sign, Exponent and Mantissa of the sum. We use the encoding process explained above to re-encode the sum in REM components and repackage the REM components into the final posit output.

The same process is followed for the subtraction of two posit numbers. However, the subtractor is negated by setting the sign bit high and taking a 2's complement of the mantissa. The flow diagram of Parameterised Posit Addition is shown in Figure 5.5.

a) Step – 5: Parmaterised Posit Sigmoid

To calculate the sigmoid function of a number efficiently in IEEE 754, we have to develop a complex arithmetic unit using Non-linear approximation of Sigmoid function. As proven in Chapter 3, the Nonlinear Approximation method is the most efficient methods of implementing sigmoid function using IEEE 754. Nonlinear Approximation involves 6 multiplication operations, and 5 addition operations to calculate the Sigmoid function.

Posits have a tapered accuracy. This tapered accuracy is achieved because of a nonlinear sampling density function on the real axis [10]. When we integrate the sampling density function, we get the cumulative density function. The cumulative density function for Posits is a sigmoid function. To get the sigmoid function of a Posit we need to left shift the Posit by 2

bits to obtain the sigmoid function efficiently. Hence, no approximation method is not required here.

The sigmoid function implemented using Posit and IEEE 754 format is shown in Figure 5.6. Here, the Sigmoid function was applied to real numbers ranging from -10 to +10 in both IEEE 754 floating point representation and Posit implementation.

For IEEE 754 represented numbers, we need to perform four exponent calculations, eight multiplications, and five addition operations to calculate the sigmoid of one input value.

For the Posit representation, for the same operation, the first bit of the posits is inverted, and then the bits are shifted right by 2 bits while appending 0's on the left [3]. So it requires only three operations i.e. – one-bit inversion operation, one 2-bit shifting and a 0-padding operation.

Both the outputs when plotted on the same axes overlap with each other indicating sigmoid representation with similar accuracy.

The complexity of Posit implementation has been measured in terms of standard library logic gates required to implement the sigmoid function, and it has been compared to that of IEEE 754. For IEEE 754, 10634 standard cells are required to implement the Sigmoid Function. Parameterized Posit implementation of the Sigmoid Function requires 447 standard cells. Both the implementations were synthesized using TSMC 180nm standard cell library on Cadence RTL Encounter.

For IEEE 754, 10634 standard cells are required to implement the Sigmoid Function. Parameterized Posit implementation of the Sigmoid Function requires 447 standard cells. Both the implementations were synthesized using TSMC 180nm standard cell library on Cadence RTL Encounter. Table 5.4 shows the comparison of the sigmoid function hardware implementation complexity using IEEE 754 and Parameterised Posit.

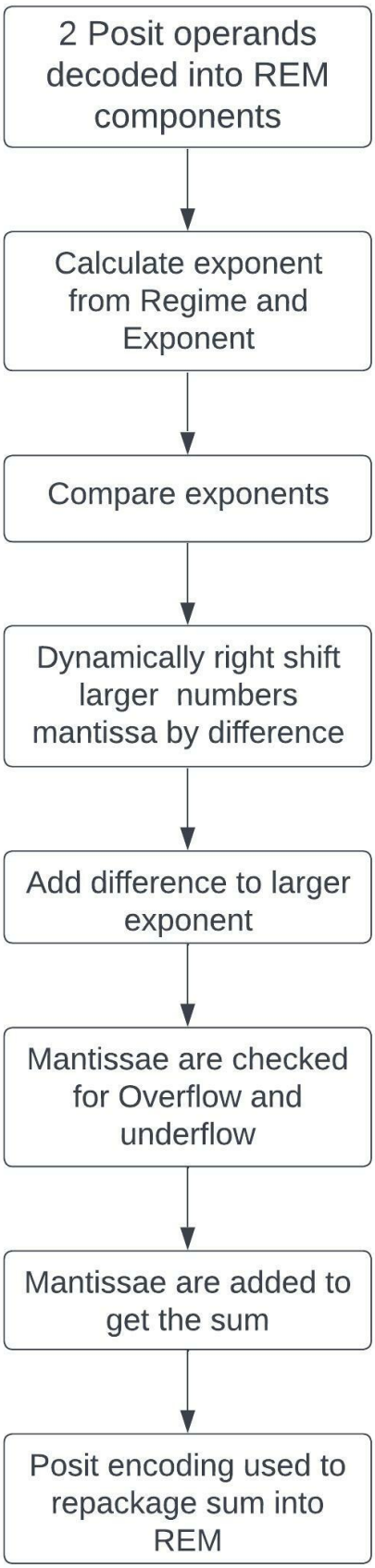


Figure 5. 5: Flow diagram of Posit Addition Unit

Table 5. 4: Comparison of the complexity of Hardware Implementation of Sigmoid Function

Implementation	Logic Gates
IEEE 754 Floating Point Representation	10634
Parameterised Posit	447

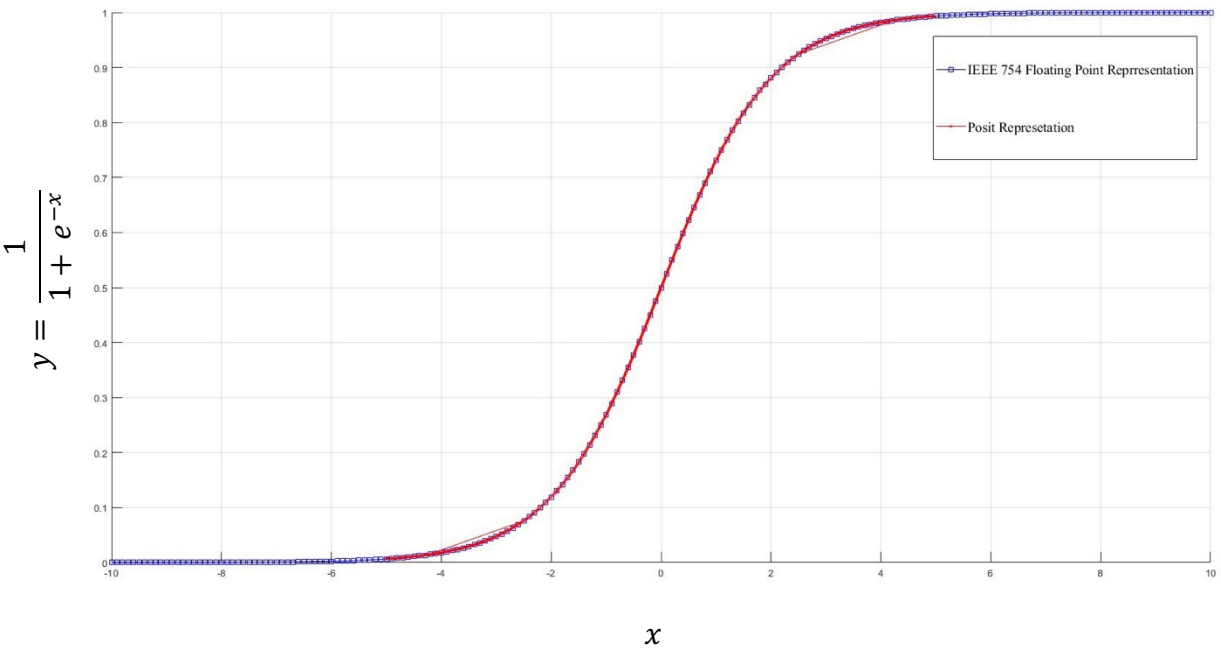


Figure 5. 6: Sigmoid function calculation comparison between IEEE 754 and Parameterized Posit representation.

5.4 Results and Observations of Proposed Smart Portable Water Quality Classification Device (WQC-Device)

5.4.1 Schematic of PPANN synthesized using TSMC 180nm technology node.

In the proposed WQC device, ANN has 4 inputs and 3 outputs. It has 3 hidden layers of neurons, each consisting of 32 neurons, and each neuron is composed of 1 sigmoid, 4 multiplications, and 1 addition unit.

The snapshot of the schematic of the implemented single neuron circuit synthesised using TSMC 180nm technology node on Cadence RTL Compiler is shown in Figure 5.7.

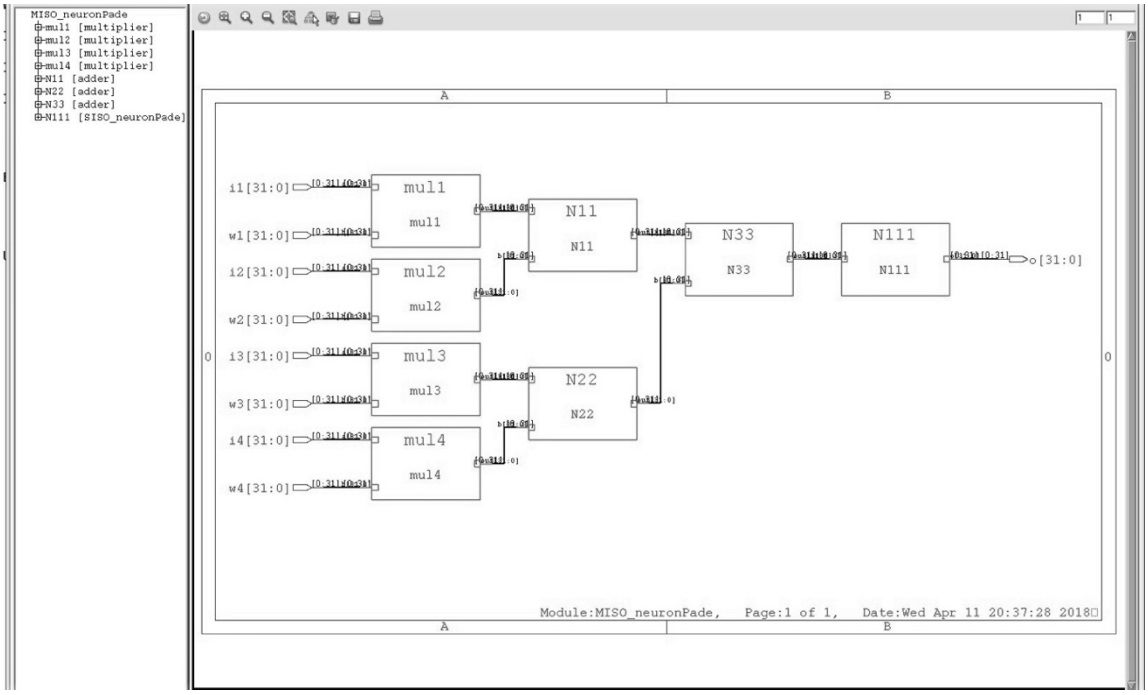


Figure 5. 7: ASIC implementation of a neuron on Cadence RTL Encounter using TSMC 180nm Standard Cell Library

5.4.2 Comparison of the results of proposed ASIC and FPGA implementation of PPANN in IEEE 754 and Parameterised Posit, respectively.

Comparison 1 – Figure 5.8: Comparison of proposed ASIC Implementation of ANN using - IEEE 754 and Parameterised Posit

Comparison 2 - Figure 5.9: Comparison of proposed FPGA Implementation of ANN using - IEEE 754 and Parameterised Posit.

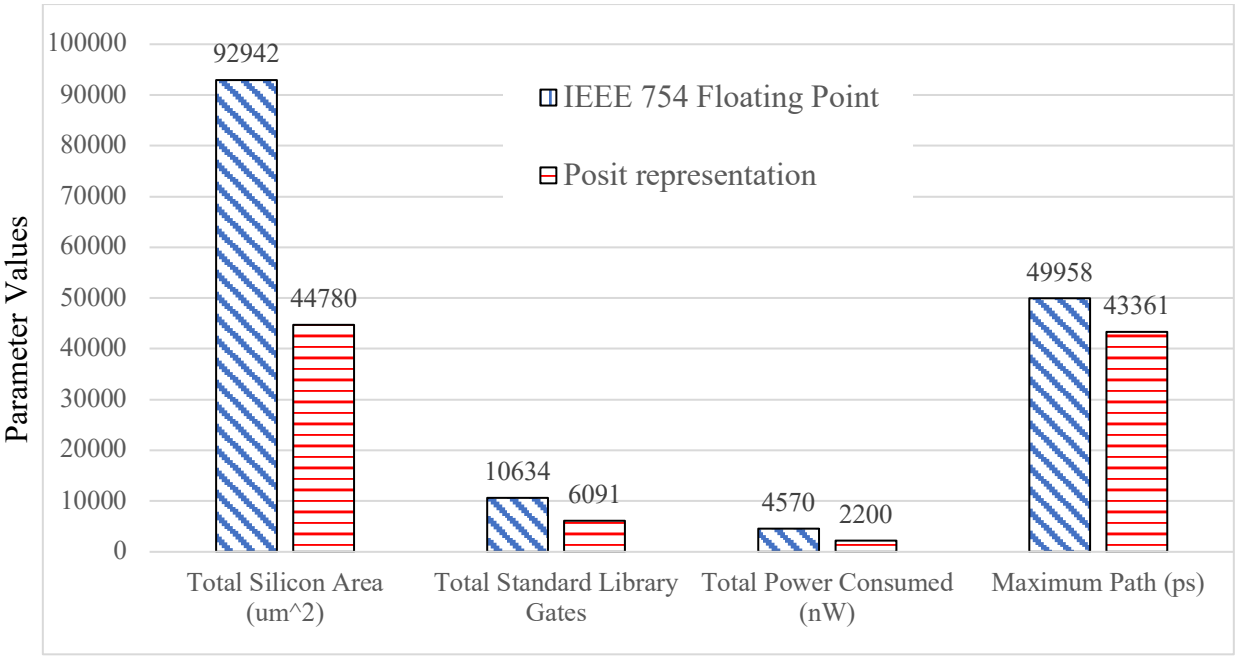


Figure 5. 8: Comparison of proposed ASIC Implementation of ANN using - IEEE 754 and Parameterized Posit

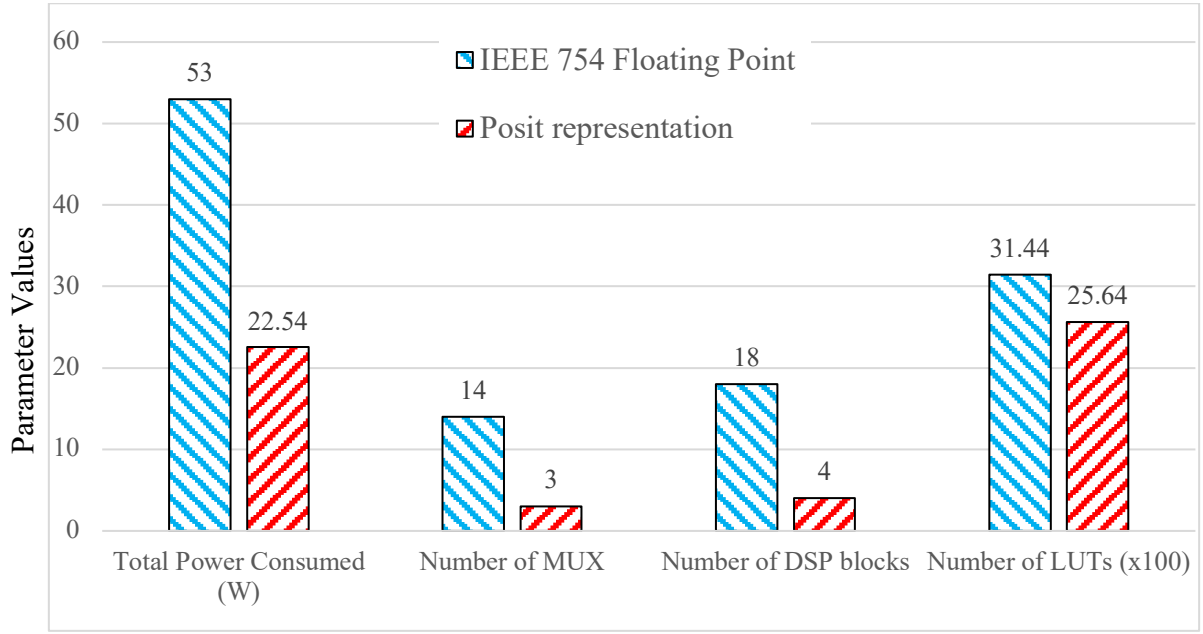


Figure 5. 9: Comparison of proposed FPGA Implementation of ANN using - IEEE 754 and Parameterized Posit

Observations

- It is observed that Parameterized Posit implementation consumes 50% less silicon area and standard cells.
- Parameterized Posit has 50% less power consumption IEEE 754 floating point representation.
- Parameterized Posit implementation has an advantage of 13.2% over IEEE 754 in Critical Path Delay.
- Parameterized Posit has comparable accuracy to that of IEEE 754.

5.5 Conclusion

In this chapter, we have implemented an ANN for Water Quality Classification using Parameterised Posits and compared the hardware performance with the IEEE 754 implementation of the same.

In comparison to ANN implemented with IEEE 754The PPANN design achieves :

- 50% lesser Resource utilisation
- 50% lesser power consumption
- 13% lesser critical path delay.
- It achieves similar accuracy (presented in Figure 5.6).

The ANN designed in this chapter has four inputs – pH, ORP, DO, and EC. The sensor electrodes required to measure EC and DO are expensive, leading to an increased cost. Further, laboratory methods like reverse osmosis for Dissolved Oxygen, are available but cannot be used in portable devices. Thus, Data augmentation is required, wherein, parameters like EC and DO, are predicted based on the data available from pH and ORP in-situ measurements. A detail of the data augmentation used in this work has been presented in Chapter 6.

The implementation of the Sigmoid function using Parameterised Posit involves 447 Logic units as compared to 10634 Logic Units required by IEEE 754. Thus, Parameterised Posit reduces the hardware complexity of the implementation of Sigmoid Function, as compared to the conventional method.

5.6 References

[1] A. Gupta, A. Gupta and R. Gupta, "Efficient ASIC Implementation of Artificial Neural Network with Posit representation of Floating-Point Numbers," in *International Conference on Next Generation Systems and Networks*, Pilani, 2022.

[2] H. So, "Introduction to Fixed Point Number Representation," UC Berkley, 28 02 2006. [Online]. Available: https://inst.eecs.berkeley.edu/~cs61c/sp06/handout/fixedpt.html. [Accessed 20 01 2023].

[3] J. L. Gustafon and I. . T. Yonemoto , "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations,* vol. 4, no. 2, p. 71–86, 2017.

[4] E. Morancho, "Unum: Adaptive Floating-Point Arithmetic," in *2016 Euromicro Conference on Digital System Design (DSD)*, Limassol, Cyprus, 2016.

[5] J. L. Gustafson, The End of Error: Unum computing, Chapman & Hall/CRC Computational Science, 2015.

[6] J. . L. Gustafson, "A radical approach to computation with real numbers," *Supercomputing Frontiers and Innovations,* vol. 3, no. 2, pp. 38 - 53, 2016.

[7] F. d. Dinechin, L. Forget, J.-. M. Muller and Y. Uguen, "Posits: the good, the bad and the ugly," in *CoNGA'19: Proceedings of the Conference for Next Generation Arithmetic 2019*, Singapore, 2019.

[8] V. Gohil, S. Walia, J. Mekie and M. Awasthi, "Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications," *arXiv:2014.04763v1 [cs.AR],* 2021.

[9] M. K. Jaiswal and H. S. K. -. Hayden, "Universal number posit arithmetic generator on FPGA," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2018.

[10] F. Xiao, F. Liang, B. Wu, J. Liang, S. Cheng and G. Zhang, "Posit Arithmetic Hardware Implementations with The Minimum Cost Divider and SquareRoot," *Electronics,* vol. 9, no. 10, 2020.

[11] J. Johnson, "Rethinking Floating Point for deep learning," *arXiv preprint,* p. https://arxiv.org/pdf/1811.01721.pdf, 2018.

[12] A. Gupta, A. Gupta and R. Gupta, "Power and Area Efficient Intelligent Hardware Design for Water Quality Applications," in *1st International Conference on on Microelectronic Devices and Technologies (MicDAT '2018) 20-22 June 2018*, Bacelona, 2018.

[13] X. Zhen-zhen and Z. Su-yu, "A Non-linear Approximation of the Sigmoid Function Based FPGA," in *Cybernetics, and Computer Engineering (ICCE2011) November 19–20, 2011, Melbourne, Australia*, Melbourne, 2011.

[14] Z. Hajduk, "High accuracy FPGA activation function implementation for neural networks," *Neurocomputing,* vol. 247, pp. 59-61, 2017.

[15] S. H. F. Langroudi, T. Pandit and D. Kudithipudi, "Deep Learning Inference on Embedded Devices: Fixed-Point vs Posit," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Williamsburg, VA, USA, 2018.

[16] S. Nambi, S. Ullah, S. S. Sahoo, A. Lohana, F. Merchant and A. Kumar, “ExPAN(N)D: Exploring Posits for Efficient Artificial Neural Network Design in FPGA-Based Systems,” *IEEE Access,* vol. 9, pp. 103691 - 103708, 20 July 2021.

[17] A. Gupta, F. M. Khan, A. Gupta and R. Gupta, “Portable Hand Held Smart device for real time Water Quality Measurement and Water Quality Classification”. India Patent Application No. - 202111017453, 14 04 2021.

[18] J. Kim, D. Seo, M. Jang and J. Kim, “Augmentation of limited input data using an artificial neural network method to improve the accuracy of water quality modeling in a large lake,” *Journal of Hydrology,* vol. 602, no. 126817, 2021.

[19] N. A. Cloete, R. Malekian and L. Nair, “Design of Smart Sensors for Real-Time Water Quality Monitoring,” *IEEE Access,* vol. 4, pp. 3975 - 3990, 2016.

[20] S. Nambi, S. Ullah, S. S. Sahoo, A. Lohana, F. Merchant and A. Kumar, “ExPAN(N)D: Exploring Posits for Efficient Artificial Neural Network Design in FPGA-Based Systems,” *IEEE Access,* vol. 9, pp. 103691 - 103708, 2021.

[21] M. Cococcioni, F. Rossi, E. Ruffaldi and S. Saponara, “Novel Arithmetics to Accelerate Machine LearningClassifiers in Autonomous Driving Applications,” in *2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Genoa, Italy, 2019.

[22] S. M. Mishra, A. Tiwari, H. S. Shekhawat, P. Guha, G. Trivedi, P. Jan and Z. Nemec, “Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithms,” in *32nd International Conference Radioelectronika (RADIOELECTRONIKA)*, Kosice, Slovakia, 2022.

[23] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson and D. Kudithipudi, “Deep Positron: A Deep Neural Network Using the Posit Number System,” *arXiv:1812.01762v2,* 2019.

[24] J. Lu, C. Fang, M. Xu, J. Lin and Z. Wang, “Evaluations on Deep Neural Networks Training Using Posit Number System,” *IEEE Transacrions on Computers,* vol. 70, no. 2, pp. 174 - 187, 2021.

Chapter 6

6. Hardware Implementation of Portable Smart device for real-time Water Quality Classification using Data Augmentation

This chapter presents the complete hardware implementation of a Portable Smart device for real-time Water Quality Classification using Data Augmentation. The implementation has been done using two design approaches – Embedded Systems and Application Specific Integrated Circuit (ASIC) Design. The results have been presented and compared with standard Water Quality Classification device.

6.1 Introduction

Conventional water quality measurement techniques include on-site sampling and subsequent laboratory-based tests; both are labour-intensive and cost-intensive processes. The measurements are not in real-time.

To accurately measure WQ, multiple parameters must be measured. In this work, we have chosen four parameters – pH, ORP, DO, and EC, to measure the water quality. The reasons for selecting these four parameters have been detailed in Chapter 3.

One of the significant challenges in making a real-time in-situ Water Quality Classification (WQC) device is the measurement of all the parameters that give us a complete WQ index. While pH and ORP can be measured easily, DO, and EC require expensive electrodes for in-situ measurement. This drives the cost of the Water Quality Classification device high. Thus, we use data augmentation to predict values of DO and EC to reduce the cost of the Water Quality Classification device.

Traditionally, data augmentation has been done using mathematical approaches, detailed in Section 6.1.1. However, ANN has proven to be more accurate in data prediction where there are no mathematical relations between the input and output parameters. This has been detailed in Section 6.1.2.

This chapter presents a Water Quality Classification device designed with two ANNs. First ANN to augment the data for DO and EC using measured pH and ORP, and then uses a second ANN to classify the water quality into one of the three classes – Potable, Agricultural, and Wastewater.

The device has been designed using two approaches – an Embedded approach and an ASIC design approach. These designs have been detailed in Section 6.3

6.1.1 Methodology for Data Augmentation

Data Augmentation becomes necessary for Water Quality Classification devices for in-situ application because measuring all the parameters contributing to the Water Quality measurement is not economical or practical. In most cases, we have to rely on Laboratory-based methods [1] [2] [3] constraining the portability of the device. For parameters like DO and EC, the cost of in-situ measurement is very high. Thus, to reduce the cost of measurement without any significant trade-off in Water Quality Classification performance, data augmentation is performed. There are two primary approaches to data augmentation – one is the mathematical approach, and the other is the ANN-based approach. The following subsections detail the two approaches.

6.1.2 Mathematical Approach

Data Augmentation has traditionally been done using mathematical approaches. Such approaches have been used for centuries, and constant development has been done in mathematical models to improve the accuracy of data augmentation. Numerical and statistical methods have been used in various fields to supplement missing data points, as described below.

- ***Linear Interpolation***

For simplicity, linear interpolation is frequently used. However, the significant variation between these points can be neglected because linear interpolation simply connects adjacent measurements with a line. Based on causality, statistical models can supplement missing data.

Linear Interpolation estimates the data assuming a straight line connects the two available data points. It ignores the possibility of local variation between the two known points on the curve.

- ***Multi Linear Regression***

Numerous studies have developed and utilized regression-based models, such as multiple linear regression (MLR) [4]. MLR was used to effectively predict daily rainfall, discharge, and groundwater elevation in [5]. MLR and daily discharge were used to predict the daily nitrogen and phosphorous content of water in [6]. MLR is effective at predicting the average trend but

has limited explanatory power for estimating extreme values and focuses on a single predictor [4].

- ***Bayesian Regression Model***

The Bayesian piecewise regression model predicted chlorophyll-a (Chl-a) concentration more accurately than the process-based model [7]. However, the Bayesian Regression model works only when there is some mapping possible between the input and output vectors.

- ***Watershed Models***

Using the outputs of a watershed model, such as the Soil and Water Assessment Tool (SWAT) [8], Hydrological Simulation Program–Fortran (HSPF) [9], and Stormwater Management Model, is an alternative way to supplement the time series for a surface water quality model (SWMM) [10]. Based on water balance and water quality interactions caused by precipitation, watershed models can predict flow rates and water quality concentrations at the outlet of subbasins. However, watershed models also require a large quantity of input data, including basic information such as topography, land use, or soil type, as well as rainfall data for each station or subbasin. In addition, the uncertainty in watershed models' conceptualization of hydrological processes, empirical equations, and estimation of various model parameters significantly impacts the precision of their results [11].

6.1.3 ANN Approach

The artificial neural network (ANN) method can be used as an alternative to augment input data by learning complex relationships between water quality variables and integrating nonlinearities. The benefit of ANN is that it can be easily extended to multivariate cases and modified by altering the network architecture, which increases the model's adaptability [12]. Due to their broad applicability, ANNs have been utilized in a number of water quality research projects. Specifically, the majority of studies have attempted to predict dissolved oxygen (DO) in a variety of environments, including rivers, lakes, reservoirs, ponds, and coastal waters [13]. In addition, it has been reported that the performance of ANN in numerous studies has been superior to that of other statistical techniques, such as regression [14]. Hence, The use of ANN methods to predict environmental water quality has increased rapidly. ANN technique can be applied to augment Water Quality parameter data and can improve the prediction accuracy of the water quality. Thus, ANN techniques can be applied to improve field measurement [4].

The implementation of the ANN based data augmentation have been detailed in Section 3.

6.2 ANN based Data Augmentation Design Flow

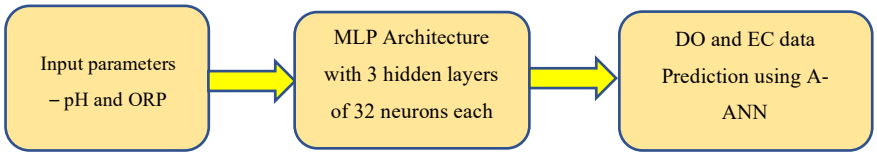


Figure 6. 1: Design flow for Complete ANN based Data Augmentation

Figure 6.1 shows the proposed ANN based data augmentation design flow.

The complete methodology of data augmentation unit is broken down into the following steps:

- Water sample collection
- Measurement of parameters using standard lab-based methods
- Measurement of pH and ORP using Arduino Uno
- Prediction of DO and EC values using Data Augmentation ANN (A-ANN)
- Hardware Implementation of A-ANN

Each of the aforementioned steps is discussed in the following subsections.

6.2.1 Water Sample Collection

1806 Ground and surface water samples have been collected from various locations in and around Pilani, Rajasthan, India. Based on knowledge, each sample was marked into one of the three categories – potable, agricultural, and wastewater. Details of Data Collection have been given in Chapter 3.

6.2.2 Lab-based parameter measurement and Collection of Training and Validation data set for Data Augmentation

1806 samples were tested for pH, ORP, DO, and conductivity using titration, spectroscopy, and solution chemistry. This is used as A-ANN and C-ANN training, testing, and validation data. Figure 6.2 exhibits the concept of digitisation of pH and ORP using Arduino Uno.

Arduino Uno has been used as the sensing circuit for electrodes and the sensing and conditioning circuits have been removed to save costs. The Arduino Uno has a 10-bit on-board ADC. For 16-bit output in IEEE 754 representation, it is observed that it requires two cycles at

9600 baud rate (which equals 1-second pulse rate), taking 2 seconds to generate the output readings.

The sensor accuracy is adjusted by the use of ANN for classification.

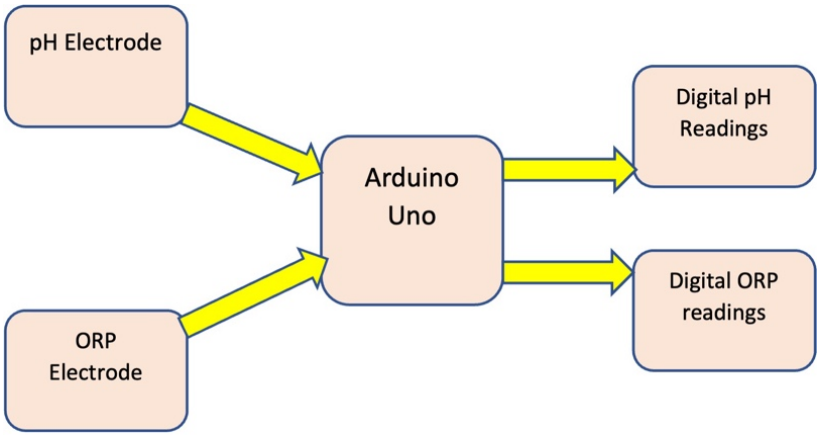


Figure 6. 2: Block diagram representing pH and ORP readings using Arduino Uno

6.2.3 Step 1: Measurement of pH and ORP using Arduino Uno

a) Measurement of pH

pH electrode voltage is read using Arduino Uno. An electrode is attached to the analog input. The Uno R3's 10-bit ADC transforms analog to digital. The Arduino Serial monitor shows digital voltages. To convert 10-bit digital pH to pH, requantise to voltage. The voltage range 0V–5.0V is quantised into 10-bits hence, the input voltage values are multiplied by the voltage range and divided by the quantization value: -

$$V_{in} = x \times \frac{5.0}{1023} \tag{1}$$

Where x represents the reading from the electrode and V_{in} is the corresponding digital value for the voltage.

This voltage value is now converted into pH reading by the Nernst equation: -

$$E = E_r + \left(\frac{2.303RT}{nF}\right)\log\left(\frac{unknown\ [H+]}{internal\ [H-]}\right) \tag{2}$$

Where E is the potential of reduction in a reaction, E_r is the standard or reference potential of oxidation/reduction reaction, R is the universal gas constant, T is the temperature in Kelvin, n is the number of electrons (in Moles), F is Faraday Constant and $[H-]$ & $[H+]$ denotes the concentration of H+ and OH- ions in the chemical reaction, respectively.

For our electrode equation (2) comes out to be: -

$$pH = \frac{((V_{in} - 512) \times 9.65)}{8.31 \times 2.302 \times 298} + 7 \quad (3)$$

Seven has been added in the above equation to offset the zero voltage to a neutral pH reading of seven.

b) Measurement of Oxidation Reduction Potential (ORP)

For the measurement of ORP, equation (1) is reused to convert digital readings into voltages (potentials). 2.25 is subtracted from the readings to offset the voltage readings by -225mV to obtain the ORP readings.

$$ORP = y \times \frac{5.0}{1023} - 2.25 \quad (4)$$

y denotes the digitized voltage reading from the ORP electrode. Like the pH electrode, the ORP electrode is also connected to the analog input of Arduino Uno.

6.2.4 Step 2: DO and EC Prediction using Augmentation ANN

Data Augmentation has been explored to predict the values parameters such as DO, EC, etc, in the literature review. These studies have been reviewed in Chapter 2. It is observed that augmentation has been done for parameters that have mathematical relations with some input parameters, such as EC and Total Dissolved Solids [15]. But parameters like EC and DO (output parameters) do not have an obvious mathematical relationship with pH and ORP (input parameters). Thus, other method needs to be explored for this purpose.

Artificial Neural Networks (ANNs) have demonstrated efficacy in predicting data when a discernible mathematical relationship between the input and output vectors is not readily apparent. ANNs possess the capability to discern variations in the output vector relative to the input vector, thereby enabling the prediction of the output vector for novel input vectors. Exploiting this characteristic, the current study leverages ANNs to forecast the values of Electrical Conductivity (EC) and Dissolved Oxygen (DO), employing pH and Oxidation Reduction Potential (ORP) as the input vector.

Prediction of DO and EC based on pH and ORP data using ANN reduces device cost Augmentation ANN forecasts DO and EC using Arduino Uno Board. Serial port transmits Arduino Uno sensor readings to Raspberry Pi (Raspberry Pi Foundation, n.d.). The Raspberry Pi microSD card holds experiment data. RaspberryPi memory stores ANN. Training, test, and validation sets are experimental data. 70% of data points were randomly selected for ANN model training, and 15% for testing.

6.3 Hardware Implementation of A-ANN

Two approaches have been used to implement A-ANN on hardware – Embedded Systems and ASIC (Application Specific Integrated Circuit).

The ANN architecture chosen for A-ANN was the same for both approaches. Figure 6.3 shows the Accuracy and Mean Square error for various architectures that were tested for the A-ANN. The ANNs were designed on MATLAB with chosen input and output vectors. The number of layers and number of neurons in each layer are varied and the accuracy and mean square error of the output are plotted using in-built functions of MATLAB.

From Figure 6.3 and Figure 6.4, we can conclude that the most suitable architecture is with 3 hidden layers with 32 neurons each because it gives maximum accuracy and minimum Mean Square Error.

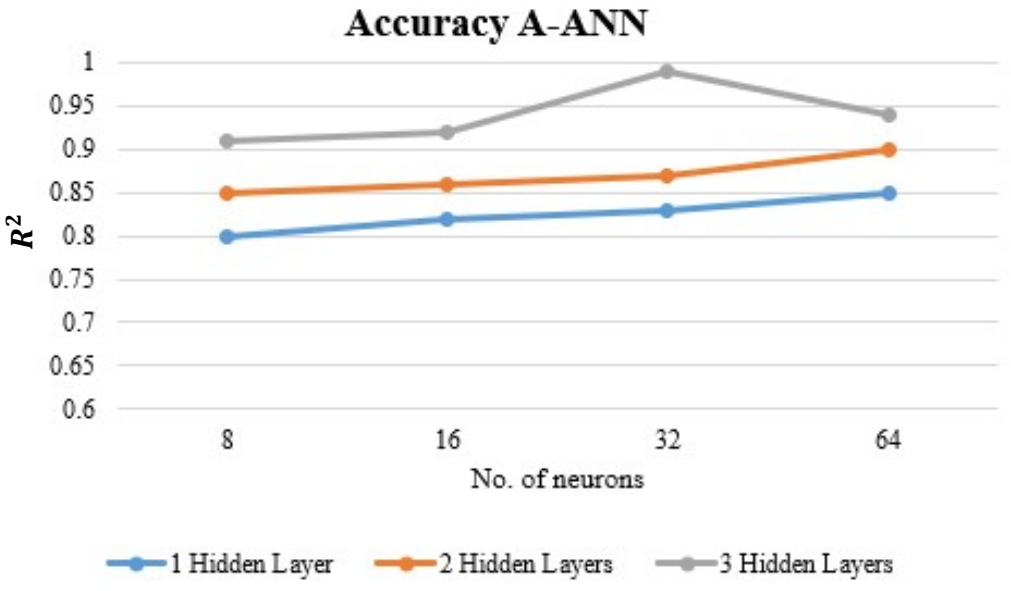


Figure 6. 3: Accuracy (R^2) of A-ANN

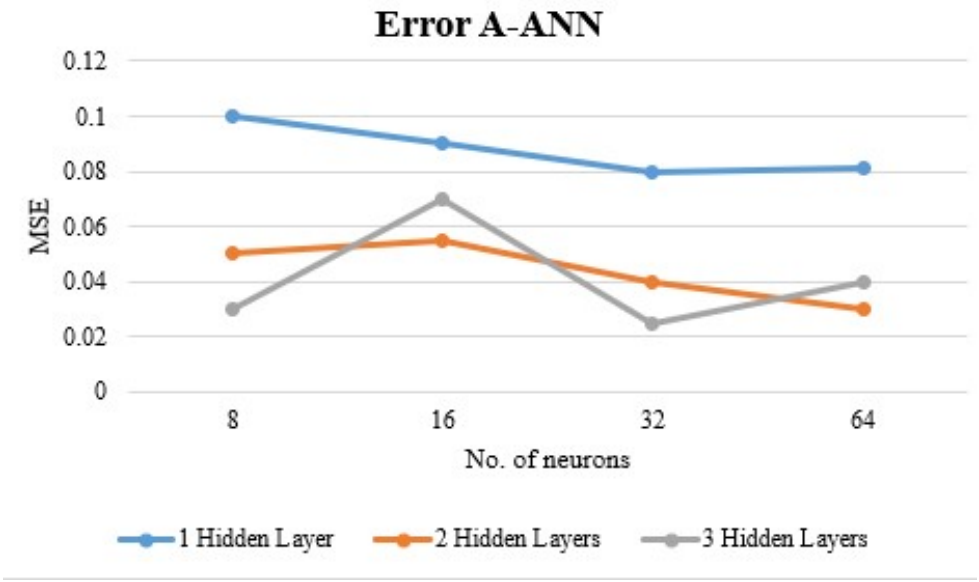


Figure 6. 4: Mean Square error for A-ANN

6.3.1 Embedded Systems Approach

The Embedded system approach involves the use of an Arduino and a Raspberry Pi board. A block-level representation of the system is shown in Figure 6.5

In this approach, the pH and ORP electrodes are connected to the Arduino Uno microcontroller board. The electrodes measure voltages in analog mode, which are then digitized and converted into respective parameter readings in the Arduino board. The process is explained in detail in section 6.2.3. These readings are passed on to the Raspberry Pi board over a serial connection.

Raspberry Pi board receives the pH and ORP readings over the serial board and stores the values in memory. These values are accessed by the Augmentation ANN python code, and the python code then predicts the values of DO and EC. The values of pH, ORP, DO and EC are added to the training data set stored in the Raspberry Pi memory. The python code and the memory locations for input data and training data sets are made accessible during boot to reduce wait time and introduce automation in the launch of ANN.

The reason behind going for the Embedded systems approach is primarily the short time to market it takes for an Embedded system-based design. Also, for a small-scale operation the Embedded Systems approach is more economical, and the design is simpler. Further, the repairability for the test device is much greater than an ASIC design.

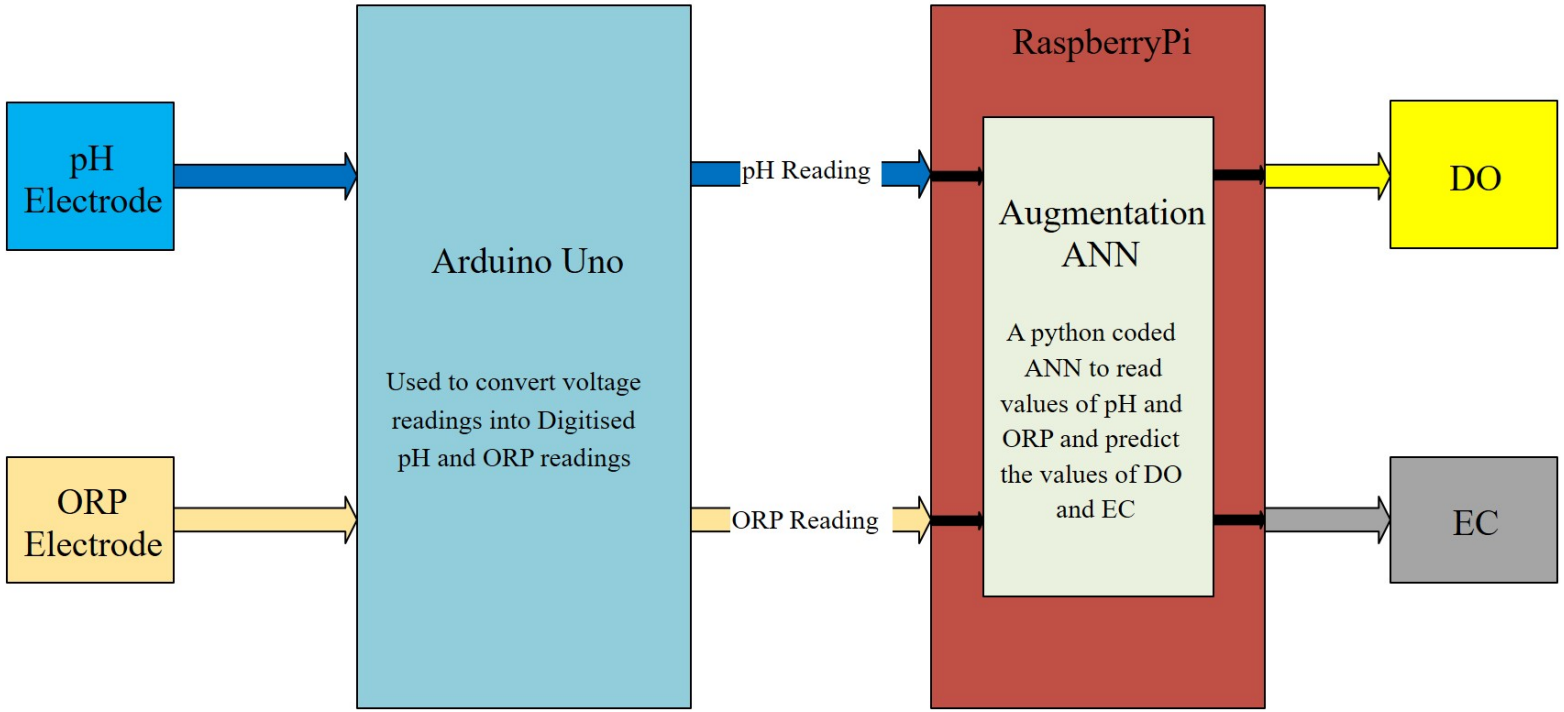


Figure 6. 5: Block-level diagram of Embedded System approach of Augmentation ANN

6.3.2 ASIC Design Approach

The ASIC approach involves the use of an Arduino and an IC designed using Verilog HDL and TSMC 180nm Standard cell library. A block level representation of the system is shown in Figure 6.6.

For the ASIC-based design approach, A-ANN has been coded using Verilog HDL based on the Posit floating point number representation system. The pH and ORP readings are taken just the same way as in the Embedded approach, using an Arduino Uno microcontroller. Since, Arduino is designed in the legacy number format systems, the pH and ORP readings are in IEEE 754 format. So, the pH and ORP readings are converted into Posit format before being transferred to an in-chip memory, where they are accessed by the A-ANN coded in Verilog. The Verilog code predicts the values of DO and EC in Posit format.

The DO and EC readings are not converted back into IEEE 754 format because they are to be used by C-ANN for classification further, which is also coded in Posit floating point representation format.

ASIC approach is more economical than Embedded Systems approach in masss production. Also, because of their very small size, and application specificity, ASICs are more power efficient and suitable for portable devices. ASICs also have the benefit of on-chip connections, leading to more reliable connections and lesser loosely connected wires. Thus, ASICs provide more reliable device at the cost of repairability.

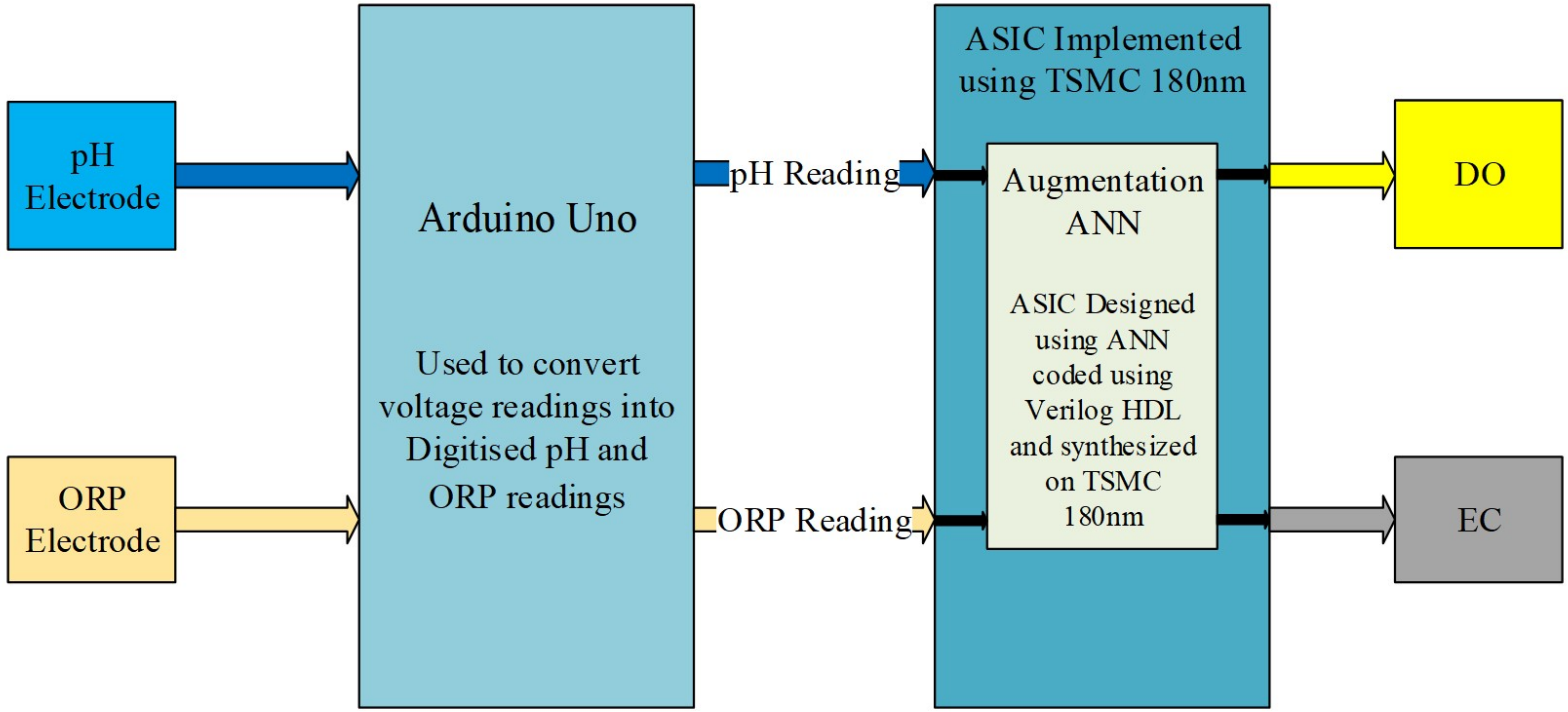


Figure 6. 6: Block-level diagram of Embedded System approach of Augmentation ANN

6.4 Implementation of Complete Water Quality Classification Device with Augmentation ANN(A-ANN) and Classification ANN(C-ANN)

The block diagram of the complete device with the classification ANN (C-ANN) is shown in Figure 6.7.

The complete device for Water Quality Classification is also designed using the two approaches, Embedded and ASIC, as mentioned in section 6.3. The following subsections detail the two implementations in detail.

Apart from the Embedded and ASIC approaches, the design is also implemented on an FPGA in order to test the functionality of the design. FPGA implementation required boards with large resource counts (> 235,000 logic blocks), which are expensive. Because of the limited number of resources available on this board, more than one FPGA board is required to implement the complete ANN. Thus, the cost of the FPGA implementation is driven high, defeating the low-cost objectives Water Quality Classification device. However, a reduced architecture has been implemented and the resource utilization and power figures are presented in Appendix C.

Similar to A-ANN, C-ANN architecture is also chosen after testing a number of different architectures. The architecture that offered maximum accuracy with minimum Mean Square Error is chosen. The method is same as described in Section 6.3.

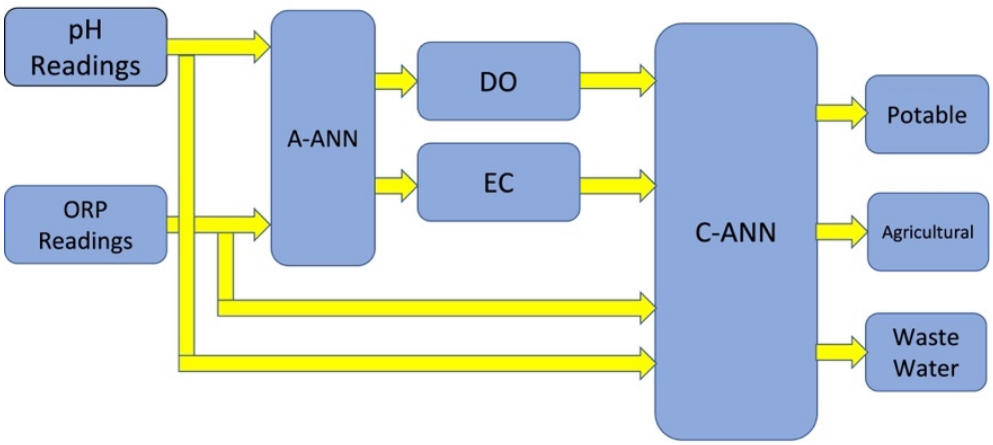


Figure 6. 7: Block diagram of Complete Water Quality Classification device

Figure 6.8 shows the plots of accuracy, and Figure 6.9 shows the Mean square errors of the different C-ANN architectures tested.

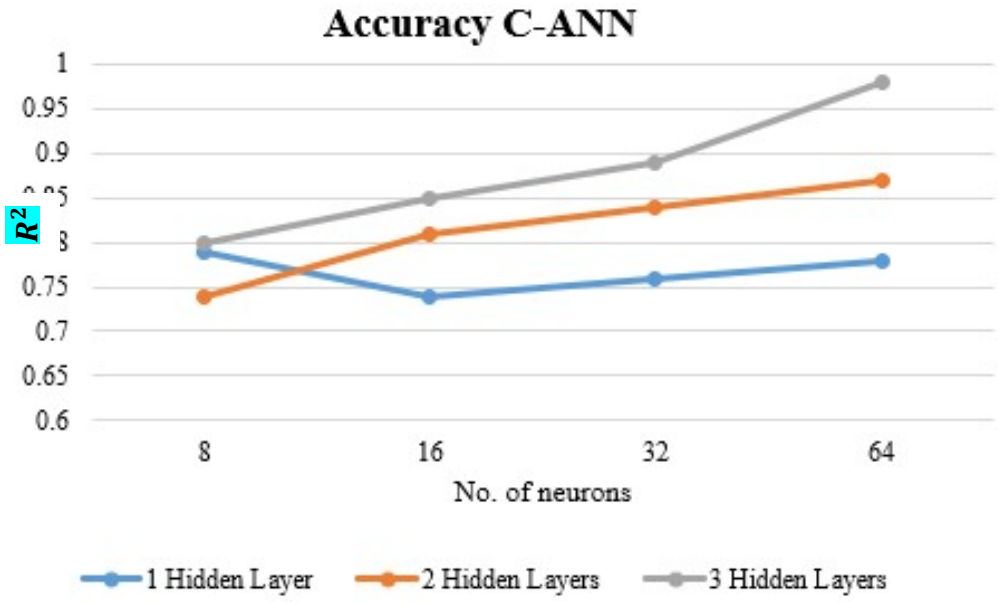


Figure 6. 8: Accuracy of C-ANN for different architectures

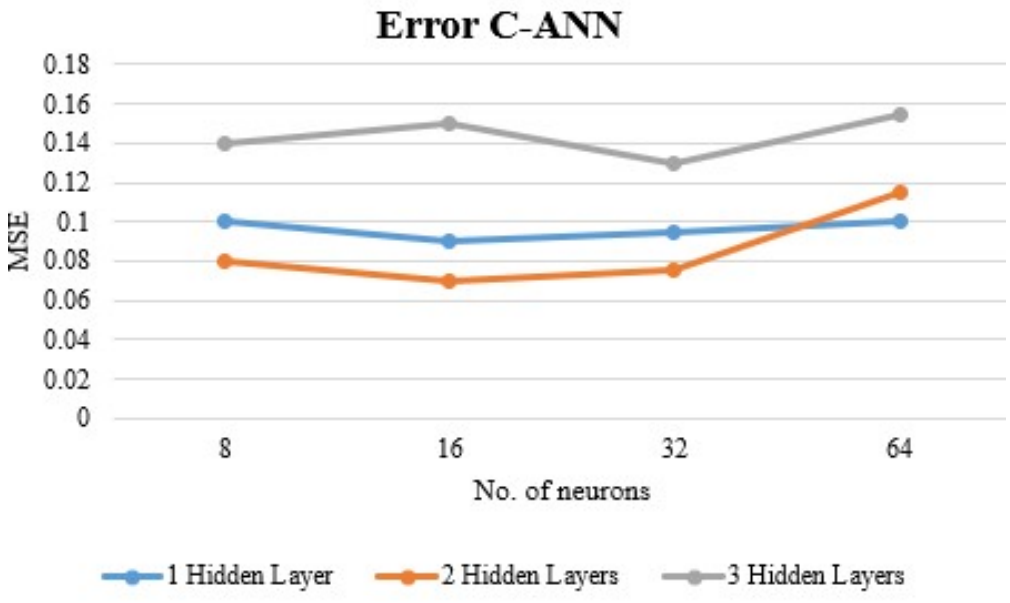


Figure 6. 9: Mean Square error for C-ANN

From figure 6.8 and 6.9 it can be observed that an architecture with 3 hidden layers each with 64 neurons gives us maximum accuracy, but the Mean Square Error also increases for this

architecture, but for 32 neurons in 3 hiddden layers, the mean square error is the lowest for the second highest accuracy.

The C-ANN architecture selected has 32 neurons each for 3 hidden layers as shown in the MATLAB model shown in figure 6.10.

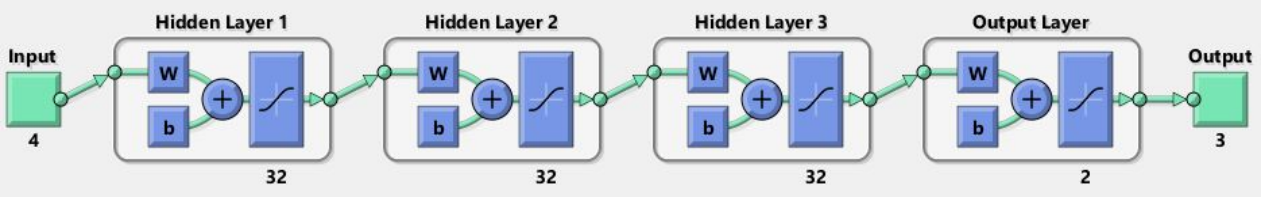


Figure 6. 10: C-ANN structure

6.4.1 Embedded Design for the Complete Water Quality Classification Device

The Embedded Design employs Arduino Uno and Raspberry Pi boards. The Arduino Uno is for digitizing the pH and ORP readings. The Raspberry Pi is used for the data augmentation and classification.

The A-ANN python code takes pH and ORP inputs and predicts the values of DO and EC. The python program for C-ANN is coded, which takes four inputs – pH, ORP, DO, and EC, and classifies the water sample into one of the three categories – potable, agricultural, and wastewater.

Both python codes are given in Appendix C. The Block diagram for the Embedded system-based design is shown in Figure 6.11. The results of the Embedded System-based design are presented in Section 6.5.

6.4.2 ASIC Design for the Complete Water Quality Classification Device

The ASIC based design involves the classification ANN as described in Chapter 4. The ANN architecture has been kept the same for both the Embedded System based design and the ASIC based design. Figure 6.12 shows the block diagram for the ASIC based design.

The A-ANN Verilog code takes pH and ORP inputs and predicts the values of DO and EC. The Verilog code for C-ANN takes four inputs – pH, ORP, DO, and EC, and classifies the water sample into one of the three categories – potable, agricultural, and wastewater.

Both Verilog codes are given in appendix C.

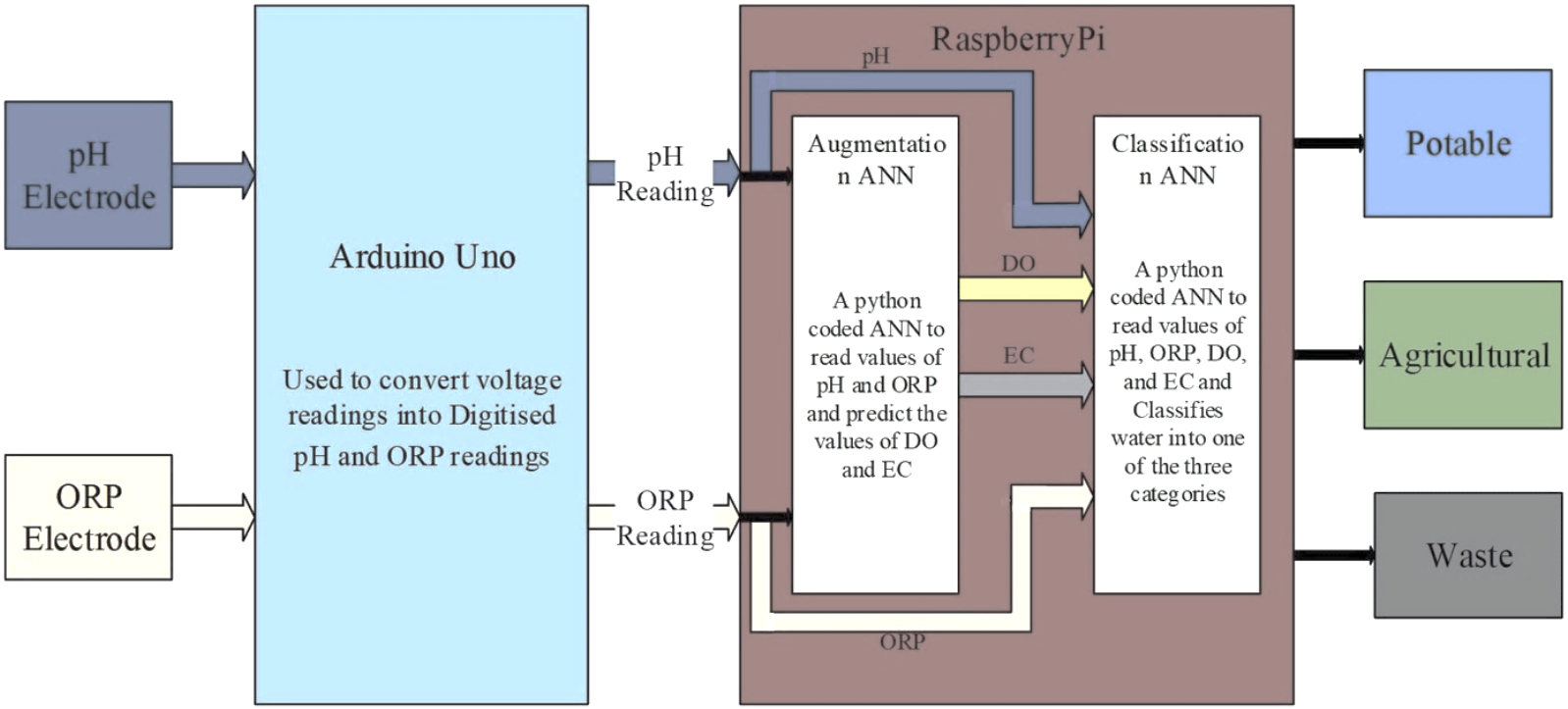


Figure 6. 11: Block diagram of complete Device using Embedded System Design

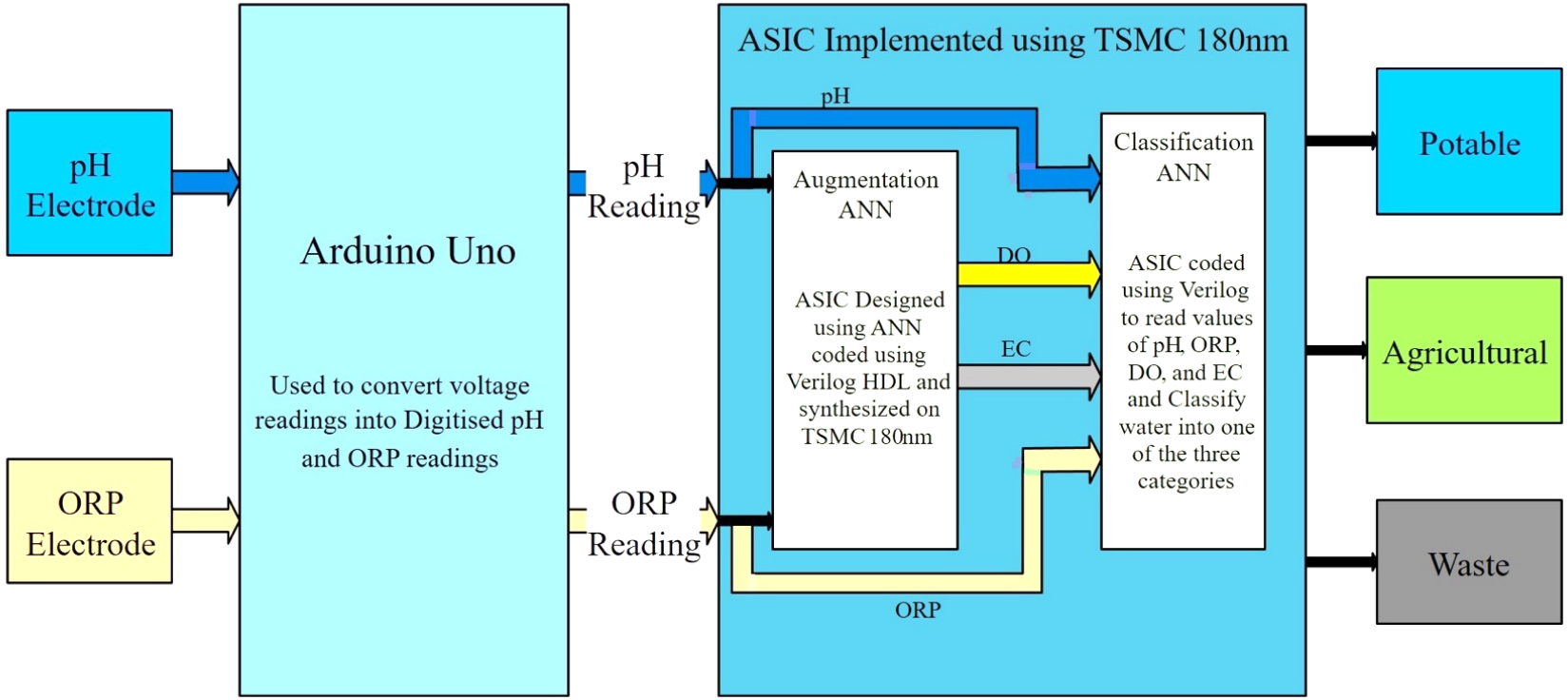


Figure 6. 12: Block diagram of complete Device using ASIC Design Approach

The ASIC Design is synthesized using TSMC 180nm standard cell library using Cadence RTL Encounter. The implementation results are presented in Section 6.5.

6.5 Results and Validation of Complete Water Quality Classification Device

The results of the complete Water Quality Classification device are presented in this section and observations are made.

Section 6.5.1 presents the prediction accuracy of A-ANN for both Embedded and ASIC approaches.

Section 6.5.2 presents the classification accuracy of C-ANN for both Embedded and ASIC approaches.

Section 6.5.3 presents the ASIC Power, Resource utilization, and critical path delay.

Section 6.5.4 presents the Cost comparison of Embedded and VLSI Water Quality Classification Devices with a standard Atlas Scientific Kit.

6.5.1 Results of Prediction Accuracy of A-ANN for both Embedded and ASIC approaches.

A total of 14 training functions are tested. From these 14 functions, only one training function (trainLM), has finished the work of regression plot and error plots. Using the Levenberg-Marquardt training function and a sigmoidal activation function (logistic function), A-ANN with 2 hidden layers and 16 neurons in each layer was optimised (logistic function).

Figures 6.13 and 6.14 exhibit A-ANN's DO and EC response plots. It is observed that DO and EC have above 97% accuracy. A-ANN architecture yields 0.98 R2 at 0.00232 RMSE.

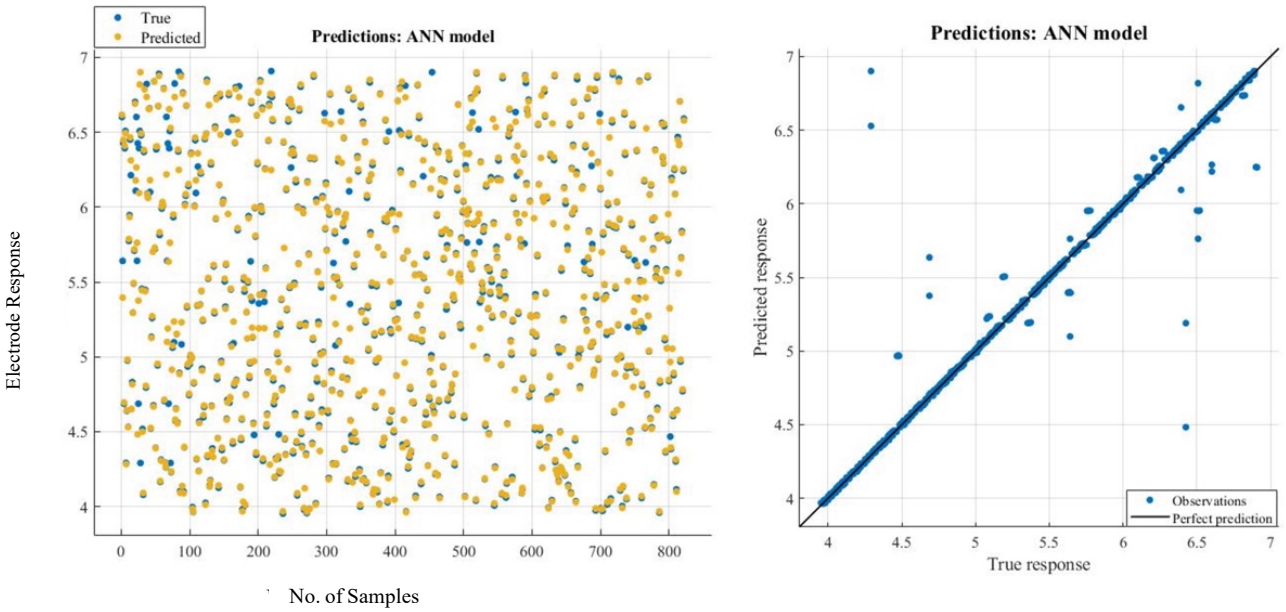


Figure 6. 13: a) Response plot of A-ANN, b) Actual vs. predicted DO value using A-ANN

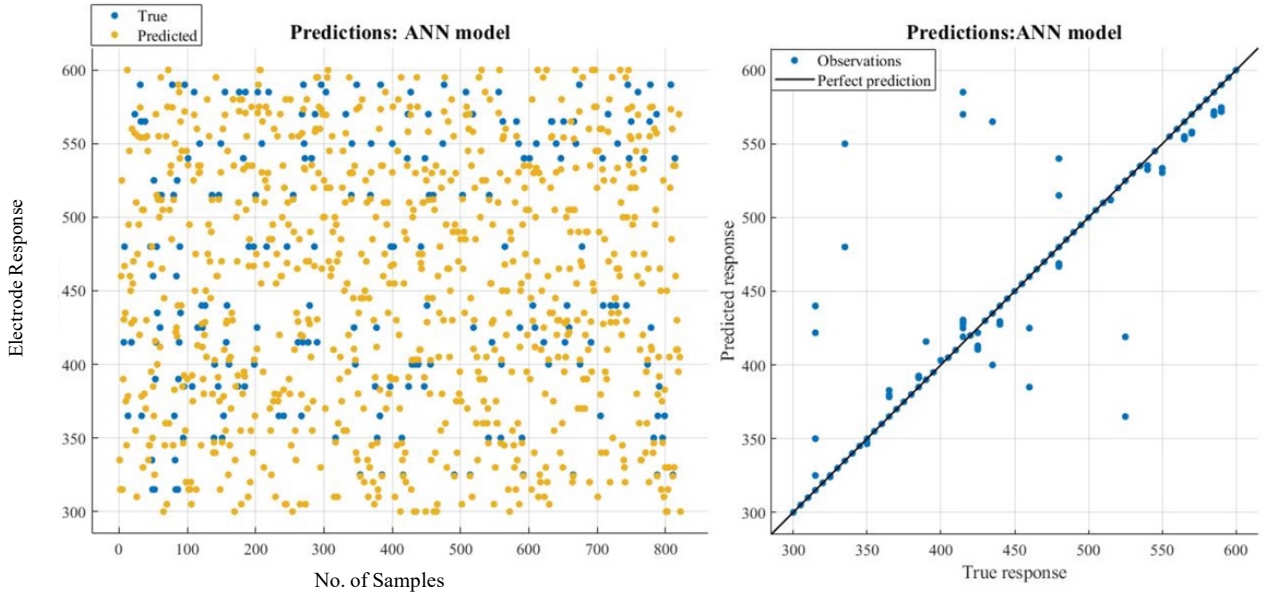


Figure 6. 14: a) Response plot of A-ANN, b) Actual vs. predicted EC value using A-ANN.

Table 6.1 compares the predicted values of 15 water samples with actual laboratory-measured values. The comparison validates the accuracy of 97%.

Table 6. 1: Validation of proposed device for real-time water quality measurement

Sample No	DO (mg/L) (Experimental)	DO (mg/L) (Predicted using A-ANN)	EC ($\mu S/cm$) (Experimental)	EC ($\mu S/cm$) (Predicted Using A-ANN)
1	9.36	9.3	1777	1745
2	9.32	9.4	1407	1398
3	9.35	9.5	912	918
4	9.36	9.3	1450	1540
5	3.81	3.3	1640	1640
6	7.36	7.4	928	908
7	6.82	7	1482	1502
8	7.89	7.8	915	915
9	7.13	7.3	1525	1500
10	5.82	6	1225	1325
11	6.34	6.3	1560	1524
12	5.56	6	857	857
13	7.31	7.3	1362	1362
14	5.18	5.2	1090	1000
15	8.13	8.1	1402	1492

6.6 Results of classification accuracy of C-ANN for both Embedded and ASIC approaches

Classification accuracy of C-ANN has been obtained in terms of four parameters - F-Score, Precision, Sensitivity, and Accuracy. These parameters are important because people who don't have access to modern technology or complex water testing kits will be able to determine whether the water is polluted or not, much more efficiently.

The following Equations (5-8) are used to compute the F-Score, Precision, Sensitivity, and Accuracy in Table 6.2:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$F - Score = \frac{2 * Sensitivity * Precision}{Sensitivity + Precision} \tag{8}$$

Figure 6.15 shows that overall accuracy in water classification is >97% in all three categories, averaging at 98%. Thus, confirming a high level of confidence in classification. The 2% error is because more false negatives are predicted than false positives, which is helpful in the case of drinking water. Table 6.2 presents the four parameters – Accuracy, Sensitivity, Precision, and F-score obtained from the confusion matrix of Figure 6.15.

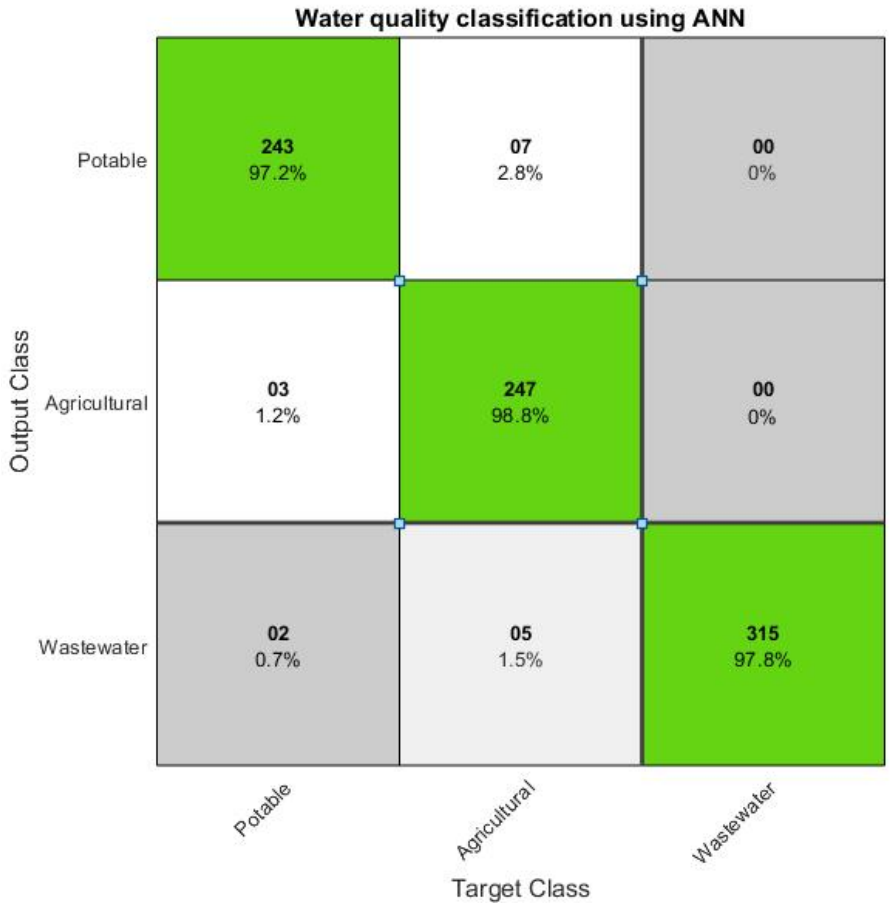


Figure 6. 15: Confusion Matrix for C-ANN

Table 6. 2: The statistical results for performance evaluation

Performance Measure	Results
Accuracy	0.98
Sensitivity	0.96
Precision	0.97
F-Score	0.97

The results obtained for both approaches are observed to be similar with the same classification accuracy value.

6.6.1 ASIC Power, Resource utilization, and critical path delay

The Complete System was also implemented using the ASIC design methodology using a semi-custom VLSI design method. The two ANNs, A-ANN and C-ANN, were coded using Verilog and synthesized using TSMC 180nm Standard Cell Library on Cadence RTL Encounter. The results of the synthesis are presented in Table 6.3.

Table 6. 3: ASIC Implementation Results of Complete Design

Parameter	Values
Power	139.4 mW
Area	3.3mm^2
Standard Cell Units	443648
Critical Path Delay	0.29ms

6.6.2 Cost comparison of Embedded and VLSI Water Quality Classification Device with standard Water Testing Atlas Scientific Kit

Table 6.4 shows the cost comparison of the proposed device with Atlas Scientific standard Electrode with Arduino Uno based Water Testing Kit. It is observed that the proposed device has cost reduction by 92% while achieving 98% accuracy.

Table 6. 4: Cost comparison of the conventional and proposed device

Component Name	Measured Parameter	Cost (Conventional Atlas Scientific Kit)	Cost (Proposed Embedded WQC Device)
Atlas Scientific DO Probe	Dissolved Oxygen	INR 21,240 [15]	NA
Atlas Scientific DO Sensor	Dissolved Oxygen	INR 4,299 [15]	NA
Aquasol ORP Electrode	Oxidation-Reduction Potential	INR 1200	INR 1200
Atlas Scientific ORP Sensor	Oxidation-Reduction Potential	INR 3,739 [15]	NA
Aquasol pH Electrode	pH	INR 900	INR 900
Atlas Scientific pH Sensor	pH	INR 3,739 [15]	NA
Atlas Scientific EC Electrode	Electrical Conductivity	INR 11,200 [15]	NA
Atlas Scientific EC Sensor	Electrical Conductivity	INR 5,600 [15]	NA
Battery Pack		INR 1,000	INR 1,000
Memory Card		NA	INR 300
Multiplexer Board		INR 11,869 [15]	NA
Arduino Uno Board		INR 330	INR 330
Raspberry Pi 3 Board		NA	INR 3,000
Total		**INR 75,116/-**	**INR 6,730/-**

VLSI (ASIC) design approach reduces the cost of production for mass-produced devices. So, ASIC approach is most suited when there is a requirement for mass production of the device, even though the embedded approach gives faster time-to-market.

Performance comparison –

The Atlas Scientific Laboratory kit is a parametric measurement kit which only measures the Water Quality Classification parameters and does not classify the water sample. The output of the A-ANN has been measured against the readings given by the Atlas Scientific kit, as shown in Table 6.1.

6.7 Conclusion

From performance observations made in Section 6.5, the following conclusions are drawn regarding hardware implementations of complete Water Quality Classification Device using Data Augmentation:

- A Multi-Layer Perceptron architecture with 3 hidden layers, each with 32 neurons, gives us optimum accuracy and the least Mean Square Error for both Augmentation ANN and Classification ANN implementation.
- The hardware implementation of the Augmentation ANN design achieves 97% accuracy and yields an R^2 of 0.98 at 0.00232 Root Mean Square Error in the prediction of Dissolved Oxygen and Electrical conductivity using pH and ORP input data.
- Hardware implementation of Classification ANN achieves classification accuracy >97% in all three categories, averaging at 98%, with a high Sensitivity of 0.96, a precision of 0.97, and an F-score value of 0.97.
- The A-ANN and C-ANN performance results obtained for both the approaches, Embedded and ASIC, are found to be similar. Also, the performance is identical to the standard Atlas Scientific lab kit for water testing.
- The proposed portable embedded Water Quality Classification device reduces the cost by 92%, while achieving 98% accuracy as compared to the Atlas Scientific lab testing kit.
- ASICs provide a much cheaper method for mass production of devices as compared to Embedded systems.

6.8 References

[1] R. T. Wilkin, M. S. McNeil, C. J. Adair and J. T. Wilson, "Field Measurement of Dissolved Oxygen: A Comparison of Methods," *Groundwater Monitoring and Remediation,* vol. 21, no. 4, pp. 124 - 132, 2007.

[2] A. Patulea, N. Baran and I. M. Calusaru, "Measurements of Dissolved Oxygen Concentration in Stationary Water," *World Environment,* vol. 2, no. 5, pp. 104 - 109, 2012.

[3] R. G. Jones, "Measurements of the electrical conductivity of water," *IEE Proceedings - Science, Measurement, and Technology,* vol. 149, no. 6, pp. 320 - 322, 2002.

[4] J. Kim, D. Seo, M. Jang and J. Kim, "Augmentation of limited input data using an artificial neural network method to improve the accuracy of water quality modeling in a large lake," *Journal of Hydrology,* vol. 602, no. 126817, 2021.

[5] B. He and K. Takase, "Application of the Artificial Neural Network Method to Estimate the Missing Hydrologic Data," *J. Japan Soc. Hydrol. & Water Resour.,* vol. 19, no. 4, pp. 249 - 257, 2006.

[6] R. M. Hirsch, D. L. Moyer and S. A. Archfield, "Weighted Regressions on Time, Discharge, and Season (WRTDS), with an Application to Chesapeake Bay River Inputs," *Journal of the American Water Resources Association (JAWRA),* vol. 46, no. 5, pp. 857 - 880, 2010.

[7] A. Kaitin, D. D. Giudice, N. S. Hall, H. W. Paerl and D. R. Obenour, "Simulating algal dynamics within a Bayesian framework to evaluate controls on estuary productivity," *Ecological Modelling,* vol. 447, no. Article No. 109497, 2021.

[8] S. L. Neitsch, J. G. Arnold, J. R. Kiniry and J. R. Williams, "Soil and water assessment tool theoretical documentation version 2009," Texas Water Resources Institute (2011), Temple, Texas, 2011.

[9] B. R. Bicknell, J. C. Imhoff, J. L. Kittle Jr., A. S. Donigian Jr. and R. C. Johanson, "Hydrological simulation program—FORTRAN user's manual for version 11.," Environmental Protection Agency Report, No. EPA/600/R-97/080, US Environmental Protection Agency, Athens, Ga (1997), Athens, GA, 1997.

[10] L. A. Rossman, "Storm water management model user's manual, version 5.0.," National Risk Management Research Laboratory, Office of Research and Development, US Environmental Protection Agency., Ohio, 2010.

[11] E. B. Daniel, J. V. Camp, E. J. LeBoeuf, J. R. Penrod, J. P. Dobbins and M. D. Abkowitz, "Watershed modeling and its applications: A state-of-the-art review.," *The Open Hydrology Journal,* vol. 5, no. 1, pp. 26 - 50, 2011.

[12] H. R. Maier and G. C. Dandy, "Application of artificial neural networks to forecasting of surface water quality variables: issues, applications and challenges," in *Artificial Neural Networks in Hydrology. Water Science and Technology Library, , vol 36*, Springer, 2000, p. 287–309.

[13] Y. Chen, L. Song, Y. Liu, L. Yang and D. Ling, "A Review of the Artificial Neural Network Models for Water Quality Prediction," *Applied Sciences,* vol. 10, no. 17, p. 5776, 2020.

[14] M. Paliwal and U. A. Kumar, "Neural networks and statistical techniques: A review of applications," *Expert Systems with Applications,* vol. 36, no. 1, pp. 2 - 17, 2009.

[15] Atlas Scientific, "Atlas Scientific | Environmental Robotics," Atlas Scientific, [Online]. Available: https://atlas-scientific.com/#. [Accessed 01 March 2021].

[16] Arduino Inc., "Arduio - Products," Arduino incorporated, [Online]. Available: https://www.arduino.cc/en/Main/Products. [Accessed 04 March 2021].

Chapter 7

7. Conclusions and Future Work

This chapter summarises the conclusions drawn throughout the study and hardware implementation of a portable smart Water Quality Classification Device using Data Augmentation. Also paves the path for future work based on the design choices made in this study and suggests new design enhancements.

7.1 Conclusions

The thesis presents a digital hardware implementation of an Artificial Neural Network for Water Quality Classification applications. The implementation is required to meet the challenges of reduced size, reduced cost, portability, and reduced power consumption while achieving accuracy comparable to the standard Atlas Scientific Testing Kit. The device is designed for in-situ water quality measurement and classification using Artificial Intelligence with high accuracy that can be used by people with limited to no literacy.

The proposed device uses 2 ANNs to augment water quality parameters and make decisions regarding water quality classification, thus eliminating the need for human expertise and laboratory testing. The device has been implemented using Embedded systems as well as the ASIC approach. The Embedded systems approach is suitable for short development time with high repairability, whereas the ASIC approach leads to compact design with higher reliability.

The important design conclusions regarding the proposed Water Quality Classification device obtained in the present work are discussed below:

- Cost Reduction - ANN-based data augmentation has been used in the design for both approaches to lower the cost of Water Quality Classification by removing the need of expensive sensors and electrodes for Dissolved Oxygen and Electrical Conductivity. The Embedded System design approach has led to a 92% cost reduction as compared to to standard Atlas Scientific Lab kit.
- In the Embedded Approach, the power consumption is controlled by the power required to drive the two embedded boards. For Raspberry Pi, the power consumption is 2.1 W, and for Arduino Uno is 0.5W. The time to generate results is also dependent on the baud rates of Arduino Uno and Raspberry Pi, which is 1s, for the design implemented.

The ASIC approach reduces the power consumption of the total device to 0.3mW and reduces the maximum time to process a result to 0.3 ms.

- For the proposed hardware implementation, Multi-Layer Perceptron architecture with 3 hidden layers with 32 neurons gives us optimum accuracy and the least Mean Square Error for both Augmentation ANN and Classification ANN implementation.
- The Sigmoid activation function is suitable as the output range is between 0 and 1 which suits the probability prediction. A non-linear approximation of the Sigmoid function is 77% power efficient and utilizes 38.5% lesser hardware resources with a 68.3% faster Critical Delay path than the Padé approximation in the IEEE 754 floating point representation system.
- ASIC implementation of the Nonlinear approximated Sigmoid function consumes lesser power by at least 3 orders of magnitude than FPGA implementation in IEEE 754.
- The proposed ANN implementation using Parameterised Posit floating point representation is 50% more power efficient using 50% fewer resources and is 13% faster than the IEEE 754 floating point representation.
- The use of data Augmentation ANN in this design helps in reducing the cost while achieving 97% prediction accuracy and yielding an R^2 of 0.98 at 0.00232 Root Mean Square Error in the prediction of Dissolved Oxygen and Electrical conductivity using pH and ORP input data.
- Hardware implementation of Classification ANN achieves a high classification accuracy above 97% in all three categories, averaging at 98%, with a high Sensitivity of 0.96, precision of 0.97, and an F-score value of 0.97.
- The A-ANN and C-ANN accuracy, sensitivity, and precision results obtained for both the approaches, Embedded and ASIC, are found to be similar.
- A reduced architecture of the design has been implemented on an FPGA board to test the functionality of the entire design prior to the complete implementation using the ASIC approach. The complete design has not been implemented using an FPGA board because it requires a large resource count (> 235,000) board(s), which are very expensive, defeating the low-cost objective of the research.

The novelty of the present work lies in i) the selection of architecture, activation function, and use of ANN-based data augmentation, and ii) the use of the Posit number system with parameterization. To the best of our knowledge, these features in hardware implementation have been used for the first time.

Analysis-based selection of simple ANN architecture, suitable activation function for a compact design to reduce cost. The method of ANN-based data augmentation used in this thesis has further reduced the cost by eliminating the need for expensive sensors yet obtaining comparable accuracy.

Using Posit Floating point representation system to improve accuracy with less computation and less hardware resource requirement. This has increased the speed of operation. To implement Posit on hardware, the biggest challenge is the variable bit width of the components of Posits. So, the technique of parametrization of Posits has been used to address this problem. The Parameterization process has enabled the flexibility of the Posit representation to be adapted as per the requirement making it application specific.

7.2 Future Direction

- In future work, more parameters like geographical information, topological information, and weather parameters, can be added to the input vectors of Augmentation ANN and Classification ANN. Augmentation ANN can also be used to predict biological and chemical contaminants based on their relationship with electrochemical parameters. This would make the device usable for a wider range of water sources, geographical locations, and wider populations within India and across the globe without a drastic increase in cost.
- In the ANN architecture field, the device has been implemented using first-order Neural Networks. More advanced Network topologies like Constructive Neural Networks and Spiking Neural Networks can be adopted to increase the accuracy and efficiency of the device.
- The design methodology can be used and customized further for the development of classification-based applications such as face recognition, air quality indexing, and other real-life applications at a low cost.
- The ASIC design approach can be further developed with IO planning and floor planning tools for final IC fabrication.
- The ASIC design was synthesized using TSMC 180nm standard cell library. The Si area and power consumption can be further reduced, and the speed of operation can be improved by using smaller technology nodes for standard cell libraries.

- The FET-based sensing technologies can be used in conjunction with the ASIC to implement the whole system in a Lab-on-Chip design approach. `

- Parameterization of Posit paves the path of further research on how the flexibility of Posit representation can be bounded and adapted to different applications, thus, maximizing accuracy with minimal increase in hardware resource requirement.

In comparison to existing Patented devices CN201343453Y (D1) [1] and CN210037776U (D2) [2], the proposed device functions independently and does not require wifi connectivity. The proposed handheld device is location independent which tests water quality without a buoy or carrier vehicle. It improves on previous research work/patented devices by adding portability, real-time monitoring, decreased reliance on external factors, use of artificial neural network (ANN) technology, customizable functionality, energy efficiency, rural accessibility, environmental awareness, cost-effectiveness, autonomous operation, and adaptability through artificial intelligence.

7.3 References

[1] X. Z. W. Y. X. Chengbin, “Automatic continuous measuring and controlling instrument for ultraviolet-oxidized water online temperature/electric conductance/total organic carbon”. China Patent CN201343453Y, 2009.

[2] Hadano, “Water quality monitoring system”. China Patent CN210037776U, 2019.

A. Appendix A

The data values of the four parameters pH, ORP, DO, and EC are presented here. 100 samples out of the complete 1806 values are presented here.

A.1. pH Data

Table A. 5: pH measurement comparison against standard devices

S. No.	pH Proposed Device Sensor	pH Atlas Scientific electrode	pH Labtornics LT-59
1	8.5	8.6	8.6
2	8.0	7.8	7.8
3	7.5	7.4	7.4
4	7.0	7.0	7.0
5	6.8	6.8	6.9
6	7.5	7.3	7.3
7	6.5	6.5	6.5
8	7	7	7
9	6.8	6.8	6.8
10	6.6	6.6	6.5
11	5.7	5.7	5.7
12	8	8	8
13	6.5	6.2	6.2
14	6.1	6.1	6.1
15	7	7	7
16	6	6.3	6.3
17	6.5	6.5	6.5
18	6.3	6.3	6.3
19	7	7	7
20	6.8	6.8	6.8
21	8	8	8
22	7	7	7
23	7	7.4	7.4
24	8	8	8
25	6	6	6

26	6.3	6.3	6.3
27	8.5	8.5	8.5
28	8	8	8
29	5.8	5.8	5.8
30	7	7	7.1
31	5.5	5.5	5.5
32	7	7	7
33	8.5	8.5	8.5
34	7.5	7.5	7.5
35	6.8	6.8	6.8
36	7	7	7
37	8.5	8.2	8.2
38	6.8	6.5	6.5
39	8	8	8
40	6.5	6.5	6.5
41	8.5	8.5	8.5
42	8	8	8
43	8.7	8.5	8.5
44	8	8	8
45	7.7	7.7	7.8
46	6	6	6
47	8	8	8
48	7	7	7
49	6.6	6.6	6.6
50	8	8	8
51	8.8	8.5	8.5
52	8.1	8.1	8.1
53	8.3	8.3	8.3
54	7	7	7
55	7.6	7.3	7.3
56	8	8	8
57	7.7	7.7	7.7
58	7.9	8	8
59	6.6	6.6	6.6

60	6.9	7	7
61	7	7	7.2
62	8	8	8
63	7	7	7
64	6.8	6.8	6.8
65	7.1	7.1	7.1
66	7.5	7.5	7.5
67	7.8	7.8	7.8
68	6.8	7	7
69	7	7	7
70	7.2	7.2	7.2
71	8.6	8.6	8.6
72	8	8	8
73	6.3	6.3	6.3
74	7.5	7.5	7.5
75	6.7	6.7	6.7
76	6.2	6.2	6.2
77	6.9	6.9	6.9
78	6.5	7	7
79	6.5	6.5	6.5
80	7	7	7
81	8.5	8.5	8.5
82	6.3	6.3	6.3
83	7.3	7.3	7.3
84	7.5	7.5	7.5
85	6.6	6.6	6.6
86	7	7	7
87	7.2	7.2	7.2
88	6.2	6.2	6.2
89	7.5	7.5	7.5
90	6.5	6.5	6.5
91	7	7	7
92	7.6	7.6	7.6
93	7.1	7.1	7.1

94	8	8.2	8.2
95	6.8	6.8	6.8
96	7.4	7.4	7.4
97	7.6	7.6	7.6
98	7	7	7
99	6.8	7	7
100	8	8	8

A.1 Oxidation Reduction Potential (ORP) Data

Table A. 6: ORP measurement comparison against standard devices

S. No.	ORP Proposed Device Sensor	ORP Atlas Scientific Electrode	ORP Labtornics LT-59
1	3.01	3.01	3.01
2	0.92	0.92	0.92
3	1.73	1.73	1.73
4	3.28	3.28	3.28
5	3.32	3.32	3.32
6	2.74	2.74	2.74
7	0.03	0.03	0.03
8	4.22	4.22	4.22
9	3.08	3.08	3.08
10	2.53	2.53	2.53
11	0.01	0.01	0.01
12	3.39	3.39	3.39
13	0.04	0.04	0.04
14	0.02	0.02	0.02
15	3.4	3.4	3.4
16	0.15	0.15	0.15
17	0.01	0.01	0.01
18	0.01	0.01	0.01
19	1.46	1.46	1.46
20	1.58	1.58	1.58
21	4.63	4.63	4.63
22	0.43	0.43	0.43
23	0.44	0.44	0.44
24	2.83	2.83	2.83
25	0.01	0.01	0.01
26	0.01	0.01	0.01
27	2.89	2.89	2.89
28	0.07	0.07	0.07

29	0.01	0.01	0.01
30	7.11	7.11	7.11
31	0.09	0.09	0.09
32	0.06	0.06	0.06
33	3.33	3.33	3.33
34	3.28	3.28	3.28
35	0.04	0.04	0.04
36	0.22	0.22	0.22
37	3.22	3.22	3.22
38	0.03	0.03	0.03
39	3.43	3.43	3.43
40	0.13	0.13	0.13
41	2.45	2.45	2.45
42	4.95	4.95	4.95
43	4.95	4.95	4.95
44	7.9	7.9	7.9
45	2.48	2.48	2.48
46	0.05	0.05	0.05
47	4.93	4.93	4.93
48	0.08	0.08	0.08
49	0.03	0.03	0.03
50	3.11	3.11	3.11
51	2.82	2.82	2.82
52	3.12	3.12	3.12
53	3.11	3.11	3.11
54	0.85	0.85	0.85
55	0.91	0.91	0.91
56	0.91	0.91	0.91
57	0.64	0.64	0.64
58	1.22	1.22	1.22
59	0.23	0.23	0.23
60	0.24	0.24	0.24

61	0.23	0.23	0.23
62	0.59	0.59	0.59
63	0.23	0.23	0.23
64	0.24	0.24	0.24
65	0.28	0.28	0.28
66	0.93	0.93	0.93
67	1.55	1.55	1.55
68	0.33	0.33	0.33
69	0.49	0.49	0.49
70	0.24	0.24	0.24
71	1.19	1.19	1.19
72	0.85	0.85	0.85
73	0.24	0.24	0.24
74	0.59	0.59	0.59
75	0.23	0.23	0.23
76	0.23	0.23	0.23
77	0.23	0.23	0.23
78	0.25	0.25	0.25
79	0.23	0.23	0.23
80	0.25	0.25	0.25
81	4.34	4.34	4.34
82	0.25	0.25	0.25
83	0.36	0.36	0.36
84	0.64	0.64	0.64
85	0.25	0.25	0.25
86	0.25	0.25	0.25
87	0.23	0.23	0.23
88	0.26	0.26	0.26
89	0.9	0.9	0.9
90	0.42	0.42	0.42
91	0.25	0.25	0.25
92	1.21	1.21	1.21

93	0.24	0.24	0.24
94	1.22	1.22	1.22
95	0.25	0.25	0.25
96	1.08	1.08	1.08
97	0.49	0.49	0.49
98	0.51	0.51	0.51
99	0.27	0.27	0.27
100	0.64	0.64	0.64

A.2 Dissolved Oxygen (DO) Data

Table A. 7: DO measurement comparison against standard devices

S. No.	DO Proposed Device Sensor	DO Atlas Scientific Electrode	DO Labtornics LT-59
1	10	10	10
2	12	11.5	12
3	14.5	14.5	14.5
4	12.5	12.5	12
5	12	12	12
6	16	16	16
7	15	15	14.5
8	14.3	14	14.3
9	10.3	10.3	10.3
10	10.4	10.4	10.4
11	13	13	13
12	16	16	16
13	15.5	15.5	15.5
14	13	13	13
15	12.5	12.5	12.5
16	13	13	13
17	11.5	11.5	11.5
18	9	10	10
19	1.7	1.7	1.7
20	10.8	10.8	10.8
21	12	12	12
22	12	12	12
23	9.5	9.5	9.5
24	11	11	11
25	9	9	9
26	12	12	12
27	10	10	10
28	8.5	8.5	8.5
29	7.5	7.5	7.5
30	8	8	8

31	9.5	9.5	9.5
32	10	10	10
33	10.8	11	10.8
34	10.5	10.5	10.5
35	9.5	9.5	9.5
36	9.7	9.7	9.7
37	9.8	9.8	10
38	9	9	9
39	7	7	7
40	9	9	9
41	10	10	10
42	10.2	10.2	10
43	8.8	8.8	8.8
44	4.5	4.5	4.5
45	9.8	9.8	9.8
46	8.5	8.5	8.5
47	5.5	5.5	5.5
48	8.2	8.2	8.2
49	7.5	7.5	7.5
50	8.2	8.2	8.2
51	7.5	7.5	7.5
52	8.5	8.5	8.5
53	7	7	7
54	8.5	8.5	8.5
55	9	9	9
56	9	9	9
57	8.5	8.5	8.5
58	9	9	9
59	8.5	8.5	8.5
60	9.5	9.5	9.5
61	8.5	9	8.5
62	7	6	6
63	10.2	10.2	10.2
64	9	9	9

65	7.8	7.8	7.8
66	7.5	7.5	7.5
67	6	6	6
68	7	7	7
69	9	9	9
70	9	9	9
71	7.3	7.3	7.3
72	6.5	6.5	6.5
73	8.5	8.5	8.5
74	8.2	8	8
75	7.8	7.8	7.8
76	8.6	8.6	8.6
77	7.4	7.4	7.4
78	8	8	8
79	7.5	7.5	7.5
80	8	8	8
81	7	7	7
82	7.6	7.6	7.6
83	7.4	7.4	7.4
84	9.5	9.5	9.5
85	9	9	9
86	7.5	7.5	7.5
87	9.2	9.2	9.2
88	9.5	9.5	9.5
89	8.3	8.3	8.3
90	8.7	8.7	8.7
91	9.6	9.6	9.6
92	6.4	6.4	6.4
93	10	10	10
94	7	7	7
95	10.5	10.5	10.5
96	8	8	8
97	9.5	9.5	9.5
98	8	8	8

99	8	8	8
100	7.5	7.5	7.5

A.3 Electrical Conductivity (EC) Data

Table A. 8: EC measurement comparison against standard devices

S. No.	EC Proposed Device Sensor	EC Atlas Scientific Electrode	EC Labtornics LT-59
1	281	281	281
2	294	294	294
3	277	277	277
4	292	292	292
5	302	302	302
6	294	294	294
7	298	298	298
8	308	308	308
9	277	277	277
10	278	278	278
11	270	270	270
12	254	254	254
13	293	293	293
14	299	299	299
15	283	283	283
16	298	298	298
17	291	291	291
18	295	295	295
19	288	288	288
20	257	257	257
21	265	265	265
22	271	271	271
23	284	284	284
24	259	259	259
25	211	211	211
26	235	235	235
27	215	215	215
28	245	245	245

29	264	264	264
30	273	273	273
31	262	262	262
32	263	263	263
33	252	252	252
34	266	266	266
35	269	269	269
36	273	273	273
37	279	279	279
38	275	275	275
39	267	267	267
40	277	277	277
41	264	264	264
42	263	263	263
43	268	268	268
44	254	254	254
45	259	259	259
46	275	275	275
47	273	273	273
48	280	280	280
49	284	284	284
50	267	267	267
51	263	263	263
52	269	269	269
53	248	248	248
54	255	255	255
55	247	247	247
56	257	257	257
57	260	260	260
58	273	273	273
59	265	265	265
60	266	266	266

61	271	271	271
62	253	253	253
63	264	264	264
64	281	281	281
65	258	258	258
66	253	253	253
67	258	258	258
68	265	265	265
69	255	255	255
70	270	270	270
71	247	247	247
72	238	238	238
73	278	278	278
74	269	269	269
75	270	270	270
76	252	252	252
77	271	271	271
78	269	269	269
79	270	270	270
80	279	279	279
81	238	238	238
82	265	265	265
83	263	263	263
84	245	245	245
85	267	267	267
86	278	278	278
87	274	274	274
88	269	269	269
89	265	265	265
90	282	282	282
91	281	281	281
92	274	274	274

93	272	272	272
94	256	256	256
95	263	263	263
96	276	276	276
97	258	258	258
98	266	266	266
99	271	271	271
100	297	297	297

A.4 Variation of DO and EC measurement in proposed device against Atlas Scientific kit.

Table A. 9: Validation of proposed device for real-time water quality measurement

S No	DO		EC	
	Atlas Scientific kit	Measured using Proposed device	Atlas Scientific Kit	Measured using Proposed device
1	9.36	9.3	1777	1745
2	9.32	9.3	1407	1407
3	9.35	9.3	912	912
4	9.36	9.3	1450	1450
5	3.81	3.8	1640	1640
6	7.36	7.3	928	928
7	6.82	6.8	1482	1482
8	7.89	7.8	915	915
9	7.13	7.1	1525	1525
10	5.82	5.8	1225	1225
11	6.34	6.3	1560	1524
12	5.56	5.5	857	857
13	7.31	7.3	1362	1362
14	5.18	5.1	1090	1090
15	8.13	8.1	1402	1402
16	8.37	8.3	1488	1474
17	5.13	5.1	1332	1332
18	7.27	7.2	225	225
19	5.82	5.8	1175	1175
20	5.57	5.5	1036	1036

21	6.15	6.1	1082	1082
22	4.51	4.5	1190	1190
23	8.37	8.3	1180	1180
24	8.54	8.5	577	577
25	5.32	5.3	1322	1322
26	8.72	8.7	1321	1321
27	5.78	5.7	1190	1190
28	7.34	7.3	1126	1126
29	5.76	5.7	1093	1093
30	5.03	5.0	340	340
31	4.41	4.4	550	550
32	6.21	6.2	405	405
33	5.24	5.2	390	390
34	5.46	5.5	305	305
35	4.99	5.0	435	435
36	4.33	4.3	420	420
37	4.01	4.0	555	555
38	6.74	6.7	350	350
39	5.95	6.0	345	345
40	4.26	4.3	360	360
41	5.80	5.8	450	450
42	4.07	4.1	550	550
43	5.06	5.1	490	490
44	5.84	5.8	450	450
45	6.71	6.7	410	410
46	4.97	5.0	560	560
47	6.76	6.7	545	545

48	4.69	4.6	475	475
49	4.91	4.9	505	505
50	6.78	6.7	515	515

A.5 All Parameter Measurements Using the Proposed Device

Table A.6: Measurement of all 4 parameters using proposed device

S. No.	pH Proposed Device Sensor	ORP Proposed Device Sensor	DO Proposed Device Sensor	EC Proposed Device Sensor
1	8.5	3.01	10	281
2	8	0.92	12	294
3	7.5	1.73	14.5	277
4	7	3.28	12.5	292
5	6.8	3.32	12	302
6	7.5	2.74	16	294
7	6.5	0.03	15	298
8	7	4.22	14.3	308
9	6.8	3.08	10.3	277
10	6.6	2.53	10.4	278
11	5.7	0.01	13	270
12	8	3.39	16	254
13	6.5	0.04	15.5	293
14	6.1	0.02	13	299
15	7	3.4	12.5	283
16	6	0.15	13	298
17	6.5	0.01	11.5	291
18	6.3	0.01	9	295
19	7	1.46	1.7	288
20	6.8	1.58	10.8	257
21	8	4.63	12	265
22	7	0.43	12	271

23	7	0.44	9.5	284
24	8	2.83	11	259
25	6	0.01	9	211
26	6.3	0.01	12	235
27	8.5	2.89	10	215
28	8	0.07	8.5	245
29	5.8	0.01	7.5	264
30	7	7.11	8	273
31	5.5	0.09	9.5	262
32	7	0.06	10	263
33	8.5	3.33	10.8	252
34	7.5	3.28	10.5	266
35	6.8	0.04	9.5	269
36	7	0.22	9.7	273
37	8.5	3.22	9.8	279
38	6.8	0.03	9	275
39	8	3.43	7	267
40	6.5	0.13	9	277
41	8.5	2.45	10	264
42	8	4.95	10.2	263
43	8.7	4.95	8.8	268
44	8	7.9	4.5	254
45	7.7	2.48	9.8	259
46	6	0.05	8.5	275
47	8	4.93	5.5	273
48	7	0.08	8.2	280
49	6.6	0.03	7.5	284

50	8	3.11	8.2	267
51	8.8	2.82	7.5	263
52	8.1	3.12	8.5	269
53	8.3	3.11	7	248
54	7	0.85	8.5	255
55	7.6	0.91	9	247
56	8	0.91	9	257
57	7.7	0.64	8.5	260
58	7.9	1.22	9	273
59	6.6	0.23	8.5	265
60	6.9	0.24	9.5	266
61	7	0.23	8.5	271
62	8	0.59	7	253
63	7	0.23	10.2	264
64	6.8	0.24	9	281
65	7.1	0.28	7.8	258
66	7.5	0.93	7.5	253
67	7.8	1.55	6	258
68	6.8	0.33	7	265
69	7	0.49	9	255
70	7.2	0.24	9	270
71	8.6	1.19	7.3	247
72	8	0.85	6.5	238
73	6.3	0.24	8.5	278
74	7.5	0.59	8.2	269
75	6.7	0.23	7.8	270
76	6.2	0.23	8.6	252

77	6.9	0.23	7.4	271
78	6.5	0.25	8	269
79	6.5	0.23	7.5	270
80	7	0.25	8	279
81	8.5	4.34	7	238
82	6.3	0.25	7.6	265
83	7.3	0.36	7.4	263
84	7.5	0.64	9.5	245
85	6.6	0.25	9	267
86	7	0.25	7.5	278
87	7.2	0.23	9.2	274
88	6.2	0.26	9.5	269
89	7.5	0.9	8.3	265
90	6.5	0.42	8.7	282
91	7	0.25	9.6	281
92	7.6	1.21	6.4	274
93	7.1	0.24	10	272
94	8	1.22	7	256
95	6.8	0.25	10.5	263
96	7.4	1.08	8	276
97	7.6	0.49	9.5	258
98	7	0.51	8	266
99	6.8	0.27	8	271
100	8	0.64	7.5	297

B. Appendix B

Verilog codes for ANN neuron using Posit representation system, Nonlinear approximation of Sigmoid function, and Padé approximation of exponent function.

B.1. Verilog Code for Posit neuron.

```
`timescale 1ns / 1ps
module neuron_posit(in1, in2, in3, in4, n_out);

    input [31:0] in1, in2, in3, in4;
    output [31:0] n_out;
    wire [15:0] w1, w2, w3, w4;
    wire [15:0] m1, m2, m3, m4;
    wire [15:0] a1, a2, add_out;
    wire start, inf, zero, done;
    wire [15:0] inp1, inp2, inp3, inp4;
    wire [15:0] sig_out;

    FP_to_posit INP1(in1, inp1);
    FP_to_posit INP2(in2, inp2);
    FP_to_posit INP3(in3, inp3);
    FP_to_posit INP4(in4, inp4);

    ///multiplying input with weights
    posit_mult M1(inp1, w1, start, m1, inf, zero, done);
    posit_mult M2(inp2, w2, start, m2, inf, zero, done);
    posit_mult M3(inp3, w3, start, m3, inf, zero, done);
    posit_mult M4(inp4, w4, start, m4, inf, zero, done);

    ///adding weighted inputs
    posit_adder A1(m1, m2, start, a1, inf, zero, done);
    posit_adder A2(m3, m4, start, a2, inf, zero, done);
    posit_adder A3(a1, a2, start, add_out, inf, zero, done);

    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];


    Posit_to_FP P2F(sig_out, n_out);


endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
//////////////////Floating Point to Posit Conversion//////////////////////////////
////////////////////////////////////////////////////////////////////////////////

module FP_to_posit(in, out);

function [31:0] log2;
input reg [31:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
                value = value>>1;
        end
endfunction

parameter N = 16;
parameter E = 5;
parameter es = 3;      //ES_max = E-1
parameter M = N-E-1;
parameter BIAS = (2**(E-1))-1;

parameter Bs = log2(N);

input [N-1:0] in;
output [N-1:0] out;

wire s_in = in[N-1];
wire [E-1:0] exp_in = in[N-2:N-1-E];
wire [M-1:0] mant_in = in[M-1:0];
wire zero_in = ~|{exp_in,mant_in};
wire inf_in = &exp_in;

wire [M:0] mant = {|exp_in, mant_in};

wire [N-1:0] LOD_in = {mant,{E{1'b0}}};
wire[Bs-1:0] Lshift;
LOD_N #(.N(N)) uut (.in(LOD_in), .out(Lshift));

wire[N-1:0] mant_tmp;
DSR_left_N_S #(.N(N), .S(Bs)) ls (.a(LOD_in),.b(Lshift),.c(mant_tmp));

wire [E:0] exp = {exp_in[E-1:1], exp_in[0] | (~|exp_in)} - BIAS - Lshift;

//Exponent and Regime Computation
wire [E:0] exp_N = exp[E] ? -exp : exp;
wire [es-1:0] e_o = (exp[E] & |exp_N[es-1:0]) ? exp[es-1:0] : exp_N[es-1:0];
```

```
wire [E-es-1:0] r_o = (~exp[E] || (exp[E] & |exp_N[es-1:0])) ? {{Bs{1'b0}},exp_N[E-1:es]} +
1'b1 : {{Bs{1'b0}},exp_N[E-1:es]};

//Exponent and Mantissa Packing
wire [2*N-1:0]tmp_o = { {N{~exp[E]}}, exp[E], e_o, mant_tmp[N-2:es]};

//Including Regime bits in Exponent-Mantissa Packing
wire [2*N-1:0] tmp1_o;
wire [Bs-1:0] diff_b;
generate
       if(E-es > Bs)  assign diff_b = |r_o[E-es-1:Bs] ? {{(Bs-2){1'b1}},2'b01} : r_o[Bs-1:0];
       else           assign diff_b = r_o;
endgenerate
DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(diff_b), .c(tmp1_o));

//Final Output
wire [N-1:0] tmp1_oN = s_in ? -tmp1_o[N-1:0] : tmp1_o[N-1:0];
assign out = inf_in|zero_in|(~mant_tmp[N-1]) ? {inf_in,{N-1{1'b0}}} : {s_in, tmp1_oN[N-1:1]};

endmodule


////////////////////////////////LOD_N/////////////////////////////////////////////////////////

module LOD_N (in, out);

  function [31:0] log2;
    input reg [31:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
       value = value>>1;
    end
  endfunction

parameter N = 64;
parameter S = log2(N);
input [N-1:0] in;
output [S-1:0] out;

wire vld;
LOD #(.N(N)) l1 (in, out, vld);
endmodule


module LOD (in, out, vld);
```

```
  function [31:0] log2;
    input reg [31:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction


parameter N = 64;
parameter S = log2(N);

   input [N-1:0] in;
   output [S-1:0] out;
   output vld;

  generate
    if (N == 2)
      begin
        assign vld = |in;
        assign out = ~in[1] & in[0];
      end
    else if (N & (N-1))
      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
    else
      begin
        wire [S-2:0] out_l, out_h;
        wire out_vl, out_vh;
        LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule


////////////////////////////////////DSR_left_N_S//////////////////////////////


module DSR_left_N_S(a,b,c);
        parameter N=16;
        parameter S=4;
        input [N-1:0] a;
        input [S-1:0] b;
```

```
        output [N-1:0] c;

wire [N-1:0] tmp [S-1:0];
assign tmp[0]  = b[0] ? a << 7'd1  : a;
genvar i;
generate
       for (i=1; i<S; i=i+1)begin:loop_blk
              assign tmp[i] = b[i] ? tmp[i-1] << 2**i : tmp[i-1];
       end
endgenerate
assign c = tmp[S-1];

endmodule

//////////////////////DSR_right_N_S/////////////////////////////////////////////

module DSR_right_N_S(a,b,c);
        parameter N=16;
        parameter S=4;
        input [N-1:0] a;
        input [S-1:0] b;
        output [N-1:0] c;

wire [N-1:0] tmp [S-1:0];
assign tmp[0]  = b[0] ? a >> 7'd1  : a;
genvar i;
generate
       for (i=1; i<S; i=i+1)begin:loop_blk
              assign tmp[i] = b[i] ? tmp[i-1] >> 2**i : tmp[i-1];
       end
endgenerate
assign c = tmp[S-1];

endmodule

////////////////////////////////////////////////////////////////////////////////
//////////////////////Posit Adder///////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

//`include "DSR_right_N_S.v"
//`include "LOD_N.v"
//`include "LZD_N.v"
//`include "DSR_left_N_S.v"
//`include "add_N.v"
//`include "sub_N.v"
//`include "data_extract.v"
```

```
//`include "add_mantovf.v"

module posit_adder (in1, in2, start, out, inf, zero, done);

function [31:0] log2;
input reg [31:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
                value = value>>1;
        end
endfunction

parameter N = 16;      //Posit Word Size
parameter Bs = log2(N);
parameter es = 3;      //Posit Exponent Size

input [N-1:0] in1, in2;
input start;
output [N-1:0] out;
output inf, zero;
output done;

wire start0= start;
wire s1 = in1[N-1];
wire s2 = in2[N-1];
wire zero_tmp1 = |in1[N-2:0];
wire zero_tmp2 = |in2[N-2:0];
wire inf1 = in1[N-1] & (~zero_tmp1),
        inf2 = in2[N-1] & (~zero_tmp2);
wire zero1 = ~(in1[N-1] | zero_tmp1),
        zero2 = ~(in2[N-1] | zero_tmp2);
assign inf = inf1 | inf2,
        zero = zero1 & zero2;

//Data Extraction
wire rc1, rc2;
wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;
wire [es-1:0] e1, e2;
wire [N-es-1:0] mant1, mant2;
wire [N-1:0] xin1 = s1 ? -in1 : in1;
wire [N-1:0] xin2 = s2 ? -in2 : in2;
data_extract  #(.N(N),.es(es))  uut_de1(.in(xin1),  .rc(rc1),  .regime(regime1),  .exp(e1),
.mant(mant1), .Lshift(Lshift1));
data_extract  #(.N(N),.es(es))  uut_de2(.in(xin2),  .rc(rc2),  .regime(regime2),  .exp(e2),
.mant(mant2), .Lshift(Lshift2));
```

```
wire [N-es:0] m1 = {zero_tmp1,mant1},
       m2 = {zero_tmp2,mant2};

//Large Checking and Assignment
wire in1_gt_in2 = xin1[N-2:0] >= xin2[N-2:0] ? 1'b1 : 1'b0;

wire ls = in1_gt_in2 ? s1 : s2;
wire op = s1 ~^ s2;

wire lrc = in1_gt_in2 ? rc1 : rc2;
wire src = in1_gt_in2 ? rc2 : rc1;

wire [Bs-1:0] lr = in1_gt_in2 ? regime1 : regime2;
wire [Bs-1:0] sr = in1_gt_in2 ? regime2 : regime1;

wire [es-1:0] le = in1_gt_in2 ? e1 : e2;
wire [es-1:0] se = in1_gt_in2 ? e2 : e1;

wire [N-es:0] lm = in1_gt_in2 ? m1 : m2;
wire [N-es:0] sm = in1_gt_in2 ? m2 : m1;

//Exponent Difference: Lower Mantissa Right Shift Amount
wire [Bs:0] r_diff11, r_diff12, r_diff2;
sub_N #(.N(Bs)) uut_sub1 (lr, sr, r_diff11);
add_N #(.N(Bs)) uut_add1 (lr, sr, r_diff12);
sub_N #(.N(Bs)) uut_sub2 (sr, lr, r_diff2);
wire [Bs:0] r_diff =  lrc ? (src ? r_diff11 : r_diff12) : r_diff2;

wire [es+Bs+1:0] diff;
sub_N #(.N(es+Bs+1)) uut_sub_diff ({r_diff,le}, {{Bs+1{1'b0}},se}, diff);
wire [Bs-1:0] exp_diff = (|diff[es+Bs:Bs]) ? {Bs{1'b1}} : diff[Bs-1:0];

//DSR Right Shifting of Small Mantissa
wire [N-1:0] DSR_right_in;
generate
       if (es >= 2)
       assign DSR_right_in = {sm,{es-1{1'b0}}};
       else
       assign DSR_right_in = sm;
endgenerate

wire [N-1:0] DSR_right_out;
wire [Bs-1:0] DSR_e_diff  = exp_diff;
DSR_right_N_S #(.N(N), .S(Bs))  dsr1(.a(DSR_right_in), .b(DSR_e_diff), .c(DSR_right_out));

//Mantissa Addition
```

```
wire [N-1:0] add_m_in1;
generate
        if (es >= 2)
        assign add_m_in1 = {lm,{es-1{1'b0}}};
        else
        assign add_m_in1 = lm;
endgenerate

wire [N:0] add_m1, add_m2;
add_N #(.N(N)) uut_add_m1 (add_m_in1, DSR_right_out, add_m1);
sub_N #(.N(N)) uut_sub_m2 (add_m_in1, DSR_right_out, add_m2);
wire [N:0] add_m = op ? add_m1 : add_m2;
wire [1:0] mant_ovf = add_m[N:N-1];

//LOD of mantissa addition result
wire [N-1:0] LOD_in = {(add_m[N] | add_m[N-1]), add_m[N-2:0]};
wire [Bs-1:0] left_shift;
LOD_N #(.N(N)) l2(.in(LOD_in), .out(left_shift));

//DSR Left Shifting of mantissa result
wire [N-1:0] DSR_left_out_t;
DSR_left_N_S #(.N(N), .S(Bs)) dsl1(.a(add_m[N:1]), .b(left_shift), .c(DSR_left_out_t));
wire [N-1:0] DSR_left_out = DSR_left_out_t[N-1] ? DSR_left_out_t[N-1:0] : {DSR_left_out_t[N-
2:0],1'b0};



//Exponent and Regime Computation
wire [Bs:0] lr_N = lrc ? {1'b0,lr} : -{1'b0,lr};
wire [es+Bs+1:0] le_o_tmp, le_o;
sub_N #(.N(es+Bs+1)) sub3 ({lr_N,le}, {{es+1{1'b0}},left_shift}, le_o_tmp);
add_mantovf #(es+Bs+1) uut_add_mantovf (le_o_tmp, mant_ovf[1], le_o);

wire [es+Bs:0] le_oN = le_o[es+Bs] ? -le_o : le_o;
wire [es-1:0] e_o = (le_o[es+Bs] & |le_oN[es-1:0]) ? le_o[es-1:0] : le_oN[es-1:0];
wire [Bs-1:0] r_o = (~le_o[es+Bs] || (le_o[es+Bs] & |le_oN[es-1:0])) ? le_oN[es+Bs-1:es] + 1'b1
: le_oN[es+Bs-1:es];

//Exponent and Mantissa Packing
wire [2*N-1:0]tmp_o = { {N{~le_o[es+Bs]}}, le_o[es+Bs], e_o, DSR_left_out[N-2:es]};
wire [2*N-1:0] tmp1_o;
DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(r_o), .c(tmp1_o));

//Final Output
wire [2*N-1:0] tmp1_oN = ls ? -tmp1_o : tmp1_o;
assign out = inf|zero|(~DSR_left_out[N-1]) ? {inf,{N-1{1'b0}}} : {ls, tmp1_oN[N-1:1]},
        done = start0;
```

```
endmodule

////////////////////////////////////////////////////////////////////////////////
//////////////////////////////Posit Multiplication//////////////////////////////
////////////////////////////////////////////////////////////////////////////////

module posit_mult (in1, in2, start, out, inf, zero, done);

function [31:0] log2;
input reg [31:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
                value = value>>1;
        end
endfunction

parameter N = 16;
parameter Bs = log2(N);
parameter es = 3;

input [N-1:0] in1, in2;
input start;
output [N-1:0] out;
output inf, zero;
output done;

wire start0= start;
wire s1 = in1[N-1];
wire s2 = in2[N-1];
wire zero_tmp1 = |in1[N-2:0];
wire zero_tmp2 = |in2[N-2:0];
wire inf1 = in1[N-1] & (~zero_tmp1),
        inf2 = in2[N-1] & (~zero_tmp2);
wire zero1 = ~(in1[N-1] | zero_tmp1),
        zero2 = ~(in2[N-1] | zero_tmp2);
assign inf = inf1 | inf2,
        zero = zero1 & zero2;

//Data Extraction
wire rc1, rc2;
wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;
wire [es-1:0] e1, e2;
wire [N-es-1:0] mant1, mant2;
wire [N-1:0] xin1 = s1 ? -in1 : in1;
wire [N-1:0] xin2 = s2 ? -in2 : in2;
```

```
data_extract   #(.N(N),.es(es))   uut_de1(.in(xin1),   .rc(rc1),   .regime(regime1),   .exp(e1),
.mant(mant1), .Lshift(Lshift1));

data_extract   #(.N(N),.es(es))   uut_de2(.in(xin2),   .rc(rc2),   .regime(regime2),   .exp(e2),
.mant(mant2), .Lshift(Lshift2));


wire [N-es:0] m1 = {zero_tmp1,mant1},

       m2 = {zero_tmp2,mant2};


//Sign, Exponent and Mantissa Computation

wire mult_s = s1 ^ s2;


wire [2*(N-es)+1:0] mult_m = m1*m2;

wire mult_m_ovf = mult_m[2*(N-es)+1];

wire [2*(N-es)+1:0] mult_mN = ~mult_m_ovf ? mult_m << 1'b1 : mult_m;


wire [Bs+1:0] r1 = rc1 ? {2'b0,regime1} : -regime1;

wire [Bs+1:0] r2 = rc2 ? {2'b0,regime2} : -regime2;

wire [Bs+es+1:0] mult_e  =  {r1, e1} + {r2, e2} + mult_m_ovf;


//Exponent and Regime Computation

wire [es+Bs:0] mult_eN = mult_e[es+Bs+1] ? -mult_e : mult_e;

wire [es-1:0] e_o = (mult_e[es+Bs+1] & |mult_eN[es-1:0]) ? mult_e[es-1:0] : mult_eN[es-1:0];

wire [Bs:0] r_o = (~mult_e[es+Bs+1] || (mult_e[es+Bs+1] & |mult_eN[es-1:0])) ? mult_eN[es+Bs:es]
+ 1'b1 : mult_eN[es+Bs:es];


//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = {{N{~mult_e[es+Bs+1]}},mult_e[es+Bs+1],e_o,mult_mN[2*(N-es):N-es+2]};



//Including Regime bits in Exponent-Mantissa Packing

wire [2*N-1:0] tmp1_o;

DSR_right_N_S #(.N(2*N), .S(Bs+1)) dsr2 (.a(tmp_o), .b(r_o[Bs] ? {Bs{1'b1}} : r_o), .c(tmp1_o));



//Final Output

wire [2*N-1:0] tmp1_oN = mult_s ? -tmp1_o : tmp1_o;

assign out = inf|zero|(~mult_mN[2*(N-es)+1]) ? {inf,{N-1{1'b0}}} : {mult_s, tmp1_oN[N-1:1]},

       done = start0;


endmodule


////////////////////////////////////////////////////////////////////////////////////////////
///////////

//////////////////////////////////////Posit                                                  to
FP//////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////
///////////


module Posit_to_FP (in, out);
```

```
function [31:0] log2;
input reg [31:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
                value = value>>1;
        end
endfunction

parameter N = 16;
parameter E = 5;
parameter es = 3;

parameter M = N-E-1;
parameter BIAS = (2**(E-1))-1;
parameter Bs = log2(N);
parameter EO = E > es+Bs ? E : es+Bs;

input [N-1:0] in;
output [N-1:0] out;

wire s = in[N-1];
wire zero_tmp = |in[N-2:0];
wire inf_in = in[N-1] & (~zero_tmp);
wire zero_in = ~(in[N-1] | zero_tmp);

//Data Extraction
wire rc;
wire [Bs-1:0] rgm, Lshift;
wire [es-1:0] e;
wire [N-es-1:0] mant;
wire [N-1:0] xin = s ? -in : in;
data_extract #(.N(N),.es(es)) uut_de1(.in(xin), .rc(rc), .regime(rgm), .exp(e), .mant(mant),
.Lshift(Lshift));

wire [N-1:0] m = {zero_tmp,mant,{es-1{1'b0}}};

//Exponent and Regime Computation
wire [EO+1:0] e_o;
assign e_o = {(rc ? {{EO-es-Bs+1{1'b0}},rgm} : -{{EO-es-Bs+1{1'b0}},rgm}),e} + BIAS;
//Final Output
assign out = inf_in|e_o[EO:E]|&e_o[E-1:0] ? {s,{E-1{1'b1}},{M{1'b0}}} : (zero_in|(~m[N-1]) ?
{s,{E-1{1'b0}},m[N-2:E]} : { s, e_o[E-1:0], m[N-2:E]} );


endmodule
```

```
//////////////////////////LZD////////////////////////////////////////////////////

module LZD_N (in, out);

  function [31:0] log2;
    input reg [31:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction

parameter N = 64;
parameter S = log2(N);
input [N-1:0] in;
output [S-1:0] out;

wire vld;
LZD #(.N(N)) l1 (in, out, vld);
endmodule


module LZD (in, out, vld);

  function [31:0] log2;
    input reg [31:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction


parameter N = 64;
parameter S = log2(N);

   input [N-1:0] in;
   output [S-1:0] out;
   output vld;

  generate
    if (N == 2)
      begin
```

```
        assign vld = ~&in;
        assign out = in[1] & ~in[0];
      end
    else if (N & (N-1))
      LZD #(1<<S) LZD ({1<<S {1'b0}} | in,out,vld);
    else
      begin
        wire [S-2:0] out_l;
        wire [S-2:0] out_h;
        wire out_vl, out_vh;
        LZD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LZD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule


//////////////////////////////////////////////LOD//////////////////////////////////////////////

////module LOD_N (in, out);

////  function [31:0] log2;
////    input reg [31:0] value;
////    begin
////      value = value-1;
////      for (log2=0; value>0; log2=log2+1)
////    value = value>>1;
////    end
////  endfunction

////parameter N = 64;
////parameter S = log2(N);
////input [N-1:0] in;
////output [S-1:0] out;

////wire vld;
////LOD #(.N(N)) l1 (in, out, vld);
////endmodule


////module LOD (in, out, vld);

////  function [31:0] log2;
////    input reg [31:0] value;
```

```
////    begin
////      value = value-1;
////      for (log2=0; value>0; log2=log2+1)
////    value = value>>1;
////    end
////  endfunction


//parameter N = 64;
//parameter S = log2(N);

//   input [N-1:0] in;
//   output [S-1:0] out;
//   output vld;

//  generate
//    if (N == 2)
//      begin
//      assign vld = |in;
//      assign out = ~in[1] & in[0];
//      end
//    else if (N & (N-1))
//      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
//    else
//      begin
//      wire [S-2:0] out_l, out_h;
//      wire out_vl, out_vh;
//      LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
//      LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
//      assign vld = out_vl | out_vh;
//      assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
//      end
//  endgenerate
//endmodule


////////////////////////////////END////////////////////////////////////
```

B.2. Verilog Code for IEEE 754 Nonlinear Approximation neuron

```
module Nonlin_sigmoid(y,x);
output [31:0] y;
input [31:0] x;

wire [31:0] x;
reg [31:0] yr;

assign y = 32'b00000000000000000000000000000000;
parameter a =  32'h3e7404ea ;        //0.2383
parameter b =  32'h3f000000  ;       // .50
parameter c =   32'h3d3f4880  ;      //0.0467
parameter d =    32'h3dfdbf48  ;     //0.1239
parameter e =    32'h3e980347  ;     //0.2969
parameter f =   32'hbd3f4880   ;     //-0.0467
parameter g =    32'h3e944674  ;     //0.2896
parameter h =   32'h3ef9f55a   ;     //0.4882
wire [31:0]i,j,k,l,m,n;         // for storing the internal variables
wire [31:0]o,p,q,r,s;
multiplier m11(x, x, i);    //   i stores x^2
multiplier m12(a,x,j);  //j stores (.2383*x)
multiplier m13(c,i,k) ; // k stores  (0.0467*x^2)
multiplier m14(d,x,l);  // l stores (.1239*x)
multiplier m15(f,i,m);  // m stores (-0.0467*x^2)
multiplier m16(g,x,n); // n stores (0.2896*x)
adder a11(j,b,o); //  o  stores (.2383*x +.50)          VAL OF Y for x(-1,1)
adder a12(k,l,p);  // p stores (0.0467x^2 + .1239x)
adder a13(p,e,q); // q stores (0.0467x^2 + .1239x+.2969)  VAL OF Y x=-1
adder a14(m,n,r); //  r stores (-0.0467*x^2 +0.2896x)
adder a15(r,h,s); // s stores (-0.0467*x^2 +0.2896x+0.4882) VAL of Y x=1
// x=1  in 754 is
always @(*)
begin
    if(x ==32'hbf800000)
    begin
        yr=q;
    end
    else if(x==32'h3f800000)
    begin
        yr=s;
    end
    else
    begin
        yr=o;
    end
```

```
end
assign y = yr;
endmodule



module fpuNEW(clk, A, B, opcode, O);
        input clk;
        input [31:0] A, B;
        input [1:0] opcode;
        output [31:0] O;

        wire [31:0] O;
        wire a_sign, b_sign;
        wire ADD, SUB, DIV, MUL;
        wire [7:0] a_exponent;
        wire [23:0] a_mantissa;
        wire [7:0] b_exponent;
        wire [23:0] b_mantissa;

        reg        o_sign;
        reg [7:0]  o_exponent;
        reg [24:0] o_mantissa;


        reg [31:0] adder_a_in;
        reg [31:0] adder_b_in;
        wire [31:0] adder_out;

        reg [31:0] multiplier_a_in;
        reg [31:0] multiplier_b_in;
        wire [31:0] multiplier_out;

        reg [31:0] divider_a_in;
        reg [31:0] divider_b_in;
        wire [31:0] divider_out;

        assign O[31] = o_sign;
        assign O[30:23] = o_exponent;
        assign O[22:0] = o_mantissa[22:0];

        assign a_sign = A[31];
        assign a_exponent[7:0] = A[30:23];
        assign a_mantissa[23:0] = {1'b1, A[22:0]};

        assign b_sign = B[31];
```

```
	assign b_exponent[7:0] = B[30:23];
	assign b_mantissa[23:0] = {1'b1, B[22:0]};

	assign ADD = !opcode[1] & !opcode[0];
	assign SUB = !opcode[1] & opcode[0];
	assign DIV = opcode[1] & !opcode[0];
	assign MUL = opcode[1] & opcode[0];

	adder A1
	(
		.a(adder_a_in),
		.b(adder_b_in),
		.out(adder_out)
	);

	multiplier M1
	(
		.a(multiplier_a_in),
		.b(multiplier_b_in),
		.out(multiplier_out)
	);

	divider D1
	(
		.a(divider_a_in),
		.b(divider_b_in),
		.out(divider_out)
	);

	always @ (posedge clk) begin
		if (ADD) begin
			//If a is NaN or b is zero return a
			if ((a_exponent == 255 && a_mantissa != 0) || (b_exponent == 0) &&
(b_mantissa == 0)) begin
				o_sign = a_sign;
				o_exponent = a_exponent;
				o_mantissa = a_mantissa;
			//If b is NaN or a is zero return b
			end else if ((b_exponent == 255 && b_mantissa != 0) || (a_exponent == 0)
&& (a_mantissa == 0)) begin
				o_sign = b_sign;
				o_exponent = b_exponent;
				o_mantissa = b_mantissa;
			//if a or b is inf return inf
			end else if ((a_exponent == 255) || (b_exponent == 255)) begin
				o_sign = a_sign ^ b_sign;
				o_exponent = 255;
```

```
                        o_mantissa = 0;
                    end else begin // Passed all corner cases
                        adder_a_in = A;
                        adder_b_in = B;
                        o_sign = adder_out[31];
                        o_exponent = adder_out[30:23];
                        o_mantissa = adder_out[22:0];
                    end
                end else if (SUB) begin
                    //If a is NaN or b is zero return a
                    if ((a_exponent == 255 && a_mantissa != 0) || (b_exponent == 0) &&
(b_mantissa == 0)) begin
                        o_sign = a_sign;
                        o_exponent = a_exponent;
                        o_mantissa = a_mantissa;
                    //If b is NaN or a is zero return b
                    end else if ((b_exponent == 255 && b_mantissa != 0) || (a_exponent == 0)
&& (a_mantissa == 0)) begin
                        o_sign = b_sign;
                        o_exponent = b_exponent;
                        o_mantissa = b_mantissa;
                    //if a or b is inf return inf
                    end else if ((a_exponent == 255) || (b_exponent == 255)) begin
                        o_sign = a_sign ^ b_sign;
                        o_exponent = 255;
                        o_mantissa = 0;
                    end else begin // Passed all corner cases
                        adder_a_in = A;
                        adder_b_in = {~B[31], B[30:0]};
                        o_sign = adder_out[31];
                        o_exponent = adder_out[30:23];
                        o_mantissa = adder_out[22:0];
                    end
                end else if (DIV) begin
                    divider_a_in = A;
                    divider_b_in = B;
                    o_sign = divider_out[31];
                    o_exponent = divider_out[30:23];
                    o_mantissa = divider_out[22:0];
                end else begin //Multiplication
                    //If a is NaN return NaN
                    if (a_exponent == 255 && a_mantissa != 0) begin
                        o_sign = a_sign;
                        o_exponent = 255;
                        o_mantissa = a_mantissa;
                    //If b is NaN return NaN
                    end else if (b_exponent == 255 && b_mantissa != 0) begin
```

```
                                o_sign = b_sign;
                                o_exponent = 255;
                                o_mantissa = b_mantissa;
                        //If a or b is 0 return 0
                        end else if ((a_exponent == 0) && (a_mantissa == 0) || (b_exponent == 0)
&& (b_mantissa == 0)) begin
                                o_sign = a_sign ^ b_sign;
                                o_exponent = 0;
                                o_mantissa = 0;
                        //if a or b is inf return inf
                        end else if ((a_exponent == 255) || (b_exponent == 255)) begin
                                o_sign = a_sign;
                                o_exponent = 255;
                                o_mantissa = 0;
                        end else begin // Passed all corner cases
                                multiplier_a_in = A;
                                multiplier_b_in = B;
                                o_sign = multiplier_out[31];
                                o_exponent = multiplier_out[30:23];
                                o_mantissa = multiplier_out[22:0];
                        end
                end
        end
endmodule

module adder(a, b, out);
  input  [31:0] a, b;
  output [31:0] out;

  wire [31:0] out;
        reg a_sign;
        reg [7:0] a_exponent;
        reg [23:0] a_mantissa;
        reg b_sign;
        reg [7:0] b_exponent;
        reg [23:0] b_mantissa;

  reg o_sign;
  reg [7:0] o_exponent;
  reg [24:0] o_mantissa;

  reg [7:0] diff;
  reg [23:0] tmp_mantissa;
  reg [7:0] tmp_exponent;


  reg  [7:0] i_e;
```

```
reg  [24:0] i_m;
wire [7:0] o_e;
wire [24:0] o_m;

addition_normaliser norm1
(
  .in_e(i_e),
  .in_m(i_m),
  .out_e(o_e),
  .out_m(o_m)
);

assign out[31] = o_sign;
assign out[30:23] = o_exponent;
assign out[22:0] = o_mantissa[22:0];

always @ ( * ) begin
             a_sign = a[31];
             if(a[30:23] == 0) begin
                    a_exponent = 8'b00000001;
                    a_mantissa = {1'b0, a[22:0]};
             end else begin
                    a_exponent = a[30:23];
                    a_mantissa = {1'b1, a[22:0]};
             end
             b_sign = b[31];
             if(b[30:23] == 0) begin
                    b_exponent = 8'b00000001;
                    b_mantissa = {1'b0, b[22:0]};
             end else begin
                    b_exponent = b[30:23];
                    b_mantissa = {1'b1, b[22:0]};
             end
  if (a_exponent == b_exponent) begin // Equal exponents
    o_exponent = a_exponent;
    if (a_sign == b_sign) begin // Equal signs = add
      o_mantissa = a_mantissa + b_mantissa;
      //Signify to shift
      o_mantissa[24] = 1;
      o_sign = a_sign;
    end else begin // Opposite signs = subtract
      if(a_mantissa > b_mantissa) begin
        o_mantissa = a_mantissa - b_mantissa;
        o_sign = a_sign;
      end else begin
        o_mantissa = b_mantissa - a_mantissa;
```

```
          o_sign = b_sign;
        end
      end
    end else begin //Unequal exponents
      if (a_exponent > b_exponent) begin // A is bigger
        o_exponent = a_exponent;
        o_sign = a_sign;
                              diff = a_exponent - b_exponent;
        tmp_mantissa = b_mantissa >> diff;
        if (a_sign == b_sign)
          o_mantissa = a_mantissa + tmp_mantissa;
        else
               o_mantissa = a_mantissa - tmp_mantissa;
      end else if (a_exponent < b_exponent) begin // B is bigger
        o_exponent = b_exponent;
        o_sign = b_sign;
        diff = b_exponent - a_exponent;
        tmp_mantissa = a_mantissa >> diff;
        if (a_sign == b_sign) begin
          o_mantissa = b_mantissa + tmp_mantissa;
        end else begin
                                      o_mantissa = b_mantissa - tmp_mantissa;
        end
      end
    end
    if(o_mantissa[24] == 1) begin
      o_exponent = o_exponent + 1;
      o_mantissa = o_mantissa >> 1;
    end else if((o_mantissa[23] != 1) && (o_exponent != 0)) begin
      i_e = o_exponent;
      i_m = o_mantissa;
      o_exponent = o_e;
      o_mantissa = o_m;
    end
  end
endmodule

module multiplier(a, b, out);
  input  [31:0] a, b;
  output [31:0] out;

  wire [31:0] out;
       reg a_sign;
  reg [7:0] a_exponent;
  reg [23:0] a_mantissa;
       reg b_sign;
```

```
reg [7:0] b_exponent;
reg [23:0] b_mantissa;

reg o_sign;
reg [7:0] o_exponent;
reg [24:0] o_mantissa;

      reg [47:0] product;

assign out[31] = o_sign;
assign out[30:23] = o_exponent;
assign out[22:0] = o_mantissa[22:0];

      reg  [7:0] i_e;
      reg  [47:0] i_m;
      wire [7:0] o_e;
      wire [47:0] o_m;

      multiplication_normaliser norm1
      (
              .in_e(i_e),
              .in_m(i_m),
              .out_e(o_e),
              .out_m(o_m)
      );


always @ ( * ) begin
              a_sign = a[31];
              if(a[30:23] == 0) begin
                      a_exponent = 8'b00000001;
                      a_mantissa = {1'b0, a[22:0]};
              end else begin
                      a_exponent = a[30:23];
                      a_mantissa = {1'b1, a[22:0]};
              end
              b_sign = b[31];
              if(b[30:23] == 0) begin
                      b_exponent = 8'b00000001;
                      b_mantissa = {1'b0, b[22:0]};
              end else begin
                      b_exponent = b[30:23];
                      b_mantissa = {1'b1, b[22:0]};
              end
  o_sign = a_sign ^ b_sign;
  o_exponent = a_exponent + b_exponent - 127;
```

```
    product = a_mantissa * b_mantissa;
                // Normalization
    if(product[47] == 1) begin
      o_exponent = o_exponent + 1;
      product = product >> 1;
    end else if((product[46] != 1) && (o_exponent != 0)) begin
      i_e = o_exponent;
      i_m = product;
      o_exponent = o_e;
      product = o_m;
    end
                o_mantissa = product[46:23];
        end
endmodule

module addition_normaliser(in_e, in_m, out_e, out_m);
  input [7:0] in_e;
  input [24:0] in_m;
  output [7:0] out_e;
  output [24:0] out_m;

  wire [7:0] in_e;
  wire [24:0] in_m;
  reg [7:0] out_e;
  reg [24:0] out_m;

  always @ ( * ) begin
                if (in_m[23:3] == 21'b000000000000000000001) begin
                        out_e = in_e - 20;
                        out_m = in_m << 20;
                end else if (in_m[23:4] == 20'b00000000000000000001) begin
                        out_e = in_e - 19;
                        out_m = in_m << 19;
                end else if (in_m[23:5] == 19'b0000000000000000001) begin
                        out_e = in_e - 18;
                        out_m = in_m << 18;
                end else if (in_m[23:6] == 18'b000000000000000001) begin
                        out_e = in_e - 17;
                        out_m = in_m << 17;
                end else if (in_m[23:7] == 17'b00000000000000001) begin
                        out_e = in_e - 16;
                        out_m = in_m << 16;
                end else if (in_m[23:8] == 16'b0000000000000001) begin
                        out_e = in_e - 15;
                        out_m = in_m << 15;
                end else if (in_m[23:9] == 15'b000000000000001) begin
```

```
                out_e = in_e - 14;
                out_m = in_m << 14;
            end else if (in_m[23:10] == 14'b00000000000001) begin
                out_e = in_e - 13;
                out_m = in_m << 13;
            end else if (in_m[23:11] == 13'b0000000000001) begin
                out_e = in_e - 12;
                out_m = in_m << 12;
            end else if (in_m[23:12] == 12'b000000000001) begin
                out_e = in_e - 11;
                out_m = in_m << 11;
            end else if (in_m[23:13] == 11'b00000000001) begin
                out_e = in_e - 10;
                out_m = in_m << 10;
            end else if (in_m[23:14] == 10'b0000000001) begin
                out_e = in_e - 9;
                out_m = in_m << 9;
            end else if (in_m[23:15] == 9'b000000001) begin
                out_e = in_e - 8;
                out_m = in_m << 8;
            end else if (in_m[23:16] == 8'b00000001) begin
                out_e = in_e - 7;
                out_m = in_m << 7;
            end else if (in_m[23:17] == 7'b0000001) begin
                out_e = in_e - 6;
                out_m = in_m << 6;
            end else if (in_m[23:18] == 6'b000001) begin
                out_e = in_e - 5;
                out_m = in_m << 5;
            end else if (in_m[23:19] == 5'b00001) begin
                out_e = in_e - 4;
                out_m = in_m << 4;
            end else if (in_m[23:20] == 4'b0001) begin
                out_e = in_e - 3;
                out_m = in_m << 3;
            end else if (in_m[23:21] == 3'b001) begin
                out_e = in_e - 2;
                out_m = in_m << 2;
            end else if (in_m[23:22] == 2'b01) begin
                out_e = in_e - 1;
                out_m = in_m << 1;
            end
  end
endmodule

module multiplication_normaliser(in_e, in_m, out_e, out_m);
```

```
  input [7:0] in_e;
  input [47:0] in_m;
  output [7:0] out_e;
  output [47:0] out_m;

  wire [7:0] in_e;
  wire [47:0] in_m;
  reg [7:0] out_e;
  reg [47:0] out_m;

  always @ ( * ) begin
         if (in_m[46:41] == 6'b000001) begin
                      out_e = in_e - 5;
                      out_m = in_m << 5;
               end else if (in_m[46:42] == 5'b00001) begin
                      out_e = in_e - 4;
                      out_m = in_m << 4;
               end else if (in_m[46:43] == 4'b0001) begin
                      out_e = in_e - 3;
                      out_m = in_m << 3;
               end else if (in_m[46:44] == 3'b001) begin
                      out_e = in_e - 2;
                      out_m = in_m << 2;
               end else if (in_m[46:45] == 2'b01) begin
                      out_e = in_e - 1;
                      out_m = in_m << 1;
               end
  end
  endmodule

module divider (a, b, out);
  input  [31:0] a, b;
  output [31:0] out;

  wire [31:0] out;
        reg a_sign;
  reg [7:0] a_exponent;
  reg [23:0] a_mantissa;
        reg b_sign;
  reg [7:0] b_exponent;
  reg [23:0] b_mantissa;

  reg o_sign;
  reg [7:0] o_exponent;
  reg [24:0] o_mantissa;
```

```
  always @(*)
  begin
    if (a_exponent >> b_exponent)
        o_exponent = (a_exponent - b_exponent);
    else
        o_exponent = (b_exponent - a_exponent);

    o_sign = a_sign  ^ b_sign;
  end
  restore_conv DIV(a_mantissa, b_mantissa, o_mantissa);
endmodule

module restore_conv(A,B,Res);
    //the size of input and output ports of the division module is generic.
    parameter WIDTH = 24;
    //input and output ports.
    input [WIDTH-1:0] A;
    input [WIDTH-1:0] B;
    output [WIDTH-1:0] Res;
    //internal variables
    reg [WIDTH-1:0] Res = 0;
    reg [WIDTH-1:0] a1,b1;
    reg [WIDTH:0] p1;
    integer i;

    always@ (A or B)
    begin
        //initialize the variables.
        a1 = A;
        b1 = B;
        p1= 0;
        for(i=0;i < WIDTH;i=i+1)    begin //start the for loop
            p1 = {p1[WIDTH-2:0],a1[WIDTH-1]};
            a1[WIDTH-1:1] = a1[WIDTH-2:0];
            p1 = p1-b1;
            if(p1[WIDTH-1] == 1)    begin
                a1[0] = 0;
                p1 = p1 + b1;   end
            else
                a1[0] = 1;
        end
        Res = a1;
    end
endmodule

//////////////////////////////////////////////////////////////////////////////////
```

B.3. Verilog Code for IEEE 754 Padé Approximation neuron

```
module Pade_Sigmoid(y,x);
output  [31:0]y;  ///IEEE 754 NOTATION
input   [31:0]x;
// parametrizing the values
parameter a =   32'h41a00000;          // Stores 20
parameter b =   32'h43340000 ;        // stores 180
parameter c=    32'h44520000 ;       //stores 840
parameter d=    32'h44d20000 ;      //stores 1680
parameter e=    32'hbddb22d1 ;       // stores -0.107
parameter f=    32'h3c3b98c8 ;            // stores 0.01145
parameter one = 32'h3f800000 ;      // stores one
parameter const =32'h399c09e1;   //  stores 2.9762*10^-4
wire [31:0]xfour,xcube,xsquare;

wire [31:0]i,j,k,l,m,n,o,p,q,r,s,t;
multiplier m19(x, x, xsquare);  // x square contains x^2
multiplier m20(xsquare,x,xcube); // xcube contains x^3
multiplier m21(xcube,x,xfour);  // xfour contains x^4
multiplier m22(a,xcube,i) ;     //  i contains (20*x^3)
multiplier m23(b,xsquare,j);  // j contains (180*x^2)
multiplier m24(c,x,k);        // k contains  (840*x)
multiplier m25(f,xfour,l); // l contains 0.01145*x^4
multiplier m26(e,xsquare,m); // m contains  -0.107x^2
// declaring Oprations
adder   a111(xfour,i,n);    // n contains x^4+20x^3
adder   a112(j,k,o);       // o contains 180x^2+840x
adder  a113(o,d,p);      // p contains  180x^2+840x+1680
adder  a114(p,n,q);   // q contains x^4+20x^3+180x^2+840x+1680
// 1st term over
// moving on 2nd term
adder  a115(one,e,r);   //  r contains 1-0.107x^2
adder a116(r,l,s);   //  s contains 1-.107x^2+0.01145x^4 // 2nd term over
// all terms over
multiplier PRE(const,s,t);  //  t contains  (1-.107x^2+0.01145x^4)*2.9762*10^-4
multiplier final(q,t,y);
//Final result is y
endmodule

module SISO_neuronPade(i,o);
output [31:0]o;
input [31:0]i;
// instantiating
```

```
Pade_Sigmoid N1(o,i);
endmodule


module MISO_neuronPade(i1,i2,i3,i4,w1,w2,w3,w4,o);
input [31:0]i1,i2,i3,i4,w1,w2,w3,w4;
output [31:0]o;
wire [31:0]A1,A2,in,m1,m2,m3,m4;
multiplier mul1(i1,w1,m1);
multiplier mul2(i2,w2,m2);
multiplier mul3(i3,w3,m3);
multiplier mul4(i4,w4,m4);
adder N11(m1,m2,A1);
adder N22(m3,m4,A2);
adder N33(A1,A2,in);
SISO_neuronPade N111(in,o);
endmodule


//// Full network//  ///TOP MODULE BECOMES Full_neuronNEW_PADE//
module
Full_neuronNEW_PADE(inp1,inp2,inp3,inp4,cloock,reset,layer1,layer2,layer3,layer4,layer5,
outL1N1,outL1N2,outL1N3,outL1N4,outL2N1,outL2N2,outL2N3,outL2N4,outL3N1,outL3N2,outL3N3,outL3N
4,
outL4N1,outL4N2,outL4N3,outL4N4,outL5N1,outL5N2,outL5N3);
wire [31:0]Final1,Final2,Final3;
input [31:0]inp1,inp2,inp3,inp4;
wire [31:0]OL1,OL2,OL3,OL4;
wire [31:0]OLL1,OLL2,OLL3,OLL4;
wire [31:0]OLLL1,OLLL2,OLLL3,OLLL4;
wire [31:0]OLLLL1,OLLLL2,OLLLL3,OLLLL4;
reg [2:0]cst,nst;  // defining the current state and next state
input layer1,layer2,layer3,layer4,layer5; // trigerring conditions for each layer
parameter S0=3'b000;
parameter S1=3'b001; //// parameter for defining the states binary encoding
parameter S2=3'b010;
parameter S3=3'b011;
parameter S4=3'b100;
input cloock,reset;
output reg [31:0]outL1N1,outL1N2,outL1N3,outL1N4; // OUTPUT of LAYER1NEURON 1 TO 4;
output reg [31:0]outL2N1,outL2N2,outL2N3,outL2N4; // OUTPUT OF LAYER2 NEURON 1 TO 4
output reg [31:0]outL3N1,outL3N2,outL3N3,outL3N4; // OUTPUT OF LAYER 3 NEURON 1 TO 4
output reg [31:0]outL4N1,outL4N2,outL4N3,outL4N4;  // output of layer 4 neuron  1 to 4
output reg [31:0]outL5N1,outL5N2,outL5N3;  //  output of layer 5  neuron  1 to 3;
////// these are variables
/// DEFINING INPUT LAYERS
SISO_neuronPade S111(inp1,OL1);
SISO_neuronPade S222(inp2,OL2);
```

```
SISO_neuronPade S333(inp3,OL3);
SISO_neuronPade S444(inp4,OL4);
// DEFINING FIRST HIDDEN LAYERS
MISO_neuronPade M1(OL1,OL2,OL3,OL4,OLL1);
MISO_neuronPade M2(OL1,OL2,OL3,OL4,OLL2);
MISO_neuronPade M3(OL1,OL2,OL3,OL4,OLL3);
MISO_neuronPade M4(OL1,OL2,OL3,OL4,OLL4);
/// DEFINING SECOND HIDDEN LAYERS
MISO_neuronPade M5(OLL1,OLL2,OLL3,OLL4,OLLL1);
MISO_neuronPade M6(OLL1,OLL2,OLL3,OLL4,OLLL2);
MISO_neuronPade M7(OLL1,OLL2,OLL3,OLL4,OLLL3);
MISO_neuronPade M8(OLL1,OLL2,OLL3,OLL4,OLLL4);
///DEFINING THIRD LAYER
MISO_neuronPade M9(OLLL1,OLLL2,OLLL3,OLLL4,OLLLL1);
MISO_neuronPade M10(OLLL1,OLLL2,OLLL3,OLLL4,OLLLL2);
MISO_neuronPade M11(OLLL1,OLLL2,OLLL3,OLLL4,OLLLL3);
MISO_neuronPade M12(OLLL1,OLLL2,OLLL3,OLLL4,OLLLL4);
//DEFINING  THE OUTPUT LAYER
MISO_neuronPade M13(OLLLL1,OLLLL2,OLLLL3,OLLLL4,Final1);
MISO_neuronPade M14(OLLLL1,OLLLL2,OLLLL3,OLLLL4,Final2);
MISO_neuronPade M15(OLLLL1,OLLLL2,OLLLL3,OLLLL4,Final3);
// The whole neuron structure is controlled by aN FSM
always @(*)
begin
case(cst)
S0:if(layer1==1'b1)
begin
nst=S1;
outL1N1=OL1; // outputs of layer 1  neuron 1 to 4  are displayed  in first state S0
outL1N2=OL2;
outL1N3=OL3;
outL1N4=OL4;
end
else
begin
nst=cst;
end
S1:if(layer2==1'b1)
begin
nst=S2;
outL2N1=OLL1;
outL2N2=OLL2;
outL2N3=OLL3;
outL2N4=OLL4;
end
else
```

```
begin
nst=cst;
end
S2:if(layer3==1'b1)
begin
outL3N1=OLLL1;
outL3N2=OLLL2;
outL3N3=OLLL3;
outL3N4=OLLL4;
nst=S3;
end
else
begin
nst=cst;
end
S3:if(layer4==1'b1)
begin
outL4N1=OLLLL1;
outL4N2=OLLLL2;
outL4N3=OLLLL3;
outL4N4=OLLLL4;
nst=S4;
end
else
begin
nst=cst;
end
S4:if(layer5==1'b1)
begin
outL5N1=Final1;
outL5N2=Final2;
outL5N3=Final3;
nst=S0;
end
else
begin
nst=cst;
end
default: nst = S0;
endcase
end

always@(posedge cloock)
          begin
           if (reset)
             cst <= S0;
```

```
            else
              cst <= nst;
           end
endmodule

/////
module fpuNEW(clk, A, B, opcode, O);
        input clk;
        input [31:0] A, B;
        input [1:0] opcode;
        output [31:0] O;

        wire [31:0] O;
        wire [7:0] a_exponent;
        wire [23:0] a_mantissa;
        wire [7:0] b_exponent;
        wire [23:0] b_mantissa;

        reg        o_sign;
        reg [7:0]  o_exponent;
        reg [24:0] o_mantissa;


        reg [31:0] adder_a_in;
        reg [31:0] adder_b_in;
        wire [31:0] adder_out;

        reg [31:0] multiplier_a_in;
        reg [31:0] multiplier_b_in;
        wire [31:0] multiplier_out;

        reg [31:0] divider_a_in;
        reg [31:0] divider_b_in;
        wire [31:0] divider_out;

        assign O[31] = o_sign;
        assign O[30:23] = o_exponent;
        assign O[22:0] = o_mantissa[22:0];

        assign a_sign = A[31];
        assign a_exponent[7:0] = A[30:23];
        assign a_mantissa[23:0] = {1'b1, A[22:0]};

        assign b_sign = B[31];
        assign b_exponent[7:0] = B[30:23];
        assign b_mantissa[23:0] = {1'b1, B[22:0]};
```

```
	assign ADD = !opcode[1] & !opcode[0];
	assign SUB = !opcode[1] & opcode[0];
	assign DIV = opcode[1] & !opcode[0];
	assign MUL = opcode[1] & opcode[0];

	adder A1
	(
		.a(adder_a_in),
		.b(adder_b_in),
		.out(adder_out)
	);

	multiplier M1
	(
		.a(multiplier_a_in),
		.b(multiplier_b_in),
		.out(multiplier_out)
	);

	divider D1
	(
		.a(divider_a_in),
		.b(divider_b_in),
		.out(divider_out)
	);

	always @ (posedge clk) begin
		if (ADD) begin
			//If a is NaN or b is zero return a
			if ((a_exponent == 255 && a_mantissa != 0) || (b_exponent == 0) &&
(b_mantissa == 0)) begin
				o_sign = a_sign;
				o_exponent = a_exponent;
				o_mantissa = a_mantissa;
			//If b is NaN or a is zero return b
			end else if ((b_exponent == 255 && b_mantissa != 0) || (a_exponent == 0)
&& (a_mantissa == 0)) begin
				o_sign = b_sign;
				o_exponent = b_exponent;
				o_mantissa = b_mantissa;
			//if a or b is inf return inf
			end else if ((a_exponent == 255) || (b_exponent == 255)) begin
				o_sign = a_sign ^ b_sign;
				o_exponent = 255;
				o_mantissa = 0;
			end else begin // Passed all corner cases
```

```
                        adder_a_in = A;
                        adder_b_in = B;
                        o_sign = adder_out[31];
                        o_exponent = adder_out[30:23];
                        o_mantissa = adder_out[22:0];
                end
        end else if (SUB) begin
                //If a is NaN or b is zero return a
                if ((a_exponent == 255 && a_mantissa != 0) || (b_exponent == 0) &&
(b_mantissa == 0)) begin
                        o_sign = a_sign;
                        o_exponent = a_exponent;
                        o_mantissa = a_mantissa;
                //If b is NaN or a is zero return b
                end else if ((b_exponent == 255 && b_mantissa != 0) || (a_exponent == 0)
&& (a_mantissa == 0)) begin
                        o_sign = b_sign;
                        o_exponent = b_exponent;
                        o_mantissa = b_mantissa;
                //if a or b is inf return inf
                end else if ((a_exponent == 255) || (b_exponent == 255)) begin
                        o_sign = a_sign ^ b_sign;
                        o_exponent = 255;
                        o_mantissa = 0;
                end else begin // Passed all corner cases
                        adder_a_in = A;
                        adder_b_in = {~B[31], B[30:0]};
                        o_sign = adder_out[31];
                        o_exponent = adder_out[30:23];
                        o_mantissa = adder_out[22:0];
                end
        end else if (DIV) begin
                divider_a_in = A;
                divider_b_in = B;
                o_sign = divider_out[31];
                o_exponent = divider_out[30:23];
                o_mantissa = divider_out[22:0];
        end else begin //Multiplication
                //If a is NaN return NaN
                if (a_exponent == 255 && a_mantissa != 0) begin
                        o_sign = a_sign;
                        o_exponent = 255;
                        o_mantissa = a_mantissa;
                //If b is NaN return NaN
                end else if (b_exponent == 255 && b_mantissa != 0) begin
                        o_sign = b_sign;
                        o_exponent = 255;
```

```
                        o_mantissa = b_mantissa;
                    //If a or b is 0 return 0
                    end else if ((a_exponent == 0) && (a_mantissa == 0) || (b_exponent == 0)
&& (b_mantissa == 0)) begin
                        o_sign = a_sign ^ b_sign;
                        o_exponent = 0;
                        o_mantissa = 0;
                    //if a or b is inf return inf
                    end else if ((a_exponent == 255) || (b_exponent == 255)) begin
                        o_sign = a_sign;
                        o_exponent = 255;
                        o_mantissa = 0;
                    end else begin // Passed all corner cases
                        multiplier_a_in = A;
                        multiplier_b_in = B;
                        o_sign = multiplier_out[31];
                        o_exponent = multiplier_out[30:23];
                        o_mantissa = multiplier_out[22:0];
                    end
            end
        end
endmodule


module adder(a, b, out);
  input  [31:0] a, b;
  output [31:0] out;

  wire [31:0] out;
        reg a_sign;
        reg [7:0] a_exponent;
        reg [23:0] a_mantissa;
        reg b_sign;
        reg [7:0] b_exponent;
        reg [23:0] b_mantissa;

  reg o_sign;
  reg [7:0] o_exponent;
  reg [24:0] o_mantissa;

  reg [7:0] diff;
  reg [23:0] tmp_mantissa;
  reg [7:0] tmp_exponent;


  reg  [7:0] i_e;
  reg  [24:0] i_m;
```

```
wire [7:0] o_e;
wire [24:0] o_m;

addition_normaliser norm1
(
  .in_e(i_e),
  .in_m(i_m),
  .out_e(o_e),
  .out_m(o_m)
);

assign out[31] = o_sign;
assign out[30:23] = o_exponent;
assign out[22:0] = o_mantissa[22:0];

always @ ( * ) begin
             a_sign = a[31];
             if(a[30:23] == 0) begin
                    a_exponent = 8'b00000001;
                    a_mantissa = {1'b0, a[22:0]};
             end else begin
                    a_exponent = a[30:23];
                    a_mantissa = {1'b1, a[22:0]};
             end
             b_sign = b[31];
             if(b[30:23] == 0) begin
                    b_exponent = 8'b00000001;
                    b_mantissa = {1'b0, b[22:0]};
             end else begin
                    b_exponent = b[30:23];
                    b_mantissa = {1'b1, b[22:0]};
             end
  if (a_exponent == b_exponent) begin // Equal exponents
    o_exponent = a_exponent;
    if (a_sign == b_sign) begin // Equal signs = add
      o_mantissa = a_mantissa + b_mantissa;
      //Signify to shift
      o_mantissa[24] = 1;
      o_sign = a_sign;
    end else begin // Opposite signs = subtract
      if(a_mantissa > b_mantissa) begin
        o_mantissa = a_mantissa - b_mantissa;
        o_sign = a_sign;
      end else begin
        o_mantissa = b_mantissa - a_mantissa;
        o_sign = b_sign;
```

```
        end
      end
    end else begin //Unequal exponents
      if (a_exponent > b_exponent) begin // A is bigger
        o_exponent = a_exponent;
        o_sign = a_sign;
                              diff = a_exponent - b_exponent;
        tmp_mantissa = b_mantissa >> diff;
        if (a_sign == b_sign)
          o_mantissa = a_mantissa + tmp_mantissa;
        else
               o_mantissa = a_mantissa - tmp_mantissa;
      end else if (a_exponent < b_exponent) begin // B is bigger
        o_exponent = b_exponent;
        o_sign = b_sign;
        diff = b_exponent - a_exponent;
        tmp_mantissa = a_mantissa >> diff;
        if (a_sign == b_sign) begin
          o_mantissa = b_mantissa + tmp_mantissa;
        end else begin
                                      o_mantissa = b_mantissa - tmp_mantissa;
        end
      end
    end
    if(o_mantissa[24] == 1) begin
      o_exponent = o_exponent + 1;
      o_mantissa = o_mantissa >> 1;
    end else if((o_mantissa[23] != 1) && (o_exponent != 0)) begin
      i_e = o_exponent;
      i_m = o_mantissa;
      o_exponent = o_e;
      o_mantissa = o_m;
    end
  end
endmodule

module multiplier(a, b, out);
  input  [31:0] a, b;
  output [31:0] out;

  wire [31:0] out;
       reg a_sign;
  reg [7:0] a_exponent;
  reg [23:0] a_mantissa;
       reg b_sign;
  reg [7:0] b_exponent;
```

```
reg [23:0] b_mantissa;

reg o_sign;
reg [7:0] o_exponent;
reg [24:0] o_mantissa;

      reg [47:0] product;

assign out[31] = o_sign;
assign out[30:23] = o_exponent;
assign out[22:0] = o_mantissa[22:0];

      reg  [7:0] i_e;
      reg  [47:0] i_m;
      wire [7:0] o_e;
      wire [47:0] o_m;

      multiplication_normaliser norm1
      (
              .in_e(i_e),
              .in_m(i_m),
              .out_e(o_e),
              .out_m(o_m)
      );


always @ ( * ) begin
              a_sign = a[31];
              if(a[30:23] == 0) begin
                      a_exponent = 8'b00000001;
                      a_mantissa = {1'b0, a[22:0]};
              end else begin
                      a_exponent = a[30:23];
                      a_mantissa = {1'b1, a[22:0]};
              end
              b_sign = b[31];
              if(b[30:23] == 0) begin
                      b_exponent = 8'b00000001;
                      b_mantissa = {1'b0, b[22:0]};
              end else begin
                      b_exponent = b[30:23];
                      b_mantissa = {1'b1, b[22:0]};
              end
  o_sign = a_sign ^ b_sign;
  o_exponent = a_exponent + b_exponent - 127;
  product = a_mantissa * b_mantissa;
```

```
                // Normalization
    if(product[47] == 1) begin
      o_exponent = o_exponent + 1;
      product = product >> 1;
    end else if((product[46] != 1) && (o_exponent != 0)) begin
      i_e = o_exponent;
      i_m = product;
      o_exponent = o_e;
      product = o_m;
    end
                o_mantissa = product[46:23];
       end
endmodule

module addition_normaliser(in_e, in_m, out_e, out_m);
  input [7:0] in_e;
  input [24:0] in_m;
  output [7:0] out_e;
  output [24:0] out_m;

  wire [7:0] in_e;
  wire [24:0] in_m;
  reg [7:0] out_e;
  reg [24:0] out_m;

  always @ ( * ) begin
                if (in_m[23:3] == 21'b000000000000000000001) begin
                        out_e = in_e - 20;
                        out_m = in_m << 20;
                end else if (in_m[23:4] == 20'b00000000000000000001) begin
                        out_e = in_e - 19;
                        out_m = in_m << 19;
                end else if (in_m[23:5] == 19'b0000000000000000001) begin
                        out_e = in_e - 18;
                        out_m = in_m << 18;
                end else if (in_m[23:6] == 18'b000000000000000001) begin
                        out_e = in_e - 17;
                        out_m = in_m << 17;
                end else if (in_m[23:7] == 17'b00000000000000001) begin
                        out_e = in_e - 16;
                        out_m = in_m << 16;
                end else if (in_m[23:8] == 16'b0000000000000001) begin
                        out_e = in_e - 15;
                        out_m = in_m << 15;
                end else if (in_m[23:9] == 15'b000000000000001) begin
                        out_e = in_e - 14;
```

```
                out_m = in_m << 14;
            end else if (in_m[23:10] == 14'b00000000000001) begin
                out_e = in_e - 13;
                out_m = in_m << 13;
            end else if (in_m[23:11] == 13'b0000000000001) begin
                out_e = in_e - 12;
                out_m = in_m << 12;
            end else if (in_m[23:12] == 12'b000000000001) begin
                out_e = in_e - 11;
                out_m = in_m << 11;
            end else if (in_m[23:13] == 11'b00000000001) begin
                out_e = in_e - 10;
                out_m = in_m << 10;
            end else if (in_m[23:14] == 10'b0000000001) begin
                out_e = in_e - 9;
                out_m = in_m << 9;
            end else if (in_m[23:15] == 9'b000000001) begin
                out_e = in_e - 8;
                out_m = in_m << 8;
            end else if (in_m[23:16] == 8'b00000001) begin
                out_e = in_e - 7;
                out_m = in_m << 7;
            end else if (in_m[23:17] == 7'b0000001) begin
                out_e = in_e - 6;
                out_m = in_m << 6;
            end else if (in_m[23:18] == 6'b000001) begin
                out_e = in_e - 5;
                out_m = in_m << 5;
            end else if (in_m[23:19] == 5'b00001) begin
                out_e = in_e - 4;
                out_m = in_m << 4;
            end else if (in_m[23:20] == 4'b0001) begin
                out_e = in_e - 3;
                out_m = in_m << 3;
            end else if (in_m[23:21] == 3'b001) begin
                out_e = in_e - 2;
                out_m = in_m << 2;
            end else if (in_m[23:22] == 2'b01) begin
                out_e = in_e - 1;
                out_m = in_m << 1;
            end
  end
endmodule
module multiplication_normaliser(in_e, in_m, out_e, out_m);
  input [7:0] in_e;
  input [47:0] in_m;
```

```
  output [7:0] out_e;
  output [47:0] out_m;

  wire [7:0] in_e;
  wire [47:0] in_m;
  reg [7:0] out_e;
  reg [47:0] out_m;

  always @ ( * ) begin
         if (in_m[46:41] == 6'b000001) begin
                      out_e = in_e - 5;
                      out_m = in_m << 5;
               end else if (in_m[46:42] == 5'b00001) begin
                      out_e = in_e - 4;
                      out_m = in_m << 4;
               end else if (in_m[46:43] == 4'b0001) begin
                      out_e = in_e - 3;
                      out_m = in_m << 3;
               end else if (in_m[46:44] == 3'b001) begin
                      out_e = in_e - 2;
                      out_m = in_m << 2;
               end else if (in_m[46:45] == 2'b01) begin
                      out_e = in_e - 1;
                      out_m = in_m << 1;
               end
  end
endmodule
```

C. Appendix C

C.1. Verilog Code for Augmentation ANN

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////

//////////////////////       Augmentation ANN        ////////////////////////////

////////////////////////////////////////////////////////////////////////////////

module AugANN(p, O, DO, EC);

input [15:0] p, O;

output [15:0] DO, EC;



wire [15:0] pp, Op;

wire [15:0] in1, in2;

wire [15:0] l [1:32];

wire [15:0] m [1:32];

wire [15:0] n [1:32];

//wire [1:32] q [15:0];



FP2posit F2P_1(p, pp);

FP2posit F2P_2(O, Op);



sigmoid_1in I1(pp, in1);

sigmoid_1in I2(Op, in2);



neuronposit L1(in1, in2, l[1]);

neuronposit L2(in1, in2, l[2]);
```

```
neuronposit L3(in1, in2, l[3]);

neuronposit L4(in1, in2, l[4]);

neuronposit L5(in1, in2, l[5]);

neuronposit L6(in1, in2, l[6]);

neuronposit L7(in1, in2, l[7]);

neuronposit L8(in1, in2, l[8]);

neuronposit L9(in1, in2, l[9]);

neuronposit L10(in1, in2, l[10]);

neuronposit L11(in1, in2, l[11]);

neuronposit L12(in1, in2, l[12]);

neuronposit L13(in1, in2, l[13]);

neuronposit L14(in1, in2, l[14]);

neuronposit L15(in1, in2, l[15]);

neuronposit L16(in1, in2, l[16]);

neuronposit L17(in1, in2, l[17]);

neuronposit L18(in1, in2, l[18]);

neuronposit L19(in1, in2, l[19]);

neuronposit L20(in1, in2, l[20]);

neuronposit L21(in1, in2, l[21]);

neuronposit L22(in1, in2, l[22]);

neuronposit L23(in1, in2, l[23]);

neuronposit L24(in1, in2, l[24]);

neuronposit L25(in1, in2, l[25]);

neuronposit L26(in1, in2, l[26]);

neuronposit L27(in1, in2, l[27]);
```

```
neuronposit L28(in1, in2, l[28]);

neuronposit L29(in1, in2, l[29]);

neuronposit L30(in1, in2, l[30]);

neuronposit L31(in1, in2, l[31]);

neuronposit L32(in1, in2, l[32]);



genvar i;



//generate

//    for (i = 1; i <= 32; i = i + 1) begin

//        neuronposit L(in1, in2, l[i]);

//        end

//endgenerate

generate

    for (i = 1; i <= 32 ; i = i + 1) begin

            neuron32in M(l[1], l[2], l[3], l[4], l[5], l[6], l[7], l[8], l[9], l[10], l[11],
l[12], l[13], l[14], l[15], l[16], l[17], l[18], l[19], l[20], l[21], l[22], l[23], l[24],
l[25], l[26], l[27], l[28], l[29], l[30], l[31], l[32], m[i] );

    end

endgenerate

generate

    for (i = 1; i <= 32 ; i = i + 1) begin

            neuron32in N(m[1], m[2], m[3], m[4], m[5], m[6], m[7], m[8], m[9], m[10], m[11],
m[12], m[13], m[14], m[15], m[16], m[17], m[18], m[19], m[20], m[21], m[22], m[23], m[24],
m[25], m[26], m[27], m[28], m[29], m[30], m[31], m[32], n[i] );

    end

endgenerate
```

```
//generate

//    for (i = 1; i <= 8 ; i = i + 1) begin

//            neuron8in P(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8], q[i] );

//    end

//endgenerate

neuron32in D(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8],n[9], n[10], n[11], n[12], n[13],
n[14], n[15], n[16],n[17], n[18], n[19], n[20], n[21], n[22], n[23], n[24],n[25], n[26], n[27],
n[28], n[29], n[30], n[31], n[32], DO);

neuron32in E(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8],n[9], n[10], n[11], n[12], n[13],
n[14], n[15], n[16],n[17], n[18], n[19], n[20], n[21], n[22], n[23], n[24],n[25], n[26], n[27],
n[28], n[29], n[30], n[31], n[32], EC);

//genvar i;

//generate

//    for (i = 1; i <= 8 ; i = i + 1) begin

//            neuron_posit L[i](in1, in2, l[i]);

//    end

//endgenerate

endmodule



//wire [15:0] I1L1, I1L2, I1L3, I1L4, I1L5, I1L6, I1L7, I1L8;

//wire [15:0] I2L1, I2L2, I2L3, I2L4, I2L5, I2L6, I2L7, I2L8;

//wire [15:0] L1M1, L1M2, L1M3, L1M4, L1M5, L1M6, L1M7, L1M8, L2M1, L2M2, L2M3, L2M4, L2M5,
L2M6, L2M7, L2M8;

//wire [15:0] L3M1, L3M2, L3M3, L3M4, L3M5, L3M6, L3M7, L3M8, L4M1, L4M2, L4M3, L4M4, L4M5,
L4M6, L4M7, L4M8;

//wire [15:0] L5M1, L5M2, L5M3, L5M4, L5M5, L5M6, L5M7, L5M8, L6M1, L6M2, L6M3, L6M4, L6M5,
L6M6, L6M7, L6M8;

//wire [15:0] L7M1, L7M2, L7M3, L7M4, L7M5, L7M6, L7M7, L7M8, L8M1, L8M2, L8M3, L8M4, L8M5,
L8M6, L8M7, L8M8;
```

```
//wire [15:0] M1N1, M1N2, M1N3, M1N4, M1N5, M1N6, M1N7, M1N8, M2N1, M2N2, M2N3, M2N4, M2N5,
M2N6, M2N7, M2N8;

//wire [15:0] M3N1, M3N2, M3N3, M3N4, M3N5, M3N6, M3N7, M3N8, M4N1, M4N2, M4N3, M4N4, M4N5,
M4N6, M4N7, M4N8;

//wire [15:0] M5N1, M5N2, M5N3, M5N4, M5N5, M5N6, M5N7, M5N8, M6N1, M6N2, M6N3, M6N4, M6N5,
M6N6, M6N7, M6N8;

//wire [15:0] M7N1, M7N2, M7N3, M7N4, M7N5, M7N6, M7N7, M7N8, M8N1, M8N2, M8N3, M8N4, M8N5,
M8N6, M8N7, M8N8;

//wire [15:0] N1D, N2D, N3D, N4D, N5D, N6D, N7D, N8D;

//wire [15:0] N1EC, N2EC, N3EC, N4EC, N5EC, N6EC, N7EC, N8EC;

////////////////////////////////////////////////////////////////////////////////

//////////////////IEEE 754 to Posit Conversion//////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

module FP2posit(pos_in, pos_out);

function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction



parameter N = 16;

parameter E = 5;

parameter es = 3;      //ES_max = E-1

parameter M = N-E-1;
```

```
parameter BIAS = (2**(E-1))-1;


parameter Bs = log2(N);


input [N-1:0] pos_in;

output [N-1:0] pos_out;


wire s_in = pos_in[N-1];

wire [E-1:0] exp_in = pos_in[N-2:N-1-E];

wire [M-1:0] mant_in = pos_in[M-1:0];

wire zero_in = ~|{exp_in,mant_in};

wire inf_in = &exp_in;

wire [M:0] mant = {|exp_in, mant_in};

wire [N-1:0] LOD_in = {mant,{E{1'b0}}};

wire[Bs-1:0] Lshift;

LOD_N #(.N(N)) uut (.in(LOD_in), .out(Lshift));

wire[N-1:0] mant_tmp;

DSR_left_N_S #(.N(N), .S(Bs)) ls (.a(LOD_in),.b(Lshift),.c(mant_tmp));


wire [E:0] exp = {exp_in[E-1:1], exp_in[0] | (~|exp_in)} - BIAS - Lshift;


//Exponent and Regime Computation

wire [E:0] exp_N = exp[E] ? -exp : exp;

wire [es-1:0] e_o = (exp[E] & |exp_N[es-1:0]) ? exp[es-1:0] : exp_N[es-1:0];

wire [E-es-1:0] r_o = (~exp[E] || (exp[E] & |exp_N[es-1:0])) ? {{Bs{1'b0}},exp_N[E-1:es]} +
1'b1 : {{Bs{1'b0}},exp_N[E-1:es]};
```

```
//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = { {N{~exp[E]}}, exp[E], e_o, mant_tmp[N-2:es]};

//Including Regime bits in Exponent-Mantissa Packing

wire [2*N-1:0] tmp1_o;

wire [Bs-1:0] diff_b;

generate

       if(E-es > Bs)  assign diff_b = |r_o[E-es-1:Bs] ? {{(Bs-2){1'b1}},2'b01} : r_o[Bs-1:0];

       else           assign diff_b = r_o;

endgenerate

DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(diff_b), .c(tmp1_o));



//Final Output

wire [N-1:0] tmp1_oN = s_in ? -tmp1_o[N-1:0] : tmp1_o[N-1:0];

assign pos_out = inf_in|zero_in|(~mant_tmp[N-1]) ? {inf_in,{N-1{1'b0}}} : {s_in, tmp1_oN[N-
1:1]};

endmodule

////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////        Neuron 32 input internal            ///////////////////////

////////////////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

module neuron32in (in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14,
in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30,
in31, in32, n_out);



    input [15:0] in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14,
in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30,
in31, in32 ;
```

```
    output [15:0] n_out;
    wire [15:0] IN [1:32];
    reg [15:0] Wt [1:32];
    wire [15:0] Mult [1:32];
    wire [15:0] a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15, a16;
    wire [15:0] add1, add2, add3, add4, add5, add6, add7, add8;
    wire [15:0] add12, add34, add56, add78;
    wire [15:0] add_out1, add_out2, add_out;
    wire start, inf, zero, done;
    wire [15:0] inp1, inp2;
    wire [15:0] sig_out;

//    FP2posit INP1(in1, inp1);
//    FP2posit INP2(in2, inp2);
    //FP_to_posit INP3(in3, inp3);
    //FP_to_posit INP4(in4, inp4);

    ///multiplying input with weights

    assign IN[1] = in1;
    assign IN[2] = in2;
    assign IN[3] = in3;
    assign IN[4] = in4;
    assign IN[5] = in5;
    assign IN[6] = in6;
```

```
assign IN[7] = in7;

assign IN[8] = in8;

assign IN[9] = in9;

assign IN[10] = in10;

assign IN[11] = in11;

assign IN[12] = in12;

assign IN[13] = in13;

assign IN[14] = in14;

assign IN[15] = in15;

assign IN[16] = in16;

assign IN[17] = in17;

assign IN[18] = in18;

assign IN[19] = in19;

assign IN[20] = in20;

assign IN[21] = in21;

assign IN[22] = in22;

assign IN[23] = in23;

assign IN[24] = in24;

assign IN[25] = in25;

assign IN[26] = in26;

assign IN[27] = in27;

assign IN[28] = in28;

assign IN[29] = in29;

assign IN[30] = in30;

assign IN[31] = in31;
```

```
    assign IN[32] = in32;


    genvar i;


    generate for (i = 1; i <= 32; i = i + 1) begin
    posit_mult M(IN[i], Wt[i], start, Mult[i], inf, zero, done);
    end
    endgenerate
//    posit_mult M1(in1, w1, start, m1, inf, zero, done);
//    posit_mult M2(in2, w2, start, m2, inf, zero, done);
//    posit_mult M3(in3, w3, start, m3, inf, zero, done);
//    posit_mult M4(in4, w4, start, m4, inf, zero, done);
//    posit_mult M5(in5, w5, start, m5, inf, zero, done);
//    posit_mult M6(in6, w6, start, m6, inf, zero, done);
//    posit_mult M7(in7, w7, start, m7, inf, zero, done);
//    posit_mult M8(in8, w8, start, m8, inf, zero, done);
    ///adding weighted inputs



    posit_adder A1(Mult[1], Mult[2], start, a1, inf, zero, done);
    posit_adder A2(Mult[3], Mult[4], start, a2, inf, zero, done);
    posit_adder A3(Mult[5], Mult[6], start, a3, inf, zero, done);
    posit_adder A4(Mult[7], Mult[8], start, a4, inf, zero, done);
    posit_adder A5(Mult[9], Mult[10], start, a5, inf, zero, done);
    posit_adder A6(Mult[11], Mult[12], start, a6, inf, zero, done);
```

```
posit_adder A7(Mult[13], Mult[14], start, a7, inf, zero, done);

posit_adder A8(Mult[15], Mult[16], start, a8, inf, zero, done);

posit_adder A9(Mult[17], Mult[18], start, a9, inf, zero, done);

posit_adder A10(Mult[19], Mult[20], start, a10, inf, zero, done);

posit_adder A11(Mult[21], Mult[22], start, a11, inf, zero, done);

posit_adder A12(Mult[23], Mult[24], start, a12, inf, zero, done);

posit_adder A13(Mult[25], Mult[26], start, a13, inf, zero, done);

posit_adder A14(Mult[27], Mult[28], start, a14, inf, zero, done);

posit_adder A15(Mult[29], Mult[30], start, a15, inf, zero, done);

posit_adder A16(Mult[31], Mult[32], start, a16, inf, zero, done);

////////////////////////////////////////////////////////////////

posit_adder A17(a1, a2, start, add1, inf, zero, done);

posit_adder A18(a3, a4, start, add2, inf, zero, done);

posit_adder A19(a5, a6, start, add3, inf, zero, done);

posit_adder A120(a7, a8, start, add4, inf, zero, done);

posit_adder A21(a9, a10, start, add5, inf, zero, done);

posit_adder A22(a11, a12, start, add6, inf, zero, done);

posit_adder A23(a13, a14, start, add7, inf, zero, done);

posit_adder A24(a15, a16, start, add8, inf, zero, done);

////////////////////////////////////////////////////////////////

posit_adder A25(add1, add2, start, add12, inf, zero, done);

posit_adder A26(add3, add4, start, add34, inf, zero, done);

posit_adder A27(add5, add6, start, add56, inf, zero, done);

posit_adder A28(add7, add8, start, add78, inf, zero, done);

////////////////////////////////////////////////////////////////
```

```
    posit_adder A29(add12, add34, start, add_out1, inf, zero, done);
    posit_adder A30(add56, add78, start, add_out2, inf, zero, done);
    ////////////////////////////////////////////////////////////////
    posit_adder Aout(add_out1, add_out2, start, add_out, inf, zero, done);

    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];

    Posit_to_FP P2F(sig_out, n_out);
endmodule

////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////Posit 1 Input Neuron//////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
module sigmoid_1in(sig_in, sig_out);
input [15:0] sig_in;
output [15:0] sig_out;

    assign sig_out[15] = ~sig_in[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = sig_in[14:2];

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////Posit 2 input Neuron////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

module neuronposit(in1, in2, n_out);

    input [15:0] in1, in2;

    output [15:0] n_out;

    wire [15:0] w1, w2;

    wire [15:0] m1, m2;

    wire [15:0] add_out;

    wire start, inf, zero, done;

    wire [15:0] inp1, inp2;

    wire [15:0] sig_out;


//    FP2posit INP1(in1, inp1);

//    FP2posit INP2(in2, inp2);

    //FP_to_posit INP3(in3, inp3);

    //FP_to_posit INP4(in4, inp4);


    ///multiplying input with weights

    posit_mult M1(in1, w1, start, m1, inf, zero, done);

    posit_mult M2(in2, w2, start, m2, inf, zero, done);

    //posit_mult M3(inp3, w3, start, m3, inf, zero, done);

    //posit_mult M4(inp4, w4, start, m4, inf, zero, done);
```

```
    ///adding weighted inputs
    posit_adder A1(m1, m2, start, add_out, inf, zero, done);
    //posit_adder A2(m3, m4, start, a2, inf, zero, done);
    //posit_adder A3(a1, a2, start, add_out, inf, zero, done);

    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];

    assign n_out = sig_out;
    //Posit_to_FP P2F(sig_out, n_out);
endmodule

////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////   Neuron 8 input internal         ////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
`timescale 1ns / 1ps
module neuron8in (in1, in2, in3, in4, in5, in6, in7, in8, n_out);

    input [15:0] in1, in2, in3, in4, in5, in6, in7, in8;
    output [15:0] n_out;
    wire [15:0] w1, w2, w3, w4, w5, w6, w7, w8;
    wire [15:0] m1, m2, m3, m4, m5, m6, m7, m8;
    wire [15:0] a1, a2, a3, a4, add1, add2, add_out;
```

```
    wire start, inf, zero, done;
    wire [15:0] inp1, inp2;
    wire [15:0] sig_out;

//    FP2posit INP1(in1, inp1);
//    FP2posit INP2(in2, inp2);
    //FP_to_posit INP3(in3, inp3);
    //FP_to_posit INP4(in4, inp4);

    ///multiplying input with weights
    posit_mult M1(in1, w1, start, m1, inf, zero, done);
    posit_mult M2(in2, w2, start, m2, inf, zero, done);
    posit_mult M3(in3, w3, start, m3, inf, zero, done);
    posit_mult M4(in4, w4, start, m4, inf, zero, done);
    posit_mult M5(in5, w5, start, m5, inf, zero, done);
    posit_mult M6(in6, w6, start, m6, inf, zero, done);
    posit_mult M7(in7, w7, start, m7, inf, zero, done);
    posit_mult M8(in8, w8, start, m8, inf, zero, done);
    ///adding weighted inputs
    posit_adder A1(m1, m2, start, a1, inf, zero, done);
    posit_adder A2(m3, m4, start, a2, inf, zero, done);
    posit_adder A3(m5, m6, start, a3, inf, zero, done);
    posit_adder A4(m7, m7, start, a4, inf, zero, done);
    posit_adder A5(a1, a2, start, add1, inf, zero, done);
    posit_adder A6(a3, a4, start, add2, inf, zero, done);
```

```
    posit_adder A7(add1, add2, start, add_out, inf, zero, done);


    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];


    assign n_out = sig_out;
    //Posit_to_FP P2F(sig_out, n_out)
endmodule
////////////////////////////////////////////////////////////////////////////////
//////////////////Floating Point to Posit Conversion//////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//module FP_to_posit(in, out);
//function [15:0] log2;
//input reg [15:0] value;
//      begin
//      value = value-1;
//      for (log2=0; value>0; log2=log2+1)
//              value = value>>1;
//              end
//endfunction

//parameter N = 16;
//parameter E = 5;
//parameter es = 3;    //ES_max = E-1
```

```
//parameter BIAS = (2**(E-1))-1;


//parameter Bs = log2(N);


//input [N-1:0] in;
//output [N-1:0] out;


//wire s_in = in[N-1];
//wire [E-1:0] exp_in = in[N-2:N-1-E];
//wire [M-1:0] mant_in = in[M-1:0];
//wire zero_in = ~|{exp_in,mant_in};
//wire inf_in = &exp_in;


//wire [M:0] mant = {|exp_in, mant_in};


//wire [N-1:0] LOD_in = {mant,{E{1'b0}}};
//wire[Bs-1:0] Lshift;
//LOD_N #(.N(N)) uut (.in(LOD_in), .out(Lshift));


//wire[N-1:0] mant_tmp;
//DSR_left_N_S #(.N(N), .S(Bs)) ls (.a(LOD_in),.b(Lshift),.c(mant_tmp));


//wire [E:0] exp = {exp_in[E-1:1], exp_in[0] | (~|exp_in)} - BIAS - Lshift;


////Exponent and Regime Computation
```

```
//wire [E:0] exp_N = exp[E] ? -exp : exp;

//wire [es-1:0] e_o = (exp[E] & |exp_N[es-1:0]) ? exp[es-1:0] : exp_N[es-1:0];

//wire [E-es-1:0] r_o = (~exp[E] || (exp[E] & |exp_N[es-1:0])) ? {{Bs{1'b0}},exp_N[E-1:es]} +
1'b1 : {{Bs{1'b0}},exp_N[E-1:es]};



////Exponent and Mantissa Packing

//wire [2*N-1:0]tmp_o = { {N{~exp[E]}}, exp[E], e_o, mant_tmp[N-2:es]};



////Including Regime bits in Exponent-Mantissa Packing

//wire [2*N-1:0] tmp1_o;

//wire [Bs-1:0] diff_b;

//generate

//      if(E-es > Bs)  assign diff_b = |r_o[E-es-1:Bs] ? {{(Bs-2){1'b1}},2'b01} : r_o[Bs-1:0];

//      else           assign diff_b = r_o;

//endgenerate

//DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(diff_b), .c(tmp1_o));



////Final Output

//wire [N-1:0] tmp1_oN = s_in ? -tmp1_o[N-1:0] : tmp1_o[N-1:0];

//assign out = inf_in|zero_in|(~mant_tmp[N-1]) ? {inf_in,{N-1{1'b0}}} : {s_in, tmp1_oN[N-1:1]};



//endmodule



////////////////////////////////LOD_N/////////////////////////////////////////////////////////

module LOD_N (in, out);

  function [15:0] log2;
```

```
    input reg [15:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction

parameter N = 64;
parameter S = log2(N);
input [N-1:0] in;
output [S-1:0] out;

wire vld;
LOD #(.N(N)) l1 (in, out, vld);
endmodule



module LOD (in, out, vld);
  function [15:0] log2;
    input reg [15:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
```

```
    end
  endfunction



parameter N = 64;
parameter S = log2(N);


   input [N-1:0] in;
   output [S-1:0] out;
   output vld;


  generate
    if (N == 2)
      begin
        assign vld = |in;
        assign out = ~in[1] & in[0];
      end
    else if (N & (N-1))
      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
    else
      begin
        wire [S-2:0] out_l, out_h;
        wire out_vl, out_vh;
        LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
```

```
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule

////////////////////////////////////DSR_left_N_S//////////////////////////////
module DSR_left_N_S(a,b,c);
        parameter N=16;
        parameter S=4;
        input [N-1:0] a;
        input [S-1:0] b;
        output [N-1:0] c;

wire [N-1:0] tmp [S-1:0];
assign tmp[0]  = b[0] ? a << 7'd1  : a;
genvar i;
generate
        for (i=1; i<S; i=i+1)begin:loop_blk
                assign tmp[i] = b[i] ? tmp[i-1] << 2**i : tmp[i-1];
        end
endgenerate
assign c = tmp[S-1];

endmodule
```

```
//////////////////////DSR_right_N_S/////////////////////////////////////////////////

module DSR_right_N_S(a,b,c);

        parameter N=16;

        parameter S=4;

        input [N-1:0] a;

        input [S-1:0] b;

        output [N-1:0] c;



wire [N-1:0] tmp [S-1:0];

assign tmp[0]  = b[0] ? a >> 7'd1  : a;

genvar i;

generate

        for (i=1; i<S; i=i+1)begin:loop_blk

                assign tmp[i] = b[i] ? tmp[i-1] >> 2**i : tmp[i-1];

        end

endgenerate

assign c = tmp[S-1];

endmodule

//////////////////////////////////////////////////////////////////////////////////

//////////////////////Posit Adder/////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////

//`include "DSR_right_N_S.v"

//`include "LOD_N.v"

//`include "LZD_N.v"
```

```
//`include "DSR_left_N_S.v"

//`include "add_N.v"

//`include "sub_N.v"

//`include "data_extract.v"

//`include "add_mantovf.v"

module posit_adder (in1, in2, start, out, inf, zero, done);

function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction

parameter N = 16;      //Posit Word Size

parameter Bs = log2(N);

parameter es = 3;      //Posit Exponent Size



input [N-1:0] in1, in2;

input start;

output [N-1:0] out;

output inf, zero;

output done;



wire start0= start;
```

```
wire s1 = in1[N-1];

wire s2 = in2[N-1];

wire zero_tmp1 = |in1[N-2:0];

wire zero_tmp2 = |in2[N-2:0];

wire inf1 = in1[N-1] & (~zero_tmp1),

       inf2 = in2[N-1] & (~zero_tmp2);

wire zero1 = ~(in1[N-1] | zero_tmp1),

       zero2 = ~(in2[N-1] | zero_tmp2);

assign inf = inf1 | inf2,

       zero = zero1 & zero2;



//Data Extraction

wire rc1, rc2;

wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;

wire [es-1:0] e1, e2;

wire [N-es-1:0] mant1, mant2;

wire [N-1:0] xin1 = s1 ? -in1 : in1;

wire [N-1:0] xin2 = s2 ? -in2 : in2;

data_extract  #(.N(N),.es(es))  uut_de1(.in(xin1),  .rc(rc1),  .regime(regime1),  .exp(e1),
.mant(mant1), .Lshift(Lshift1));

data_extract  #(.N(N),.es(es))  uut_de2(.in(xin2),  .rc(rc2),  .regime(regime2),  .exp(e2),
.mant(mant2), .Lshift(Lshift2));



wire [N-es:0] m1 = {zero_tmp1,mant1},

       m2 = {zero_tmp2,mant2};
```

```
//Large Checking and Assignment

wire in1_gt_in2 = xin1[N-2:0] >= xin2[N-2:0] ? 1'b1 : 1'b0;

wire ls = in1_gt_in2 ? s1 : s2;

wire op = s1 ~^ s2;



wire lrc = in1_gt_in2 ? rc1 : rc2;

wire src = in1_gt_in2 ? rc2 : rc1;



wire [Bs-1:0] lr = in1_gt_in2 ? regime1 : regime2;

wire [Bs-1:0] sr = in1_gt_in2 ? regime2 : regime1;



wire [es-1:0] le = in1_gt_in2 ? e1 : e2;

wire [es-1:0] se = in1_gt_in2 ? e2 : e1;



wire [N-es:0] lm = in1_gt_in2 ? m1 : m2;

wire [N-es:0] sm = in1_gt_in2 ? m2 : m1;



//Exponent Difference: Lower Mantissa Right Shift Amount

wire [Bs:0] r_diff11, r_diff12, r_diff2;

sub_N #(.N(Bs)) uut_sub1 (lr, sr, r_diff11);

add_N #(.N(Bs)) uut_add1 (lr, sr, r_diff12);

sub_N #(.N(Bs)) uut_sub2 (sr, lr, r_diff2);

wire [Bs:0] r_diff =  lrc ? (src ? r_diff11 : r_diff12) : r_diff2;



wire [es+Bs+1:0] diff;
```

```
sub_N #(.N(es+Bs+1)) uut_sub_diff ({r_diff,le}, {{Bs+1{1'b0}},se}, diff);

wire [Bs-1:0] exp_diff = (|diff[es+Bs:Bs]) ? {Bs{1'b1}} : diff[Bs-1:0];



//DSR Right Shifting of Small Mantissa

wire [N-1:0] DSR_right_in;

generate

        if (es >= 2)

        assign DSR_right_in = {sm,{es-1{1'b0}}};

        else

        assign DSR_right_in = sm;

endgenerate



wire [N-1:0] DSR_right_out;

wire [Bs-1:0] DSR_e_diff  = exp_diff;

DSR_right_N_S #(.N(N), .S(Bs))  dsr1(.a(DSR_right_in), .b(DSR_e_diff), .c(DSR_right_out));



//Mantissa Addition

wire [N-1:0] add_m_in1;

generate

        if (es >= 2)

        assign add_m_in1 = {lm,{es-1{1'b0}}};

        else

        assign add_m_in1 = lm;

endgenerate
```

```
wire [N:0] add_m1, add_m2;

add_N #(.N(N)) uut_add_m1 (add_m_in1, DSR_right_out, add_m1);

sub_N #(.N(N)) uut_sub_m2 (add_m_in1, DSR_right_out, add_m2);

wire [N:0] add_m = op ? add_m1 : add_m2;

wire [1:0] mant_ovf = add_m[N:N-1];



//LOD of mantissa addition result

wire [N-1:0] LOD_in = {(add_m[N] | add_m[N-1]), add_m[N-2:0]};

wire [Bs-1:0] left_shift;

LOD_N #(.N(N)) l2(.in(LOD_in), .out(left_shift));



//DSR Left Shifting of mantissa result

wire [N-1:0] DSR_left_out_t;

DSR_left_N_S #(.N(N), .S(Bs)) dsl1(.a(add_m[N:1]), .b(left_shift), .c(DSR_left_out_t));

wire [N-1:0] DSR_left_out = DSR_left_out_t[N-1] ? DSR_left_out_t[N-1:0] : {DSR_left_out_t[N-
2:0],1'b0};

//Exponent and Regime Computation

wire [Bs:0] lr_N = lrc ? {1'b0,lr} : -{1'b0,lr};

wire [es+Bs+1:0] le_o_tmp, le_o;

sub_N #(.N(es+Bs+1)) sub3 ({lr_N,le}, {{es+1{1'b0}},left_shift}, le_o_tmp);

add_mantovf #(es+Bs+1) uut_add_mantovf (le_o_tmp, mant_ovf[1], le_o);



wire [es+Bs:0] le_oN = le_o[es+Bs] ? -le_o : le_o;

wire [es-1:0] e_o = (le_o[es+Bs] & |le_oN[es-1:0]) ? le_o[es-1:0] : le_oN[es-1:0];

wire [Bs-1:0] r_o = (~le_o[es+Bs] || (le_o[es+Bs] & |le_oN[es-1:0])) ? le_oN[es+Bs-1:es] + 1'b1
: le_oN[es+Bs-1:es];
```

```
//Exponent and Mantissa Packing
wire [2*N-1:0]tmp_o = { {N{~le_o[es+Bs]}}, le_o[es+Bs], e_o, DSR_left_out[N-2:es]};
wire [2*N-1:0] tmp1_o;
DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(r_o), .c(tmp1_o));

//Final Output
wire [2*N-1:0] tmp1_oN = ls ? -tmp1_o : tmp1_o;
assign out = inf|zero|(~DSR_left_out[N-1]) ? {inf,{N-1{1'b0}}} : {ls, tmp1_oN[N-1:1]},
        done = start0;
endmodule
////////////////////////////////////////////////////////////////////////////////
////////////////////////////Posit Multiplication////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

module posit_mult (in1, in2, start, out, inf, zero, done);
function [15:0] log2;
input reg [15:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
                value = value>>1;
        end
endfunction
```

```
parameter N = 16;

parameter Bs = log2(N);

parameter es = 3;



input [N-1:0] in1, in2;

input start;

output [N-1:0] out;

output inf, zero;

output done;



wire start0= start;

wire s1 = in1[N-1];

wire s2 = in2[N-1];

wire zero_tmp1 = |in1[N-2:0];

wire zero_tmp2 = |in2[N-2:0];

wire inf1 = in1[N-1] & (~zero_tmp1),

        inf2 = in2[N-1] & (~zero_tmp2);

wire zero1 = ~(in1[N-1] | zero_tmp1),

        zero2 = ~(in2[N-1] | zero_tmp2);

assign inf = inf1 | inf2,

        zero = zero1 & zero2;



//Data Extraction

wire rc1, rc2;

wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;
```

```
wire [es-1:0] e1, e2;

wire [N-es-1:0] mant1, mant2;

wire [N-1:0] xin1 = s1 ? -in1 : in1;

wire [N-1:0] xin2 = s2 ? -in2 : in2;

data_extract  #(.N(N),.es(es))  uut_de1(.in(xin1),  .rc(rc1),  .regime(regime1),  .exp(e1),
.mant(mant1), .Lshift(Lshift1));

data_extract  #(.N(N),.es(es))  uut_de2(.in(xin2),  .rc(rc2),  .regime(regime2),  .exp(e2),
.mant(mant2), .Lshift(Lshift2));



wire [N-es:0] m1 = {zero_tmp1,mant1},

       m2 = {zero_tmp2,mant2};



//Sign, Exponent and Mantissa Computation

wire mult_s = s1 ^ s2;



wire [2*(N-es)+1:0] mult_m = m1*m2;

wire mult_m_ovf = mult_m[2*(N-es)+1];

wire [2*(N-es)+1:0] mult_mN = ~mult_m_ovf ? mult_m << 1'b1 : mult_m;



wire [Bs+1:0] r1 = rc1 ? {2'b0,regime1} : -regime1;

wire [Bs+1:0] r2 = rc2 ? {2'b0,regime2} : -regime2;

wire [Bs+es+1:0] mult_e  =  {r1, e1} + {r2, e2} + mult_m_ovf;



//Exponent and Regime Computation

wire [es+Bs:0] mult_eN = mult_e[es+Bs+1] ? -mult_e : mult_e;

wire [es-1:0] e_o = (mult_e[es+Bs+1] & |mult_eN[es-1:0]) ? mult_e[es-1:0] : mult_eN[es-1:0];
```

```
wire [Bs:0] r_o = (~mult_e[es+Bs+1] || (mult_e[es+Bs+1] & |mult_eN[es-1:0])) ? mult_eN[es+Bs:es]
+ 1'b1 : mult_eN[es+Bs:es];


//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = {{N{~mult_e[es+Bs+1]}},mult_e[es+Bs+1],e_o,mult_mN[2*(N-es):N-es+2]};


//Including Regime bits in Exponent-Mantissa Packing

wire [2*N-1:0] tmp1_o;

DSR_right_N_S #(.N(2*N), .S(Bs+1)) dsr2 (.a(tmp_o), .b(r_o[Bs] ? {Bs{1'b1}} : r_o), .c(tmp1_o));


//Final Output

wire [2*N-1:0] tmp1_oN = mult_s ? -tmp1_o : tmp1_o;

assign out = inf|zero|(~mult_mN[2*(N-es)+1]) ? {inf,{N-1{1'b0}}} : {mult_s, tmp1_oN[N-1:1]},

        done = start0;

endmodule


////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////Posit to FP/////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////


module Posit_to_FP (in, out);

function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)
```

```
                value = value>>1;
        end
endfunction

parameter N = 16;
parameter E = 5;
parameter es = 3;

parameter M = N-E-1;
parameter BIAS = (2**(E-1))-1;
parameter Bs = log2(N);
parameter EO = E > es+Bs ? E : es+Bs;

input [N-1:0] in;
output [N-1:0] out;

wire s = in[N-1];
wire zero_tmp = |in[N-2:0];
wire inf_in = in[N-1] & (~zero_tmp);
wire zero_in = ~(in[N-1] | zero_tmp);

//Data Extraction
wire rc;
wire [Bs-1:0] rgm, Lshift;
wire [es-1:0] e;
```

```
wire [N-es-1:0] mant;

wire [N-1:0] xin = s ? -in : in;

data_extract #(.N(N),.es(es)) uut_de1(.in(xin), .rc(rc), .regime(rgm), .exp(e), .mant(mant),
.Lshift(Lshift));



wire [N-1:0] m = {zero_tmp,mant,{es-1{1'b0}}};



//Exponent and Regime Computation

wire [EO+1:0] e_o;

assign e_o = {(rc ? {{EO-es-Bs+1{1'b0}},rgm} : -{{EO-es-Bs+1{1'b0}},rgm}),e} + BIAS;

//Final Output

assign out = inf_in|e_o[EO:E]|&e_o[E-1:0] ? {s,{E-1{1'b1}},{M{1'b0}}} : (zero_in|(~m[N-1]) ?
{s,{E-1{1'b0}},m[N-2:E]} : { s, e_o[E-1:0], m[N-2:E]} );

endmodule



//////////////////////////LZD//////////////////////////////////////////////////////

module LZD_N (in, out);

  function [15:0] log2;

    input reg [15:0] value;

    begin

      value = value-1;

      for (log2=0; value>0; log2=log2+1)

        value = value>>1;

    end

  endfunction
```

```
parameter N = 64;
parameter S = log2(N);
input [N-1:0] in;
output [S-1:0] out;

wire vld;
LZD #(.N(N)) l1 (in, out, vld);
endmodule

module LZD (in, out, vld);
  function [15:0] log2;
    input reg [15:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction

parameter N = 64;
parameter S = log2(N);

   input [N-1:0] in;
   output [S-1:0] out;
   output vld;
```

```
  generate
    if (N == 2)
      begin
        assign vld = ~&in;
        assign out = in[1] & ~in[0];
      end
    else if (N & (N-1))
      LZD #(1<<S) LZD ({1<<S {1'b0}} | in,out,vld);
    else
      begin
        wire [S-2:0] out_l;
        wire [S-2:0] out_h;
        wire out_vl, out_vh;
        LZD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LZD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule
////////////////////////////////////////////LOD/////////////////////////////////////////////
////module LOD_N (in, out);
////  function [15:0] log2;
////    input reg [15:0] value;
```

```
////    begin
////      value = value-1;
////      for (log2=0; value>0; log2=log2+1)
////   value = value>>1;
////    end
////  endfunction


////parameter N = 64;
////parameter S = log2(N);
////input [N-1:0] in;
////output [S-1:0] out;


////wire vld;
////LOD #(.N(N)) l1 (in, out, vld);
////endmodule


////module LOD (in, out, vld);
////  function [15:0] log2;
////    input reg [15:0] value;
////    begin
////      value = value-1;
////      for (log2=0; value>0; log2=log2+1)
////   value = value>>1;
////    end
////  endfunction
```

```
//parameter N = 64;
//parameter S = log2(N);

//   input [N-1:0] in;
//   output [S-1:0] out;
//   output vld;

//  generate
//    if (N == 2)
//      begin
//      assign vld = |in;
//      assign out = ~in[1] & in[0];
//      end
//    else if (N & (N-1))
//      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
//    else
//      begin
//      wire [S-2:0] out_l, out_h;
//      wire out_vl, out_vh;
//      LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
//      LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
//      assign vld = out_vl | out_vh;
//      assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
```

```
//      end
//  endgenerate
//endmodule


////////////////////////////////END/////////////////////////////////
`timescale 1ns / 1ps
module neuron_posit(in1, in2, in3, in4, n_out);
    input [15:0] in1, in2, in3, in4;
    output [15:0] n_out;
    wire [15:0] w1, w2, w3, w4;
    wire [15:0] m1, m2, m3, m4;
    wire [15:0] a1, a2, add_out;
    wire start, inf, zero, done;
    wire [15:0] inp1, inp2, inp3, inp4;
    wire [15:0] sig_out;

    FP_to_posit INP1(in1, inp1);
    FP_to_posit INP2(in2, inp2);
    FP_to_posit INP3(in3, inp3);
    FP_to_posit INP4(in4, inp4);

    ///multiplying input with weights
    posit_mult M1(inp1, w1, start, m1, inf, zero, done);
    posit_mult M2(inp2, w2, start, m2, inf, zero, done);
    posit_mult M3(inp3, w3, start, m3, inf, zero, done);
```

```
    posit_mult M4(inp4, w4, start, m4, inf, zero, done);

    ///adding weighted inputs
    posit_adder A1(m1, m2, start, a1, inf, zero, done);
    posit_adder A2(m3, m4, start, a2, inf, zero, done);
    posit_adder A3(a1, a2, start, add_out, inf, zero, done);

    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];

    Posit_to_FP P2F(sig_out, n_out);
endmodule

////////////////////////////////////////////////////////////////////////////////
//////////////////Floating Point to Posit Conversion//////////////////////////////
////////////////////////////////////////////////////////////////////////////////

module FP_to_posit(in, out);
function [15:0] log2;
input reg [15:0] value;
        begin
        value = value-1;
        for (log2=0; value>0; log2=log2+1)
```

```
                value = value>>1;

        end

endfunction


parameter N = 16;

parameter E = 5;

parameter es = 3;      //ES_max = E-1

parameter M = N-E-1;

parameter BIAS = (2**(E-1))-1;


parameter Bs = log2(N);


input [N-1:0] in;

output [N-1:0] out;


wire s_in = in[N-1];

wire [E-1:0] exp_in = in[N-2:N-1-E];

wire [M-1:0] mant_in = in[M-1:0];

wire zero_in = ~|{exp_in,mant_in};

wire inf_in = &exp_in;


wire [M:0] mant = {|exp_in, mant_in};


wire [N-1:0] LOD_in = {mant,{E{1'b0}}};

wire[Bs-1:0] Lshift;
```

```
LOD_N #(.N(N)) uut (.in(LOD_in), .out(Lshift));


wire[N-1:0] mant_tmp;

DSR_left_N_S #(.N(N), .S(Bs)) ls (.a(LOD_in),.b(Lshift),.c(mant_tmp));


wire [E:0] exp = {exp_in[E-1:1], exp_in[0] | (~|exp_in)} - BIAS - Lshift;


//Exponent and Regime Computation

wire [E:0] exp_N = exp[E] ? -exp : exp;

wire [es-1:0] e_o = (exp[E] & |exp_N[es-1:0]) ? exp[es-1:0] : exp_N[es-1:0];

wire [E-es-1:0] r_o = (~exp[E] || (exp[E] & |exp_N[es-1:0])) ? {{Bs{1'b0}},exp_N[E-1:es]} +
1'b1 : {{Bs{1'b0}},exp_N[E-1:es]};


//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = { {N{~exp[E]}}, exp[E], e_o, mant_tmp[N-2:es]};


//Including Regime bits in Exponent-Mantissa Packing

wire [2*N-1:0] tmp1_o;

wire [Bs-1:0] diff_b;

generate

       if(E-es > Bs)  assign diff_b = |r_o[E-es-1:Bs] ? {{(Bs-2){1'b1}},2'b01} : r_o[Bs-1:0];

       else           assign diff_b = r_o;

endgenerate

DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(diff_b), .c(tmp1_o));


//Final Output
```

```
wire [N-1:0] tmp1_oN = s_in ? -tmp1_o[N-1:0] : tmp1_o[N-1:0];

assign out = inf_in|zero_in|(~mant_tmp[N-1]) ? {inf_in,{N-1{1'b0}}} : {s_in, tmp1_oN[N-1:1]};

endmodule



////////////////////////////////LOD_N/////////////////////////////////////////////////////////

module LOD_N (in, out);

  function [15:0] log2;

    input reg [15:0] value;

    begin

      value = value-1;

      for (log2=0; value>0; log2=log2+1)

        value = value>>1;

    end

  endfunction



parameter N = 64;

parameter S = log2(N);

input [N-1:0] in;

output [S-1:0] out;



wire vld;

LOD #(.N(N)) l1 (in, out, vld);

endmodule
```

```
module LOD (in, out, vld);
  function [15:0] log2;
    input reg [15:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction
parameter N = 64;
parameter S = log2(N);

   input [N-1:0] in;
   output [S-1:0] out;
   output vld;

  generate
    if (N == 2)
      begin
        assign vld = |in;
        assign out = ~in[1] & in[0];
      end
    else if (N & (N-1))
      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
    else
```

```
      begin
        wire [S-2:0] out_l, out_h;
        wire out_vl, out_vh;
        LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule
////////////////////////////////////DSR_left_N_S//////////////////////////////
module DSR_left_N_S(a,b,c);
        parameter N=16;
        parameter S=4;
        input [N-1:0] a;
        input [S-1:0] b;
        output [N-1:0] c;

wire [N-1:0] tmp [S-1:0];
assign tmp[0]  = b[0] ? a << 7'd1  : a;
genvar i;
generate
        for (i=1; i<S; i=i+1)begin:loop_blk
                assign tmp[i] = b[i] ? tmp[i-1] << 2**i : tmp[i-1];
        end
```

```
endgenerate

assign c = tmp[S-1];

endmodule



//////////////////////DSR_right_N_S/////////////////////////////////////////////

module DSR_right_N_S(a,b,c);

        parameter N=16;

        parameter S=4;

        input [N-1:0] a;

        input [S-1:0] b;

        output [N-1:0] c;



wire [N-1:0] tmp [S-1:0];

assign tmp[0]  = b[0] ? a >> 7'd1  : a;

genvar i;

generate

        for (i=1; i<S; i=i+1)begin:loop_blk

                assign tmp[i] = b[i] ? tmp[i-1] >> 2**i : tmp[i-1];

        end

endgenerate

assign c = tmp[S-1];

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////

//////////////////////Posit Adder///////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////

//`include "DSR_right_N_S.v"

//`include "LOD_N.v"

//`include "LZD_N.v"

//`include "DSR_left_N_S.v"

//`include "add_N.v"

//`include "sub_N.v"

//`include "data_extract.v"

//`include "add_mantovf.v"

module posit_adder (in1, in2, start, out, inf, zero, done);

function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction



parameter N = 16;      //Posit Word Size

parameter Bs = log2(N);

parameter es = 3;      //Posit Exponent Size
```

```
input [N-1:0] in1, in2;

input start;

output [N-1:0] out;

output inf, zero;

output done;



wire start0= start;

wire s1 = in1[N-1];

wire s2 = in2[N-1];

wire zero_tmp1 = |in1[N-2:0];

wire zero_tmp2 = |in2[N-2:0];

wire inf1 = in1[N-1] & (~zero_tmp1),

        inf2 = in2[N-1] & (~zero_tmp2);

wire zero1 = ~(in1[N-1] | zero_tmp1),

        zero2 = ~(in2[N-1] | zero_tmp2);

assign inf = inf1 | inf2,

        zero = zero1 & zero2;



//Data Extraction

wire rc1, rc2;

wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;

wire [es-1:0] e1, e2;

wire [N-es-1:0] mant1, mant2;

wire [N-1:0] xin1 = s1 ? -in1 : in1;

wire [N-1:0] xin2 = s2 ? -in2 : in2;
```

```
data_extract  #(.N(N),.es(es))  uut_de1(.in(xin1),  .rc(rc1),  .regime(regime1),  .exp(e1),
.mant(mant1), .Lshift(Lshift1));

data_extract  #(.N(N),.es(es))  uut_de2(.in(xin2),  .rc(rc2),  .regime(regime2),  .exp(e2),
.mant(mant2), .Lshift(Lshift2));



wire [N-es:0] m1 = {zero_tmp1,mant1},

       m2 = {zero_tmp2,mant2};



//Large Checking and Assignment

wire in1_gt_in2 = xin1[N-2:0] >= xin2[N-2:0] ? 1'b1 : 1'b0;



wire ls = in1_gt_in2 ? s1 : s2;

wire op = s1 ~^ s2;



wire lrc = in1_gt_in2 ? rc1 : rc2;

wire src = in1_gt_in2 ? rc2 : rc1;



wire [Bs-1:0] lr = in1_gt_in2 ? regime1 : regime2;

wire [Bs-1:0] sr = in1_gt_in2 ? regime2 : regime1;



wire [es-1:0] le = in1_gt_in2 ? e1 : e2;

wire [es-1:0] se = in1_gt_in2 ? e2 : e1;



wire [N-es:0] lm = in1_gt_in2 ? m1 : m2;

wire [N-es:0] sm = in1_gt_in2 ? m2 : m1;
```

```
//Exponent Difference: Lower Mantissa Right Shift Amount

wire [Bs:0] r_diff11, r_diff12, r_diff2;

sub_N #(.N(Bs)) uut_sub1 (lr, sr, r_diff11);

add_N #(.N(Bs)) uut_add1 (lr, sr, r_diff12);

sub_N #(.N(Bs)) uut_sub2 (sr, lr, r_diff2);

wire [Bs:0] r_diff =  lrc ? (src ? r_diff11 : r_diff12) : r_diff2;



wire [es+Bs+1:0] diff;

sub_N #(.N(es+Bs+1)) uut_sub_diff ({r_diff,le}, {{Bs+1{1'b0}},se}, diff);

wire [Bs-1:0] exp_diff = (|diff[es+Bs:Bs]) ? {Bs{1'b1}} : diff[Bs-1:0];



//DSR Right Shifting of Small Mantissa

wire [N-1:0] DSR_right_in;

generate

        if (es >= 2)

        assign DSR_right_in = {sm,{es-1{1'b0}}};

        else

        assign DSR_right_in = sm;

endgenerate



wire [N-1:0] DSR_right_out;

wire [Bs-1:0] DSR_e_diff  = exp_diff;

DSR_right_N_S #(.N(N), .S(Bs))  dsr1(.a(DSR_right_in), .b(DSR_e_diff), .c(DSR_right_out));



//Mantissa Addition
```

```
wire [N-1:0] add_m_in1;

generate

        if (es >= 2)

        assign add_m_in1 = {lm,{es-1{1'b0}}};

        else

        assign add_m_in1 = lm;

endgenerate



wire [N:0] add_m1, add_m2;

add_N #(.N(N)) uut_add_m1 (add_m_in1, DSR_right_out, add_m1);

sub_N #(.N(N)) uut_sub_m2 (add_m_in1, DSR_right_out, add_m2);

wire [N:0] add_m = op ? add_m1 : add_m2;

wire [1:0] mant_ovf = add_m[N:N-1];



//LOD of mantissa addition result

wire [N-1:0] LOD_in = {(add_m[N] | add_m[N-1]), add_m[N-2:0]};

wire [Bs-1:0] left_shift;

LOD_N #(.N(N)) l2(.in(LOD_in), .out(left_shift));



//DSR Left Shifting of mantissa result

wire [N-1:0] DSR_left_out_t;

DSR_left_N_S #(.N(N), .S(Bs)) dsl1(.a(add_m[N:1]), .b(left_shift), .c(DSR_left_out_t));

wire [N-1:0] DSR_left_out = DSR_left_out_t[N-1] ? DSR_left_out_t[N-1:0] : {DSR_left_out_t[N-
2:0],1'b0};
```

```
//Exponent and Regime Computation

wire [Bs:0] lr_N = lrc ? {1'b0,lr} : -{1'b0,lr};

wire [es+Bs+1:0] le_o_tmp, le_o;

sub_N #(.N(es+Bs+1)) sub3 ({lr_N,le}, {{es+1{1'b0}},left_shift}, le_o_tmp);

add_mantovf #(es+Bs+1) uut_add_mantovf (le_o_tmp, mant_ovf[1], le_o);



wire [es+Bs:0] le_oN = le_o[es+Bs] ? -le_o : le_o;

wire [es-1:0] e_o = (le_o[es+Bs] & |le_oN[es-1:0]) ? le_o[es-1:0] : le_oN[es-1:0];

wire [Bs-1:0] r_o = (~le_o[es+Bs] || (le_o[es+Bs] & |le_oN[es-1:0])) ? le_oN[es+Bs-1:es] + 1'b1
: le_oN[es+Bs-1:es];



//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = { {N{~le_o[es+Bs]}}, le_o[es+Bs], e_o, DSR_left_out[N-2:es]};

wire [2*N-1:0] tmp1_o;

DSR_right_N_S #(.N(2*N), .S(Bs)) dsr2 (.a(tmp_o), .b(r_o), .c(tmp1_o));



//Final Output

wire [2*N-1:0] tmp1_oN = ls ? -tmp1_o : tmp1_o;

assign out = inf|zero|(~DSR_left_out[N-1]) ? {inf,{N-1{1'b0}}} : {ls, tmp1_oN[N-1:1]},

       done = start0;

endmodule



////////////////////////////////////////////////////////////////////////////////

//////////////////////////////Posit Multiplication//////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
```

```
module posit_mult (in1, in2, start, out, inf, zero, done);


function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction


parameter N = 16;

parameter Bs = log2(N);

parameter es = 3;


input [N-1:0] in1, in2;

input start;

output [N-1:0] out;

output inf, zero;

output done;


wire start0= start;

wire s1 = in1[N-1];

wire s2 = in2[N-1];

wire zero_tmp1 = |in1[N-2:0];
```

```
wire zero_tmp2 = |in2[N-2:0];

wire inf1 = in1[N-1] & (~zero_tmp1),

       inf2 = in2[N-1] & (~zero_tmp2);

wire zero1 = ~(in1[N-1] | zero_tmp1),

       zero2 = ~(in2[N-1] | zero_tmp2);

assign inf = inf1 | inf2,

       zero = zero1 & zero2;



//Data Extraction

wire rc1, rc2;

wire [Bs-1:0] regime1, regime2, Lshift1, Lshift2;

wire [es-1:0] e1, e2;

wire [N-es-1:0] mant1, mant2;

wire [N-1:0] xin1 = s1 ? -in1 : in1;

wire [N-1:0] xin2 = s2 ? -in2 : in2;

data_extract  #(.N(N),.es(es))  uut_de1(.in(xin1),  .rc(rc1),  .regime(regime1),  .exp(e1),
.mant(mant1), .Lshift(Lshift1));

data_extract  #(.N(N),.es(es))  uut_de2(.in(xin2),  .rc(rc2),  .regime(regime2),  .exp(e2),
.mant(mant2), .Lshift(Lshift2));



wire [N-es:0] m1 = {zero_tmp1,mant1},

       m2 = {zero_tmp2,mant2};



//Sign, Exponent and Mantissa Computation

wire mult_s = s1 ^ s2;
```

```
wire [2*(N-es)+1:0] mult_m = m1*m2;

wire mult_m_ovf = mult_m[2*(N-es)+1];

wire [2*(N-es)+1:0] mult_mN = ~mult_m_ovf ? mult_m << 1'b1 : mult_m;



wire [Bs+1:0] r1 = rc1 ? {2'b0,regime1} : -regime1;

wire [Bs+1:0] r2 = rc2 ? {2'b0,regime2} : -regime2;

wire [Bs+es+1:0] mult_e  =  {r1, e1} + {r2, e2} + mult_m_ovf;



//Exponent and Regime Computation

wire [es+Bs:0] mult_eN = mult_e[es+Bs+1] ? -mult_e : mult_e;

wire [es-1:0] e_o = (mult_e[es+Bs+1] & |mult_eN[es-1:0]) ? mult_e[es-1:0] : mult_eN[es-1:0];

wire [Bs:0] r_o = (~mult_e[es+Bs+1] || (mult_e[es+Bs+1] & |mult_eN[es-1:0])) ? mult_eN[es+Bs:es]
+ 1'b1 : mult_eN[es+Bs:es];



//Exponent and Mantissa Packing

wire [2*N-1:0]tmp_o = {{N{~mult_e[es+Bs+1]}},mult_e[es+Bs+1],e_o,mult_mN[2*(N-es):N-es+2]};




//Including Regime bits in Exponent-Mantissa Packing

wire [2*N-1:0] tmp1_o;

DSR_right_N_S #(.N(2*N), .S(Bs+1)) dsr2 (.a(tmp_o), .b(r_o[Bs] ? {Bs{1'b1}} : r_o), .c(tmp1_o));




//Final Output

wire [2*N-1:0] tmp1_oN = mult_s ? -tmp1_o : tmp1_o;

assign out = inf|zero|(~mult_mN[2*(N-es)+1]) ? {inf,{N-1{1'b0}}} : {mult_s, tmp1_oN[N-1:1]},
```

```
        done = start0;


endmodule


////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////Posit to FP ////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////


module Posit_to_FP (in, out);

function [15:0] log2;

input reg [15:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction


parameter N = 16;

parameter E = 5;

parameter es = 3;


parameter M = N-E-1;

parameter BIAS = (2**(E-1))-1;

parameter Bs = log2(N);
```

```
parameter EO = E > es+Bs ? E : es+Bs;

input [N-1:0] in;

output [N-1:0] out;



wire s = in[N-1];

wire zero_tmp = |in[N-2:0];

wire inf_in = in[N-1] & (~zero_tmp);

wire zero_in = ~(in[N-1] | zero_tmp);



//Data Extraction

wire rc;

wire [Bs-1:0] rgm, Lshift;

wire [es-1:0] e;

wire [N-es-1:0] mant;

wire [N-1:0] xin = s ? -in : in;

data_extract #(.N(N),.es(es)) uut_de1(.in(xin), .rc(rc), .regime(rgm), .exp(e), .mant(mant),
.Lshift(Lshift));



wire [N-1:0] m = {zero_tmp,mant,{es-1{1'b0}}};



//Exponent and Regime Computation

wire [EO+1:0] e_o;

assign e_o = {(rc ? {{EO-es-Bs+1{1'b0}},rgm} : -{{EO-es-Bs+1{1'b0}},rgm}),e} + BIAS;

//Final Output

assign out = inf_in|e_o[EO:E]|&e_o[E-1:0] ? {s,{E-1{1'b1}},{M{1'b0}}} : (zero_in|(~m[N-1]) ?
{s,{E-1{1'b0}},m[N-2:E]} : { s, e_o[E-1:0], m[N-2:E]} );
```

```
endmodule


//////////////////////////LZD////////////////////////////////////////////////////

module LZD_N (in, out);

  function [15:0] log2;

    input reg [15:0] value;

    begin

      value = value-1;

      for (log2=0; value>0; log2=log2+1)

        value = value>>1;

    end

  endfunction


parameter N = 64;

parameter S = log2(N);

input [N-1:0] in;

output [S-1:0] out;


wire vld;

LZD #(.N(N)) l1 (in, out, vld);

endmodule
```

```
module LZD (in, out, vld);


  function [15:0] log2;
    input reg [15:0] value;
    begin
      value = value-1;
      for (log2=0; value>0; log2=log2+1)
        value = value>>1;
    end
  endfunction


parameter N = 64;
parameter S = log2(N);


   input [N-1:0] in;
   output [S-1:0] out;
   output vld;
  generate
    if (N == 2)
      begin
        assign vld = ~&in;
        assign out = in[1] & ~in[0];
      end
    else if (N & (N-1))
      LZD #(1<<S) LZD ({1<<S {1'b0}} | in,out,vld);
```

```
    else
      begin
        wire [S-2:0] out_l;
        wire [S-2:0] out_h;
        wire out_vl, out_vh;
        LZD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
        LZD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
        assign vld = out_vl | out_vh;
        assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
      end
  endgenerate
endmodule

//////////////////////////////////////////////LOD//////////////////////////////////////////////
////module LOD_N (in, out);
////  function [15:0] log2;
////    input reg [15:0] value;
////    begin
////      value = value-1;
////      for (log2=0; value>0; log2=log2+1)
////    value = value>>1;
////    end
////  endfunction

////parameter N = 64;
```

```
////parameter S = log2(N);

////input [N-1:0] in;

////output [S-1:0] out;



////wire vld;

////LOD #(.N(N)) l1 (in, out, vld);

////endmodule



////module LOD (in, out, vld);

////  function [15:0] log2;

////    input reg [15:0] value;

////    begin

////      value = value-1;

////      for (log2=0; value>0; log2=log2+1)

////    value = value>>1;

////    end

////  endfunction



//parameter N = 64;

//parameter S = log2(N);



//   input [N-1:0] in;

//   output [S-1:0] out;

//   output vld;
```

```
//  generate
//    if (N == 2)
//      begin
//      assign vld = |in;
//      assign out = ~in[1] & in[0];
//      end
//    else if (N & (N-1))
//      LOD #(1<<S) LOD ({1<<S {1'b0}} | in,out,vld);
//    else
//      begin
//      wire [S-2:0] out_l, out_h;
//      wire out_vl, out_vh;
//      LOD #(N>>1) l(in[(N>>1)-1:0],out_l,out_vl);
//      LOD #(N>>1) h(in[N-1:N>>1],out_h,out_vh);
//      assign vld = out_vl | out_vh;
//      assign out = out_vh ? {1'b0,out_h} : {out_vl,out_l};
//      end
//  endgenerate
//endmodule



////////////////////////////////END////////////////////////////////////
```

C.2. Verilog Code for Classification ANN

```
////////////////////////////////////////////////////////////////////////////////

//////////////////////       Classification C-ANN        ////////////////////////////

////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

module ClasANN(pH, ORP, DO, EC, Pot, Agri, Waste);



input [15:0] pH, ORP, DO, EC;

output [15:0] Pot, Agri, Waste;



wire [15:0] in1, in2, in3, in4;

wire [15:0] l [1:32];

wire [15:0] m [1:32];

wire [15:0] n [1:32];

wire [15:0] Pot_out, Agri_out, Waste_out;



sigmoid_1in I1(pH, in1);

sigmoid_1in I2(ORP, in2);

sigmoid_1in I3(DO, in3);

sigmoid_1in I4(EC, in4);



genvar i;

generate

    for (i = 1; i <= 32; i = i + 1) begin

        neuron4in L(in1, in2, in3, in4, l[i]);
```

```
        end

endgenerate

generate

    for (i = 1; i <= 32 ; i = i + 1) begin

            neuron32in M(l[1], l[2], l[3], l[4], l[5], l[6], l[7], l[8], l[9], l[10], l[11],
l[12], l[13], l[14], l[15], l[16], l[17], l[18], l[19], l[20], l[21], l[22], l[23], l[24],
l[25], l[26], l[27], l[28], l[29], l[30], l[31], l[32], m[i] );

    end

endgenerate

generate

    for (i = 1; i <= 32 ; i = i + 1) begin

            neuron32in N(m[1], m[2], m[3], m[4], m[5], m[6], m[7], m[8], m[9], m[10], m[11],
m[12], m[13], m[14], m[15], m[16], m[17], m[18], m[19], m[20], m[21], m[22], m[23], m[24],
m[25], m[26], m[27], m[28], m[29], m[30], m[31], m[32], n[i] );

    end

endgenerate



neuron32in P(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8],n[9], n[10], n[11], n[12], n[13],
n[14], n[15], n[16],n[17], n[18], n[19], n[20], n[21], n[22], n[23], n[24],n[25], n[26], n[27],
n[28], n[29], n[30], n[31], n[32], Pot_out);

neuron32in A(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8],n[9], n[10], n[11], n[12], n[13],
n[14], n[15], n[16],n[17], n[18], n[19], n[20], n[21], n[22], n[23], n[24],n[25], n[26], n[27],
n[28], n[29], n[30], n[31], n[32], Agri_out);

neuron32in W(n[1], n[2], n[3], n[4], n[5], n[6], n[7], n[8],n[9], n[10], n[11], n[12], n[13],
n[14], n[15], n[16],n[17], n[18], n[19], n[20], n[21], n[22], n[23], n[24],n[25], n[26], n[27],
n[28], n[29], n[30], n[31], n[32], Waste_out);



Posit_to_FP POT(Pot_out, Pot);

Posit_to_FP AGRI(Agri_out, Agri);

Posit_to_FP WASTE(Waste_out, Waste);
```

```
endmodule


////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////          Neuron 32 input internal            //////////////////////

////////////////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps

module neuron32in (in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14,
in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30,
in31, in32, n_out);



    input [31:0] in1, in2, in3, in4, in5, in6, in7, in8, in9, in10, in11, in12, in13, in14,
in15, in16, in17, in18, in19, in20, in21, in22, in23, in24, in25, in26, in27, in28, in29, in30,
in31, in32 ;

    output [31:0] n_out;

    wire [15:0] IN [1:32];

    reg [15:0] Wt [1:32];

    wire [15:0] Mult [1:32];

    wire [15:0] a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14, a15, a16;

    wire [15:0] add1, add2, add3, add4, add5, add6, add7, add8;

    wire [15:0] add12, add34, add56, add78;

    wire [15:0] add_out1, add_out2, add_out;

    wire start, inf, zero, done;

    wire [15:0] inp1, inp2;

    wire [15:0] sig_out;



//    FP2posit INP1(in1, inp1);

//    FP2posit INP2(in2, inp2);
```

```
//FP_to_posit INP3(in3, inp3);

//FP_to_posit INP4(in4, inp4);


///multiplying input with weights


assign IN[1] = in1;

assign IN[2] = in2;

assign IN[3] = in3;

assign IN[4] = in4;

assign IN[5] = in5;

assign IN[6] = in6;

assign IN[7] = in7;

assign IN[8] = in8;

assign IN[9] = in9;

assign IN[10] = in10;

assign IN[11] = in11;

assign IN[12] = in12;

assign IN[13] = in13;

assign IN[14] = in14;

assign IN[15] = in15;

assign IN[16] = in16;

assign IN[17] = in17;

assign IN[18] = in18;

assign IN[19] = in19;

assign IN[20] = in20;
```

```
    assign IN[21] = in21;
    assign IN[22] = in22;
    assign IN[23] = in23;
    assign IN[24] = in24;
    assign IN[25] = in25;
    assign IN[26] = in26;
    assign IN[27] = in27;
    assign IN[28] = in28;
    assign IN[29] = in29;
    assign IN[30] = in30;
    assign IN[31] = in31;
    assign IN[32] = in32;


    genvar i;
    generate for (i = 1; i <= 32; i = i + 1) begin
    posit_mult M(IN[i], Wt[i], start, Mult[i], inf, zero, done);
    end
    endgenerate
//    posit_mult M1(in1, w1, start, m1, inf, zero, done);
//    posit_mult M2(in2, w2, start, m2, inf, zero, done);
//    posit_mult M3(in3, w3, start, m3, inf, zero, done);
//    posit_mult M4(in4, w4, start, m4, inf, zero, done);
//    posit_mult M5(in5, w5, start, m5, inf, zero, done);
//    posit_mult M6(in6, w6, start, m6, inf, zero, done);
//    posit_mult M7(in7, w7, start, m7, inf, zero, done);
```

```
//    posit_mult M8(in8, w8, start, m8, inf, zero, done);

    ///adding weighted inputs

    posit_adder A1(Mult[1], Mult[2], start, a1, inf, zero, done);

    posit_adder A2(Mult[3], Mult[4], start, a2, inf, zero, done);

    posit_adder A3(Mult[5], Mult[6], start, a3, inf, zero, done);

    posit_adder A4(Mult[7], Mult[8], start, a4, inf, zero, done);

    posit_adder A5(Mult[9], Mult[10], start, a5, inf, zero, done);

    posit_adder A6(Mult[11], Mult[12], start, a6, inf, zero, done);

    posit_adder A7(Mult[13], Mult[14], start, a7, inf, zero, done);

    posit_adder A8(Mult[15], Mult[16], start, a8, inf, zero, done);

    posit_adder A9(Mult[17], Mult[18], start, a9, inf, zero, done);

    posit_adder A10(Mult[19], Mult[20], start, a10, inf, zero, done);

    posit_adder A11(Mult[21], Mult[22], start, a11, inf, zero, done);

    posit_adder A12(Mult[23], Mult[24], start, a12, inf, zero, done);

    posit_adder A13(Mult[25], Mult[26], start, a13, inf, zero, done);

    posit_adder A14(Mult[27], Mult[28], start, a14, inf, zero, done);

    posit_adder A15(Mult[29], Mult[30], start, a15, inf, zero, done);

    posit_adder A16(Mult[31], Mult[32], start, a16, inf, zero, done);

    ////////////////////////////////////////////////////////////////

    posit_adder A17(a1, a2, start, add1, inf, zero, done);

    posit_adder A18(a3, a4, start, add2, inf, zero, done);

    posit_adder A19(a5, a6, start, add3, inf, zero, done);

    posit_adder A120(a7, a8, start, add4, inf, zero, done);

    posit_adder A21(a9, a10, start, add5, inf, zero, done);

    posit_adder A22(a11, a12, start, add6, inf, zero, done);
```

```
    posit_adder A23(a13, a14, start, add7, inf, zero, done);
    posit_adder A24(a15, a16, start, add8, inf, zero, done);
    ////////////////////////////////////////////////////////////////
    posit_adder A25(add1, add2, start, add12, inf, zero, done);
    posit_adder A26(add3, add4, start, add34, inf, zero, done);
    posit_adder A27(add5, add6, start, add56, inf, zero, done);
    posit_adder A28(add7, add8, start, add78, inf, zero, done);
    ////////////////////////////////////////////////////////////////
    posit_adder A29(add12, add34, start, add_out1, inf, zero, done);
    posit_adder A30(add56, add78, start, add_out2, inf, zero, done);
    ////////////////////////////////////////////////////////////////
    posit_adder Aout(add_out1, add_out2, start, add_out, inf, zero, done);


    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];


    assign n_out = sig_out;
    //Posit_to_FP P2F(sig_out, n_out);
endmodule


////////////////////////////////////////////////////////////////////////////////////////////////
/////////////////////////////////Posit 1 Input Neuron///////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////
```

```
module sigmoid_1in(sig_in, sig_out);
input [15:0] sig_in;
output [15:0] sig_out;

    assign sig_out[15] = ~sig_in[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = sig_in[14:2];
endmodule

////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////Posit 4 input Neuron////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps
module neuron4in(in1, in2, in3, in4, n_out);
    input [31:0] in1, in2, in3, in4;
    output [31:0] n_out;
    wire [15:0] w1, w2, w3, w4;
    wire [15:0] m1, m2, m3, m4;
    wire [15:0] a1, a2, add_out;
    wire start, inf, zero, done;
    //wire [15:0] inp1, inp2;
    wire [15:0] sig_out;

//    FP2posit INP1(in1, inp1);
```

```
//    FP2posit INP2(in2, inp2);

    //FP_to_posit INP3(in3, inp3);

    //FP_to_posit INP4(in4, inp4);



    ///multiplying input with weights

    posit_mult M1(in1, w1, start, m1, inf, zero, done);

    posit_mult M2(in2, w2, start, m2, inf, zero, done);

    posit_mult M3(in3, w3, start, m3, inf, zero, done);

    posit_mult M4(in4, w4, start, m4, inf, zero, done);



    ///adding weighted inputs

    posit_adder A1(m1, m2, start, a1_out, inf, zero, done);

    posit_adder A2(m3, m4, start, a2, inf, zero, done);

    posit_adder A3(a1, a2, start, add_out, inf, zero, done);



    assign sig_out[15] = ~add_out[15];

    assign sig_out[14:13] = 2'b00;

    assign sig_out[12:0] = add_out[14:2];



    assign n_out = sig_out;

    //Posit_to_FP P2F(sig_out, n_out);

endmodule

////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////Neuron 8 input internal///////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////////
```

```
`timescale 1ns / 1ps

module neuron8in (in1, in2, in3, in4, in5, in6, in7, in8, n_out);

    input [31:0] in1, in2, in3, in4, in5, in6, in7, in8;

    output [31:0] n_out;

    wire [15:0] w1, w2, w3, w4, w5, w6, w7, w8;

    wire [15:0] m1, m2, m3, m4, m5, m6, m7, m8;

    wire [15:0] a1, a2, a3, a4, add1, add2, add_out;

    wire start, inf, zero, done;

    wire [15:0] inp1, inp2;

    wire [15:0] sig_out;


//    FP2posit INP1(in1, inp1);

//    FP2posit INP2(in2, inp2);

    //FP_to_posit INP3(in3, inp3);

    //FP_to_posit INP4(in4, inp4);


    ///multiplying input with weights

    posit_mult M1(in1, w1, start, m1, inf, zero, done);

    posit_mult M2(in2, w2, start, m2, inf, zero, done);

    posit_mult M3(in3, w3, start, m3, inf, zero, done);

    posit_mult M4(in4, w4, start, m4, inf, zero, done);

    posit_mult M5(in5, w5, start, m5, inf, zero, done);

    posit_mult M6(in6, w6, start, m6, inf, zero, done);

    posit_mult M7(in7, w7, start, m7, inf, zero, done);
```

```
    posit_mult M8(in8, w8, start, m8, inf, zero, done);
    ///adding weighted inputs
    posit_adder A1(m1, m2, start, a1, inf, zero, done);
    posit_adder A2(m3, m4, start, a2, inf, zero, done);
    posit_adder A3(m5, m6, start, a3, inf, zero, done);
    posit_adder A4(m7, m7, start, a4, inf, zero, done);
    posit_adder A5(a1, a2, start, add1, inf, zero, done);
    posit_adder A6(a3, a4, start, add2, inf, zero, done);
    posit_adder A7(add1, add2, start, add_out, inf, zero, done);

    assign sig_out[15] = ~add_out[15];
    assign sig_out[14:13] = 2'b00;
    assign sig_out[12:0] = add_out[14:2];
    assign n_out = sig_out;
    //Posit_to_FP P2F(sig_out, n_out);
endmodule
////////////////////////////////////////////////////////////////////////////////
module add_mantovf (a,mant_ovf,c);
parameter N=10;
input [N:0] a;
input mant_ovf;
output [N:0] c;
assign c = a + mant_ovf;
endmodule
```

```
////////////////////////////////////////////////////////////////////////////////

module add_N (a,b,c);

parameter N=10;

input [N-1:0] a,b;

output [N:0] c;

assign c = {1'b0,a} + {1'b0,b};

endmodule

////////////////////////////////////////////////////////////////////////////////

module sub_N (a,b,c);

parameter N=10;

input [N-1:0] a,b;

output [N:0] c;

assign c = {1'b0,a} - {1'b0,b};

endmodule

////////////////////////////////////////////////////////////////////////////////

module data_extract(in, rc, regime, exp, mant, Lshift);

function [31:0] log2;

input reg [31:0] value;

        begin

        value = value-1;

        for (log2=0; value>0; log2=log2+1)

                value = value>>1;

        end

endfunction
```

```
parameter N=16;

parameter Bs=log2(N);

parameter es = 2;

input [N-1:0] in;

output rc;

output [Bs-1:0] regime, Lshift;

output [es-1:0] exp;

output [N-es-1:0] mant;



wire [N-1:0] xin = in;

assign rc = xin[N-2];

wire [Bs-1:0] k0, k1;

LOD_N #(.N(N)) xinst_k0(.in({xin[N-2:0],1'b0}), .out(k0));

LZD_N #(.N(N)) xinst_k1(.in({xin[N-3:0],2'b0}), .out(k1));



assign regime = xin[N-2] ? k1 : k0;

assign Lshift = xin[N-2] ? k1+1 : k0;



wire [N-1:0] xin_tmp;

DSR_left_N_S #(.N(N), .S(Bs)) ls (.a({xin[N-3:0],2'b0}),.b(Lshift),.c(xin_tmp));

assign exp= xin_tmp[N-1:N-es];

assign mant= xin_tmp[N-es-1:0];

endmodule



////////////////////////////////////////////////////////////////////////////////
```

C.3. Python Code for Augmentation ANN

```
#!/usr/bin/env python

# coding: utf-8



# In[58]:



#import the pandas module

import pandas as pd

#import the numpy module

import numpy as np



colnames=['I1', 'I2', 'I3', 'T1', 'T2', 'P1', 'P2']



# In[59]:



#Reading and exploring the data by coverting to pandas dataframe

df = pd.read_excel(r'C:\Users\user\Desktop\watertesting.xlsx')

print(df.head(5))



# In[67]:



#splitting into train-test set for model training of the data from test2.tsv file (0.9 test,,
0.1 train)

from sklearn.model_selection import train_test_split

y = df.drop(['I1', 'I2', 'I3', 'T1', 'T2'], axis =1)
```

```
X = df.drop(['T1', 'T2', 'P1', 'P2'], axis =1)


X = df[['I1','I2','I3']]

y = df[['P1','P2']]


print(X.head())

print(y.head())


'''X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2)

print("\nX_train:\n")

print(X_train.head())

print(X_train.shape)


print("\nX_test:\n")

print(X_test.head())

print(X_test.shape)


print("\ny_train:\n")

print(y_train.head())

print(y_train.shape)


print("\ny_test:\n")

print(y_test.head())

print(y_test.shape)'''
```

```
# In[54]:



import keras

import numpy as np

import matplotlib.pyplot as plt

from keras import layers

from keras import optimizers

from keras.layers import Dense, Flatten, Activation, Dropout

from keras import applications

from keras.models import Sequential, Model, load_model

from keras.applications import VGG16, InceptionV3, ResNet50



from skimage.io import imread,imsave

from keras.models import Model,load_model

from keras.optimizers import SGD,Adam

from keras.layers import *

from skimage.util import pad,crop

from skimage.transform import resize

import os

from sklearn.model_selection import train_test_split

from keras.callbacks import ReduceLROnPlateau,ModelCheckpoint,CSVLogger,Callback,EarlyStopping

import tensorflow as tf

import keras.backend as K
```

```
from keras.utils import plot_model

import matplotlib.pyplot as plt


# In[88]:


from numpy import zeros, newaxis

numpy_X = X.as_matrix()

numpy_X1 = numpy_X[:, :, newaxis]

numpy_X1.shape


# In[89]:


numpy_y = y.as_matrix()

numpy_y1 = numpy_y[:, :, newaxis]

numpy_y1.shape


# In[95]:


#building a neural network of 3 layers

inputs = Input((numpy_X1.shape[0], numpy_X1.shape[1]))

x = Dense(512, activation='relu')(inputs)

#x = Dropout(0.5)(x)

x = Dense(256, activation='relu')(x)

#x = Dropout(0.5)(x)

x = Dense(2, activation='sigmoid')(x)
```

```
model = Model(inputs = inputs,outputs = x)


model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])

model.summary()


# In[93]:


#training or fitting the model

model.compile(optimizer='Adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])




# Train model

model.fit(numpy_X1, numpy_y1, epochs=4, validation_split=0.2)


# In[ ]:
```

C.4. Python Code for Classification ANN

```
#!/usr/bin/env python

# coding: utf-8


# In[58]:

#import the pandas module

import pandas as pd
```

```
#import the numpy module

import numpy as np



colnames=['I1', 'I2', 'I3', 'T1', 'T2', 'P1', 'P2']



# In[59]:

#Reading and exploring the data by coverting to pandas dataframe

df = pd.read_excel(r'C:\Users\user\Desktop\watertesting.xlsx')

print(df.head(5))



# In[67]:

#splitting into train-test set for model training of the data from test2.tsv file (0.9 test,,
0.1 train)

from sklearn.model_selection import train_test_split

y = df.drop(['I1', 'I2', 'I3', 'T1', 'T2'], axis =1)

X = df.drop(['T1', 'T2', 'P1', 'P2'], axis =1)



X = df[['I1','I2','I3']]

y = df[['P1','P2']]

print(X.head())

print(y.head())



'''X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2)

print("\nX_train:\n")

print(X_train.head())

print(X_train.shape)
```

```
print("\nX_test:\n")

print(X_test.head())

print(X_test.shape)



print("\ny_train:\n")

print(y_train.head())

print(y_train.shape)



print("\ny_test:\n")

print(y_test.head())

print(y_test.shape)'''



# In[54]:

import keras

import numpy as np

import matplotlib.pyplot as plt

from keras import layers

from keras import optimizers

from keras.layers import Dense, Flatten, Activation, Dropout

from keras import applications

from keras.models import Sequential, Model, load_model

from keras.applications import VGG16, InceptionV3, ResNet50



from skimage.io import imread,imsave
```

```
from keras.models import Model,load_model

from keras.optimizers import SGD,Adam

from keras.layers import *

from skimage.util import pad,crop

from skimage.transform import resize

import os

from sklearn.model_selection import train_test_split

from keras.callbacks import ReduceLROnPlateau,ModelCheckpoint,CSVLogger,Callback,EarlyStopping

import tensorflow as tf

import keras.backend as K

from keras.utils import plot_model

import matplotlib.pyplot as plt



# In[88]:

from numpy import zeros, newaxis

numpy_X = X.as_matrix()

numpy_X1 = numpy_X[:, :, newaxis]

numpy_X1.shape



# In[89]:

numpy_y = y.as_matrix()

numpy_y1 = numpy_y[:, :, newaxis]

numpy_y1.shape



# In[95]:
```

```
#building a neural network of 3 layers

inputs = Input((numpy_X1.shape[0], numpy_X1.shape[1]))



x = Dense(512, activation='relu')(inputs)

#x = Dropout(0.5)(x)

x = Dense(256, activation='relu')(x)

#x = Dropout(0.5)(x)

x = Dense(2, activation='sigmoid')(x)



model = Model(inputs = inputs,outputs = x)



model.compile(loss='binary_crossentropy',optimizer='adam', metrics=['accuracy'])

model.summary()



# In[93]:

#training or fitting the model

model.compile(optimizer='Adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])




# Train model

model.fit(numpy_X1, numpy_y1, epochs=4, validation_split=0.2)



# In[ ]:
```

C.5. List of Resources used in Hardware Implementation of 20 neurons ANN in IEEE 754 and Posit Representation

For IEEE 754 Nonlinear Approximation

```
============================================================

  Generated by:           Encounter(R) RTL Compiler v08.10-s116_1

  Generated on:           Jul 13 2021  11:38:45 PM

  Module:                 Nonlin_sigmoid

  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3

  Operating conditions:   _nominal_ (balanced_tree)

  Wireload mode:          enclosed

  Area mode:              timing library

============================================================

    Gate      Instances    Area               Library

------------------------------------------------------------------------

AN2B1RLX1             4     24.000    fsd0k_a_generic_core_1d0vtc

AN2B1RLXLP          101    505.000    fsd0k_a_generic_core_1d0vtc

AN2RLX1              38    190.000    fsd0k_a_generic_core_1d0vtc

AN2RLXLP            229   1145.000    fsd0k_a_generic_core_1d0vtc

AN3B1RLX1P            2     20.000    fsd0k_a_generic_core_1d0vtc

AN3B1RLXLP            5     40.000    fsd0k_a_generic_core_1d0vtc

AN3B2RLX1            14     98.000    fsd0k_a_generic_core_1d0vtc

AN3B2RLXLP           38    266.000    fsd0k_a_generic_core_1d0vtc

AN3RLX1              11     77.000    fsd0k_a_generic_core_1d0vtc

AN3RLXLP              8     56.000    fsd0k_a_generic_core_1d0vtc

AN4B1RLXLP            4     32.000    fsd0k_a_generic_core_1d0vtc
```

AN4B2RLXLP	16	128.000	fsd0k_a_generic_core_1d0vtc
AN4RLX1	3	30.000	fsd0k_a_generic_core_1d0vtc
AO112RLXLP	4	36.000	fsd0k_a_generic_core_1d0vtc
AO12RLXLP	162	1134.000	fsd0k_a_generic_core_1d0vtc
AO13RLXLP	1	9.000	fsd0k_a_generic_core_1d0vtc
AO222RLX1	10	120.000	fsd0k_a_generic_core_1d0vtc
AO222RLXLP	365	4380.000	fsd0k_a_generic_core_1d0vtc
AO22RLXLP	14	126.000	fsd0k_a_generic_core_1d0vtc
AOI112RLXLP	52	364.000	fsd0k_a_generic_core_1d0vtc
AOI122RLXLP	190	1710.000	fsd0k_a_generic_core_1d0vtc
AOI12B2RLXLP	1	9.000	fsd0k_a_generic_core_1d0vtc
AOI12RLX1	3	18.000	fsd0k_a_generic_core_1d0vtc
AOI12RLXLP	130	780.000	fsd0k_a_generic_core_1d0vtc
AOI13RLXLP	47	329.000	fsd0k_a_generic_core_1d0vtc
AOI222RLXLP	194	2134.000	fsd0k_a_generic_core_1d0vtc
AOI22RLXLP	467	3269.000	fsd0k_a_generic_core_1d0vtc
AOI23RLX1	1	8.000	fsd0k_a_generic_core_1d0vtc
AOI23RLXLP	141	1128.000	fsd0k_a_generic_core_1d0vtc
AOI33RLXLP	70	700.000	fsd0k_a_generic_core_1d0vtc
FA1RLX1	863	25890.000	fsd0k_a_generic_core_1d0vtc
HA1RLX1	94	1410.000	fsd0k_a_generic_core_1d0vtc
INVCKRLXLP	6	18.000	fsd0k_a_generic_core_1d0vtc
INVRLX1	1255	3765.000	fsd0k_a_generic_core_1d0vtc
INVRLXLP	49	147.000	fsd0k_a_generic_core_1d0vtc
MAO222RLXLP	295	2655.000	fsd0k_a_generic_core_1d0vtc

MAOI1RLXLP	37	333.000	fsd0k_a_generic_core_1d0vtc
MAOI222RLX1	165	1320.000	fsd0k_a_generic_core_1d0vtc
MOAI1RLXLP	472	3776.000	fsd0k_a_generic_core_1d0vtc
MUX2RLXLP	56	504.000	fsd0k_a_generic_core_1d0vtc
MUXB2RLXLP	2	20.000	fsd0k_a_generic_core_1d0vtc
MXL2RLXLP	726	5082.000	fsd0k_a_generic_core_1d0vtc
ND2RLX1	10	40.000	fsd0k_a_generic_core_1d0vtc
ND2RLXLP	865	3460.000	fsd0k_a_generic_core_1d0vtc
ND3RLX1	1	6.000	fsd0k_a_generic_core_1d0vtc
ND3RLXLP	142	852.000	fsd0k_a_generic_core_1d0vtc
NR2RLX1	46	184.000	fsd0k_a_generic_core_1d0vtc
NR2RLX2	1	7.000	fsd0k_a_generic_core_1d0vtc
NR2RLXLP	864	3456.000	fsd0k_a_generic_core_1d0vtc
NR3RLX1	19	114.000	fsd0k_a_generic_core_1d0vtc
NR3RLXLP	24	144.000	fsd0k_a_generic_core_1d0vtc
OA112RLX1	5	45.000	fsd0k_a_generic_core_1d0vtc
OA112RLXLP	6	54.000	fsd0k_a_generic_core_1d0vtc
OA12RLXLP	34	238.000	fsd0k_a_generic_core_1d0vtc
OA13RLXLP	2	16.000	fsd0k_a_generic_core_1d0vtc
OA222RLXLP	16	192.000	fsd0k_a_generic_core_1d0vtc
OA22RLXLP	18	162.000	fsd0k_a_generic_core_1d0vtc
OAI112RLX1	1	7.000	fsd0k_a_generic_core_1d0vtc
OAI112RLXLP	105	735.000	fsd0k_a_generic_core_1d0vtc
OAI122RLXLP	126	1134.000	fsd0k_a_generic_core_1d0vtc
OAI12RLX1	4	24.000	fsd0k_a_generic_core_1d0vtc

OAI12RLXLP	294	1764.000	fsd0k_a_generic_core_1d0vtc
OAI13RLXLP	66	462.000	fsd0k_a_generic_core_1d0vtc
OAI222RLXLP	18	198.000	fsd0k_a_generic_core_1d0vtc
OAI22RLXLP	184	1288.000	fsd0k_a_generic_core_1d0vtc
OAI23RLXLP	70	560.000	fsd0k_a_generic_core_1d0vtc
OAI33RLXLP	115	1035.000	fsd0k_a_generic_core_1d0vtc
OR2B1RLXLP	229	1145.000	fsd0k_a_generic_core_1d0vtc
OR2RLX1	3	15.000	fsd0k_a_generic_core_1d0vtc
OR2RLXLP	37	185.000	fsd0k_a_generic_core_1d0vtc
OR3B1RLXLP	10	80.000	fsd0k_a_generic_core_1d0vtc
OR3B2RLX1	1	7.000	fsd0k_a_generic_core_1d0vtc
OR3B2RLXLP	10	70.000	fsd0k_a_generic_core_1d0vtc
OR3RLX1	16	112.000	fsd0k_a_generic_core_1d0vtc
OR3RLXLP	1	7.000	fsd0k_a_generic_core_1d0vtc
OR4B1RLX1	2	16.000	fsd0k_a_generic_core_1d0vtc
OR4B1RLXLP	54	432.000	fsd0k_a_generic_core_1d0vtc
OR4B2RLXLP	70	560.000	fsd0k_a_generic_core_1d0vtc
OR4RLXLP	7	70.000	fsd0k_a_generic_core_1d0vtc
QDLAHRLX1	717	9321.000	fsd0k_a_generic_core_1d0vtc
XNR2RLX1	5	50.000	fsd0k_a_generic_core_1d0vtc
XNR2RLXLP	16	160.000	fsd0k_a_generic_core_1d0vtc
XOR2RLX1	75	750.000	fsd0k_a_generic_core_1d0vtc
total	10634	92942.000	

```
   Type    Instances    Area   Area %
--------------------------------------
sequential       717  9321.000   10.0
inverter        1310  3930.000    4.2
logic           8607 79691.000   85.7
--------------------------------------
total          10634 92942.000  100.0
```

For Posit number Representation

```
============================================================
  Generated by:           Encounter(R) RTL Compiler v08.10-s116_1
  Generated on:           Jul 28 2021  12:19:54 AM
  Module:                 neuron_posit
  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================
    Gate     Instances    Area               Library
------------------------------------------------------------------
AN2B1RLXLP         325   1625.000    fsd0k_a_generic_core_1d0vtc
AN2RLX1             11     55.000    fsd0k_a_generic_core_1d0vtc
AN2RLXLP            80    400.000    fsd0k_a_generic_core_1d0vtc
AN3B1RLX1            2     18.000    fsd0k_a_generic_core_1d0vtc
```

AN3B1RLXLP	4	32.000	fsd0k_a_generic_core_1d0vtc
AN3B2RLXLP	27	189.000	fsd0k_a_generic_core_1d0vtc
AN4B1RLXLP	4	32.000	fsd0k_a_generic_core_1d0vtc
AO112RLX1	3	27.000	fsd0k_a_generic_core_1d0vtc
AO112RLXLP	12	108.000	fsd0k_a_generic_core_1d0vtc
AO12RLXLP	19	133.000	fsd0k_a_generic_core_1d0vtc
AO13RLXLP	3	27.000	fsd0k_a_generic_core_1d0vtc
AO222RLXLP	37	444.000	fsd0k_a_generic_core_1d0vtc
AO22RLXLP	176	1584.000	fsd0k_a_generic_core_1d0vtc
AOI112RLXLP	7	49.000	fsd0k_a_generic_core_1d0vtc
AOI122RLXLP	126	1134.000	fsd0k_a_generic_core_1d0vtc
AOI12RLX1	1	6.000	fsd0k_a_generic_core_1d0vtc
AOI12RLXLP	34	204.000	fsd0k_a_generic_core_1d0vtc
AOI13RLXLP	3	21.000	fsd0k_a_generic_core_1d0vtc
AOI222RLXLP	35	385.000	fsd0k_a_generic_core_1d0vtc
AOI22RLXLP	279	1953.000	fsd0k_a_generic_core_1d0vtc
AOI23RLXLP	13	104.000	fsd0k_a_generic_core_1d0vtc
BUFRLX3	6	48.000	fsd0k_a_generic_core_1d0vtc
FA1RLX1	160	4800.000	fsd0k_a_generic_core_1d0vtc
HA1RLX1	95	1425.000	fsd0k_a_generic_core_1d0vtc
INVCKRLXLP	1	3.000	fsd0k_a_generic_core_1d0vtc
INVRLX1	530	1590.000	fsd0k_a_generic_core_1d0vtc
INVRLXLP	27	81.000	fsd0k_a_generic_core_1d0vtc
MAO222RLXLP	144	1296.000	fsd0k_a_generic_core_1d0vtc
MAOI1RLXLP	133	1197.000	fsd0k_a_generic_core_1d0vtc

MAOI222RLX1	161	1288.000	fsd0k_a_generic_core_1d0vtc
MOAI1RLX1	4	32.000	fsd0k_a_generic_core_1d0vtc
MOAI1RLXLP	665	5320.000	fsd0k_a_generic_core_1d0vtc
MUX2RLXLP	347	3123.000	fsd0k_a_generic_core_1d0vtc
MUXB2RLX1	8	80.000	fsd0k_a_generic_core_1d0vtc
MUXB2RLXLP	4	40.000	fsd0k_a_generic_core_1d0vtc
MXL2RLXLP	677	4739.000	fsd0k_a_generic_core_1d0vtc
ND2RLX1	4	16.000	fsd0k_a_generic_core_1d0vtc
ND2RLXLP	380	1520.000	fsd0k_a_generic_core_1d0vtc
ND3RLXLP	8	48.000	fsd0k_a_generic_core_1d0vtc
NR2RLX1	26	104.000	fsd0k_a_generic_core_1d0vtc
NR2RLXLP	286	1144.000	fsd0k_a_generic_core_1d0vtc
NR3RLXLP	9	54.000	fsd0k_a_generic_core_1d0vtc
NR4RLXLP	4	44.000	fsd0k_a_generic_core_1d0vtc
OA12RLXLP	33	231.000	fsd0k_a_generic_core_1d0vtc
OA13RLXLP	6	48.000	fsd0k_a_generic_core_1d0vtc
OA22RLXLP	57	513.000	fsd0k_a_generic_core_1d0vtc
OAI112RLXLP	28	196.000	fsd0k_a_generic_core_1d0vtc
OAI122RLXLP	70	630.000	fsd0k_a_generic_core_1d0vtc
OAI12RLX1	3	18.000	fsd0k_a_generic_core_1d0vtc
OAI12RLXLP	75	450.000	fsd0k_a_generic_core_1d0vtc
OAI13RLXLP	7	49.000	fsd0k_a_generic_core_1d0vtc
OAI222RLXLP	30	330.000	fsd0k_a_generic_core_1d0vtc
OAI22RLX1	4	28.000	fsd0k_a_generic_core_1d0vtc
OAI22RLXLP	213	1491.000	fsd0k_a_generic_core_1d0vtc

```
OAI23RLXLP           7     56.000    fsd0k_a_generic_core_1d0vtc

OAI33RLXLP           8     72.000    fsd0k_a_generic_core_1d0vtc

OR2B1RLXLP         301   1505.000    fsd0k_a_generic_core_1d0vtc

OR2RLXLP           115    575.000    fsd0k_a_generic_core_1d0vtc

OR3B1RLXLP           3     24.000    fsd0k_a_generic_core_1d0vtc

OR3B2RLXLP           6     42.000    fsd0k_a_generic_core_1d0vtc

OR3RLX1              4     28.000    fsd0k_a_generic_core_1d0vtc

OR4B1RLXLP          24    192.000    fsd0k_a_generic_core_1d0vtc

OR4RLXLP            41    410.000    fsd0k_a_generic_core_1d0vtc

XNR2RLX1             3     30.000    fsd0k_a_generic_core_1d0vtc

XNR2RLXLP           54    540.000    fsd0k_a_generic_core_1d0vtc

XOR2RLX1            80    800.000    fsd0k_a_generic_core_1d0vtc

-------------------------------------------------------------------

total             6052  44780.000


   Type    Instances    Area   Area %

---------------------------------------

inverter          558  1674.000    3.7

buffer              6    48.000    0.1

unresolved         39     0.000    0.0

logic            5488 43058.000   96.2

---------------------------------------

total            6091 44780.000  100.0
```

C.6. List of Resources used in Hardware Implementation for Complete WQI Device using 100 Neurons using Posit Representation

```
============================================================

  Generated by:           Encounter(R) RTL Compiler v08.10-s116_1

  Generated on:           Feb 23 2023  01:57:45 AM

  Module:                 WQI

  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3

  Operating conditions:   _nominal_ (balanced_tree)

  Wireload mode:          enclosed

  Area mode:              timing library

============================================================

    Gate       Instances       Area    Library

-----------------------------------------------------------------------

AN2B1RLXLP       14020    70100.000    fsd0k_a_generic_core_1d0vtc

AN2RLX1            923     4615.000    fsd0k_a_generic_core_1d0vtc

AN2RLXLP          5201    26005.000    fsd0k_a_generic_core_1d0vtc

AN3B1RLX1          270     2430.000    fsd0k_a_generic_core_1d0vtc

AN3B1RLXLP         283     2264.000    fsd0k_a_generic_core_1d0vtc

AN3B2RLXLP        2028    14196.000    fsd0k_a_generic_core_1d0vtc

AN4B1RLX1           13      104.000    fsd0k_a_generic_core_1d0vtc

AO112RLX1          283     2547.000    fsd0k_a_generic_core_1d0vtc

AO112RLXLP         984     8856.000    fsd0k_a_generic_core_1d0vtc

AO12RLXLP         1607    11249.000    fsd0k_a_generic_core_1d0vtc

AO13RLXLP          283     2547.000    fsd0k_a_generic_core_1d0vtc

AO222RLXLP        1981    23772.000    fsd0k_a_generic_core_1d0vtc
```

AO22RLX1	534	4806.000	fsd0k_a_generic_core_1d0vtc
AO22RLXLP	12626	113634.000	fsd0k_a_generic_core_1d0vtc
AOI112RLXLP	611	4277.000	fsd0k_a_generic_core_1d0vtc
AOI122RLXLP	9178	82602.000	fsd0k_a_generic_core_1d0vtc
AOI12RLX1	51	306.000	fsd0k_a_generic_core_1d0vtc
AOI12RLXLP	2615	15690.000	fsd0k_a_generic_core_1d0vtc
AOI13RLXLP	283	1981.000	fsd0k_a_generic_core_1d0vtc
AOI222RLXLP	2399	26389.000	fsd0k_a_generic_core_1d0vtc
AOI22RLXLP	14616	102312.000	fsd0k_a_generic_core_1d0vtc
AOI23RLXLP	849	6792.000	fsd0k_a_generic_core_1d0vtc
BUFRLX12	8	208.000	fsd0k_a_generic_core_1d0vtc
BUFRLX20	558	22878.000	fsd0k_a_generic_core_1d0vtc
FA1RLX1	12744	382320.000	fsd0k_a_generic_core_1d0vtc
HA1RLX1	8411	126165.000	fsd0k_a_generic_core_1d0vtc
INVCKRLX1	8	24.000	fsd0k_a_generic_core_1d0vtc
INVCKRLX2	36	144.000	fsd0k_a_generic_core_1d0vtc
INVCKRLXLP	13	39.000	fsd0k_a_generic_core_1d0vtc
INVRLX1	37428	112284.000	fsd0k_a_generic_core_1d0vtc
INVRLX2	2554	10216.000	fsd0k_a_generic_core_1d0vtc
INVRLX4	1566	10962.000	fsd0k_a_generic_core_1d0vtc
INVRLXLP	2338	7014.000	fsd0k_a_generic_core_1d0vtc
MAO222RLXLP	12136	109224.000	fsd0k_a_generic_core_1d0vtc
MAOI1RLXLP	5084	45756.000	fsd0k_a_generic_core_1d0vtc
MAOI222RLX1	13361	106888.000	fsd0k_a_generic_core_1d0vtc
MOAI1RLX1	484	3872.000	fsd0k_a_generic_core_1d0vtc

MOAI1RLXLP	49572	396576.000	fsd0k_a_generic_core_1d0vtc
MUX2RLXLP	29303	263727.000	fsd0k_a_generic_core_1d0vtc
MUXB2RLX1	360	3600.000	fsd0k_a_generic_core_1d0vtc
MUXB2RLXLP	640	6400.000	fsd0k_a_generic_core_1d0vtc
MXL2RLXLP	54615	382305.000	fsd0k_a_generic_core_1d0vtc
ND2RLXLP	26095	104380.000	fsd0k_a_generic_core_1d0vtc
NR2RLX1	2026	8104.000	fsd0k_a_generic_core_1d0vtc
NR2RLXLP	22227	88908.000	fsd0k_a_generic_core_1d0vtc
NR3RLXLP	754	4524.000	fsd0k_a_generic_core_1d0vtc
OA12RLXLP	2582	18074.000	fsd0k_a_generic_core_1d0vtc
OA13RLXLP	566	4528.000	fsd0k_a_generic_core_1d0vtc
OA22RLXLP	5261	47349.000	fsd0k_a_generic_core_1d0vtc
OAI112RLXLP	1788	12516.000	fsd0k_a_generic_core_1d0vtc
OAI122RLXLP	6033	54297.000	fsd0k_a_generic_core_1d0vtc
OAI12RLX1	283	1698.000	fsd0k_a_generic_core_1d0vtc
OAI12RLXLP	5524	33144.000	fsd0k_a_generic_core_1d0vtc
OAI13RLXLP	283	1981.000	fsd0k_a_generic_core_1d0vtc
OAI222RLXLP	2830	31130.000	fsd0k_a_generic_core_1d0vtc
OAI22RLX1	328	2296.000	fsd0k_a_generic_core_1d0vtc
OAI22RLXLP	16316	114212.000	fsd0k_a_generic_core_1d0vtc
OAI23RLXLP	283	2264.000	fsd0k_a_generic_core_1d0vtc
OAI33RLXLP	656	5904.000	fsd0k_a_generic_core_1d0vtc
OR2B1RLXLP	20774	103870.000	fsd0k_a_generic_core_1d0vtc
OR2RLXLP	6618	33090.000	fsd0k_a_generic_core_1d0vtc
OR3B1RLXLP	283	2264.000	fsd0k_a_generic_core_1d0vtc

```
OR3B2RLXLP          334     2338.000    fsd0k_a_generic_core_1d0vtc

OR3RLX1             328     2296.000    fsd0k_a_generic_core_1d0vtc

OR4B1RLXLP         1460    11680.000    fsd0k_a_generic_core_1d0vtc

OR4B2RLXLP          328     2624.000    fsd0k_a_generic_core_1d0vtc

OR4RLXLP           2473    24730.000    fsd0k_a_generic_core_1d0vtc

XNR2RLX1            127     1270.000    fsd0k_a_generic_core_1d0vtc

XNR2RLXLP          3806    38060.000    fsd0k_a_generic_core_1d0vtc

XOR2RLX1           6936    69360.000    fsd0k_a_generic_core_1d0vtc

------------------------------------------------------------------

total            440159  3344967.000
```

```
   Type    Instances     Area    Area %

----------------------------------------

inverter       43943  140683.000    4.2

buffer           566   23086.000    0.7

unresolved      3489       0.000    0.0

logic         395650 3181198.000   95.1

----------------------------------------

total         443648 3344967.000  100.0
```

C.7. FPGA Results for Reduced Complete Water Quality Classification device

Table C. 1: FPGA Results for Reduced Complete Water Quality Classification device

FPGA Resource Parameter	Values
LUTs	45,024
MUXs	100
DSPs	56
Power	35.836 W

List of Publications

Published/Accepted: -

1. Abheek Gupta, Anu Gupta, and Rajiv Gupta, " Power and Area Efficient Intelligent Hardware Design for Water Quality Applications", Sensors and Transducers Journal, 2018, Vol 227(11), pp 67 - 78. **Status – Published.** *Scopus Indexed*
2. Abheek Gupta, Anu Gupta, and Rajiv Gupta, "Efficient ASIC Implementation of Artificial Neural Network with Posit representation of Floating-Point Numbers", International Conference on Next Generation Systems and Networks (BITS EEECon 2022), 4 – 5 November 2022, BITS Pilani, Pilani, India; considered for publication in Scopus Indexed Springer book series "Lecture Notes in Networks and Systems". **Status – Published.** *Scopus Indexed*
3. Abheek Gupta, Anu Gupta, and Rajiv Gupta, "Low-cost Artificial Intelligence Enhanced Hardware Design for Data Augmentation", The 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering, Tenerife, Spain, to be included in the final proceedings for the submission to the IEEE Xplore. **Status – Accepted.** *Scopus Indexed*
4. Abheek Gupta, Anu Gupta, and Rajiv Gupta, "High speed and Power efficient Digital VLSI Architecture of Artificial Neural Network for reliable in-situ Water Quality Application", Sustainable Water Resources Management, 2024, **Status – Under Review**. *SCIe Indexed.*

Communicated: -

1. Abheek Gupta, Anu Gupta, and Rajiv Gupta, "High speed and Power efficient Digital VLSI Architecture of Artificial Neural Network for reliable in-situ Water Quality Application", Sustainable Water Resources Management, 2024. Status – 1st Review completed. SCIe Journal
2. Abheek Gupta, Anu Gupta, Rajiv Gupta, Chandra Shekhar, " A Low-cost Embedded Instrument for Smart Water Quality Classification using ANN based Data Augmentation"*, Communicated to Sustainable Water Resources Management, Springer Nature, SCIe Indexed, Paper Under Review*
3. Abheek Gupta, Anu Gupta, Rajiv Gupta, Chandra Shekhar, "Low-Cost Power Efficient VLSI Implementation of Artificial Neural Network with Bounded Posit Floating-Point Format"; *Communicated to IETE Technical Review, Taylor and Francis, SCIe Indexed, Paper under review.*
4. Abheek Gupta, Anu Gupta, Rajiv Gupta, Chandra Shekhar, "A Low-cost Embedded System Design for Intelligent Water Classification using ANN based Data Augmentation" *Communicated to IETE Journal of Research, Taylor and Francis, SCIe Indexed Paper Under Review*.
5. Abheek Gupta, Anu Gupta, Rajiv Gupta, "High speed and Power efficient Digital VLSI Architecture of Artificial Neural Network for Portable Water Quality Classification" *Communicated to "Integration: A VLSI Journal*", *Elsevier, SCI Indexed*.
6. Abheek Gupta, Anu Gupta, Rajiv Gupta, "Water quality monitoring using artificial neural network for sustainable development: A case study of Jhunjhunu City, Rajasthan", *Communicated to Iranian Journal of Science and Technology, Transactions of Electrical Engineering, Springer, SCIe Indexed.*

Patent: -

1. A Portable Real-Time Colorimetric Detection Device and Method Of Using The Same", The patent application 202111017453 (filed on 14 April 2021). Status - FER Generated.

Biography of the Research Scholar

1. **Abheek Gupta** is a research scholar at Birla Institute of Science and Technology, Pilani, Rajasthan. He did his Master of Technology in Microelectronics from Manipal University, Jaipur, Rajasthan and his Bachelor of Technology in Electronics and Communication Engineering from ICFAI University, Dehradun, Uttarakhand. His research interests include Digital VLSI Design, Digital VLSI Architectures, Artificial Neural Networks, Water Quality Monitoring and Management.

Biography of the Supervisors

1. **Prof. Anu Gupta** is a Professor at Birla Institute of Technology and Science (BITS) Pilani, Rajasthan. Since joining the institute, she is involved in research in high-performance, low-power, and digital, analog and mixed-signal design for FPGA and ASIC applications. She has published over 100 research Papers and guided 3 PhD scholars. Prof. Anu Gupta is the Supervisor of this thesis.
2. **Sr. Prof. Rajiv Gupta** is Senior Professor of Civil Engineering at BITS, Pilani. He has completed his B.E., M.E and Ph.D. from BITS, Pilani. In his last 30 years of teaching and research, he has published more than 150 research papers in peer-reviewed journals and presented in conferences in India and abroad and authored a number of books and course development material. He has guided more than 10 Ph.D. scholars apart from being involved in teaching around 30 courses and reviewed more than 125 books, projects, and papers in reputed journals. His fields of interest are Water-Energy conservation, GIS and RS, the Application of Artificial Intelligence, and Concrete Technology. He is involved in a number of research and development projects worth more than Rs. 650 lacs of World Bank, UGC, DST, University of Virginia, and other sponsored organizations. He has also worked in different capacities of administration like Warden, Head of Department, and Dean of Engineering Services and Hardware. He was instrumental in developing a number of infrastructure facilities at Pilani, Goa, and Hyderabad campuses. Sr. Prof. Rajiv Gupta is the co-supervisor of this thesis.