VLSI Implementation of Neural Network Driven Augmented FSM

by

Jimmy Kirtikumar Patel
202011048

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

May, 2022

Declaration

I hereby declare that

i) The thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) Due acknowledgment has been made in the text to all the reference material used.

Jimmy Kirtikumar Patel

Certificate

This is to certify that the thesis work entitled VLSI Implementation of Neural Network Driven Augmented FSM has been carried out by Jimmy Kirtikumar Patel for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.

Tapas Kumar Maiti
Thesis Supervisor

Acknowledgments

First and foremost, grateful and thankfulness to God, the Almighty, for His showers of graces during my study endeavour, which enabled me to complete the research successfully.

I would like to thank my research supervisor, Prof. Tapas Kumar Maiti, for allowing me the chance to do research and for offering vital assistance during this process. His energy, vision, genuineness, and determination have all left an indelible impression on me. He taught me the approach for conducting the study and presenting the findings as simply as possible. Working and researching under his direction was a tremendous honour and privilege. I am grateful for everything he has done for me. I'd also want to thank him for his friendship, kindness, and fantastic warm personality.

I am truly grateful for their love, faith, care, and sacrifices in teaching and prepared me for the future. Also I express my thanks to my sister for her support and valuable inspiration.

I would like to thank one of my friends and research colleague, Harsh Advani, for his constant support throughout this research. I'd like to thank the M.Tech. students from Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar for their assistance with my study. I thank the management of Dhirubhai Ambani Institute of Information and Communication Technology, Gandhinagar for their support to do this work.

Once again, I want to convey my gratitude to everyone who supported me in completing the research work, whether directly or indirectly.

Contents

Abstract

This thesis reports the VLSI implementation of an NN (Neural Network) based emergent behavior model for high-speed robot control. Augmented FSM (Finite-State Machine) is considered to implement the emergent behavior. We performed a system-level simulation using our proposed model. For system level simulation, we have used Python base TensorFlow model to implement the Neural Network. Then, we transformed the model to RTL (Register-Transfer Level) for circuit simulation. For RTL modeling we have used Verilog (Xilinx, Quartus Prime and iVerilog) and for simulation we have used (Modelsim and GTK wave). In this study, we considered multiple inputs and multiple-outputs NN. Our implementation method improves the speed of execution and accuracy and compares the result with the conventional neural network. For activation function in NN, we implemented sigmoid function with second-order approximation to reduce complexity. We used the walking gesture of the Kondo KHR-3HV robot to verify the model. Finally, we design NN based augmented-AI chip for high-speed robotics applications.

List of Principal Symbols and Acronyms

AFSM Augmented Finite State Machine

AI Artificial Intelligence

ANN Artificial Neural Network

DL Deep Learning

FSM Finite State Machine

ML Machine Learning

NN Neural Network

RL Reinforcement Learning

RNN Recurrent Neural Networks

SNN Simulated Neural Network

VLSI Very Large Scale Integration

List of Tables

List of Figures

CHAPTER 1
Introduction

Our civilization has been heading toward full automation over the last few decades, whether in manufacturing, production, surveillance, security, or other areas. Nowadays, for automation, AI (Artificial Intelligence) and neural networks are used for classification and identification [1], [2]. Brooks et al., reported a new method for the application of AI to robotics, known as emergent behaviour-based robotics [3]. Simple control system was represented by AFSM (Augmented Finite-State Machine). However, for a complex robotic system, knowledge-based behaviour is also required for full control which can be implemented as a neural network [2].

1.1 Modern Automation

Future automation refers to a variety of technologies that lessen the need for human intervention in robot operations. The requirement for human involvement can be reduced by associated activities, nested links, predetermining decision criteria and encoding such predeterminations in robot [4]. Complete automation requires various technology and control systems, such as manufacturing operations, machinery, boilers and switching on telephone networks, stabilisation of ships, steering and other applications and vehicles, with minimum human contact [5] which includes a wide range of applications, ranging from a household thermostat controlling a boiler to a large control systems with large numbers of input and feedback signals [6]. In terms of control strategy, it ranges from a simple switch-control to multi-variable algorithms.

1.1.1 Machine Learning (ML)

Machine learning (ML) is applicable to intelligent artificial systems that may improve themselves dynamically via knowledge and data use. It is thought to be a sub-part of artificial intelligence (AI). ML algorithms construct a model with the help of sample data, referred to as training data, in order to predict or finalize decisions without being explicitly programmed in an artificial system [7]. ML algorithms are commonly used in numerous applied areas such as machine vision, speech recognition, medicine, email filtering, etc., when traditional algorithms are difficult or impossible to build[8].

1.1.2 Deep Learning (DL)

Deep Learning (DL) which is also known as deep structured learning, is a subset of ML technique that relies on artificial neural networks (ANN). Learning may occurs under supervision, unsupervised or semi supervised [9]. Architectures such as deep neural networks (DNN), recurrent neural networks (RNN), deep reinforcement learning (Deep-RL), and convolutional neural networks (CNN) applied to various domains such as machine vision, natural language processing, machine translation, speech recognition, medical image analysis, drug design, climate research, material inspection, board game programmes, and producing results that are equivalent to and, in some cases, superior to traditional methods [49] [10] [11].

1.2 Neural Network (NN)

A neural network (NN) is a network or circuit of biological neurons, while an artificial neural network (ANN) is one comprised of artificial neurons or nodes in the modern sense [19]. As a result, a neural network can be either a biological neural network made up of biological neurons or an artificial neural network intended to address artificial intelligence (AI) challenges. Artificial neural networks mimic biological neuron connections as weights between nodes. Positive weights represent activating connections, whereas negative weights represent regulatory connections. All entries are weighted and totaled. This is known as a linear combination. Finally, the NN output amplitude is controlled by an activation function. For example, an acceptable NN output range is normally between 0 and 1, although it might also be between -1 and 1.

1.2.1 Artificial Neural Networks (ANN)

Artificial neurons were originally proposed in 1943 by neurophysiologist Warren Sturgis and logician Walter Pitts, who were both affiliated with the University of Chicago at the time [17]. The concept of neural networks appears to have been first proposed by Alan Turing in the 1948 paper Intelligent Machinery, where he called them "chaotic B-type machines" [18]. An artificial neural networks (ANN) is used for predictive modeling, adaptive control, etc., that is generally trained on the basis of datasets. The experience-based self-learning is done within a network where conclusions is drawn from a complex and seemingly irrelevant set of datasets [20].

In case of an ANN, a simulated neural-network (SNN) forms a mathematical or computational model for processing information, of natural or artificial neurons. An interconnected group is considered for calculations based on a connectivity-oriented approach. ANN system is often an adaptive system that changes its structure in response to an external or an internal information passing through the network. NN in practical terms, are nonlinear statistical data modelling or decision-making tools. These may be used to describe complicated interactions between inputs and outputs of an environment, as well as to discover patterns in datasets. ANNs are made up of basic processing elements such as artificial neurons that may exhibit complicated global behaviour based on connections between processing elements and element characteristics.

1.2.2 Working Principle of Artificial Neural Networks

Artificial Neural Networks (ANNs), sometimes known as Neural Networks (NNs), are computing systems that are inspired by the biological neural networks that comprise an animal's brain. ANN is built on a network of linked units or nodes known as artificial neurons. Artificial neurons roughly model the neurons of the biological brain. Like the synapses of the biological brain, each connection can signal other neurons. Artificial neurons can receive and process signals and send signals to the neurons connected to them. The "signal" at the link is a real number, and each neuron's output is determined by a non-linear function of the sum of its inputs. The link is referred to as an edge. Neurons and edges usually have weights that are adjusted as learning progresses. Weights increase or decrease the signal strength of the connection. You can set a threshold on a neuron so that the signal is sent only when the sum of the signals exceeds that threshold. Neurons are normally organised in layers. Different layers may apply various modifica-

tions to their inputs. Signals go from the first layer (the input layer) to the last layer (the output layer), sometimes many times.

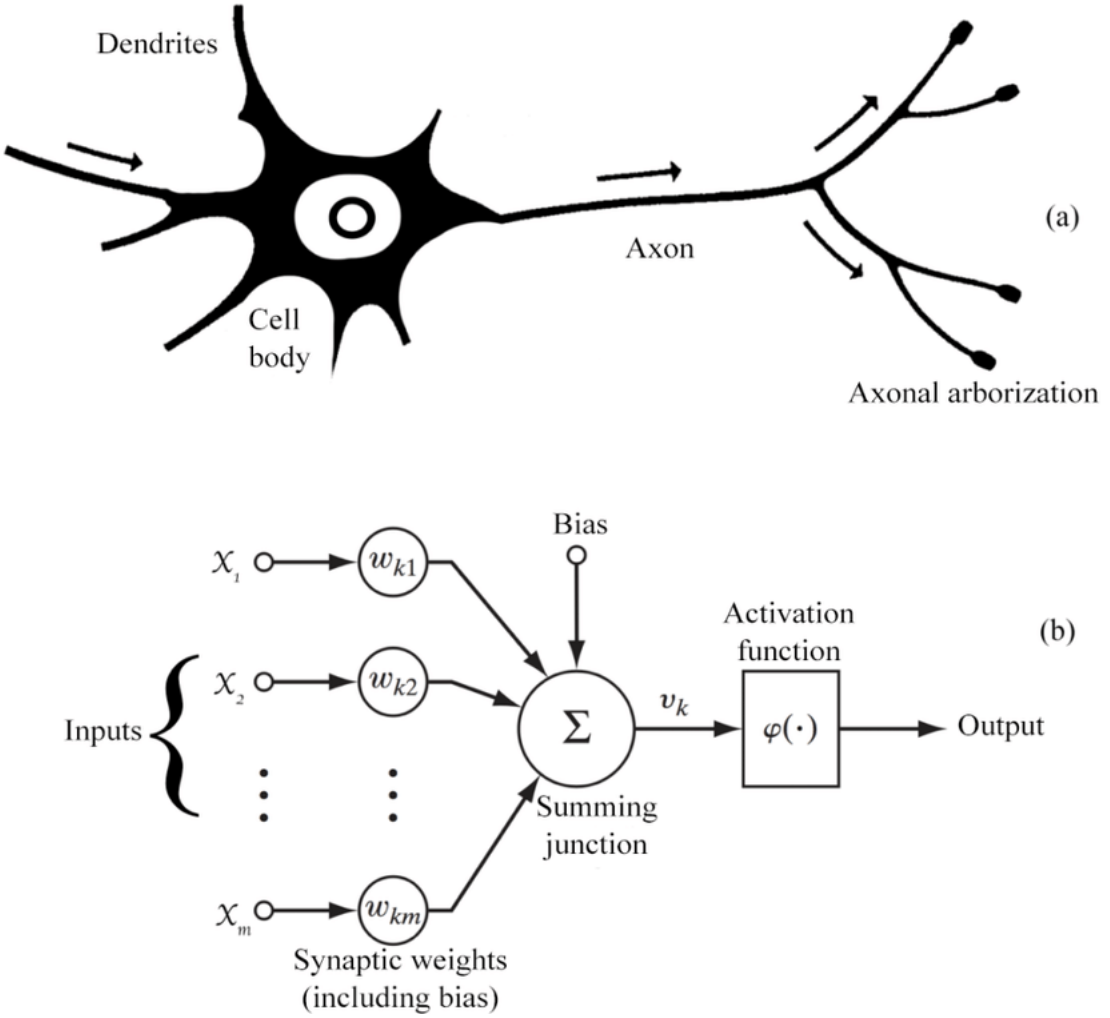

Figure 1.1: (a) Biological neural-network (NN), (b) Representation of artificial neural-network (ANN) [27]

1.2.3 Artificial Neuron

ANN is made up of artificial neurons that have been theoretically developed from biological neurons. Each artificial neuron receives an input and generates a single output that may be shared by several neurons [27]. The input can be a feature value of a sample of external data such as an image or document, or it can be the output of another neuron. To find the output of a neuron, we first need to take the weighted sum of all the inputs and weight them by the weight of the connection from the input to the neuron. Add a bias term to this sum [28]. This weighted sum is sometimes called activation. This weighted sum then passes through an

activation function (usually non-linear) to produce an output. The first input is external data such as images and documents. The final output performs tasks such as B. Recognize the objects in the image [29].

1.3 Training of NN

A training dataset is a sample dataset that is used during the training process to adjust parameters such as classifiers (such as weights) [23] [24]. The supervised learning method analyses the training dataset to find or train the ideal combination of variables to build the right prediction model for classification tasks [25]. The goal is to create a trained (fit) model that successfully generalizes to new unknown data [26]. The fitted model is tested against "new" instances from the retained datasets (validation and test datasets), and its accuracy in categorising new data is assessed [22]. Avoid using test dataset samples for model training to avoid issues like overfitting [22]. Most methods for obtaining training data for empirical relationships overfit the data. This implies you can spot and capitalise on evident correlations in training data that are often overlooked.

1.4 Testing of NN

The test dataset is independent of the training dataset, although it has the same probability distribution. Overfitting is reduced if the model that works well on the training dataset also works well on the test dataset (shown in Figure 1.2). Overfitting is generally indicated by a strong fit of the training dataset to the test dataset.

A check set is really a predetermined collection of examples that are best evaluate the performance (i.e. generalization) of a totally targeted classifier [23] [24]. To do this, the very last version of check set is used to expect classifications of examples withinside the check set. Those predictions are as compared to the examples' authentic classifications to evaluate the version's accuracy [25]. In a situation in which each validation and check datasets are used, the check statistics set is commonly used to evaluate the very last version this is decided on all through the validation process. In the case in which the authentic statistics set is partitioned into subsets (education and check datasets), the check statistics set would possibly determine the version best once (e.g., withinside the holdout method) [21]. Note that a few reassets propose towards such a technique [26]. However, while the use of a technique consisting of cross-validation, walls may be enough and pow-

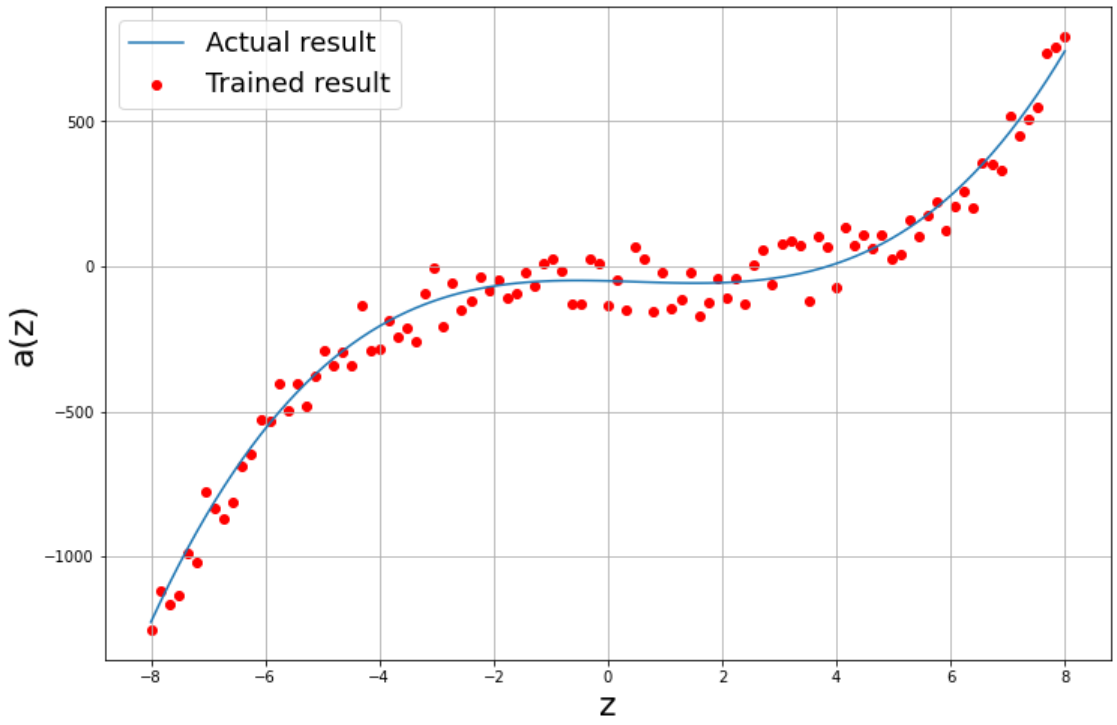

Figure 1.2: Trained model for testing the new datasets in [a.u.].

erful given that effects are averaged after repeated rounds of version education and checking out to assist lessen bias and variability [22] [26].

1.5 Types of Artificial Neural Network

ANN has grown into a diverse family of technologies that have enhanced cutting-edge technology in a variety of fields. The most basic kinds have one or more static components, such as the number of units, layers, unit weights, and topology. Learning allows you to grow one or more of these dynamic categories. The latter is significantly more difficult, but it can shorten learning time and produce greater outcomes. Some kinds allow the operator to "supervise" learning, whilst others function autonomously. Some are totally hardware-based, while others are entirely software-based and operate on general-purpose computers.

The main breakthroughs are Convolutional neural networks that have proven to be particularly successful in processing visual and other 2D data [31] [32] Long short-term memory avoids the vanishing gradient problem [33] and signals that represent mixing. Low-frequency and high-frequency components that support speech recognition for large vocabularies [34] [35] Text-to-speech synthesis, [36] [37] photorealistic speakers, [38] Generative hostile networks, etc. Competitive

network. Networks (different structures) compete for tasks such as winning the game [39] and deceiving opponents in terms of input authenticity [40].

1.6 Outline of Thesis

In this thesis, we considered a hybrid approach which is the combination of both knowledge-based behaviour and sensor control priority-based behaviour based on AFSM. The main objective of this work is the VLSI implementation of a behavioural model with lesser delay and complexity compared to conventional implementation to speed up the robot control.

Normally, implementation of a behavioural based AI model using Python results in higher processing time (millisecond) in robot control the Python based implementation is cover in chapter-2. It contains the behevioural implementation of the Artificial Neural Network using Python and also include the results obtain by the network. To implement the ANN, we approximated the sigmoid function instead of the conventional method and the flow of that method is discussed in chapter-2.

The hardware implementation is discussed in chapter-3. The comparison of the Python based results and Verilog based results are discussed in chapter-3. This work focusses on the implementation of emergent behaviour model at the hardware level and all the archetectures and behavioural analysis related to hardware implementation are discussed in chapter-3. We have also included the data flow analysis for the NN in chapter-3, obtained almost the same S-curve with lesser complexity. We have implemented AFSM with respect to four states of sensor inputs. The RTL level implementation and analysis of NN is described in Chapter-3.

In Chapter-4, we have discussed about the chip design for the proposed architecture. It also contain information regarding the flow of layout design and the modules which we have use to design our layout.

CHAPTER 2
System Level Implementation

ANNs (Artificial Neural Networks) usually known as NNs are computing systems inspired by the biological neural network that contains animal brains [1]. In ANN, we used artificial neurons to perform different mathematical functions like transfer function, sigmoid function, summation, multiplication, etc. In NN, each artificial neuron is connected to other artificial neurons and all artificial neurons are categorized in different layers. A simple ANN consists of at least 2 layers, one input layer, and one output layer but in some complex neural network, there are some hidden layers to improve the accuracy of the results and reduce the errors. As each artificial neuron is connected to other artificial neurons, creates a dense and complex network, thus by adding more layers, we increased the complexity of the network, thereby the accuracy is increased.

2.1 Flow of Implementation

To implement the neural network, weight, and bias values are needed which are obtained by training the network which is performed by using TensorFlow and Keras. After training the network, we collected all the necessary parameters. Using those parameters, we tested the network, measure accuracy, and error margin. Here, we have used this neural network to classify the motion of the humanoid robot such as forward motion, backward motion, right motion, and left motion.

2.1.1 Collecting Data for Network

We also included sensor data responses which we collected from sensors, connected to the robot. Figure 2.1 shows the sensor data are corresponding to 22 different servo motor angles, used as input data for the neural network. The data we have used is a 1-dimensional form.

Since we have 22-input data, we considered 22 neurons in the input layer and,

the output layer consists of 4 neurons. We classified four different motions after classifying the motion, we have four different combinations of data by using that predict next motion for the robot and, by implementing this neural network-on-chip, we can reduce delay and can perform prediction in real-time. Here, we avoided the hidden layers because of the lesser complexity of our application if the complexity of classification is higher.

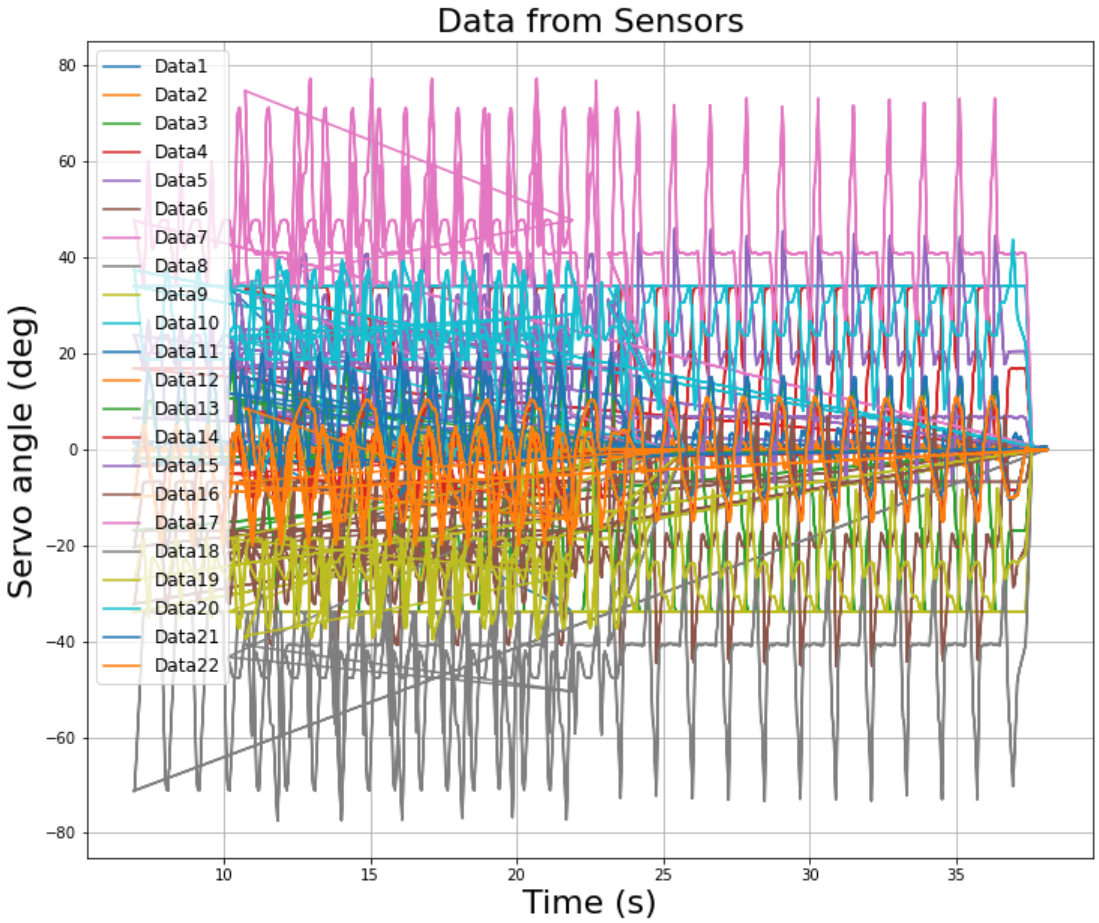

Figure 2.1: Data of 22 different Servo angles are obtained using the Kondo KHR-3HV robot [42].

2.1.2 Implementing Network using TensorFlow

NN-based classified data is shown in Figure 2.4, Whereas Figure 2.2 shows each layer except the input layer, there are artificial neurons, and each neuron operates on two basic functions i.e., transfer function and sigmoid function. 1) In the transfer function, there are two operations, multiplying weight values with input and then adding it with bias value. 2) The sigmoid activation function decides how likely the prediction is true for a given input set. As shown in Figure 2.2 there

are two operations, which are performed to achieve the predicted output range between 0 to 1. That's why the sigmoid function is used as an activation function to get an S-shaped curve that has an output range between 0 to 1. Therefore, the sigmoid function has a very important operation that must be implemented very precisely.

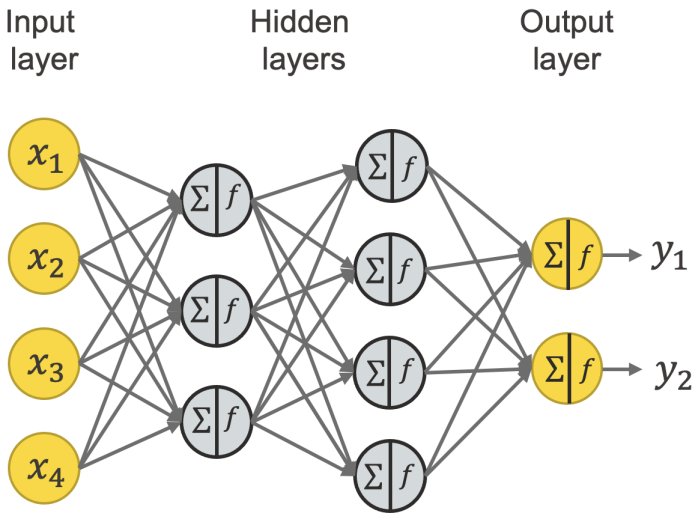

Figure 2.2: Simple dense artificial neural-network (ANN) containing two-hidden layers.

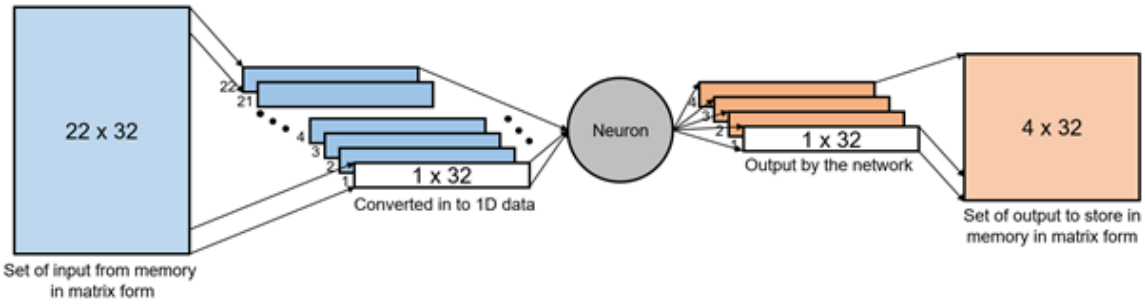

Figure 2.3: Matrix based representation of proposed ANN.

As shown in Fig 2.3, we have collected the data and store it in memory and read it as matrix (22 x 32). Here all inputs are stored in each row, then we have separated those input and make it to 1D data for the process. We provided all the inputs to the network which we have created using TensorFlow. NN captured the output as 1D data after that to store the result in to memory we have created matrix (4 x 32) where each row contains the output results generated by the output layer, and the results are stored in the memory. Based on the output, we tuned the sensor data to control the humanoid robot, and classified the motion to predict the

next motion by just adding one more data like proximity sensor data which gives the information about the obstacle around the robot. Here are the output results which we have generated after training and testing the network. As we can see in the Figure 2.4, we have successfully classified the different motion (Forward motion, back motion, right motion, and left motion).

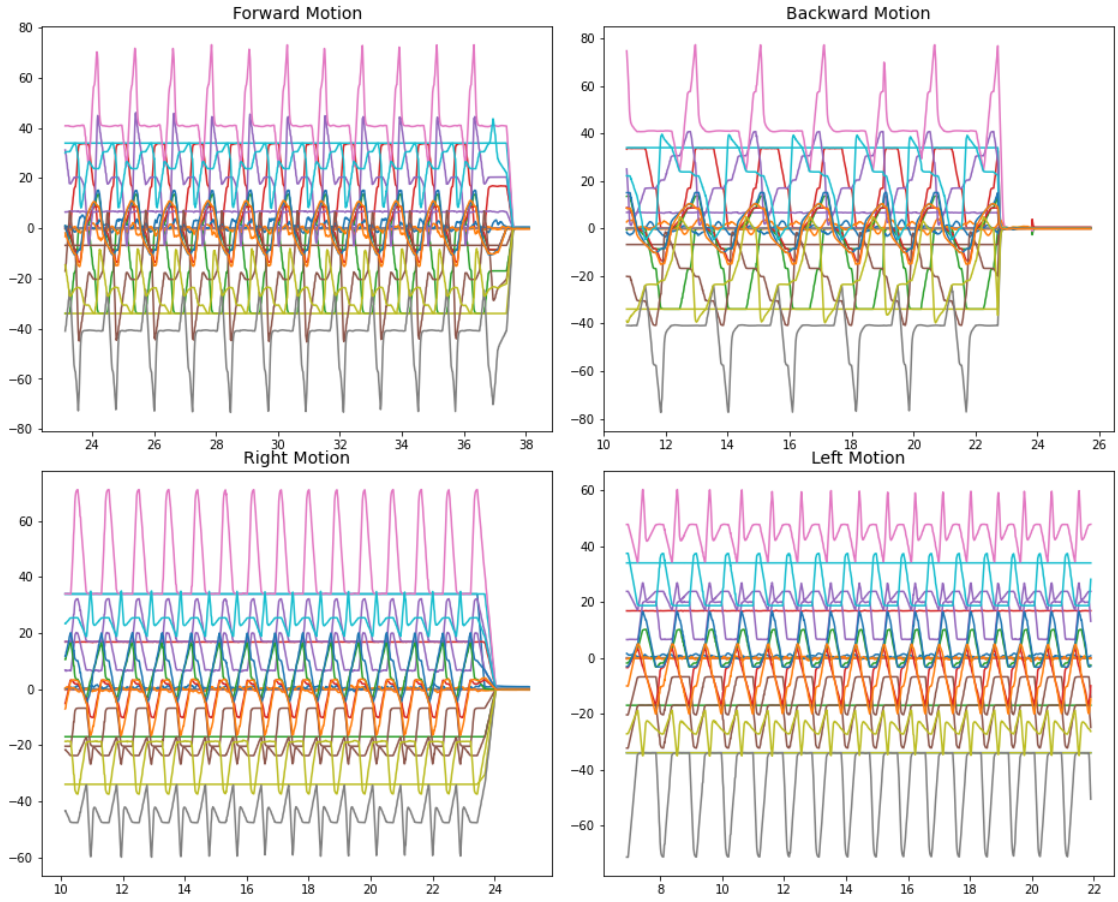

Figure 2.4: Classified data obtained using ANN created using Python.

2.2 Transfer Function in NN

Transfer functions are mathematical functions that describe a system's output $y(t)$ (a time-varying function) in relation to its input $x(t)$. It is worth noting that both the input and the output have a notation that specifies their pattern. A transfer function seeks the mathematical link between the input pattern and the desired pattern (output).

Figure 2.5 shows that the transfer function collects multiplication of the input data and weights values and adds them and feeds to the activation function in

this case it feeds the output to the sigmoid function for activation of the neuron. Equation (1.1) describes this function,

$$Z = B + \sum_{i=1}^{n} w_i * x_i \qquad \ldots \quad (2.1)$$

Here, Z is the final output that feeds to the sigmoid function. We considered n (1, 2, 3, ..., n) number of inputs, corresponding weight values wi varies from w1 to wn same for input data, varies from x1 to xn. Here B is the bias value which is fixed for each neuron. The bias and weight values are obtained after completing the training of the network.

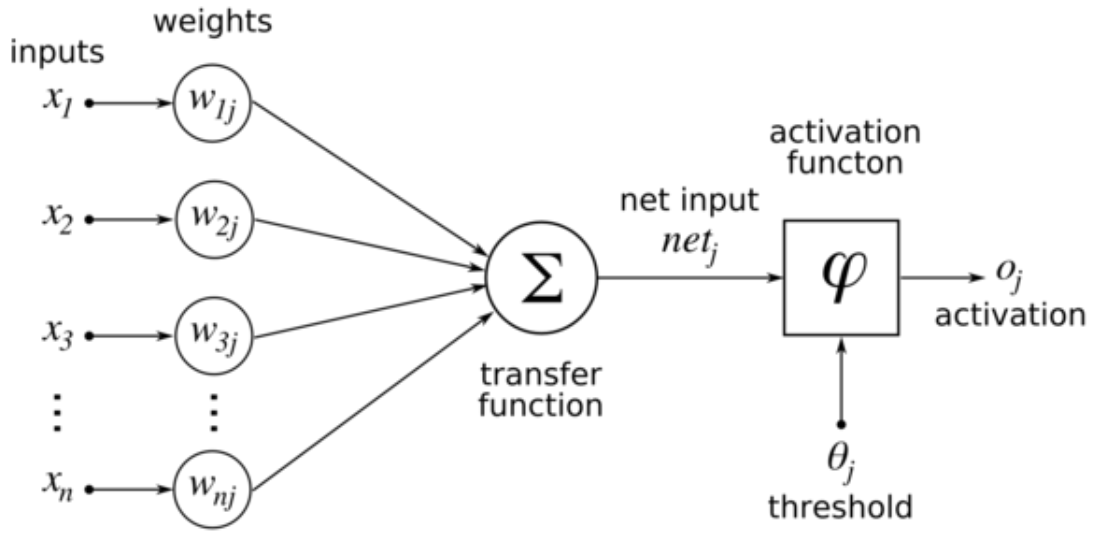

Figure 2.5: Transfer function and activation function used in ANN.

2.3 Activation Function in NN

Each neuron in an artificial neural network creates a weighted sum of its inputs and sends the resulting scalar value through a function known as an activation function or transfer function. If a neuron has n inputs $x_1, x_2, ...x_n$ the output or activation of a neuron is $a = g(w_1x_1 + w_2x_2 + w_3x_3 + ...w_nx_n + b)$. The activation function is denoted by the function g. The neuron performs linear regression or classification if the function g is considered as the linear function $g(z) = z$. To do nonlinear regression and solve classification issues that are not linearly separable, g is assumed to be a nonlinear function. When g is assumed to be a sigmoidal or's' shaped function with values ranging from 0 to 1 or -1 to 1, the neuron's

output value can be read as a YES/NO answer or binary choice. In deep networks, however, a saturating activation function might result in the vanishing gradient issue. For the first time, deeper networks may be trained by replacing saturating sigmoidal activation functions with activation functions with greater derivative values, such as ReLU. Non-monotonic and oscillatory activation functions that surpass ReLU have subsequently been discovered [41].

2.3.1 Linear or Identity Activation Function

A linear function is one with a straight line as its graph. The linear function is written as $y = f(x) = a + bx$. Linear functions include independent variables and dependent variables. The independent variable is x, and the dependent variable is y.

Range: (Infinity to Infinity) Not useful for the complexity and various parameters of normal data supplied to neural networks.

2.3.2 Non-linear Activation Function

The nonlinear activation function is the most used activation function. Non-linear means that the graph is not a straight line. The non-linear function graph is a curve. A curve is a line that changes direction constantly. Note: Economists are accustomed to calling every line in a graph a curve, both a straight line and a line that is actually a curve. This makes it easier to generalize or fit different data in the model and distinguish the output.

1. Sigmoid Activation Function

The sigmoid function curve is shaped like a S. The sigmoid function is useful because it occurs between (0 and 1). Figure 2.6 shows the S shape curve generated by the Sigmoid function. Therefore, this is especially used for models that need to predict probabilities as output. Sigmoid is the right choice because the probability of something is only between 0 and 1.

$$a = \frac{1}{1 + e^{-z}} \qquad \ldots \quad (2.2)$$

The function is differentiable. That is, you can find the slope of the sigmoid curve at any two points. The function is monotonous, but the derivative of the function is not monotonous. Logistic sigmoid functions can cause neural networks to hang during training.

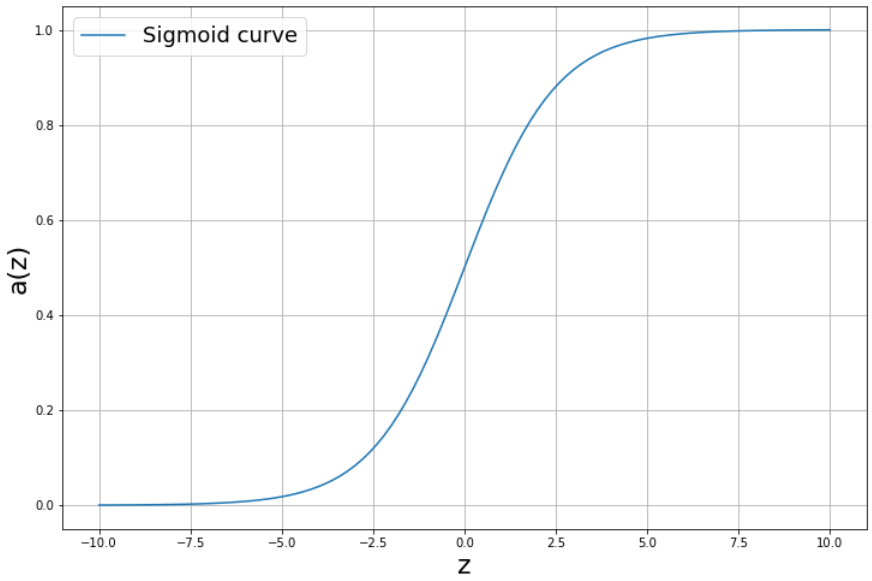

Figure 2.6: Sigmoid activation function curve in [a.u.].

2. Hyperbolic Tangent or tanh Activation Function

tanh is related to, but superior to, logistic sigmoid. The tanh function has a value range of (-1 to 1). Tanh is a sigmoid as well (S-shaped). The curve obtained using tanh function is shown in Figure 2.7.

$$a = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad \ldots \quad (2.3)$$

In the tanh plot, negative inputs are strongly negatively mapped, whereas zero inputs are mapped near zero. The function can be differentiated. Although the function is monotonic, its derivative is not.

3. ReLU Activation Function

ReLU (Rectified Linear Unit) is the most widely utilised activation function in the world today. It has since been used in practically all convolutional neural networks and deep learning algorithms.

$$a = max(0, z) \qquad \ldots \quad (2.4)$$

Range: [0 to infinity), both the function and its derivatives are monotonous. The concern is that all negative numbers rapidly become zero. This affects the model's capacity to correctly fit or train the data. This implies that a negative

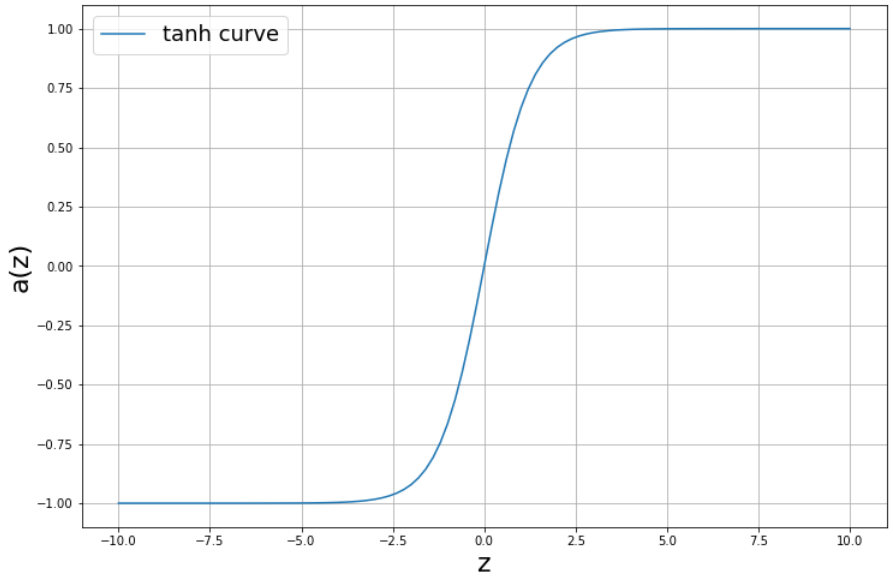

Figure 2.7: tanh activation function curve in [a.u.].

input to the ReLU activation function will immediately reset the chart's value to zero. This has an effect on the resultant chart by incorrectly mapping negative values. Figure 2.8 shows the curve generated by the ReLU activation function.

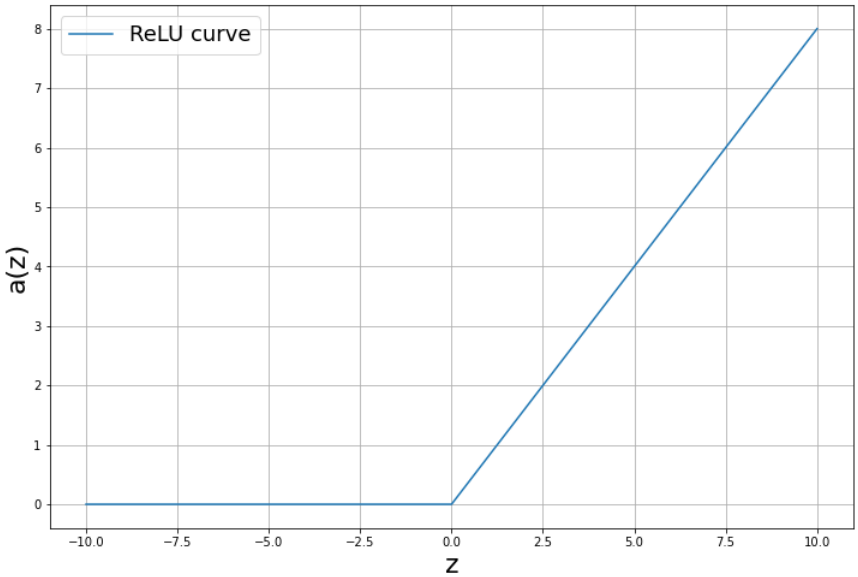

Figure 2.8: ReLU activation function curve in [a.u.].

2.4 Sigmoid Activation Function

The sigmoid function is a form of activation function that is commonly employed in artificial neural networks (ANN) [45] [46]. A sigmoid equation is a mathe-

matical function with a characteristic "S" shaped curve or sigmoid curve, and its output range is typically 0 to 1. Because of the form and output range of the curve, NN uses it.

$$s(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1} \qquad \ldots \quad (2.5)$$

Two basic operations are observed in the equation which are an exponential component and division. Direct implementation of these two operations at the hardware level is quite complex. Therefore, instead of direct implementing Equation (2.5), we can approximate it up to second-order which is described below.

2.4.1 Second-order Sigmoid Approximation

Now, as we have seen that sigmoid function contains various complicated operations such as exponential and division operation which makes hardware-level implementation complicated. So, to make it easier we have proposed an efficient way of implementing sigmoid function using second-order approximation. The proposed function contains operations such as multiplication, addition, and subtraction which are not complex as compared to exponential and division operations. The proposed equation of the sigmoid function using second-order approximation is mentioned in Equation (1.6). The result of second-order approximation is almost accurate, also the complexity of the equation is reduced significantly as it requires multiplication, addition, and subtraction operation instead of exponential, so the implementation of this equation is simpler compared to the conventional one. According to our discussion on the proposed equation, the flow for the equation is depicted in Fig. 2.9.

$$s(z) = \begin{cases} 0, & z < -4 \\ \frac{1}{2}(\frac{z}{2^2}+1)^2, & -4 \leq z \leq 0 \\ 1-\frac{1}{2}(\frac{z}{2^2}-1)^2, & 0 < z \leq 4 \\ 1, & z \geq 4 \end{cases} \qquad \ldots \quad (2.6)$$

As we can see in the Figure 2.9 there is only three major operations are going to perform based on the input we provide, and the operation are addition, subtraction, and multiplication there is not any division or exponential operation are required now so the complexity of the sigmoid activation function is reduced significantly. For the comparison we have implemented the original sigmoid function with exponential and division term in it, and approximated sigmoid function we have implemented with help of Equation (2.6) which we have derived using

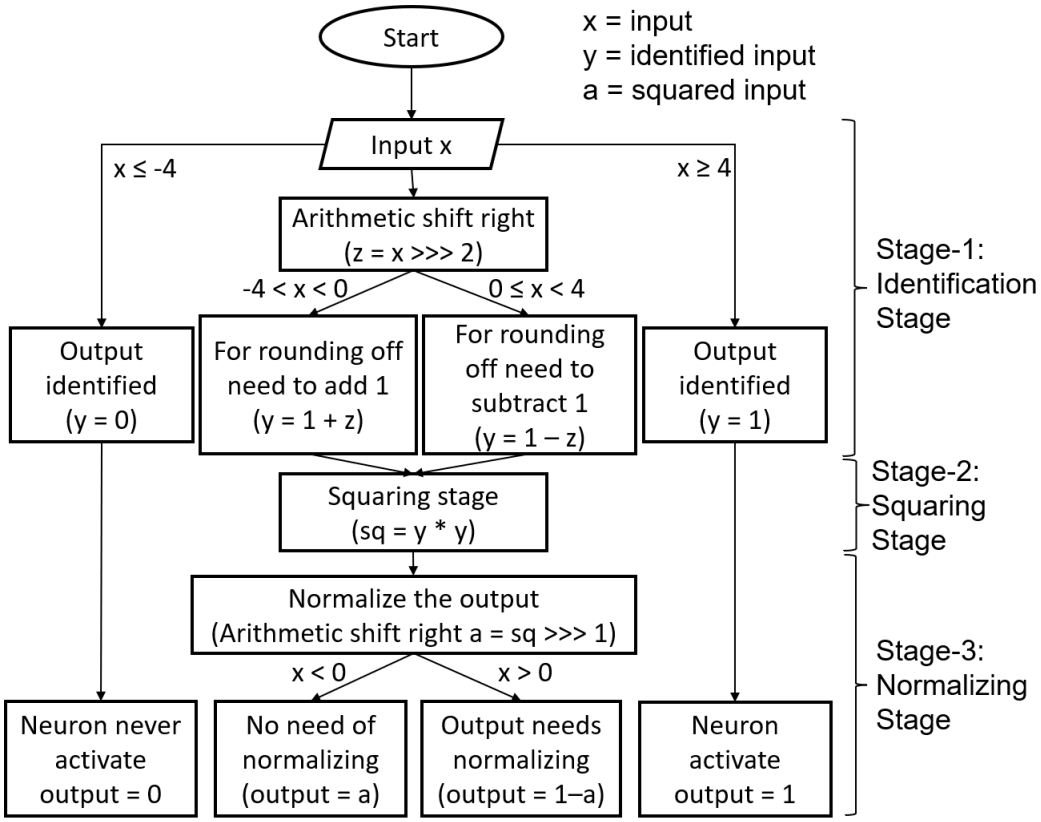

Figure 2.9: Flow of proposed approximated sigmoid function.

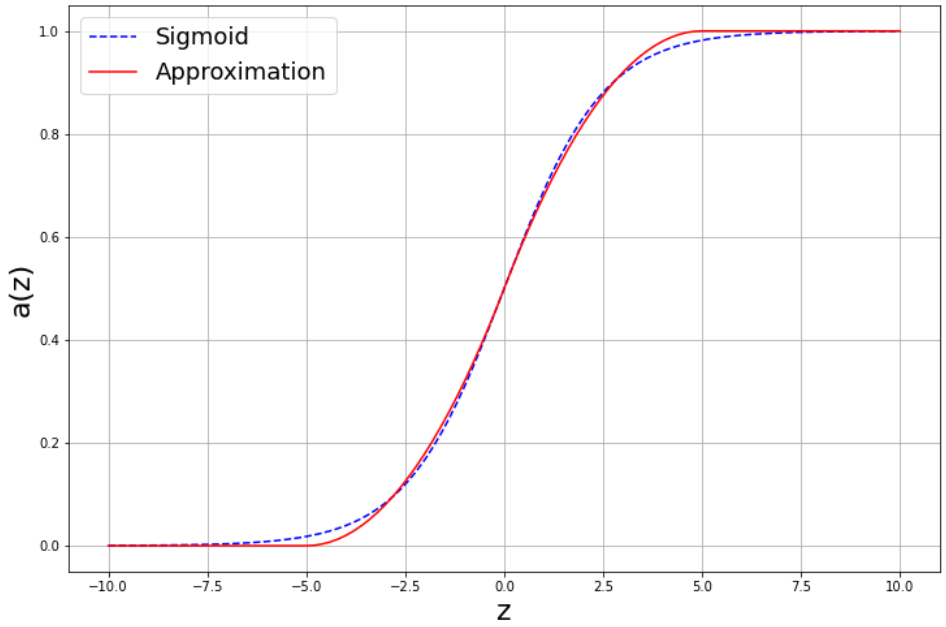

Figure 2.10: Comparison between conventional sigmoid function and proposed sigmoid function.

approximation method. Using the proposed sigmoid function, we avoided the exponential operation, also the accuracy is very high. Figure 2.10 shows the comparison between conventional sigmoid function and proposed sigmoid function. We observed that the curve of the proposed sigmoid function is S-shaped, almost identical to the conventional one.

As shown in Figure 2.10 blue doted line represents the output of conventional sigmoid function whereas the red line represents the output obtained by the second-order approximation of sigmoid function. Hence, the results obtained are almost identical and thus we have used second-order approximation function.

CHAPTER 3
Architecture of Neural Network and AFSM

As we have discussed in previous chapter we are using Artificial Neural Network to classifying the humanoid robot motions, after that we are predicting the next motion for the robot by using the classified data and the extra input in our case its proximity sensor data. Using this data we have build Finite State Machine which can provide next action which should be perform by the robot. Proposed architecture for robot motion detection is shown in Figure 3.1.

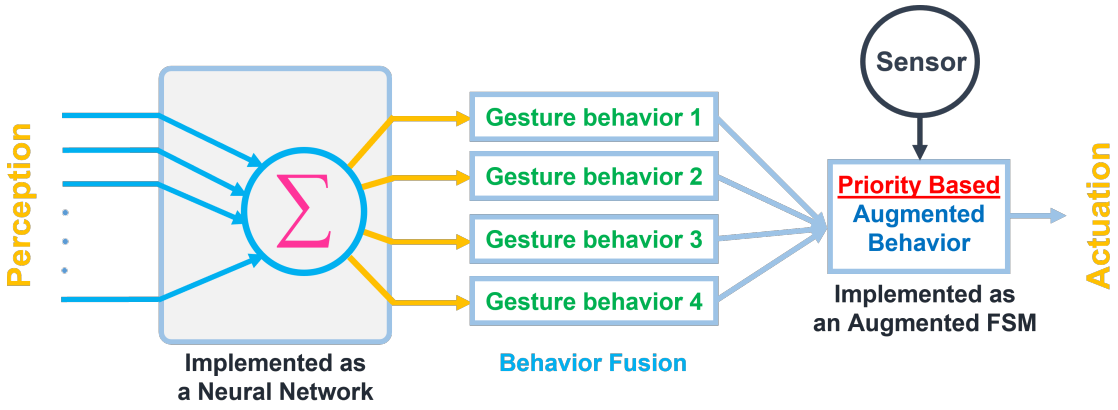

Figure 3.1: Proposed behavior-based architecture implemented using a neural network (NN) and an augmented finite-state machine (AFSM).

Figure 3.1 is the proposed method for predicting the next motion and to predict the motion first step requires the collecting the data and that has to be provided to the Artificial Neural Network for classification. Based on the values obtained from each neuron i.e., in the range of 0 to 1 and the proximity sensor data, the next motion is being predicted.

3.1 Sigmoid Activation Function

Since activation function is the heart of ANN, it has to function as fast as possible and also it should be accurate because the neuron activation depends on the

activation function. Here we have chosen sigmoid function as activation function because it gives 'S' shape curve and the output range it provides is 0 to 1 so we can easily identify the activated neuron. Since sigmoid function contains exponential and division operation we have proposed new approach to perform sigmoid function. We have used second order approximation method to approximate the results of sigmoid function which are almost identical to conventional sigmoid function.

3.1.1 Proposed Architecture of Sigmoid Function

Hardware implementation of the sigmoid function requires adder, comparator, registers, and multiplier [47] [48] [50]. Figure 3.2 shows the simplified block diagram for the proposed architecture where Figure 3.3 gives the information of the data path for the digital logic design.

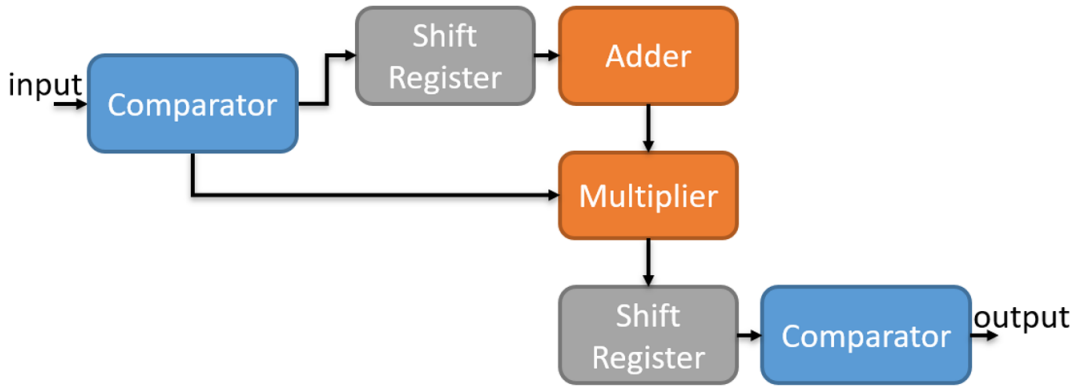

Figure 3.2: Simplified block diagram of proposed sigmoid function.

Proposed architecture doesn't consist of any exponential or division operation. Instead, only addition, subtraction, and multiplication operations are used as shown in Equation (2.6). Here we have divided the flow in three parts (for pipeline) so we can improve the speed of the operation more. The first stage known as Identification stage, second stage is known as squaring stage and the third stage is known as normalizing stage.

RTL generated for the proposed architecture is shown in Figure 3.4 and we have used pipeline method to optimize the architecture and it will improve the speed of operation by significant margin.

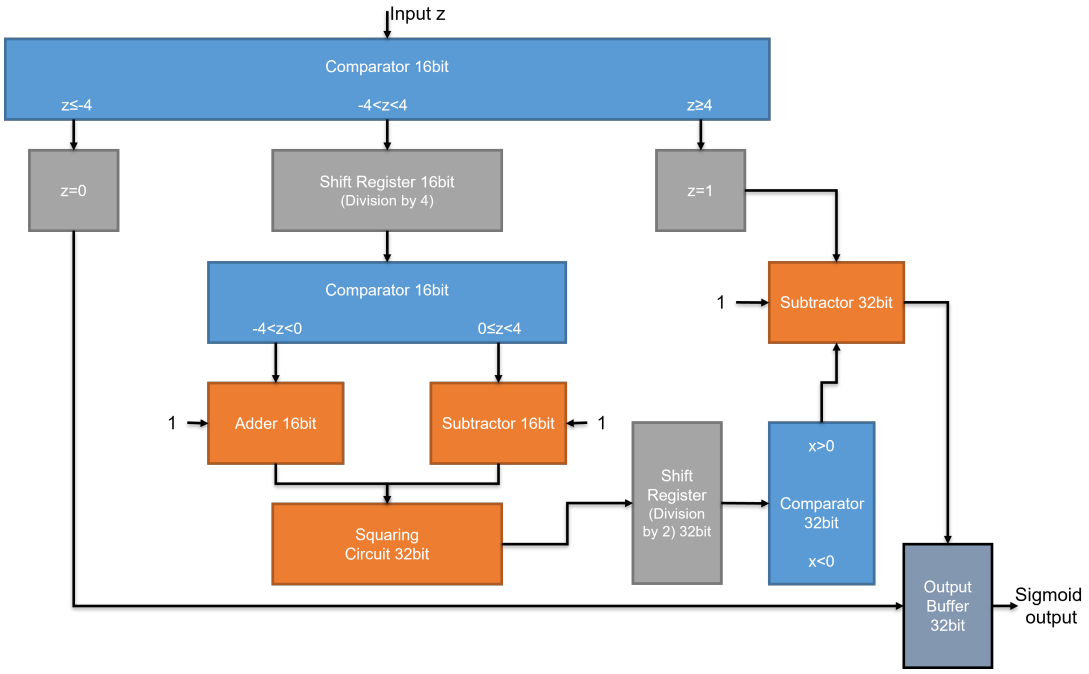

Figure 3.3: Architecture of proposed sigmoid function.

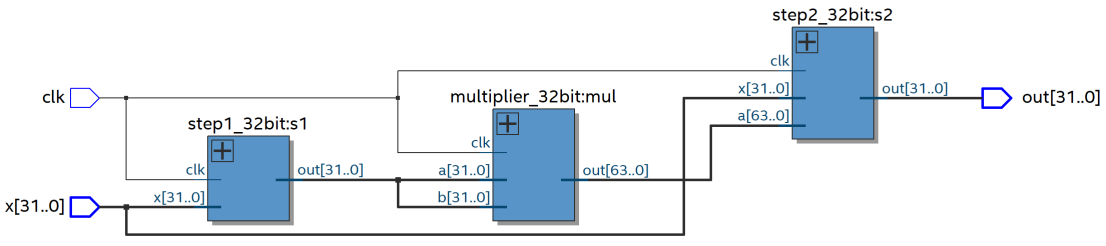

Figure 3.4: RTL overview of the proposed sigmoid function.

Stage-1: Identification Stage

RTL generated for the stage-1 (Identification stage) is shown in Figure 3.5 which contains comparators, adders and subtractors.

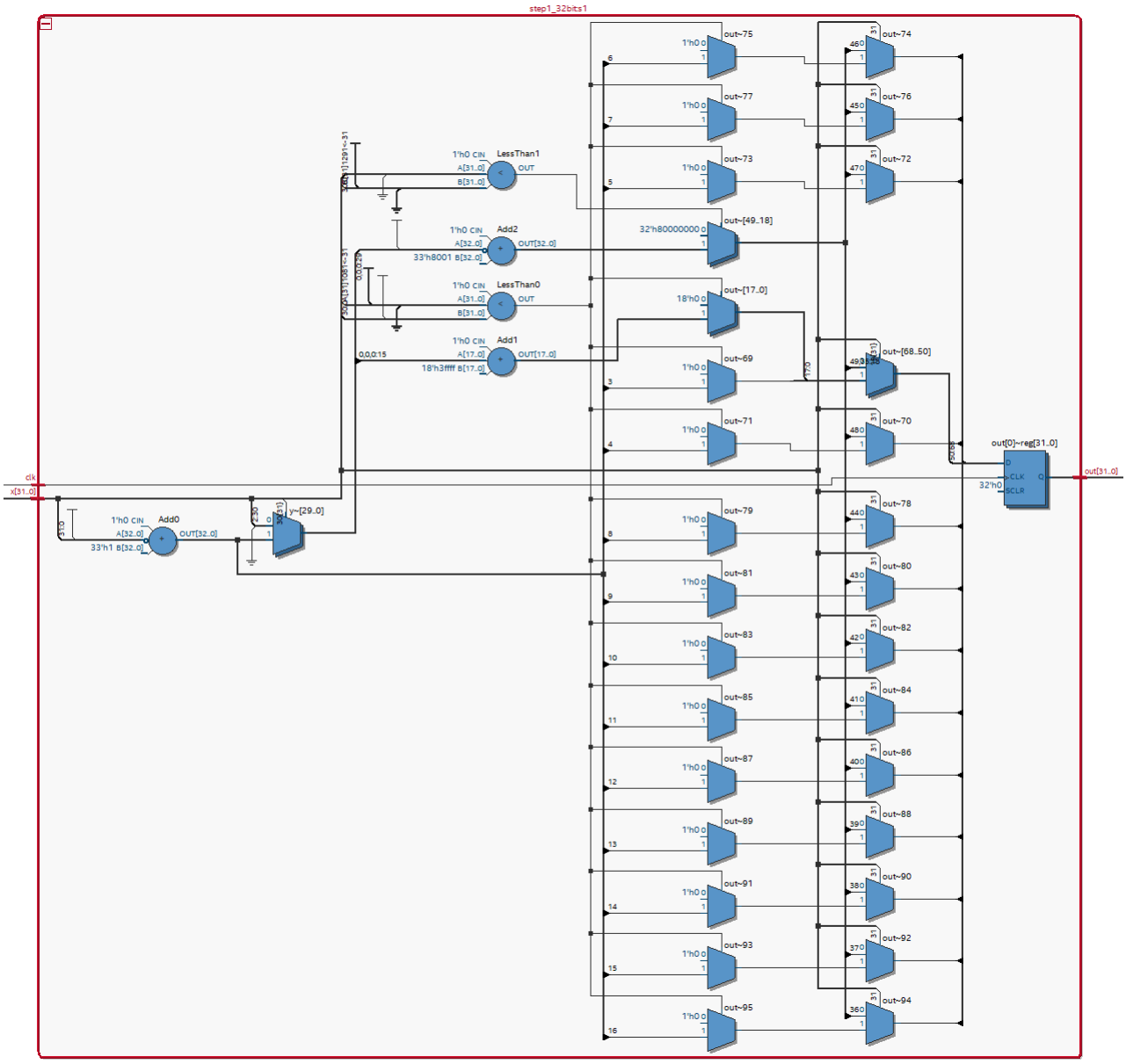

Figure 3.5: RTL for stage-1 (identification stage).

In this stage we are identifying the input range in three different range whether the input is less than -4 or it is in between -4 to 4 or it is greater than 4. Based on the input range we are performing arithmetic shift right operation (divide by 4) and add 1 or subtract 1 according to the input, if input is negative then we add 1 and if its positive then we subtract 1. So, in simple term stage one is performing two operations based on the input range and the operations are mention below

$$\frac{z}{2^2} + 1, \qquad if \quad -4 < input < 0 \qquad ... \quad (3.1)$$

$$\frac{z}{2^2} - 1, \qquad if \quad -4 < input < 0 \qquad ... \quad (3.2)$$

Stage-2: Squaring Stage

In this stage we are performing multiplication operation to generate the square of the output generated by the stage 1. Compare to adder and subtractor multiplier is more complex that's why second stage only contains one operation in order to generate output faster. Squaring stage will give the output in from of

$$(\frac{z}{2^2} + 1)^2, \qquad if \quad -4 < input < 0 \qquad ... \quad (3.3)$$

$$(\frac{z}{2^2} - 1)^2, \qquad if \quad -4 < input < 0 \qquad ... \quad (3.4)$$

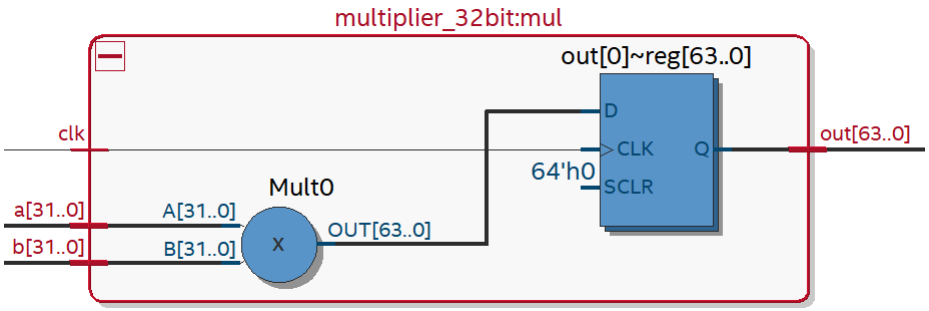

Figure 3.6: RTL for stage-2 (squaring stage).

Figure 3.6 is the RTL generated for the stage-2 (Squaring stage). Here we have used array multiplier for simplicity. Also, to optimize the performance, we can use Modified booth multiplier or Wallace tree multiplier.

Stage-3: Normalizing Stage

In this stage we are normalizing the output generated by the squaring stage and also reduced the number of bits of the output to saving space in memory. Since the output generated by the sigmoid function is in range of 0 to 1 we can remove the MSB bits because its always remains zero. Here we have use fixed point representation so we can change the range of floating bits according to requirement. RTL for the stage-3 is shown in Figure 3.7.

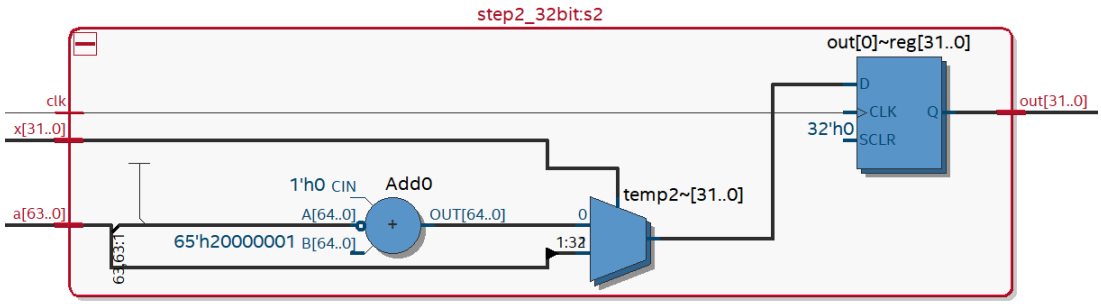

Figure 3.7: RTL for stage-3 (normalizing stage).

3.1.2 Behavioural Analysis of the Sigmoid Function

For sigmoid function, we used multiple components and, we have computed the output in three stage-1 (Identification stage), stage-2 (Squaring stage), and stage-3 (Normalizing stage). Identification stage performs addition and subtraction based on the given input and initially. The data passing input through the comparator to identify which operation should be perform on the input after that we are performing the squaring (multiplication) the output generated by the Identification stage and in the final Normalizing stage. The final output generated by the Normalizing stage is in range of 0 to 1 same as sigmoid function output for that we used to fix point conversion where, we have fixed 28 bits as fraction bits and remaining 4 bits for signed integer the reason for taking more number of bits for fraction is because the output, generated is in range of 0 to 1.

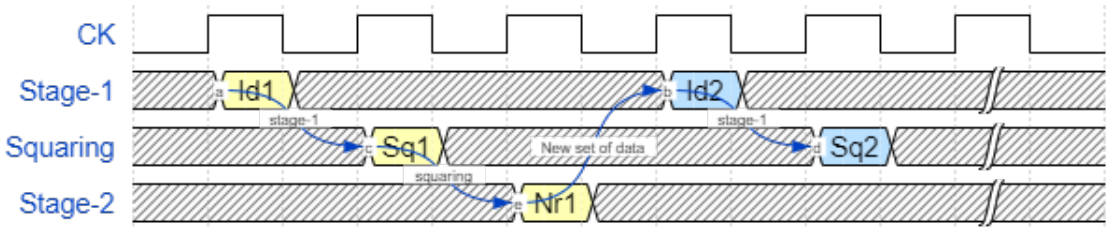

Figure 3.8: Data flow for Sigmoid function without pipeline.

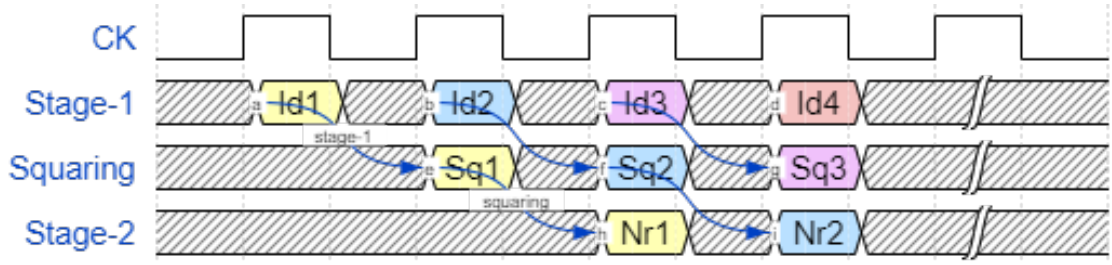

Figure 3.9: Data flow for Sigmoid function with pipeline.

As we discussed in data flow for sigmoid function, we have used three stages which are observed in the wave form and after completing Normalizing stage. It will load new data for the process, also by using pipeline we have optimize and reduce time to generate output. We generated the output in each clock cycle instead of generating output every third clock. Figure 3.8 and Figure 3.9 are for without pipeline and with pipeline and we can observe that with pipeline we can generate output much faster than the architecture without pipeline.

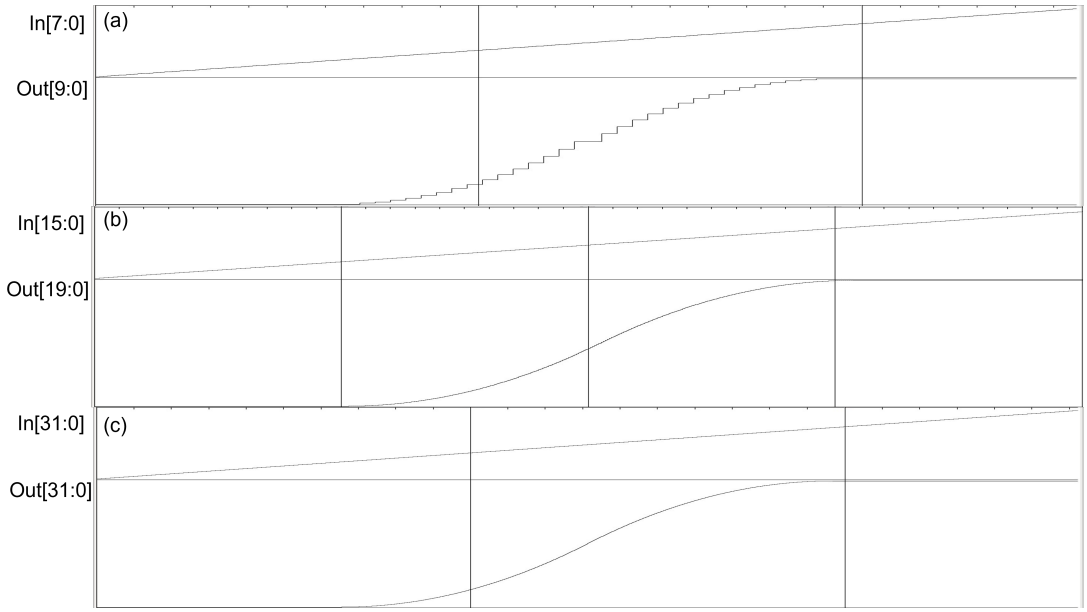

Figure 3.10: Simulated output of a proposed Sigmoid function with various precision. (a) with 8bit of input, (b) with 16bit of input and (c) with 32bit of input, respectively.

We have discussed earlier in Chapter-2 is that an activation function plays a crucial role in NN, which makes it necessary for the hardware implementation to be accurate, with minimal errors. The hardware-level implementation is performed for different numbers of input bits i.e., the higher the number of bits, the higher is the accuracy, as well as precision which we can observe in Figure 3.10. During testing, we have considered 8-bit, 16-bit, and 32-bit fixed numbers as an input to the proposed sigmoid function, one at a time. Of course, 32-bit fixed input would have higher precision, also the shape of the curve is sharper compared to the other two inputs i.e., 8-bit and 16-bit, but at the cost of more memory space. In 8-bit fixed input, 4 bits are used for representing fractions. In 16-bit, 8 bits are used for representing fractions. In 32-bit, 14 bits are used for the same. Hence, 8-bit fixed input is having a precision of 6.25×10-2, for 16-bit it becomes 3.9×10-3 and for 32-bit it becomes 6.1×10-5. For output the case is different, 8-bit, 16-bit, and 28-bit are used to represent fractions, also the precision comes out to

be 3.9×10-3, 1.52×10-5, and 3.72×10-9 respectively for 8-bit, 16-bit, and 32-bit fixed inputs.The precision values for input data and output data is mention in the Table 3.1 and Table 3.2 respectively. Hence, we have proposed the hardware design of the sigmoid function with 32-bit fixed input for higher precision.

Representation for input	No. of fraction bit for input	Input precision
8 bits	4 bits	$6.25 * 10^{-2}$
16 bits	8 bits	$3.9 * 10^{-3}$
32 bits	14 bits	$6.1 * 10^{-5}$

Table 3.1: Comparing precision for different input.

Representation for output	No. of fraction bit for output	Output precision
12 bits	8 bits	$3.9 * 10^{-3}$
20 bits	16 bits	$1.52 * 10^{-5}$
32 bits	28 bits	$3.72 * 10^{-9}$

Table 3.2: Comparing precision for different output.

3.1.3 Power Analysis of the Sigmoid Function

As we can see in the Table 3.3 we have calculated power for technology 180nm in Quartus Prime (intel tool) we can reduce dynamic power by selecting the 8-bit sigmoid function but the precision of the 8bit sigmoid function is very less compare to other two also I/O thermal power is half compare to 32bit sigmoid function. So, we can select 16 bit sigmoid function for overall performance and power optimization. The data collected from the tool is mention in the Table 3.3.

Input size	Core Dynamic(mW)	Core Static(mW)	I/O Thermal(mW)	Total(mW)
8bits	2.18	89.93	14.69	106.80
16bits	2.59	89.96	20.78	113.33
32bits	5.33	90.00	28.64	123.97

Table 3.3: Power comparison for different set of inputs.

3.2 Architecture for the Transfer Function

Transfer function only contains array of multiplier an adder and the adders are arrange in stages according to number of inputs in our case there are 7 stages of adders since we have 22 inputs. Here we need weight values and bias values for

each neuron and all the weight and bias values are stored in SRAM and it can be obtained by training the network. We have trained our network using Python TensorFlow and collected those values for testing the network. The architecture of the transfer function is shown in Figure 3.11.

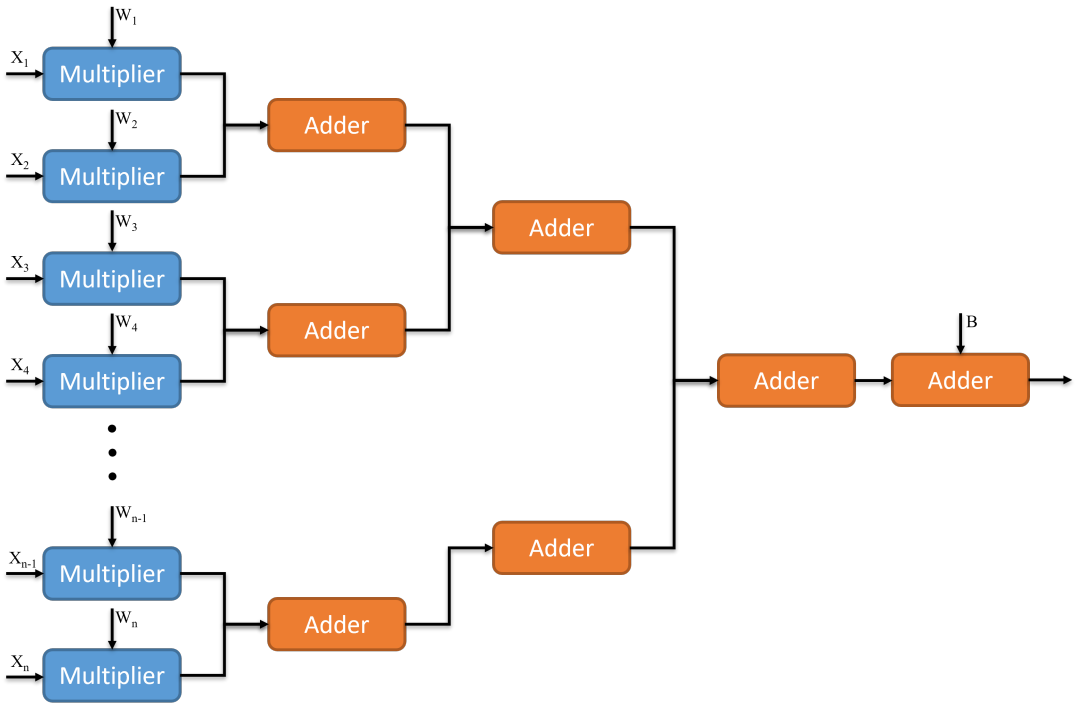

Figure 3.11: Architectural representation of Transfer function.

As per the equation we have to first multiply all the weight values with the respective input. So for that we have use the weight values which we have store in SRAM and multiply with the inputs in first stage then for addition we have use savaral stages and in the last stage we have added the bias value which we have obtained during the training phase. The output generated by the transfer function is given to sigmoid function for the enabling or disabling the particular neuron.

3.2.1 Behavioural Analysis of the Transfer Function

For transfer function we have used multiple multiplier and adder. Since the transfer function is performing only two operations, we have used two SRAM one for the storing weight and bias values, and second for storing input data and output data generated by the final circuit. Circuit consists of multiple input and multiple output, and each input and output data is having 32 bits. We used SRAM for storing all the data for the reduction of number of pins.

We can observe the waveform shown in Figure 3.12 and Figure 3.13 that there

are multiple stages to generate final output, and adder stages are dependent on the number of inputs. For example if we have 8 different inputs then we need 4 adder stages, and one multiplication stage. As we have 22 inputs, 6 adder stages and one multiplication stage is needed.

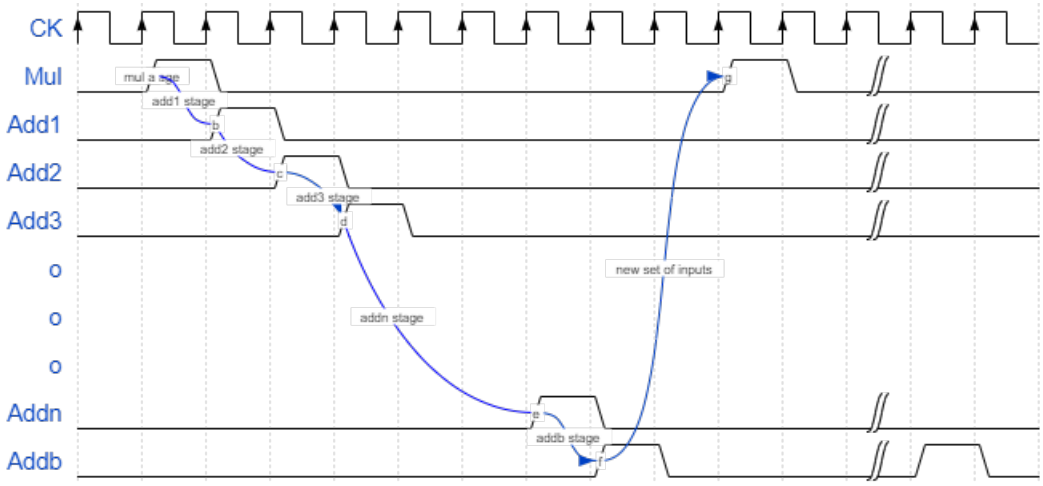

Figure 3.12: Data flow for transfer function without pipeline.

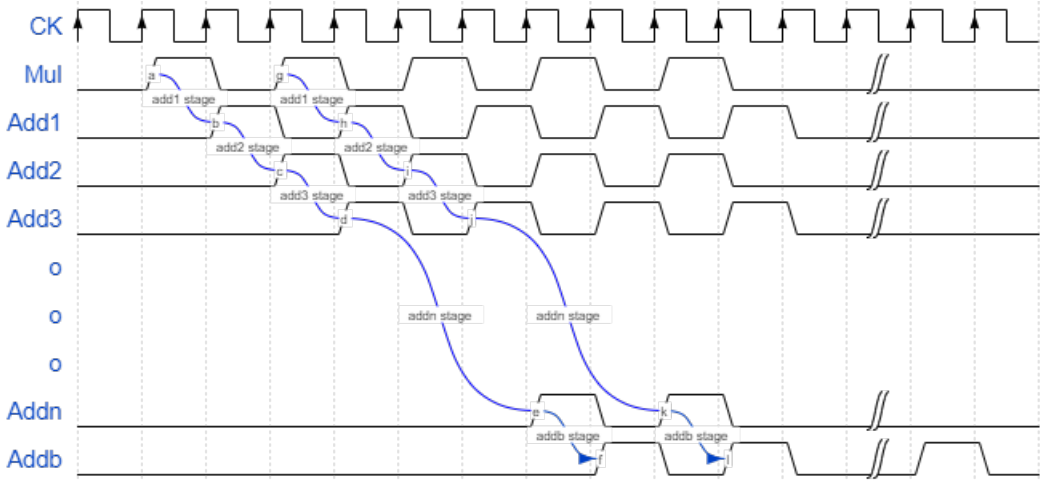

Figure 3.13: Data flow for transfer function with pipeline.

3.3 Architecture for the Artificial Neural Network

To make Artificial Neural Network we require Artificial Neuron and as we have seen earlier To mane Neuron we need Transfer function as well as Sigmoid function as activation function by connecting several neuron we can create Artificial Neural Network the architecture shown in fig. is simplified version of connecting those neuron with the inputs in such a way that we can get dense network

because here each neuron is connecting with all inputs. Also we have provided weight and bias values from the SRAM.

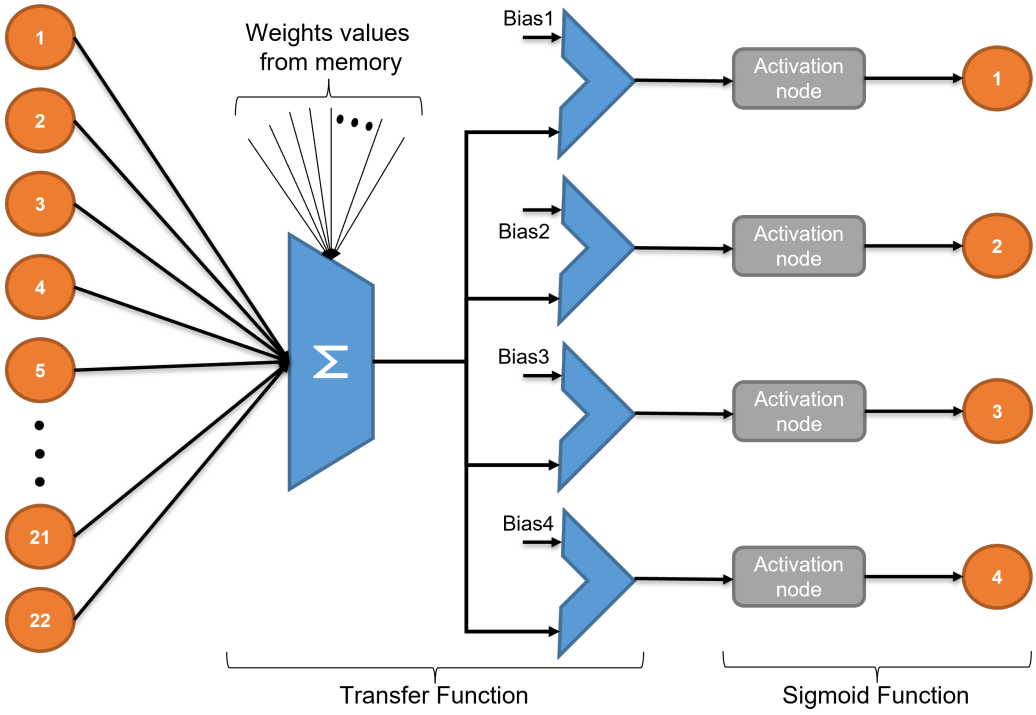

Figure 3.14: Architectural representation of Artificial Neural Network (ANN).

As shown in Figure 3.14 the architecture we have used for the Neural Network contains two major section first is transfer function and other one is sigmoid function. Here output of transfer function is fed to the sigmoid function. As we know that the output of sigmoid function is in range of 0 to 1 so to identify which neuron is activated, we need to check the neuron with value near to 1. Here We have used four neuron in output layer because we are classifying four motions. Also, we can add more number neurons as per the number of different motions being added. Also, we can add hidden layer to get more accurate results.

3.4 AFSM Design

After classifying the motion, we have used one more sensor (Proximity sensor) data to predict next motion. Initially we are trying with four basic motion forward motion, backward motion, right motion, and left motion. As shown in Figure 3.15 the state diagram we have created has many possibilities and all possible combination are mentioned in the table, since we have 4 motion we have total 16 different combination of input set Which are shown in Table 3.5 also prioritise the motion according to requirement, here we have given forward motion highest priority and backward motion has lowest priority and priorities are mention in Table 3.4.

State	Representation	Priority
Forward	00	Highest
Right	01	High
Left	10	Low
Back	11	Lowest

Table 3.4: State based on motion with their priority.

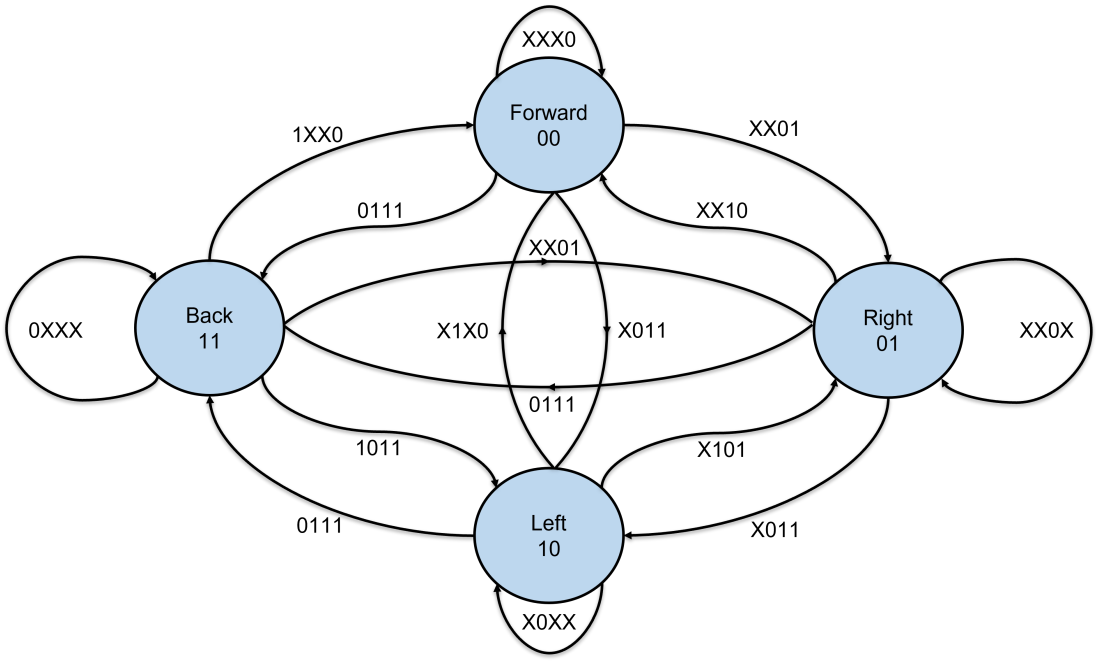

Figure 3.15: State diagram for motion prediction.

Here, we have used 4 proximity sensors in each direction to collect data and based on that data we are predicting the next motion. Here, we have given priority to forward motion. Also, we can add more motion such as jump, sit, moving

Possible input set	Representation
No obstacles	0000
Obstacle in forward	0001
Obstacle in right	0010
Obstacle in forward and right	0011
Obstacle in left	0100
Obstacle in forward and left	0101
Obstacle in right and left	0110
Obstacle in forward, right and left	0111
Obstacle in back	1000
Obstacle in forward and back	1001
Obstacle in right and back	1010
Obstacle in forward, right and back	1011
Obstacle in left and back	1100
Obstacle in forward, left and back	1101
Obstacle in right, left and back	1110
Obstacle in all directions	1111

Table 3.5: Input combination for given FSM.

upstairs, moving downstairs, etc. The state diagram will change according to the number of motion we want to classifying and based on that we can add more number of state. Complexity of the FSM will increase by adding more number of motion.

3.4.1 Behavioural Analysis of the FSM

As we have discussed earlier AFSM is use to predict motion based on the classification data and four proximity sensor data. For prediction we can define priorities and it is mention in Table 3.4 and according to the priority we have simulated the results and the results are shown in Figure 3.16.

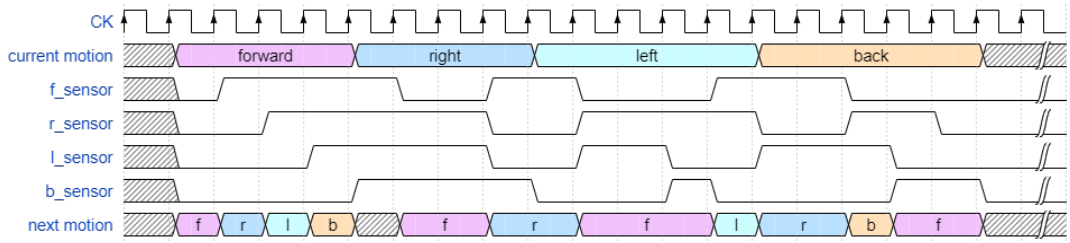

Figure 3.16: Waveform of FSM with different set of inputs.

Form the Figure 3.16 and Table 3.4 we can observe that next motion can be decided by the current motion and the sensor data coming from proximity sensors. In no obstacle case robot will take previous motion and if the obstacle come in to

the same direction where robot is moving it will take next motion accourding to the priority we have given, so in our case it will first check forward then right, left and back respectively.

3.5 Speed Analysis

As we can see in the Table 3.6 there is significant time difference when we compare the results of Python and Verilog. So, by using hardware we can improve speed significantly also there is huge difference in processing time when we compare conventional sigmoid function with the proposed sigmoid function. We can reduce around 50% time by using second-order approximation method for sigmoid function. When we compare the results of the sigmoid function with different numbers of bits we have different time so from the result we can say that 16bit sigmoid function has accurate output as well as time it takes is less. Here results obtained from Python required more processing time compared to that of Verilog.

Implementation method	Average time
ANN Using Python	900ms
ANN Using Verilog	44ns
Conventional Sigmoid (Python)	2ms
Proposed sigmoid (Python)	1ms
Proposed sigmoid 8 bits (Verilog)	2.5ns
Proposed sigmoid 16 bits (Verilog)	4.2ns
Proposed sigmoid 32 bits (Verilog)	6.5ns

Table 3.6: Speed analysis.

CHAPTER 4
Chip Design

4.1 Introduction

Electronic systems are extremely important in human life. EDA (Electronic Design Automation) tools are used to develop and produce electronic systems spanning from Integrated Circuits (ICs) to PCB (Printed Circuit Boards). EDA is a set of techniques, methodologies, and tools for automating electronic system design, verification, and testing. Cadence, Mentor Graphics, Synopsys, and other commercial EDA tools are industry standards and extremely expensive to licence. Students and teachers can only learn and execute their ideas by altering the source code using Free/Open-Source Software (FOSS) EDA tools. From schematics through layout, the Electric VLSI Design System is a powerful open-source CAD system that can handle a range of technologies such as CMOS and Bipolar. It includes a number of generic analysis and synthesis tools that can automate design and save time and effort. It also supports EDIF, VHDL, GDS, LEF/DEF, and other common interchange and manufacturing formats [52], [53], [57].

4.2 VLSI Design Flow

Figure 1.1 depicts the various degrees of VLSI design abstraction [53], [54]. These VLSI design abstraction levels are used by hardware designers to ensure that all of the standards and major goals, such as power consumption and speed, are met. The major goal at the system level using the given set of standards is to define the system divisions and their interfaces. The next stage is to use high-level programming languages to model the behaviour of the subsystem. We could determine whether the model satisfied the requirements using behavioural verification. The next phase in the design process is to specify the design at the Register Transfer Level after the specifications have been confirmed (RTL) [56]. Timing and data transfers between simple functional units like registers, full-adders, de-

coders, counters, and so on are defined in RTL. The design is made up of gates, latches, and flip-flops at the gate level. The precise area and delay are estimated at this level [55]. It also completes the design's floor planning, positioning, and sequencing. The designer can utilise the Electric VLSI tool to design an IC from the gate level to the physical level. It cannot simulate Verilog code because it lacks an integrated Verilog simulator, but it can create Verilog decks for simulation in other Verilog simulators.

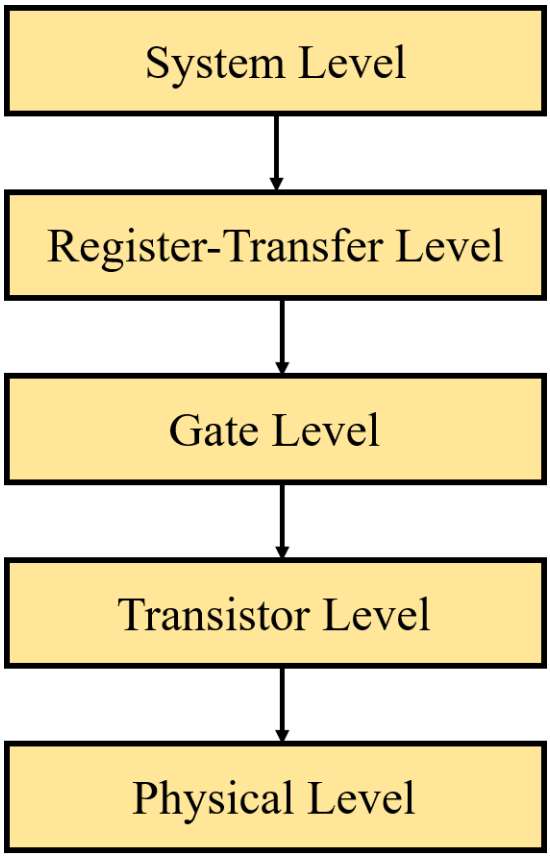

Figure 4.1: VLSI design abstraction levels.

4.3 Electric EDA Tool Design Flow

Figure 4.2 shows the flow for the EDA tool for layout design. So, to make layout of the circuit we have to follow these steps and each step several process such as Schematic design, Schematic verification, Schematic simulation, Layout design, etc.

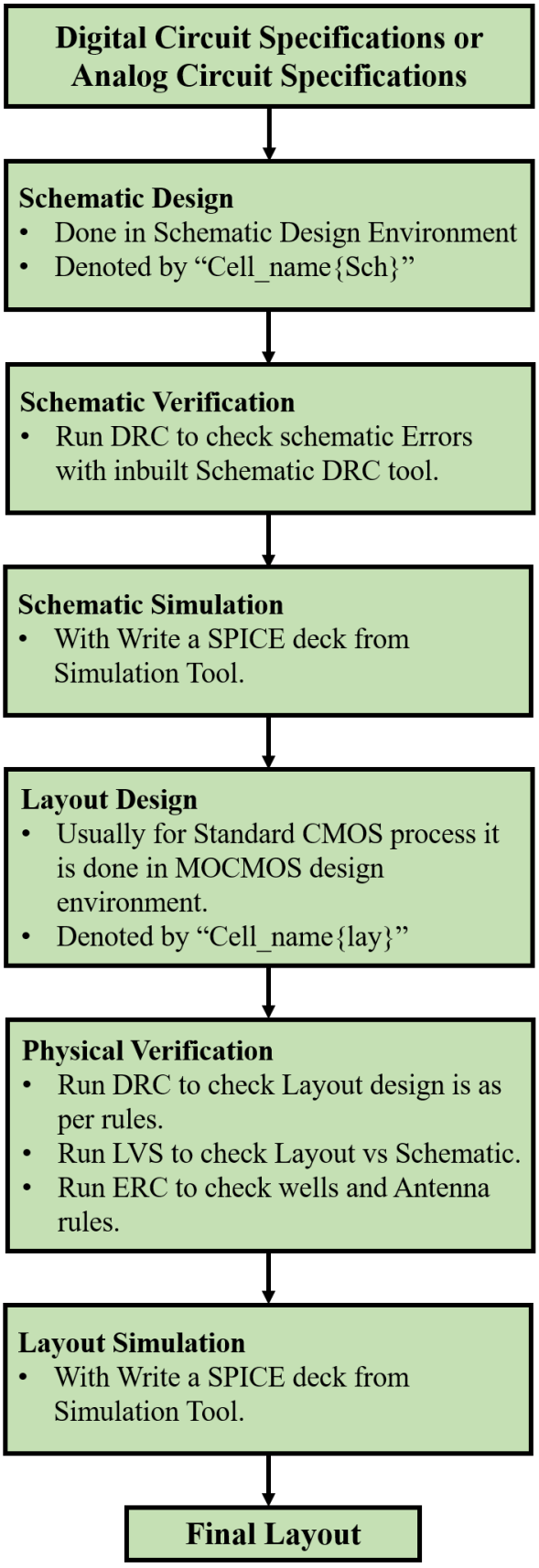

Figure 4.2: EDA tool design flow.

4.4 Digital Integrated Circuits using Electric VLSI

All of the physical level designs were created using MOSIS design guidelines for TSMC 180nm technology. The transistor model files were downloaded from the MOSIS website.

As we have discussed in previous chapters to design Neural Network we require several mathematical operations to be perform and to design those elements we need logic elements such as AND, OR, NAND, NOR, Shift register, Comparator etc. Where as for arithmetic operation we need Adder, Subtractor, Multiplier, Counter, etc.

4.4.1 Logic and Arithmetic Elements

For designing Logic elements we need some standard cell such as 2 input AND, OR, NAND, NOR and also we need 3 input and 4 input standard cells to design 32 bit logical elements.

Using those standard cell we have design 32bit Comparator we have use several 2 input, 3 input and 4 input AND, OR, Ex-OR, and inverters and the design we have created for Comparator, similarly to design arithmetic unit we need full adder as basic element and using that adder we can design Multiplier, Subtractor, Counter, etc. so for designing the transfer function we require multiple adder as well as adder in each stage, for that we have use sevaral adder and design array of multiplier and array of adder which can perform addition and multiplication as per the function suggest the cross section of that array of multiplier and array of adder is shown in Figure 4.3. and the layout of transfer function and sigmoid function are shown in Figure 4.4 and Figure 4.5 respectively. As shown in the Figure 4.4 layout there are two major components in Transfer function i.e., array of multiplier and array of adder. Sigmoid function contains comparator, multiplexer, register for holding data for short time, adder, subtractor and multiplier which is shown in Figure 4.5.

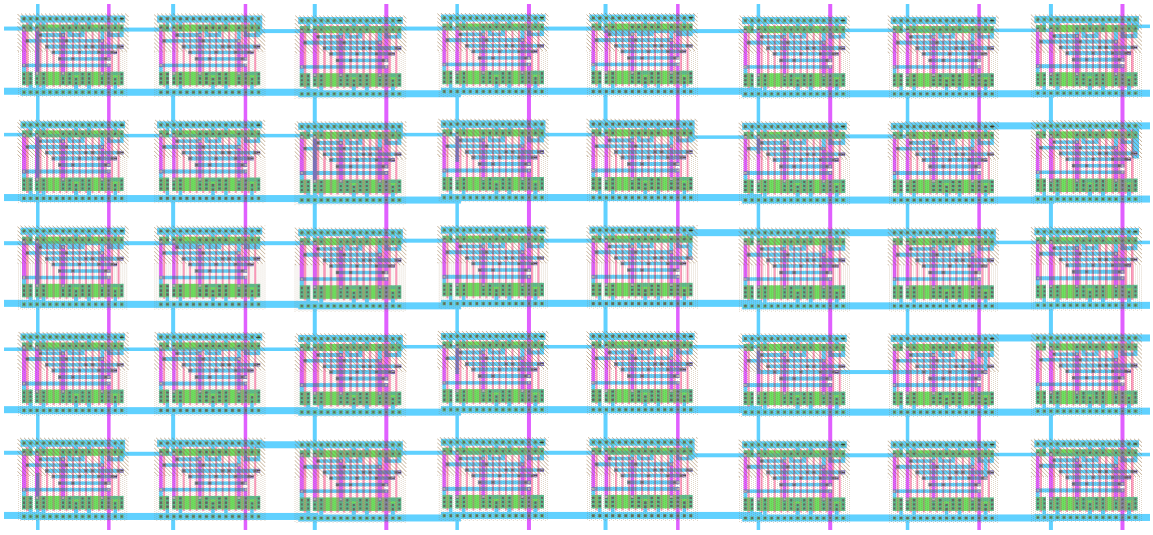

Figure 4.3: Cross section of the array of adder and multiplier.

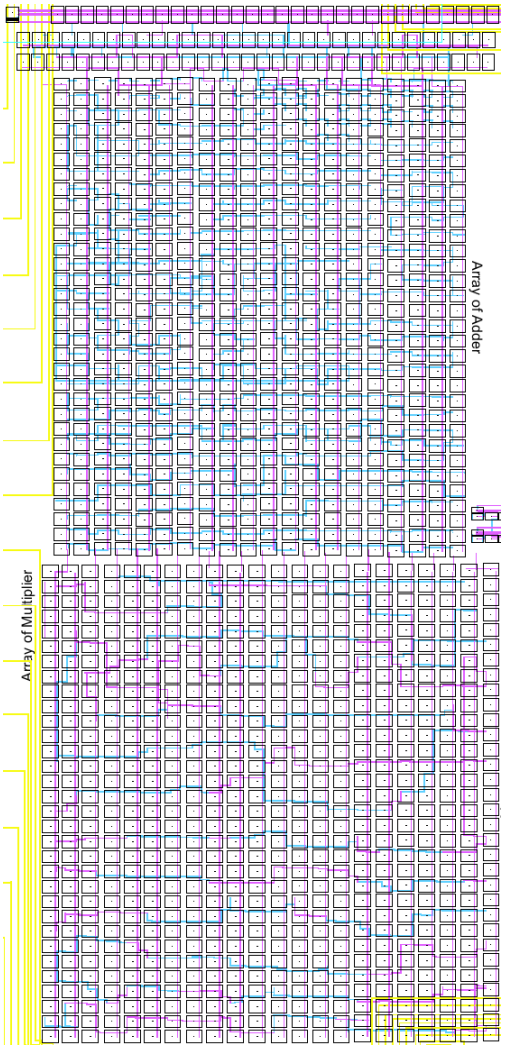

Figure 4.4: Layout of transfer function.

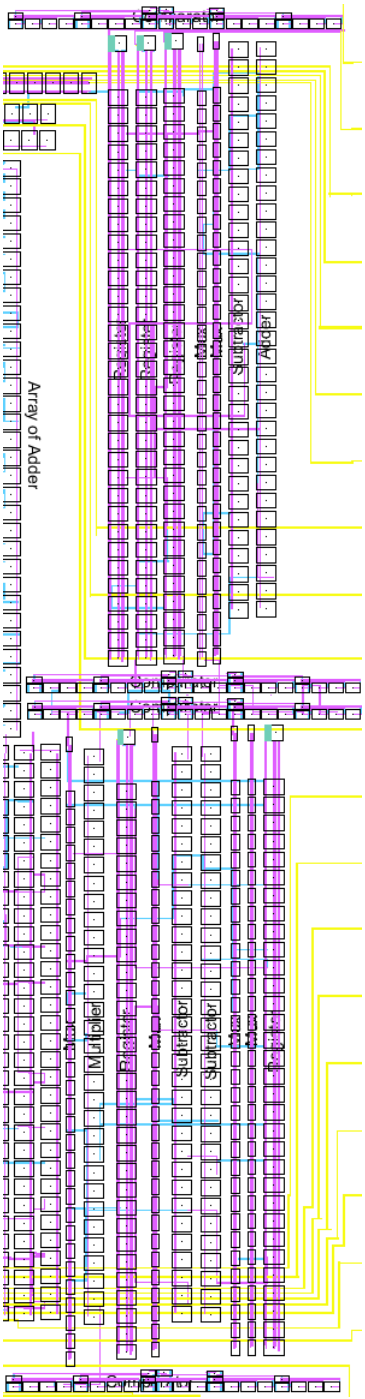

Figure 4.5: Layout of Sigmoid function.

4.4.2 Designing of Storage Element

Storage element is important element in the design because it will stores the weights, bias and some other important values which are very important for proper operation. Also we need Registers which are very fast and good for holding data for short time period. To improve the speed of operation the read and write operation perform in storage element should be as fast as possible.

Register design

Registers can be used as temporary storage elements because it is fastest storage element we can get and for some case we have hold data for short time so for that we can use registers instead of SRAM. To design register we can use flip-flop as we all know flip-flop can also hold data so by using flip-flop we can make register which can hold data for short time. Here we have use master slave D flip-flop to make register. The schematic of 1 bit master slave D flip-flop is shown in figure 4.6.

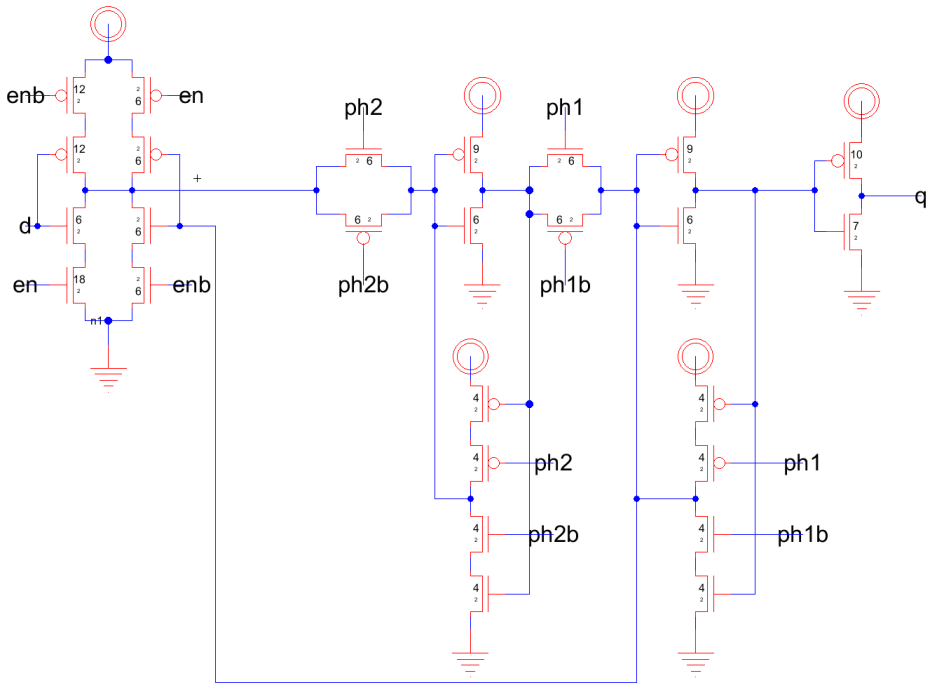

Figure 4.6: Schematic of the master slave D flip-flop.

For designing Shift register we have use D flip-flop for designing the simple 32bit register and use that register and provide some logic elements using basic gates for shifting operation and also provide some extra signals to control or selecting the Shift register. Figure 4.2 shows the cross section of layout of the shift

register.

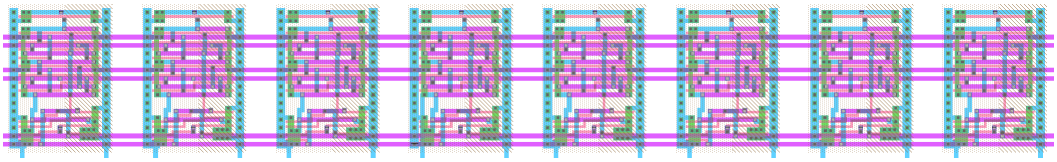

Figure 4.7: Cross section of the register.

SRAM Design

We have used SRAM as we need a small amount of memory. Also using SRAM gives a faster access to data. It is treated as a cache memory also. Conventional SRAMs have less reliability while holding the data in presence of noise, since 12T SRAM provides this as an advantage over the conventional SRAMs.[15] [16] We thought of using 12T SRAM, as our application is used for prediction and needs data that is error free i.e., less affected by noise. Though this would increase area requirement and a bit of power consumption. Figure 4.8 shows the schematic of the 12T SRAM.

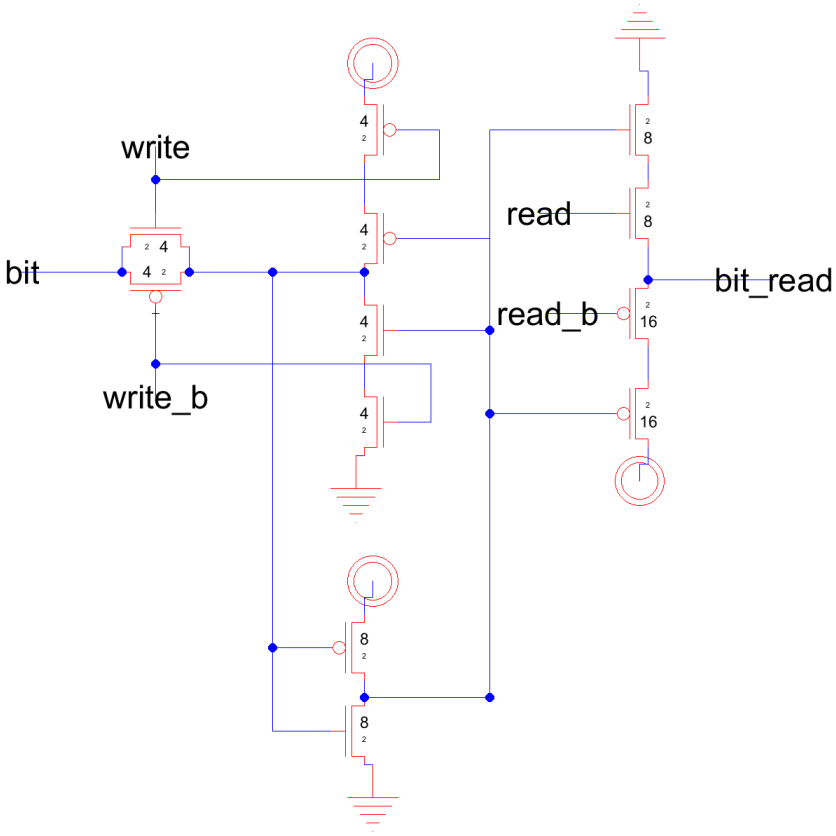

Figure 4.8: Schematic of the 12T SRAM.

We have used 2 SRAMs of size 64×4 bits each designed using 12T SRAM cell. SRAM1 is used for weight and bias values and SRAM2 is used to store input data and output data generated by the sigmoid function. The layout of SRAM is shown in Figure 4.9 where the cross section of the SRAM layout is shown in Figure 4.10.

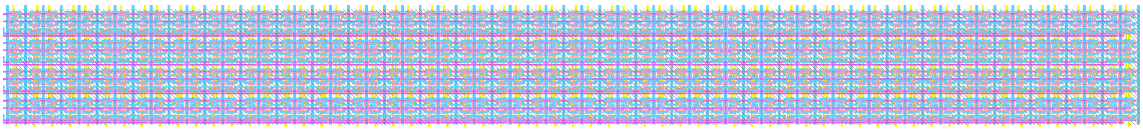

Figure 4.9: Layout of the SRAM.

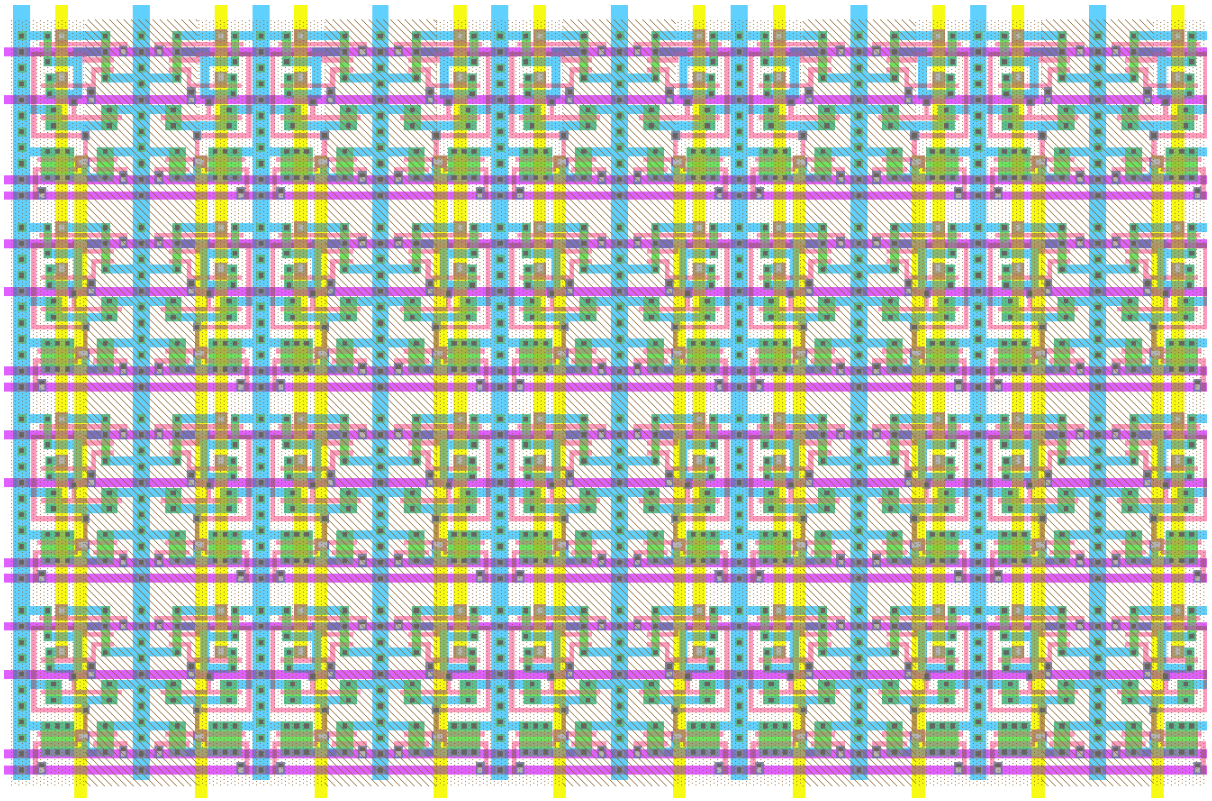

Figure 4.10: Cross section of the SRAM layout.

4.5 Final Chip

The Neural network chip is having 72 pins in total where 32 pins are used for input and output, there are 2 pins are for two different clocks, since we have use 2 SRAM for storing data of weight, bias, input, and output data we need read, write, enable, select, etc. The pin description is mentioned in Table 4.1.

PIN no.	DESCRIPTION
1	VCC
2	GND
3-18	Data [0:15]
19-20	Data select [0:1]
21	Select multiplication
22-24	Select adder [0:2]
25	Select subtractor
26-27	Select multiplexer [0:1]
28-30	Select register [0:2]
31	Select Transfer function
32	Select Sigmoid function
33-34	Select Sigmoid stage [0:1]
35-50	Data [16:31]
51	SRAM1 write
52	SRAM1 read
53	SRAM1 clock
54	SRAM1 enable
55-57	Select1 [0:2]
58-61	Address1 [0:3]
62-65	Address2 [0:3]
66-68	Select2 [0:2]
69	SRAM2 enable
70	SRAM2 clock
71	SRAM2 read
72	SRAM2 write

Table 4.1: Possible input combination for given FSM.

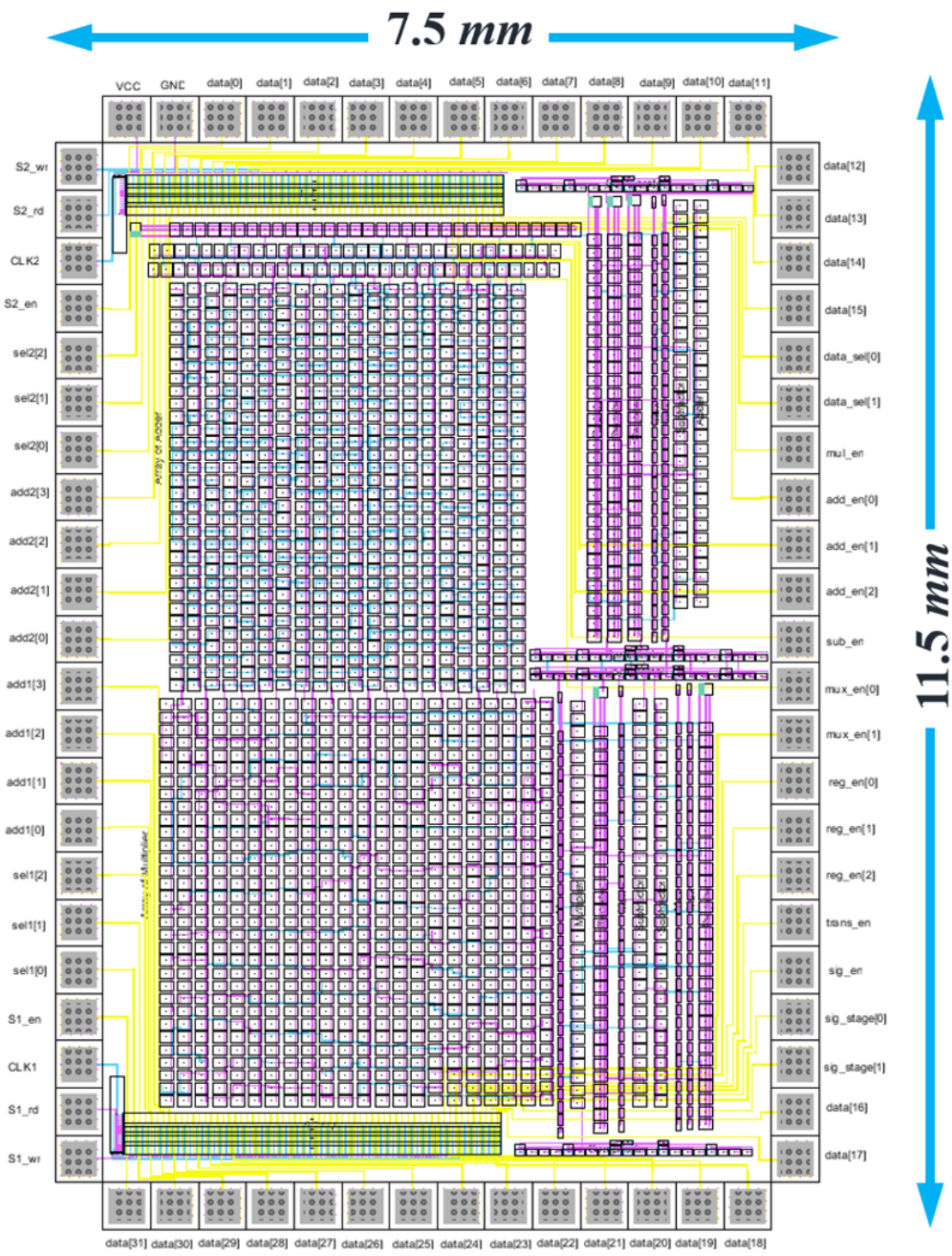

Figure 4.11: Data flow for the FSM with different set of input combination.

4.5.1 Area

Area of the final chip is $7.5mm * 11.5mm$ its around $86.25mm^2$ and all other modules area are mention in the Table 4.2.

Layout	Area (mm^2)
Adder	0.566
Subtractor	0.592
Multiplier	0.692
Register	0.656
Comparator	0.355
SRAM	3.448
Array of multiplier	16.543
Array of adder	14.848
Transfer function	40.553
Sigmoid function	25.264

Table 4.2: Speed analysis.

CHAPTER 5
CONCLUSION AND FUTURE RESEARCH

5.1 Summary and Conclusions

To summarize the whole work, we first started with the research and reviewed many papers and articles about the research going on in the field of neural-network and found that there is less research done on reducing the processing time. This gave an ideology to us that leads to use of hardware instead of software for AI which is the base of neural network. First, we tried to analyze the existing projects done on neural network for walking robot (section II) describes the same. We used Python to implement the training part and succeeded in obtaining the desired results. Now for testing part we implemented the hardware level coding on Xilinx ISE/Quartus Prime. Once the hardware level coding was done, we started working on layout in open-source tool Electric VLSI. Here we first started with standard cell layouts and their simulations. Then we implemented large modules using those standard cells, and finally we designed layout for the whole Network. At the end we used that layout and made a chip out of it, which was the final target of our thesis.

In this work, we have designed the layout using 180nm technology, as our priority for now is to implement the neural network on VLSI accelerator. Once we start getting the desired outputs in 180nm technology, we will scale it down. As mentioned above we need to implement dynamic coding and some addition of functionalities, seeing this there would be some changes in the hardware also which would lead to layout design changes. So, it would be more practical to stick to 180nm until we have incorporated all the changes and reached to desired outputs.

5.2 Future Research

We can add more motion for better functionality and also we can improve each module sigmoid function, transfer function and FSM to generate faster and accurate output. To improve sigmoid function we can use faster multiplier such as Modified Booth-Wallace Tree multiplier which has lowest critical delay but the complexity is increased and also it is difficult to implement.

Same thing we can do for transfer function we can use more efficient multiplier and adder so the run time can be reduce also we can reduce the number of stages in transfer function so that we can generate output more faster. To improve FSM we can add more number of motions and use different scenarios such as if robot is stuck at any location, if robot falls. Also, we can identify the type of obstacle in front of the robot.

Publications from This Thesis

1. **Jimmy Patel**, Harsh Advani, Tapas Kumar Maiti, "VLSI Implementation of Neural Network Based Emergent Behavior Model for Robot Control" **Submitted** in International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics May, 2022

2. Harsh Advani, **Jimmy Patel**, Tapas Kumar Maiti, "Hardware-Efficient *Q*-Learning Accelerator for Robot Path Planning", **Published** in Springer, 5th International Symposium on Devices, Circuits and Systems, March, 2022.

3. **Jimmy Patel**, Harsh Advani, Tapas Kumar Maiti, "VLSI Implementation of Neural Network Driven Augmented FSM", **In progress**

4. Harsh Advani, **Jimmy Patel**, Tapas Kumar Maiti, "*Q*-Learning Accelerator Chip Design for Robot Path Planning", **In progress**

References

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th US ed., Pearson, Apr. 2020.

[2] R. R. Murphy and R. C. Arkin, Introduction to AI Robotics, 1st ed., MIT Press, Jan. 2001.

[3] R. A. Brooks, Cambrian Intelligence-The Early History of the New AI. MIT Press, Cambridge, MA, 1999.

[4] Groover, Mikell (2014). Fundamentals of Modern Manufacturing: Materials, Processes, and Systems.

[5] Lyshevski, S.E. Electromechanical Systems and Devices 1st Edition. CRC Press, 2008. ISBN 1420069721.

[6] Lamb, Frank. Industrial Automation: Hands On (English Edition). NC, McGraw-Hill Education, 2013. ISBN 978-0071816458

[7] Mitchell, Tom (1997). Machine Learning. New York: McGraw Hill. ISBN 0-07-042807-7. OCLC 36417892.

[8] Hu, J.; Niu, H.; Carrasco, J.; Lennox, B.; Arvin, F., "Voronoi-Based Multi-Robot Autonomous Exploration in Unknown Environments via Deep Reinforcement Learning" IEEE Transactions on Vehicular Technology, 2020.

[9] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey (2015). "Deep Learning". Nature. 521 (7553): 436–444. Bibcode:2015Natur.521..436L. doi:10.1038/nature14539. PMID 26017442. S2CID 3074096.

[10] arXiv:1202.2745. doi:10.1109/cvpr.2012.6248110. ISBN 978-1-4673-1228-8. S2CID 2161592.

[11] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffry (2012). "ImageNet Classification with Deep Convolutional Neural Networks" (PDF). NIPS 2012: Neu-

ral Information Processing Systems, Lake Tahoe, Nevada. Archived (PDF) from the original on 2017-01-10. Retrieved 2017-05-24.

[12] A. Bermak and A. Bouzerdoum, “VLSI Implementation of a Neural Network Classifier Based on the Saturating Linear Activation Function”, Proceedings of the 9th International Conference on Neural Information Processing (ICONIP), pp. 1-4, Nov. 2002, Singapore

[13] H. M. EI-Bakry and N. Mastorakis “A Simple Design and Implementation of Reconfigurable Neural Networks”, International Joint Conference on Neural Networks (IJCNN), Jun 2009, Atlanta, GA, USA.

[14] J. Sun and H. Jiao, "A 12T Low-Power Standard-Cell Based SRAM Circuit for Ultra-Low-Voltage Operations," IEEE International Conference on IC Design and Technology (ICICDT), pp. 1-4, August 2019.

[15] J. Jiang, D. Lin, J. Xiao and S. Zou, "A Novel Highly Reliable 12T SRAM Bitcell Design," IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC), pp. 1-2, July 2019.

[16] R. Navajothi and A. K. Rahuman, "Implementation of high performance 12T SRAM cell," IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE), pp. 1-6, December 2017.

[17] McCulloch, Warren; Pitts, Walter (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115–133. doi:10.1007/BF02478259.

[18] Copeland, B. Jack, ed. (2004). The Essential Turing. Oxford University Press. p. 403. ISBN 978-0-19-825080-7.

[19] Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities". Proc. Natl. Acad. Sci. U.S.A. 79 (8): 2554–2558. Bibcode:1982PNAS...79.2554H. doi:10.1073/pnas.79.8.2554. PMC 346238. PMID 6953413.

[20] "Neural Net or Neural Network - Gartner IT Glossary". www.gartner.com.

[21] Kohavi, Ron (2001-03-03). "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection"

[22] Brownlee, Jason (2017-07-13). "What is the Difference Between Test and Validation Datasets?". Retrieved 2017-10-12.

[23] Ripley, B.D. (1996) Pattern Recognition and Neural Networks, Cambridge: Cambridge University Press, p. 354

[24] "Subject: What are the population, sample, training set, design set, validation set, and test set?", Neural Network FAQ, part 1 of 7: Introduction (txt), comp.ai.neural-nets, Sarle, W.S., ed. (1997, last modified 2002-05-17)

[25] Larose, D. T.; Larose, C. D. (2014). Discovering knowledge in data : an introduction to data mining. Hoboken: Wiley. doi:10.1002/9781118874059. ISBN 978-0-470-90874-7. OCLC 869460667.

[26] Xu, Yun; Goodacre, Royston (2018). "On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning". Journal of Analysis and Testing. Springer Science and Business Media LLC. 2 (3): 249–262. doi:10.1007/s41664-018-0068-2. ISSN 2096-241X.

[27] Abbod, Maysam F (2007). "Application of Artificial Intelligence to the Management of Urological Cancer". The Journal of Urology. 178 (4): 1150–1156. doi:10.1016/j.juro.2007.05.122. PMID 17698099.

[28] DAWSON, CHRISTIAN W (1998). "An artificial neural network approach to rainfall-runoff modelling". Hydrological Sciences Journal. 43 (1): 47–66. doi:10.1080/02626669809492102.

[29] "The Machine Learning Dictionary". www.cse.unsw.edu.au. Archived from the original on 26 August 2018. Retrieved 4 November 2009.

[30] Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). "Flexible, High Performance Convolutional Neural Networks for Image Classification". Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence-Volume Volume Two. 2: 1237–1242. Retrieved 17 November 2013.

[31] LeCun et al., "Backpropagation Applied to Handwritten Zip Code Recognition," Neural Computation, 1, pp. 541–551, 1989.

[32] Yann LeCun (2016). Slides on Deep Learning Online

[33] Hochreiter, Sepp; Schmidhuber, Jürgen (1 November 1997). "Long Short-Term Memory". Neural Computation. 9 (8): 1735–1780. doi:10.1162/neco.1997.9.8.1735. ISSN 0899-7667. PMID 9377276. S2CID 1915014.

[34] Sak, Hasim; Senior, Andrew; Beaufays, Francoise (2014). "Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling" (PDF). Archived from the original (PDF) on 24 April 2018.

[35] Li, Xiangang; Wu, Xihong (15 October 2014). "Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition". arXiv:1410.4281 [cs.CL].

[36] Fan, Y.; Qian, Y.; Xie, F.; Soong, F. K. (2014). "TTS synthesis with bidirectional LSTM based Recurrent Neural Networks". Proceedings of the Annual Conference of the International Speech Communication Association, Interspeech: 1964–1968. Retrieved 13 June 2017.

[37] Zen, Heiga; Sak, Hasim (2015). "Unidirectional Long Short-Term Memory Recurrent Neural Network with Recurrent Output Layer for Low-Latency Speech Synthesis" (PDF). Google.com. ICASSP. pp. 4470–4474.

[38] Fan, Bo; Wang, Lijuan; Soong, Frank K.; Xie, Lei (2015). "Photo-Real Talking Head with Deep Bidirectional LSTM" (PDF). Proceedings of ICASSP.

[39] Silver, David; Hubert, Thomas; Schrittwieser, Julian; Antonoglou, Ioannis; Lai, Matthew; Guez, Arthur; Lanctot, Marc; Sifre, Laurent; Kumaran, Dharshan; Graepel, Thore; Lillicrap, Timothy; Simonyan, Karen; Hassabis, Demis (5 December 2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". arXiv:1712.01815 [cs.AI].

[40] Goodfellow, Ian; Pouget-Abadie, Jean; Mirza, Mehdi; Xu, Bing; Warde-Farley, David; Ozair, Sherjil; Courville, Aaron; Bengio, Yoshua (2014). Generative Adversarial Networks (PDF). Proceedings of the International Conference on Neural Information Processing Systems (NIPS 2014). pp. 2672–2680.

[41] Gustineli, Murilo (2022-04-06). "A survey on recently proposed activation functions for Deep Learning". arXiv:2204.02921 [cs]. doi:10.48550/arxiv.2204.02921.

[42] Specification of KHR-3HV Ver.2 Robot, Dec. 2018, [online] Available: https://kondo-robot.com/product/03178.

[43] Rifkin, Jeremy (1995). The End of Work: The Decline of the Global Labor Force and the Dawn of the Post-Market Era. Putnam Publishing Group. pp. 66, 75. ISBN 978-0-87477-779-6.

[44] M. F. E. Rohmer and S. P. N. Singh, “V-rep: a versatile and scalable robot simulation framework”, Proceeding of the International Conference on Intelligent Robots and Systems (IROS), pp.1321-1326, Nov. 2013, Tokyo, Japan.

[45] C.-W. Lin and J.-S. Wang, “A Digital Circuit Design of Hyperbolic Tangent Sigmoid Function for Neural Networks”, IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-4, 2008, Seattle, WA, USA.

[46] S. Satyanarayana and H. P. Graf, “A Reconfigurable VLSI Neural Network”, IEEE Journal of Solid-State Circuits, pp. 67-81, vol. 27, Iss. 1, Jan 1992.

[47] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi, “Efficient Hardware Implementation of the Hyperbolic Tangent Sigmoid Function” IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-4, May 2009, Taipei, Taiwan

[48] I. Tsmots, V. Rabyk, and O. Skorokhoda, “Hardware Implementation of Sigmoid Activation Functions using FPGA”, IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), pp. 1-4, Mar. 2019, Polyana, Ukraine.

[49] Ciresan, D.; Meier, U.; Schmidhuber, J. (2012). "Multi-column deep neural networks for image classification". 2012 IEEE Conference on Computer Vision and Pattern Recognition. pp. 3642–3649.

[50] S. Xing and C. Wu, “Implementation of A Neuron Using Sigmoid Activation Function with CMOS”, IEEE 5th International Conference on Integrated Circuits and Microsystems (ICICM), pp. 1-4, Oct. 2020, Nanjing, China.

[51] P. Treleaven, M. Pacheco, and M. Vellasco “VLSI Architectures for Neural Network”, pp. 1-4, May 1989, Portland, OR, USA

[52] A.H. Farrahi, D.J. Hathaway, M. Wang, and M. Sarrafzadeh, Quality of eda cad tools: definitions, metrics and directions, Quality Electronic Design, 2000. ISQED 2000. Proceedings. IEEE 2000 First International Symposium on, 2000, pp. 395–405.

[53] Nanoelectronic Mixed-Signal System Design, no. 9780071825719 and 0071825711, McGraw-Hill Education, 2015.

[54] Energy and Transient Power Minimization during Behavioral Synthesis, Ph.D. thesis, Department of Computer Science and Engineering, University of South Florida, Fall, 2003.

[55] Saraju P Mohanty, N Ranganathan, and Vamsi Krishna, Datapath scheduling using dynamic frequency clocking, VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on, IEEE, 2002, pp. 58–63.

[56] Saraju P Mohanty, Ramakrishna Velagapudi, and Elias Kougianos, Physical-aware simulated annealing optimization of gate leakage in nanoscale datapath circuits, Design, Automation and Test in Europe, 2006. DATE'06. Proceedings, vol. 1, IEEE, 2006, pp. 6–pp.

[57] Steven M. Rubin, Using the electric vlsi design system, R.L. Ranch Press, 2015, Oracle and Static Free Software , ISBN 0972751432.

[58] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng, Electronic design automation: synthesis, verification, and test, Morgan Kaufmann, 2009.

CHAPTER A
Standard Cell Layout and Simulation

A.1 Layout of Standard Cell

Here are the some of the basic standard cell layout design which we have use to design our final chip.

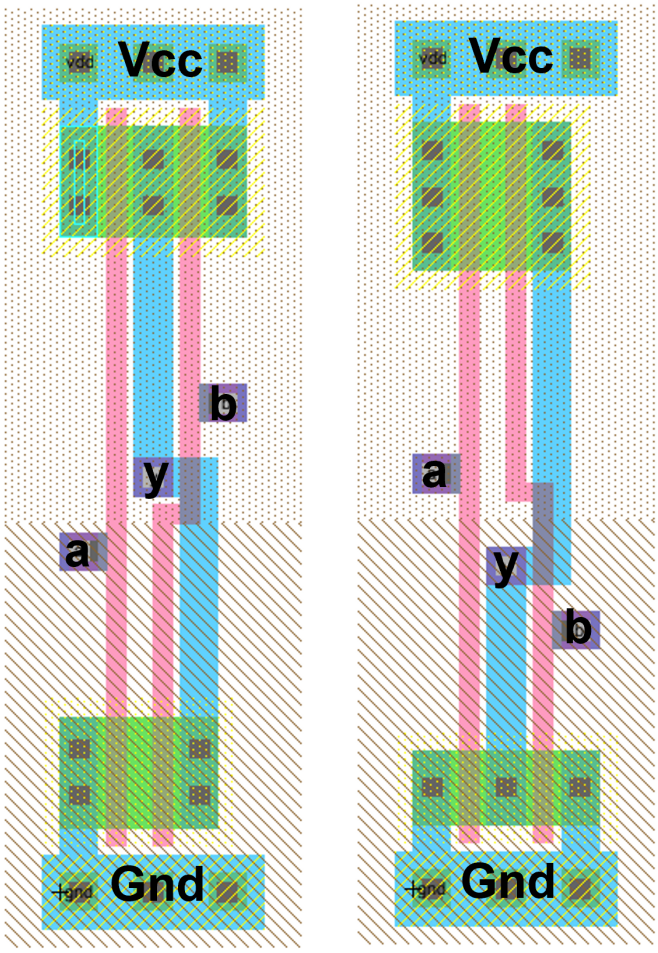

Figure A.1: (a) NAND gate layout, (b) NOR gate layout.

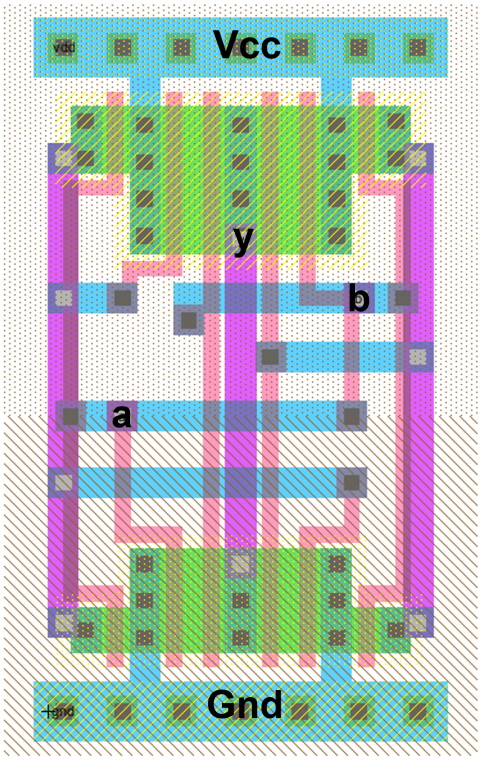

Figure A.2: XOR gate layout.

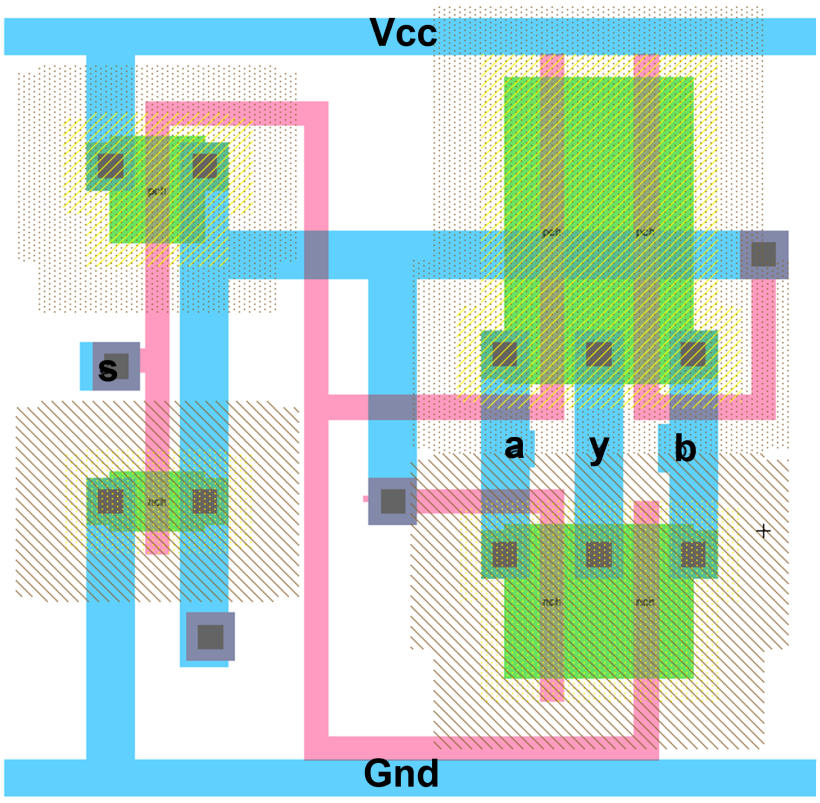

Figure A.3: 2x1 Multiplexer layout.

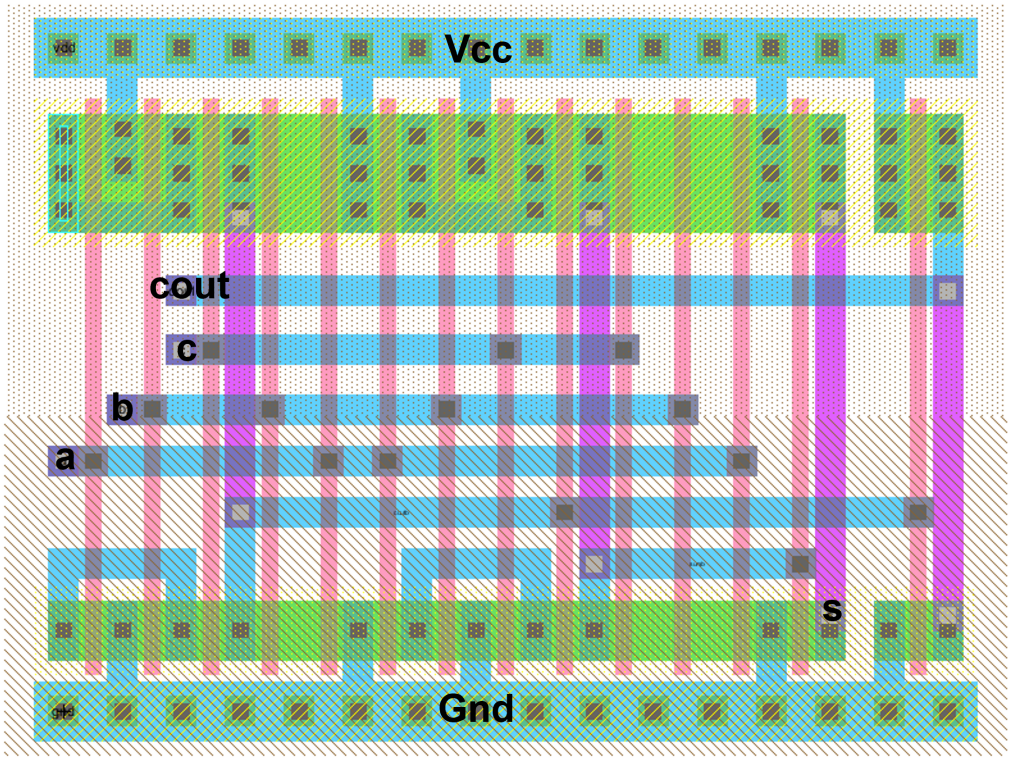

Figure A.4: 1bit Full-Adder layout.

A.2 Simulation

Here are the simulated wave forms of all the standard cell which we have discussed in section A.1.

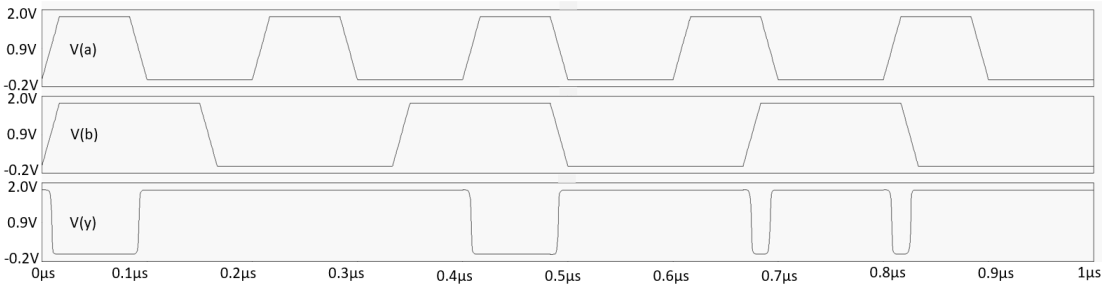

Figure A.5: NAND gate waveform.

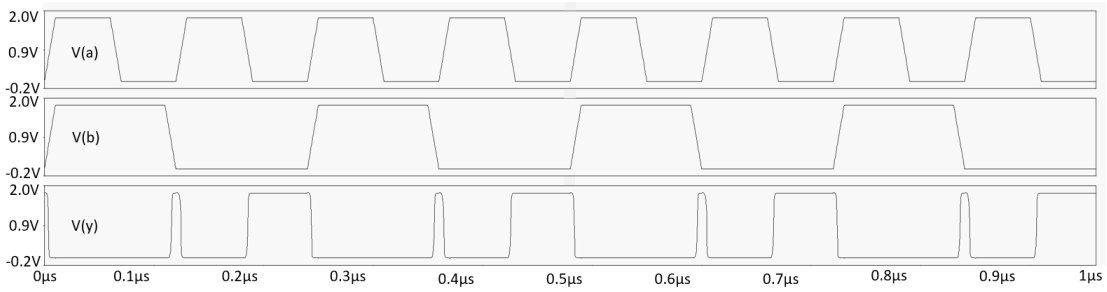

Figure A.6: NOR gate waveform.

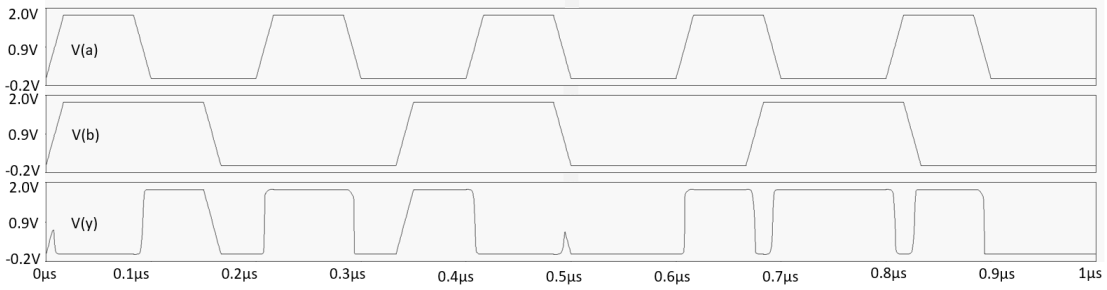

Figure A.7: XOR gate waveform.

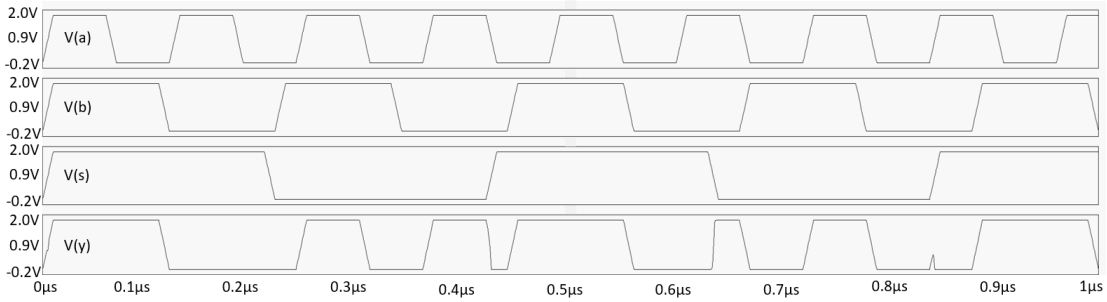

Figure A.8: 2x1 Multiplexer waveform.

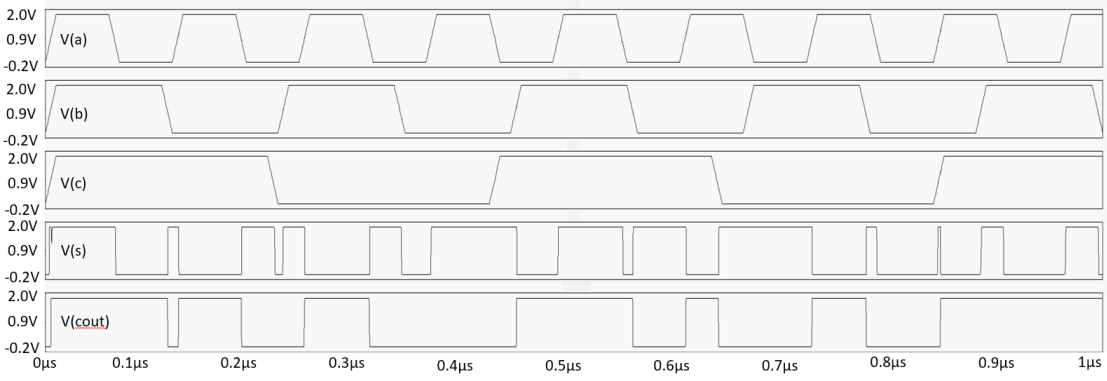

Figure A.9: 1-bit Full-Adder waveform.