The Pennsylvania State University

The Graduate School

College of Engineering

A LOW POWER AND AREA EFFICIENT CMOS IMPLEMENTATION OF MULTILAYER FEEDFORWARD ARTIFICIAL NEURAL NETWORK

A Thesis in

Electrical Engineering

by

Mayuresh P Patki

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2017

ABSTRACT

With several advancements in medical science being carried out over the past few decades, there has been a constant need to process information artificially, the way it is processed inside the human body. This inherent attribute of Artificial Intelligence (AI) is achieved in practice using Artificial Neural Networks (ANNs). ANNs have been around since 1943 and used since then for artificial information processing and neural computation.

This thesis focuses on the hardware implementation of an artificial neural network using CMOS technology. The design is carried out in the analog domain to exploit certain advantages of analog integrated circuit design, such as, high efficiency, in terms of area and power, and ease of computation. The neural architecture designed is a multilayer feedforward neural network to solve the XOR classification problem, which serves as a benchmark for several complex classification problems that are not linearly separable. Each component circuit of the network, such as the synapse circuit that performs the multiplication operation and the non-linear activation function circuit that acts as squashing function, is designed using MOSFETs operating in the sub-threshold (weak inversion) region.

The schematic designs are carried out using Cadence OrCAD Capture version 16.6 EDA software and simulated using PSPICE version 16.6, an in-built simulation tool within OrCAD capture. The layout of the individual components and the overall schematic is also done using Electric VLSI Design software version 9.06 on a 200 nm design scale. A consistency check is carried out to ensure equivalency of layout with the schematic, for a potential scope towards chip fabrication using Metal Oxide Semiconductor Implementation Service (MOSIS) foundry. The layout of the proposed neural architecture is found to occupy an area of 0.065 mm^2, indicating design compactness to a moderate level.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Seth Wolpert, my thesis advisor, for his continued support and guidance throughout the learning process of this thesis. His constant encouragement and timely mentoring helped me in conducting research and writing the thesis. Dr. Wolpert was available, whenever I required his advice in moving forward with the research work and helped me clear any doubts I had, during the process.

I would also like to thank my thesis committee members, Dr. Scott von Tonningen and Dr. Wolfram Bettermann for their continued encouragement, insightful comments and timely feedback, while reviewing my research work. Their valuable suggestions helped me a long way in improving the content of my thesis, making it more informative.

Chapter 1

Introduction

1.1 Research Motivation

Much time and effort is being given to the research and implementation of artificial neural networks for reliable information processing and generating accurate results. The primary difficulty faced by designers in typical neural network implementations is striking a good balance between silicon area overhead, power consumption and speed of operation, before going into the actual fabrication of a neural network chip. The main issue while implementing any neural architecture is that the design can either be area efficient or power efficient or speed efficient, but not all three at the same time [1]. The operating speed is decided by the number of components on a chip and is not much of a problem for designs having moderate level of complexity. However, area occupancy and power consumption are the major deciding factors that need to be taken care of in any Integrated Circuit (IC) design, irrespective of the associated complexity level.

Analog implementations of artificial neural networks are known to have higher efficiency in terms of power consumption and with some careful design techniques, the implementations can be made area efficient as well. The motivation for the research done in this thesis stems from the requirement to have a multilayer neural network implementation that is area efficient and has low power dissipation, at both, the individual component level and the overall design level.

1.2 The need for VLSI Implementation

Almost all integrated circuits in use today are fabricated using Very Large Scale Integration (VLSI) technology. VLSI is presently the technology of choice owing to several advantages it offers, such as, low cost, high speed implementation and minimization in terms of chip area and power consumption. In almost all applications today, there is a constant requirement to have multi-objective optimization that implies achieving the desired functionality with a minimum hardware resource utilization [2]. Implementation of neural networks in hardware using VLSI technology faces a challenge, given that the design constraints for biological systems implemented in silicon media are not the same as those in biological medium. So, it becomes necessary that the overall architectural framework be researched upon before forging ahead and designing electronic structures for individual neural network components. Moreover, it is both practical and feasible to have silicon media impose restrictions on the learning capabilities of neural networks for having more effective computations.

Traditional neural network systems have separate resource allocations for memory and processors used for computing information. VLSI implementation allows one to have a local memory for storing the operating data, resulting in a more compact design. This phenomenon is also known as ‘Computation in Memory’ [3]. Parallel computation using neural networks has received much attention since past several years, owing to the multiple benefits it has to offer, like simultaneous weight adaptions and more tolerance in terms of individual component failure. Another cited benefit is the algorithmic speed up obtained through weight vector multiplication operations performed in parallel [3]. VLSI implementation takes advantage of this inherent parallelism and leads to a more efficient design.

1.3 Analog versus Digital

Considerable amount of research is being carried out in neural network implementations using analog and digital array processors on the same chip in order to exploit the benefits of parallel computing. There are, however, certain advantages and disadvantages involved with such implementations that need to be taken into account, before deciding on the technology to go for. The choice is based mainly on the application where the neural network will potentially be employed. For high end applications where a lot of hardware designing is involved, programmability and flexibility of the implemented design are of a major concern. In such cases, digital implementation is preferred, since it provides good amount of flexibility when it comes to making minor changes in the existing design. Such changes can be easily accomplished through software modifications [3]. However, there are certain problems encountered with digital VLSI implementation of neural networks in applications involving moderate designs. Digital weight storage involves the use of buses that complicate the existing hardware. Multiplication and summation operations in digital domain require several transistors on board. Moreover, the realization of a non-linear activation function is difficult using digital logic. Finally, when it comes to interfacing digital logic with existing analog circuitry, Digital to Analog Converters (DACs) are required. This requires extra hardware that could otherwise be devoted to computation. So, for moderate designs, analog implementation is by far, the most preferred choice.

Analog circuits are more compact in terms of chip area and circuit complexity. Analog multiplication, summation and non-linear function implementation can be realized using only a handful of transistors [3]. Since analog outputs are in terms of either voltage or current, buses are not required for data storage. With least amount of area, analog circuits perform efficient computations at minimal power consumption and higher speed of operation. In the proposed design, we go for an ‘All Analog’ implementation with a view to achieve optimization in terms of area and power, and also since the design is of a moderate scaling.

1.4 Thesis Structure

The thesis structure has been divided into the following chapters. Chapter 2 outlines the literature review done in the field of neural computation. Chapter 3 provides a brief comparison of how neural networks function, both in biological and artificial domain and details the various electronic circuits that are required for hardware implementation of neural networks. Chapter 4 details the different mechanisms for synaptic weight storage and the commonly used algorithms for neural network training. Chapter 5 shows the implementation of the proposed design in a step by step manner. Chapter 6 demonstrates the results obtained on simulating the design using various software packages, for individual functionality, power dissipation and solution to the XOR classification problem. Finally, chapter 7 provides a general conclusion on the research carried out in the thesis with a briefing on the future research work.

Chapter 2

Literature Review

2.1 Background

Significant advancements in the field of medical science during the last several decades have led to a constant need to have certain alternative solutions be implemented for computing and processing biological information. Biological systems excel far more in cognitive abilities than most of the supercomputers available today. However, alternate methodologies still turn out to be the need of the hour to have some sort of redundancy, when it comes to real time information processing. The requirement is to have quick and efficient solution(s) that should be able to generate correct results. Reliability, counted only on the effectiveness of medical instruments can turn out to be catastrophic, if multiple device failures and/or instrumental errors are involved. It can also lead to increased costs when it comes to rectifying those failures, let alone the time spent in doing so.

Since past several decades, researchers and engineers from all over the globe have come up with several novel ideas to implement biological information in an artificial and reliable manner. This led to the development of 'Artificial Intelligence (AI)' in the ever emerging field of neural computation [4]. AI involves computing and processing biological information using artificial methodologies. These methods are in no way, a replacement to existing biological systems and processes, but are meant to make them more versatile. One way to implement this idea is by using electronic hardware structures known as ‘Artificial Neural Networks (ANNs)’. They are basically, information computing and processing systems, wherein multiple electronic hardware blocks come together in unison to process the available data and generate relevant results. In simple terms, they are electronic counterparts of biological systems. Fig. 2.1 shows the structure of a simple feedforward artificial neural network [1].

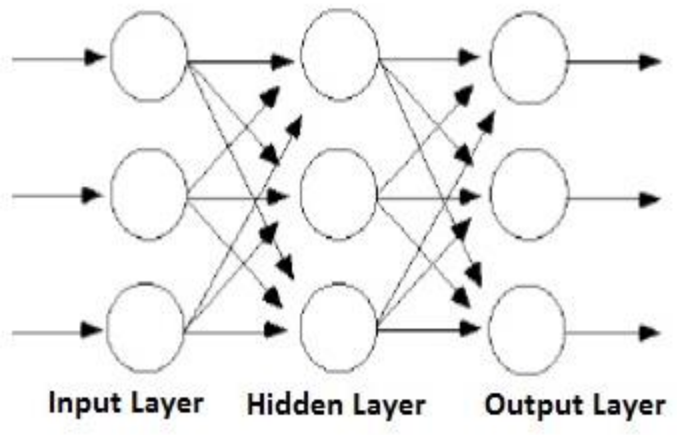


Fig. 2.1 Basic Structure of a Feedforward Neural Network [1]

Fig. 2.1 is an example of a feedforward neural network which is used in a wide variety of neural network applications. The network consists of inputs, synaptic weights (not shown in the figure) that get multiplied with the inputs, hidden layer and outputs that are generated after the weighted inputs are passed through an activation function. The activation function used is non-linear and is generally, the hyperbolic tangent or the sigmoid function. Most of the approaches employ feedforward implementation, since the algorithmic design associated with the feedback implementation is considerably more difficult to implement [1]. While this is a very basic form of a structure, several complex structures with more number of input and output layers as well as hidden layers can be realized in practice [4].

2.2 History of Neural Networks

From a historical perspective, the origin of artificial neural networks can be traced back to the McCulloch and Pitts neuron model developed by Warren McCulloch and Walter Pitts in 1943 [5]. This model formed the basis of neural computing and the evolution of several other neural network models. The neuron model to a basic level, consisted of two inputs and one output as shown in fig 2.2. For the neuron to fire, both the inputs were required to be in their active state. The weights for each input were assumed to be equal and the neuron would fire only when the weighted inputs summed to have a value above a certain pre-defined threshold level. The output of the neuron was considered to be binary; an output of one implied firing of a neuron and output of zero implied the neuron to be in its inactive state [6].

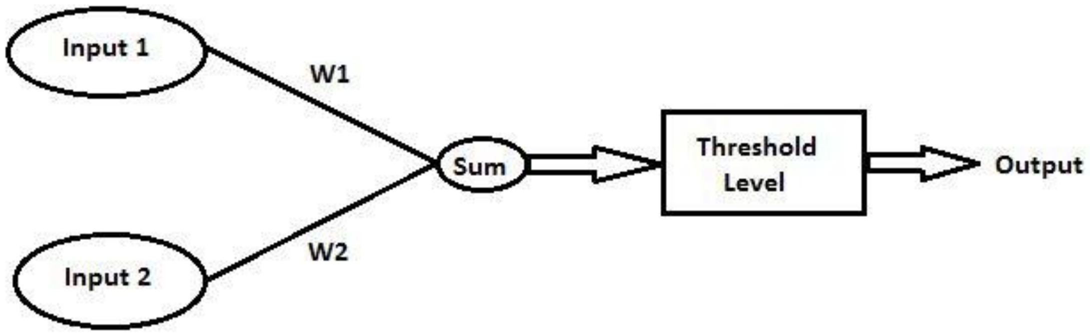


Fig. 2.2 Basic McCulloch and Pitts Neuron Model

In an attempt to model the various electrochemical processes that take place in the human brain, the 'Perceptron' was developed by Rosenblatt in 1958 [7]. The perceptron is a very simple mathematical model of a neuron and generally has multiple inputs and a single output. Rosenblatt tried to randomly interconnect the perceptrons and used random weights through a trial and error procedure to achieve learning. He proved that if there are two classes of data that are linearly separable, the perceptron algorithm converges and positions itself to form a decision surface in the form of a straight line or a plane separating the two classes. Fig. 2.3 shows a single layer perceptron model [7].

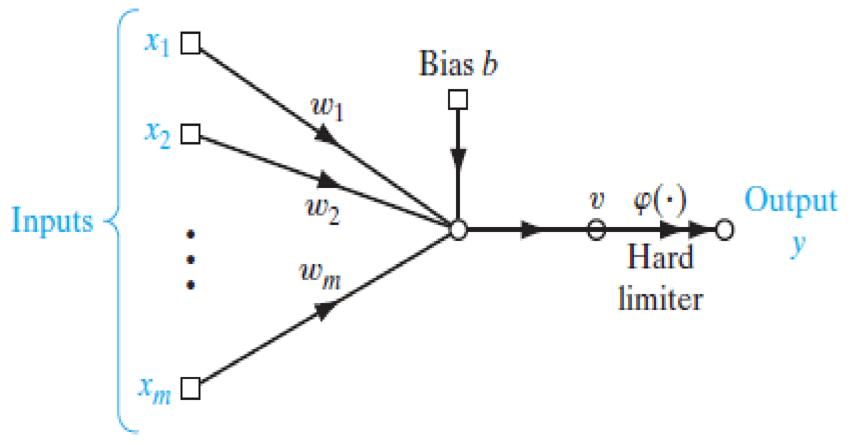


Fig. 2.3 Single Layer Perceptron Model [7]

The weighted sum of n inputs with an optional bias term was passed through a hard limiter (activation function) with a certain threshold level to get the desired output. While Rosenblatt used trial and error to perform weight modifications, Oliver Selfridge in 1958, used a direction vector to make the weight adjustments [8]. If adjusting the weights through a randomly chosen direction vector did not make the performance any better, the weights were returned to their original value and a new direction vector was chosen.

While the methods proposed by both Rosenblatt and Selfridge for network learning were trial and error based, there was no proper mathematical technique available at that time to train the perceptron. Widrow and Hoff in 1960 came up with a mathematical method for updating the weights [9]. Assuming the existence of the desired response, they implemented a gradient search method for minimizing the square of the error function. This method was later known as the Least Mean Squared (LMS) algorithm [9]. The LMS algorithm not only provided a reliable method for network training, but also significantly reduced the overall computation time compared to the trial and error method. Immediately following the invention of the LMS algorithm, perceptrons became widely used in several applications and were known to solve any kind of computational problem. In 1969, Minsky and Papert in a book on perceptrons [10] pointed out, that single layer perceptrons with ‘n’ nodes can only solve linearly separable problems, also known as two-separable problems. The single layer perceptron was inefficient in solving n-separable problems against what was previously believed [10].

In 1974, Werbos came up with a novel idea to solve this issue [11]. The training algorithm he used involved, propagating errors backward into the multilayer perceptron network. The technique was named as the ‘Backpropagation Algorithm’. The weight adaptions were carried out starting from the output layer and working backwards to the input layer [11]. The commonly used neuron activation function was the sigmoid function, against the signum function used with the single layer perceptron. The main advantage of using the sigmoid function instead of the signum function is that, the former is differentiable. The backpropagation algorithm was essentially, a generalization of the LMS algorithm developed by Widrow-Hoff [9] and was later separately rediscovered by Parker in 1985 [12] and Rummelhart and McClelland in 1986 [35]. The algorithm was able to solve non-linearly separable problems by training perceptrons in a multilayer configuration. The year 1986 saw the ushering of modern day neural networks and significantly increased the amount of research conducted in neural computing that is prevalent even today.

Chapter 3

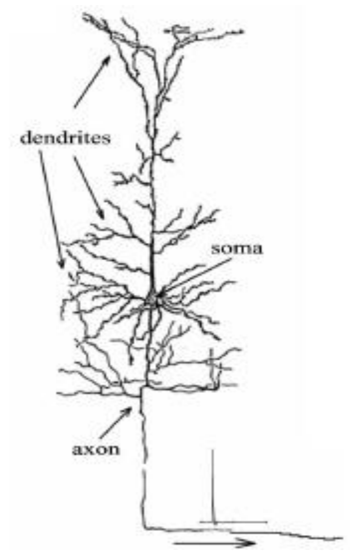
Neural Networks Theory

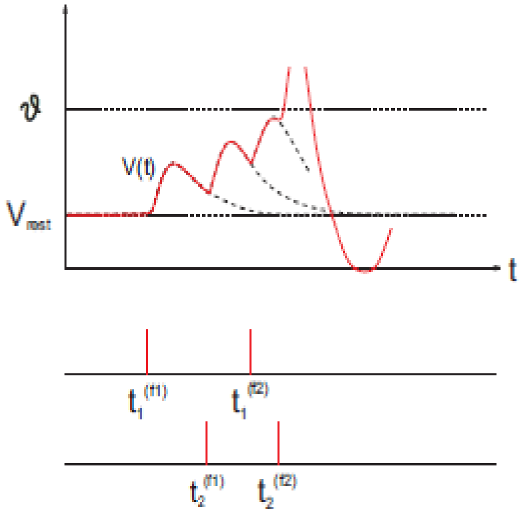
3.1 Neural Computation and Machine Learning

Developing on the concept of artificial intelligence introduced in chapter 1, the requirement is to have neural computation done using artificial electronic circuitry that in a way should generate results analogous to those with the biological systems. In order to understand the analogous nature between the two, it is beneficial to understand the way in which a network of highly interconnected biological nerve cells compute and process information.

3.1.1 Computation in Biological and Artificial domain

Human brain consists of a very dense network of nerve cells, also called ‘neurons’ that occupy an area of about 1.5 square meters owing to their nested architecture [13]. The cerebral cortex of the human brain consists of about 10000 neurons per cubic millimeter that are massively interconnected and receive information from other neurons. This kind of a nested structure gives the human brain its unique and incomparable computational power. Fig 3.1(a) shows the basic structure of a biological nerve cell [13].





(a) (b)

Fig. 3.1. (a) Basic structure of a Biological Nerve Cell [13]
(b) Plot of membrane potential vs time and action potential (spikes) [13]

The body of a nerve cell is called 'soma' that is about 10 to 50 microns in length. The input information to the neuron is given by 'dendrites' connected to the soma. The information output is through 'axon' that branches into several axonal structures, known as 'axonic trees' and couples to the dendrites of other neurons for information transfer. The information transfer takes place through 'synapses' that act as coupling elements for axons and dendrites [13]. The synapses act as information processing elements that take information from the axon of a particular neuron, make the necessary modifications to the input information in terms of its strength and pass the modified information down to the neuron located at the end of the synapse. Every nerve cell has a membrane potential and a membrane capacitance associated with it. If the pre-synaptic input is weighed so as to be stronger than the membrane potential, the accounting synapse is said to be 'excitatory' in nature; if weighed to be weaker than the membrane potential, then it is considered as 'inhibitory' in nature. Neurons communicate with each other through action potentials that are short spikes of about 1 ms in duration, generated in the soma and which travel to the axon as shown in fig 3.1(b) [13]. These are meant to modify the post-synaptic potential of a particular synapse that adds to the membrane potential. The cumulative effect is to have the neuron fire, when a particular threshold voltage is reached. Firing of a neuron implies transfer of information to several other interconnected neurons in a repetitive manner.

Modelling a biological neuron in hardware is accomplished using several electronic circuits each designed to serve a specific purpose [14] [15]. The input information is modelled in terms of voltage signals to form a vector of voltage values. A synapse circuit is used to multiply the input data with stored weight values (again a vector), add them up and provide the result to a circuit that implements the activation function required to generate the output [15]. The synapse circuit together with the circuit implementing the activation function forms the neuron circuit. Details about the design and operation of these circuits will be discussed in section 3.2. The output is generated subject to a certain pre-defined threshold value, in a manner similar to the biological neuron. In some cases, an optional bias term is added to the weighted sum of inputs so as to make the overall input stronger (excitatory) or weaker (inhibitory).

The introductory chapter briefly explained the basic structure of an artificial neural network but did not manifest the various mathematical operations that take place within the network. Fig 3.2 shows a two-layer neural network depicting the mathematical operations that are carried out to process the input information [3].

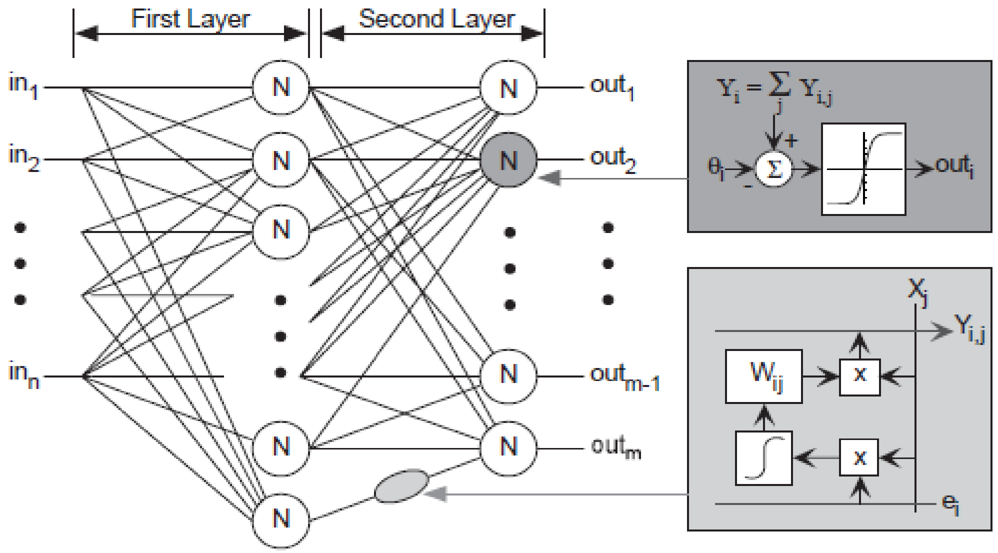


Fig. 3.2 A two-layer artificial neural network depicting input and output layers and various mathematical operations carried out in the network [3]

It can be seen that the inputs from the first layer are connected to every other neuron and those neurons in turn, are connected to several other neurons to form a nested architecture, just as in the case of biological systems. A basic synapse circuit should be able to store the weight values, perform a multiplication of its input with the corresponding weight value and then perform a weight adaption in response to certain learning criteria and fed-back error signal. This computation is expressed mathematically as [3]:

$$y_i = W_{ij}x_j \qquad [2.1]$$

where y_i is the i^{th} output that forms the input to the activation function, x_j is the j^{th} input and W_{ij} is the weight stored at position (i, j). The result of this computation is passed through a non-linear activation function to generate the i^{th} output (actual neuron output) given by [3]:

$$out_i = \tanh(a(y_i - \theta_i)) \qquad [2.2]$$

Where, θ_i is the threshold value and 'a' is the gain factor.

The outputs are generally currents since they can be easily added using Kirchhoff’s Current Law (KCL). Voltages are preferred for inputs since they are easy to propagate through the network. The synapse circuit performs some adaptions on the stored weight values based on the fed-back error signal, that is a result of the difference between the actual output and the desired output. These weight adaptions are governed by certain learning rules that enable the network to learn from its previously computed values. Before going further into various aspects of neural network learning and the training algorithms used to perform weight adaptions, it is worthwhile to concisely discuss an important concept of machine learning in the field of artificial intelligence.

3.1.2 Basics of Machine Learning

Machine Learning is a mechanism through which a computer learns to perform a specific set of actions depending on a pre-determined set of inputs and previously computed results based on those inputs, without the need for explicit programming. It is a process of iteratively learning from data using certain learning algorithms [16]. Artificial neural networks take advantage of this process to make synaptic weight modifications, that further enable the network to perform efficient computations using the least amount of time. This essentially leads to feedback computation in neural networks. While feedforward computation evaluates the output using equations [2.1] and [2.2] above, feedback computation computes and propagates the error back into the system, achieved as a result of the difference between the actual and the desired output. The task is then to minimize the error down to zero. Two of the commonly used machine learning algorithms for doing so are supervised learning and unsupervised learning, discussed next.

3.1.2.1 Supervised Learning

In supervised learning, the system knows the correct output(s) for a specific set of input data [17]. The learning algorithm is given a set of correct outputs for a corresponding set of inputs and the algorithm is expected to make the necessary adaptions, by comparing the actual output with the desired output to minimize the resulting error. Linear regression and Classification are some techniques that are based on supervised learning. Linear regression, in its simplest form, involves predicting a specific value based on a pre-defined set of data values relating the input and output parameters [17].

Regression is also used to estimate the relationship between a dependent variable and one or more independent variables [17]. A few examples employing linear regression technique are to predict the property prices from some database that relates the cost with property area; estimating the likelihood of insurance claim filing by a person, based on the claims filed earlier by the same person. While regression is used primarily for estimation, a typical classification problem involves classifying a set of data points into two or more categories. One example of a neural classification problem is to classify a cancerous tumor as either benign or malignant depending on the previously generated data set, after testing a few patients having either of the two forms of cancerous tumor [17] [18].

3.1.2.2 Unsupervised Learning

Contrary to supervised learning, in unsupervised learning, the system does not know the correct output(s) for the specified input parameters [18]. The responsibility of estimating the relation between input and output values is given to the learning algorithm. The learning algorithm tries to figure out some common structure or a pattern by exploring the available database, that may give a fairly accurate estimation of the relation between the input and output parameters. A common example where unsupervised leaning is used is in cluster analysis, that finds the hidden attributes in a cluster or a collection of data points [18]. Other examples of unsupervised learning are in understanding genomics of an individual, based on a DNA micro-grade data that decides whether a particular group of individuals have a specific gene or not [18]. The algorithm is also used in social networking sites such as Facebook, Twitter to categorize individuals, based on a clustering algorithm and some analysis of their activities on these sites [18].

3.2 Electronic Circuits for Neural Networks

Implementation of an artificial neural network in hardware is accomplished by designing individual electronic circuits, each designed to serve a specific purpose. The three basic operations required are multiplication, summation and a non-linear activation function acting on the summed up values. The circuits for providing these operations are designed, keeping in mind the optimization in terms of area, power and operating speed. The three basic circuits used are the synapse circuit, circuit for implementing the activation function and the neuron circuit that are discussed in the next few sub-sections.

3.2.1 Synapse Circuit

The synapse is an important element in artificial neural networks [19]. The basic role of a synapse element is to multiply the input signals with stored weight values and later adapt those values through some learning algorithm. Owing to the criticality of the synapse structure in neural network implementations, it is viable at this point to state five basic synapse properties as mentioned below [19]:

1. The synapse should be able to store the weight values permanently in absence of any learning algorithm.

2. The synapse should be able to compute an output current in response to the multiplication of input voltage with the stored weight value.

3. The synapse should be able to adapt the stored weight values when a certain learning algorithm is applied to train the network.

4. The synapse circuit should consume minimum area on chip so as to maximize the number of synapse elements in a given area.

5. The synapse circuit should dissipate minimum amount of power.

Ideal scenario is to have efficient synapse circuit implementations satisfying all of the above criteria. In practice, it requires rigorous efforts and detailed discussions for meeting most of the conditions listed above. One of the major bottlenecks in synapse circuit implementation is in performing the multiplication operation of input voltage signals with stored weight values and producing an output current signifying the result of the operation. Since input voltages transform into an output current, a multiplier circuit serving as a transconductance element, needs to be implemented. One such circuit is the Gilbert Multiplier Cell shown in fig. 3.3 [20].

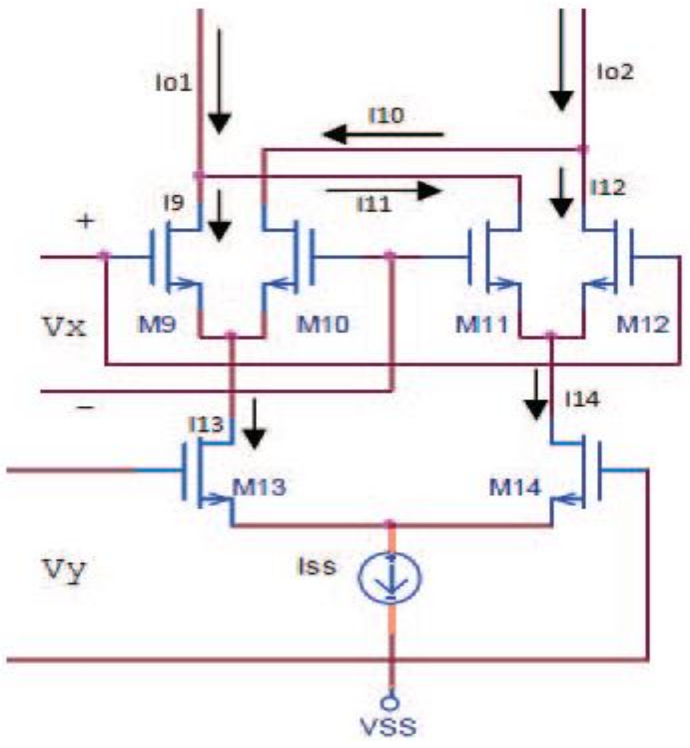


Fig. 3.3 CMOS Gilbert Multiplier Cell [20]

The Gilbert Multiplier Cell is used to convert the product of two voltages applied at the input into a proportional output current. Synapses can be implemented using this multiplier cell to generate an output current in response to the multiplication of input voltages with the corresponding weight values. For design simplicity, all transistors are assumed to be biased in the saturation region and the device sizes and parameters are matched, so that the transconductance values for all of them are equal. After some mathematical operations shown in [20] [21], the expression for the output current is:

$$I_o = \sqrt{2K_aK_b}V_xV_y \qquad [2.3]$$

In order to maintain a good amount of linearity between output current and input voltages, both V_x and V_y should be kept small.

3.2.2 Activation Function Circuit

In most applications, artificial neural networks use a non-linear activation function to act on their weighted sum of input patterns and produce a corresponding output pattern. The activation functions generally used are the hyperbolic tangent or sigmoid functions, implemented using CMOS circuits operating in the subthreshold region. The reason behind using a non-linear activation function in neural network implementation is to take advantage of the inherent non-linearities of CMOS circuits [22]. These non-linearities are governed by a non-linear relation between currents and voltages, primarily in the saturation region of MOSFETs. Fig. 3.4 shows the plot of a typical hyperbolic tangent activation function with three operating regions clearly defined [23].

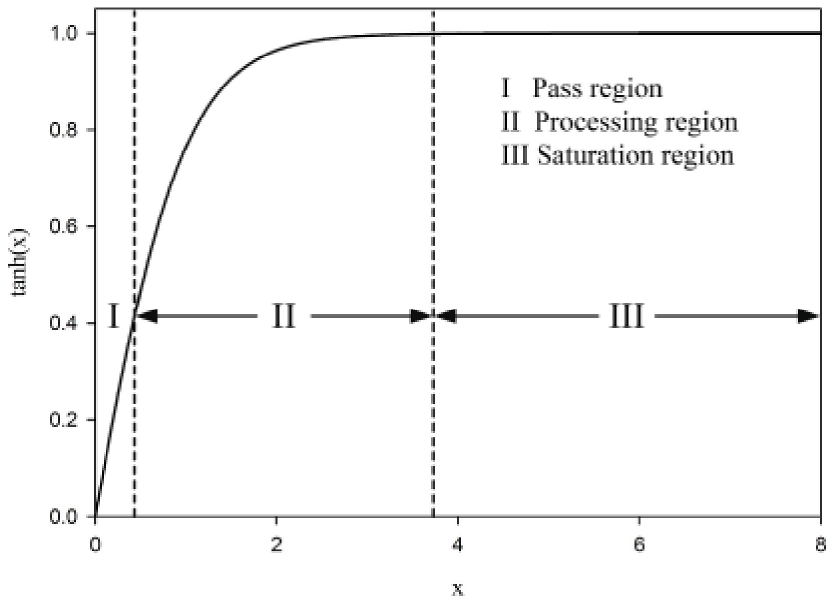


Fig. 3.4 Hyperbolic Tangent Activation Function with Regions of Operation [23]

In the pass region, the input is approximately equal to the output, hence can be approximated by a straight line from the origin. In region 3 which is the saturation region, there is less output variation in response to the input, so the linear relation no longer holds. The processing region, labelled as region 2, acts as a transition region between the pass and the saturation region. In this thesis, the proposed design of the activation function circuit uses the CMOS differential amplifier stage [21]. A differential amplifier is used to amplify or accentuate the difference between two input voltage signals and reject the average or common mode values of those signals. Fig. 3.5 shows the differential amplifier design for implementing the activation function [21].

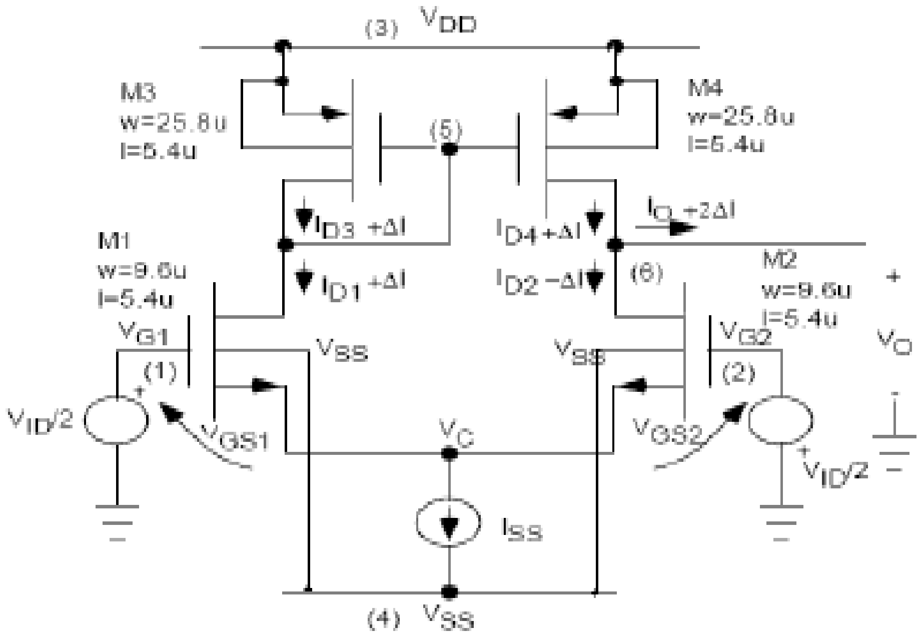


Fig. 3.5 Implementation of Activation Function using CMOS Differential Amplifier [21]

The transistors are again biased to be in the saturation region to take advantage of the square law relation that exists between the current and voltage in that region. The biasing also requires addition of drain currents of the transistors to be exactly equal to I_{ss}. In practice, this is very difficult to achieve. A feedback circuit employing a current mirror combination is used for this purpose [21]. The transistors M3 and M4 shown in fig 3.5 above, act as current mirrors with transistor M3 connected as a diode (gate connected to the drain) and driving transistor M4. If both M3 and M4 have the same β, then the drain current of M3 gets matched to that of M4. When a differential voltage is applied between the gates of M1 and M2, half of it gets applied to gate-source of M1 and the other half to gate source of M2. This increases the drain current I_{D1} and decreases I_{D2} by an equal amount ΔI. This increase in I_{D1} gets mirrored across to increase I_{D4} through the current mirror combination. The net result of ΔI increase in I_{D4} and ΔI decrease in I_{D2} is to have the output sink a resultant current of $2\Delta I$. An advantage of this kind of design is that the differential output is converted into a single ended output, without the need for any extra circuitry. This saves a lot of chip area when it comes to the actual hardware implementation [21].

3.2.3 Neuron Circuit

The synapse circuit in combination with the activation function circuit forms the neuron circuit. The neuron circuit is used to compute the overall network output based on the input given to the activation function circuit. For multiple input patterns and layers, the neuron circuits can be implemented in parallel thereby taking advantage of the inherent parallelism offered by neural networks for computation. Since all the circuits are wired in parallel, computing operations can be carried out simultaneously and the errors can also be propagated backwards, all at one go. This saves a lot of computation time, unlike with series implementations and results in an enhanced overall system performance [24].

Chapter 4

Weight Storage and Network Training

4.1 Weight Storage Mechanisms

Section 3.2.1 of chapter 3 outlined the importance of the synapse element for information processing in neural networks. This section aims at discussing the various mechanisms through which a synapse can store weight values and retain them for computation. A synapse circuit cannot be deemed as efficient if it is not able to store the weight values and hold them for longer periods of time [19,20]. Researchers have invested a lot of time and efforts into analyzing and implementing techniques for efficient synaptic weight storage [3,24,25,27,28,30,31]. The following sub-sections discuss three mechanisms of synaptic weight storage, among many others. The section starts by explaining the long utilized concept of Floating Gate Storage and then goes on further to explain the more recently researched techniques of weight storage, such as the Hybrid SRAM based Storage and the Memristor based Storage, respectively.

4.1.1 Floating Gate Synaptic Storage

The weight storage mechanism discussed in this sub-section is governed by the mechanism of non-volatile charge storage in a floating gate transistor. A floating gate transistor basically involves a transistor polysilicon gate surrounded by a high quality insulator, such as silicon dioxide (SiO2) as shown in Fig 4.1 [3]. Owing to the excellent insulating capabilities of SiO2 layer, the floating gate is able to retain the stored charge for longer periods of time, thereby providing long term analog memory. Signals on the transistor gate terminal capacitively couple into the channel by means of the floating gate. The floating gate charges are added through the process of hot electron injection and removed by means of electron tunneling [3].

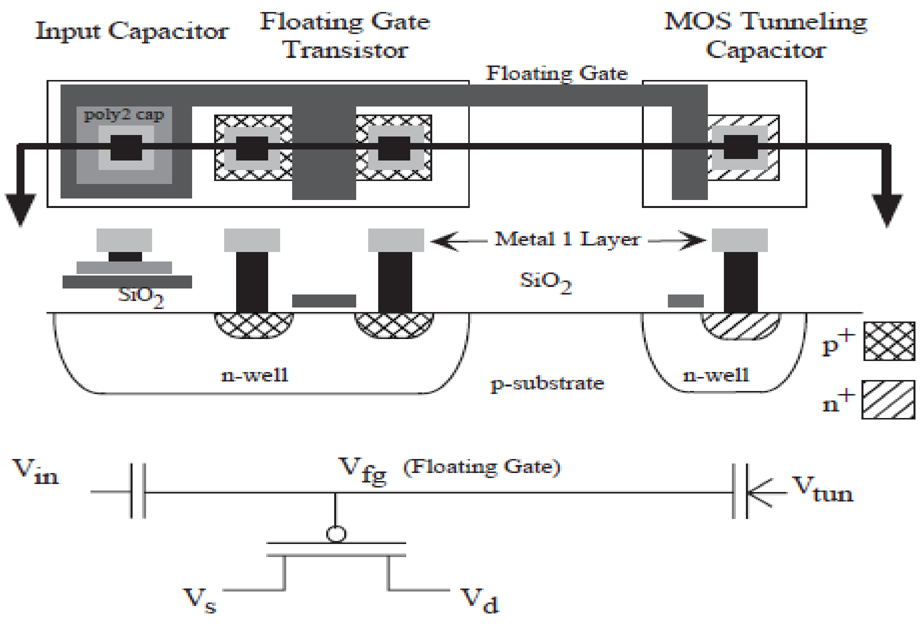


Fig. 4.1 Basic Structure of a Floating Gate pFET Transistor [3]

The concept of floating gate charge storage is used by a Single Transistor Learning Synapse (STLS) that stores the synaptic weights in a non-volatile manner and also performs weight updation [24]. Since past couple of decades, STLS has found several applications in circuit implementations involving continuous time analog signals for both supervised and unsupervised learning techniques. The STLS computes a Post-Synaptic Current (PSC) in response to the input given by the pre-synaptic computation block. The input is typically, a triangular wave generated at the gate of the transistor. The STLS then manifests a bidirectional weight adaption mechanism, based on the time difference between the pre and post-synaptic spike times [24]. This relation between weight updation and spike timing led to the mechanism of Spike Timing Dependent Plasticity (STDP). The arrival of pre-synaptic spike before the post-synaptic one leads to an increase in the synaptic weight. Conversely, the arrival of pre-synaptic spike after the post-synaptic causes the synaptic weight to decrease. The former mechanism is also called ‘Long Term Potentiation’ and the latter, ‘Long Term Depression’. Fig 4.2 shows the non-volatile storage circuit for the STLS and the corresponding plots for gate voltage, weight updation and drain current [25].

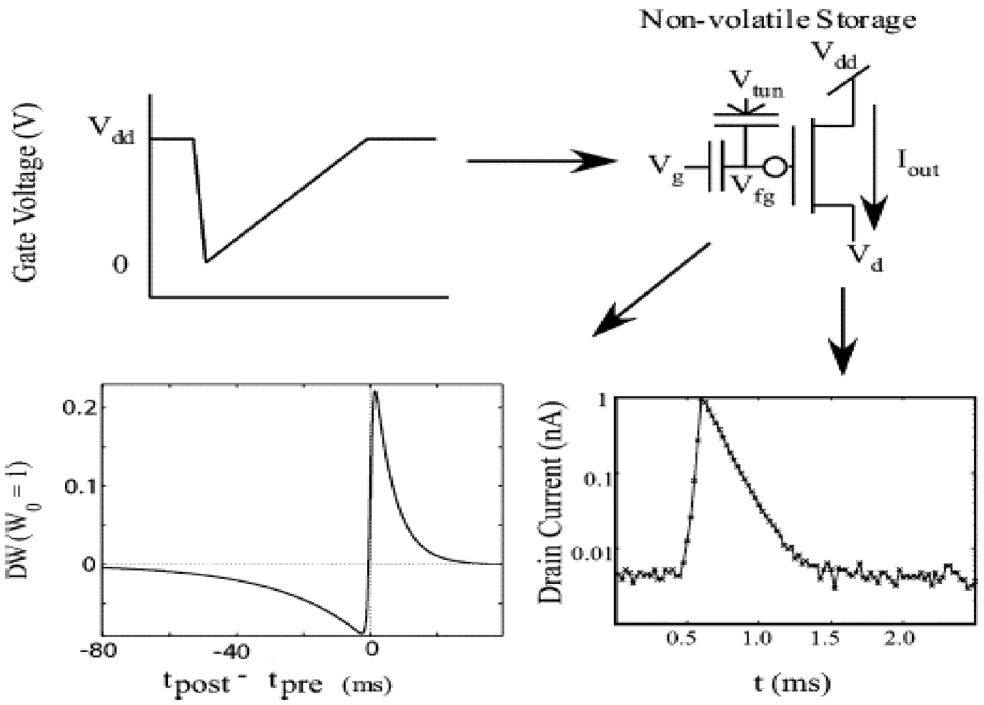


Fig. 4.2 Basic Structure of a Floating Gate (FG) transistor with waveforms for input gate voltage, weight updation and drain current [25]

From an analytical perspective, it is assumed that the source and drain of the floating gate transistor are tied to V_{dd}. The synaptic current as a result of the input given to the gate terminal is given by [25]:

$$I_{syn} \propto te^{-t/\tau_{rise}} \qquad [4.1]$$

The proportionality constant in the equation above determines the strength of the synapse. The rise time is usually of the order of 0.1 ms. The gate voltage is a triangular waveform with nearly zero drain current at the highest gate voltage. For the saturated floating gate pFET structure shown in fig 4.1, the drain current as a function of gate voltage is given by [25]:

$$I_d = I_{bias} we^{-\Delta V_g/V_{gc}} \qquad [4.2]$$

where, I_{bias} is the bias current at the quiescent operating point, ΔV_g is the change in gate voltage and V_{gc} is a constant, based on the capacitive coupling between the gate and surface channel potential. The synaptic weight is related to the floating gate voltage through the relation [25]:

$$\mathrm{w} \propto e^{-V_{fg}} \qquad [4.3]$$

This relation determines the amount of weight stored by the floating gate structure. This stored weight value is proportional to the post synaptic potential, so it is necessary to have the pre-synaptic circuitry generate a waveform by equating [4.2] and [4.3] resulting in a solution for ΔV_g.

The weight updation is based on the injection and tunneling currents and these currents in turn, have an exponential dependency on the tunneling and floating gate voltages. A detailed discussion entailing models for injection and tunneling currents is given in [26]. The general weight update equation is given as:

$$\Delta w \propto I_{inj} - I_{tun} \qquad [4.4]$$

When injection current dominates over the tunneling current, there is an increase in the synaptic weight (long term potentiation) and a decrease (long term depression), when tunneling current takes over the injection current.

4.1.2 Hybrid SRAM based Storage

Among the novel weight storage mechanisms, Static Random Access Memory (SRAM) based synaptic weight storage is discussed in this sub-section [27,28,29]. SRAMs are one of the preferred choices for weight storage, owing to their faster processing speed, robustness and ease of fabrication. In a typical multilayer artificial neural network involving dense interconnections among neurons, the synapses outnumber the neurons by several orders of magnitude. There is a substantial contribution from on-chip synaptic memory to the overall power consumption, in a typical CMOS VLSI implementation of ANNs. In such cases, SRAMs can be used to store the synaptic weights. Fig. 4.3 shows the basic structure of a 6T and 8T SRAM [27].

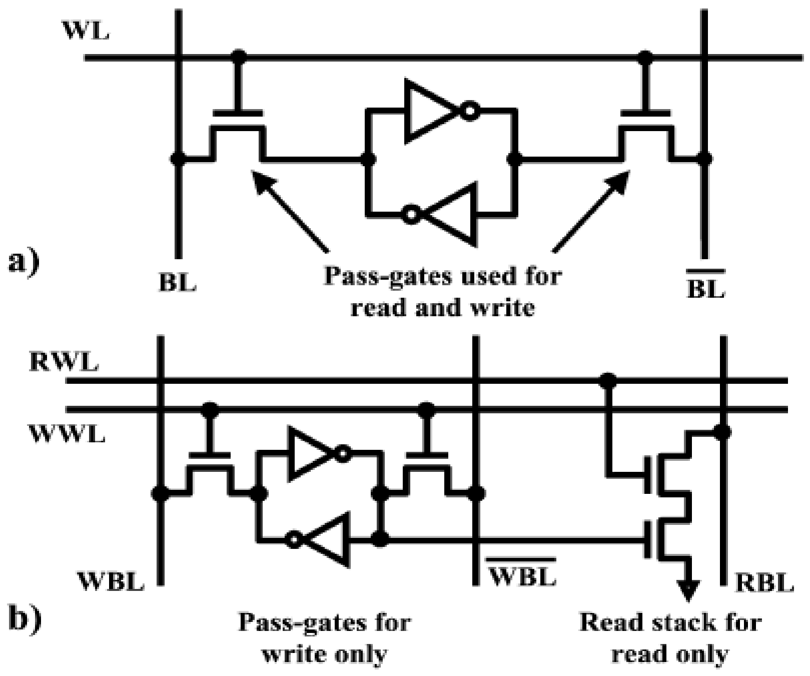


Fig. 4.3 (a) A 6T SRAM Cell. (b) A 8T SRAM Cell [27]

For the 6T SRAM, the pass transistors are connected to the wordline and also to a pair of cross coupled inverters. Read operation is enabled by turning ON the wordline with both the bitlines in their respective HIGH state. Write operation is enabled by pulling any one bitline LOW with wordline still in the ON state. The SRAM goes into its IDLE state and holds onto the stored values when the wordline is turned OFF. The 8T SRAM is essentially similar in operation to the 6T SRAM, but with a difference, that the additional transistors from the read stack enable the structure to have an independent control for both read and write mechanisms. Although this advantage comes with an additional hardware cost, through proper designing and floorplanning techniques, the area overhead can be efficiently minimized. The basic configuration of synaptic weight storage using a 6T SRAM cell is shown Fig. 4.4 below [28].

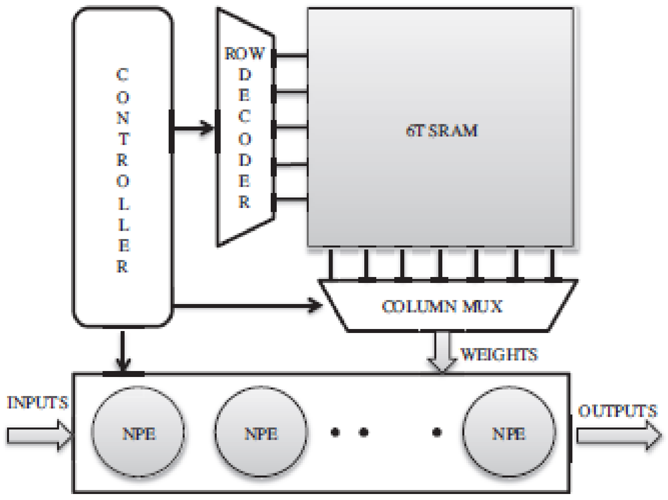


Fig. 4.4 Synaptic Weight Storage using a 6T SRAM [28]

The NPEs are the Neural Processing Elements, modelled as ANNs for core neural computations. The controller coordinates the sequence of operations that take place between the NPEs and synaptic memory. This configuration involving only a single 6T SRAM works well under normal supply voltage conditions. At scaled supply voltage conditions, however, the 6T SRAM is more prone to bitcell failures [29]. This further warrants to have an efficient weight storage mechanism at scaled supply voltage conditions. So, a more proven approach is to use a combination of 6T and 8T SRAM cells for weight storage. For typical neural network applications, such as handwritten digit recognition, the Least Significant Bits (LSBs) of the input training datasets are less prone to bitcell failures and manifest an intrinsic error resiliency. The Most Significant Bits (MSBs) on the other hand, are more prone to bitcell failures at scaled supply voltages. The classification accuracy of the neural network can be impaired if MSBs of the synapses are corrupted. A more efficient approach is to have the LSBs of synaptic weights stored in 6T SRAMs and MSBs stored in more robust 8T SRAMs, resulting in a hybrid SRAM configuration for synaptic weight storage. This ensures correct classification accuracy and enhances the overall capability of the system.

4.1.3 Memristor based Storage

Traditional weight storage mechanisms consist of passive devices, like resistors and capacitors and active structures, like floating gate transistors. They suffer from issues arising due to permanent fabrication of resistors (making them invariant to change once fabricated), shorter weight retention time of capacitors and higher synaptic non-linearity of floating gate transistors. SRAM based storage discussed in the previous sub-section, has the drawback of significant static and dynamic power dissipation. This sub-section discusses a more recent approach for weight storage that involves the use of memristors. A memristor is a programmable resistor that retains charge in a non-volatile manner depending on the voltage applied across it or the current going through it [30]. Its existence was first postulated by Leon Chua in 1971, as the fourth basic element of electronic circuits after resistor, inductor and capacitor. The idea was then generalized to a whole new class of dynamic systems called memristive devices.

As the voltage is varied across the memristor to change its memristance, after a certain point in time, the memristor ‘memorizes’ that memristance value and holds on to that value even in the absence of an applied voltage. Renewed interest among researchers for memristors and memristive devices can be attributed to the fact, that memristors behave in much the same manner as biological synapses do [30] [31]. Additionally, memristors are nanocomponents and can be fabricated using VLSI technology at a very small silicon area overhead. Memristors are used to not only store weight values, but also perform a multiplication operation of stored weights with the input voltage values. Fig. 4.5 shows the basic neural synapse configuration built using four identical memristors arranged in a classical Wheatstone bridge like fashion [32].

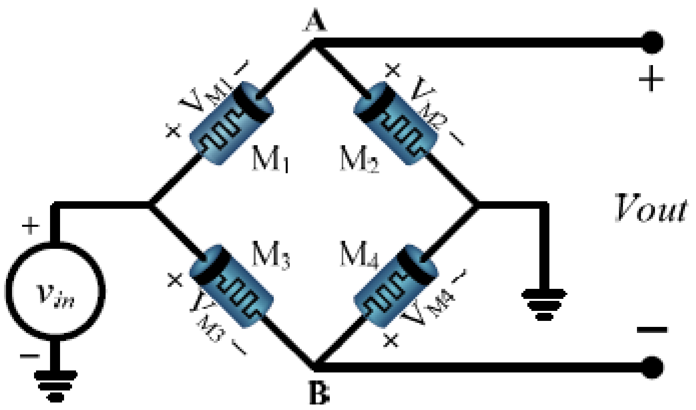


Fig. 4.5 Memristor Bridge Circuit Configuration [32]

M1, M2, M3 and M4 are the memristors arranged in a manner, similar to a Wheatstone's bridge that uses resistors; v_{in} is the input voltage and v_{out} is the output voltage. The voltage at points A and B can be obtained by applying the potential divider formula. It may also be noted that the voltage across memristor M2, is the voltage at node A with respect to ground and that across memristor M4, is voltage at node B with respect to ground. The output voltage is then simply the difference of potentials at nodes A and B. When a positive input voltage is applied, the memristance of M1 and M4 decreases, while that of M2 and M3 increases [32]. This results in potential at node A being higher than that at node B, thus giving a positive v_{out} alongwith a subsequent increase in synaptic weight. On the contrary, application of negative v_{in} results in a reverse scenario with potential at node B being higher than that at node A. The effect is to have a negative v_{out} implying a synaptic weight decrease. A balanced condition occurs when the voltages at nodes A and B are equal, in which case, the output voltage is zero resulting in a zero synaptic weight [32] [33]. In multilayer neural networks where one input connects to several other inputs, it is both convenient and feasible to have output currents rather than output voltages, so that KCL can be directly applied at a particular node to get the resultant current from that node. The output from above memristor bridge circuit can be converted into an output current using a differential amplifier, that basically functions as a voltage to current converter. Fig. 4.6 shows the circuit configuration [33].

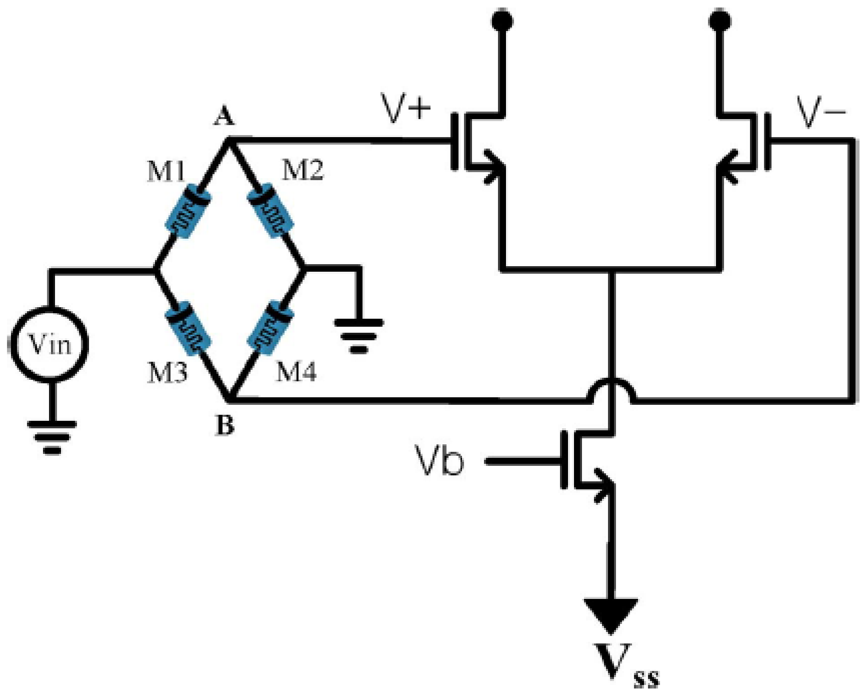


Fig. 4.6 Memristor synaptic circuit showing the configuration for converting output voltage of memristor bridge into an output current using a differential amplifier [33]

The weighing operation is performed normally by the memristor bridge. The differential amplifier with active load acts as a current source and also serves as an activation unit. The output current from the differential amplifier then feeds into other network layers, such as the hidden layer, if available, or can just be compared with the target output current to evaluate the resultant error. When the desired final output is voltage instead of current, it can be achieved by connecting a resistor across the output terminals of the differential amplifier. Since the output current is usually of the order of several milliamperes, resistor values typically in the range of 100 to 500 kilo-ohms are used.

4.2 Training Algorithms in Neural Networks

In multilayer artificial neural networks, a given set of inputs is expected to produce a desired set of output(s) with an aim to meet or closely approximate the pre-determined target output. Such networks, usually consist of an input layer, one or more hidden layers and an output layer. Information flow is directed in a feed forward manner from the input layer to the hidden layer and then to the output layer. When the information produced at the output does not meet the target specifications, some mechanism is necessary to take care of the difference between the computed and the target value. This difference, in its simplest terms, is called an error; the network is expected to learn from this error and further train itself in a consistent manner until the error goes down to a minimum. This process of network learning or network training is governed by several learning algorithms. In this section, two of the most widely used learning algorithms are discussed.

4.2.1 The Backpropagation Algorithm

The backpropagation algorithm, originally proposed by Rumelhart and McClelland in 1986 [34] is a very widely and commonly used algorithm for neural network training. It is an example of supervised learning algorithm that was discussed in section 3.1.2.1. The network knows the target output and performs an adaption mechanism on its own output to drive the resultant error to a minimum. The process involves computing the network output by passing the weighted inputs through a non-linear activation function and then propagating the acquired error signal backwards into the network to perform a weight updation procedure. This backward propagation of error has given this algorithm its nomenclature. The training process generally takes place in an iterative manner through the evaluation of negative gradient of Mean Square Error (MSE) function to calculate new weight parameters [35]. There are two phases associated with backpropagation, one is the 'feedforward' phase and the other is the backward propagation, also known as 'feedback' phase. In the feedforward phase, a pattern of input data is presented to the input layer that gets multiplied with the stored weight values and passes forward to the hidden layer after mapping to a non-linear activation function. A similar process occurs from the hidden layer to the output layer. In this way, the input information is transferred from layer to layer until a pattern gets generated at the output. The output being a function of the net input is calculated as [36]:

$$output = f(net_i) \qquad [4.5]$$

$$net_i = \sum_i W_{ij} O_j + \theta_i \qquad [4.6]$$

where, W_{ij} is the weight vector between nodes i and j, O_j is the output of node j and θ_i is the bias of node i. The computed net_i is then passed through an activation function that is generally, a sigmoid function given by [36]:

$$f(net_i) = \frac{1}{1+e^{-net_i}} \quad [4.7]$$

In the backward propagation phase, the generated output is compared to the target output to calculate the error, if any. The error signal depends on the type of error function used. A commonly used error function is the sigmoid function expressed as [36]:

$$E = \frac{1}{2}\sum_k (t_{kj} - o_{kj})^2 \quad [4.8]$$

The error signal then propagates backward into the network structure to adjust the weight parameters and compute new weight values for subsequent iterations. The process then repeats until convergence is achieved. Furthermore, for backpropagation to be effective, it is important that the activation function be differentiable.

Although very widely used in several neural network applications, the backpropagation algorithm is stymied from a few drawbacks, such as slow convergence, possibility of getting trapped into a local minima and learning rate adjustment. The convergence speed is primarily attributed to the kind of activation function used to calculate the output. The first derivative of the activation function is usually the major source of poor convergence. In order to speed up the convergence process, researchers have proposed a variety of techniques like the one proposed in [37]. It utilizes an arc-tangent activation function for mapping from the input layer to the hidden layer and sigmoid activation function for mapping from the hidden layer to the output layer.

The problem of local minima occurs when the output approaches either extremes of 0 or +/-1. In that case, the derivative of the activation function is almost zero, which leads to a very small weight change, owing to a very small backpropagated error signal. This can cause the network to give an incorrect output without any significant weight change and further slowdown the convergence process. To avoid the problem of local minima, several mechanisms have been proposed in [38] that involve using two learning rate parameters instead of one to improve the convergence. This not only helps avoid getting trapped into a local minimum, but also speeds up the overall training process, thereby ensuring efficient error minimization through backpropagation.

4.2.2 The Levenberg-Marquardt Algorithm

Another commonly used technique for neural network training is the Levenberg-Marquardt (LM) algorithm, originally published by Kenneth Levenberg in 1944 and later optimized and rediscovered by Donald Marquardt in 1960 [39]. The algorithm is one of the most efficient method for neural network training and is widely used for solving problems involving non-linear least square functions. It attempts to find the minimum of a multivariate objective function, expressed as a sum of squares of non-linear real valued functions [39]. The LM algorithm can be thought of as an approximation to the classical Gauss-Newton algorithm that involves minimizing least square functions using a Hessian matrix, a square matrix consisting of partial second order derivatives of a scalar valued function. The algorithm can be considered as a combination of gradient descent (principle of backpropagation) and Gauss-Newton method. The algorithm works without actually calculating the Hessian matrix, but rather calculates the gradient vector and the Jacobian matrix. The computation of the Jacobian matrix is an important step in this algorithm [39].

If we have a function $f(x)$, that we want to minimize with respect to some parameter value x and if we consider a sum of squared error function $E(x)$ given by [40] [41]:

$$E(x) = \sum_{i=1}^{N} e_i^2(x) \qquad [4.9]$$

then, according to Newton's method:

$$\Delta x = -[\nabla^2 E(x)]^{-1} \nabla E(x) \qquad [4.10]$$

where, $\nabla^2 E(x)$ is the Hessian matrix and $\nabla E(x)$ is the gradient vector governed respectively, by the following equations [40] [41]:

$$\nabla E(x) = J^T(x) e(x) \qquad [4.11]$$

and

$$\nabla^2 E(x) = J^T(x) J(x) + S(x) \qquad [4.12]$$

Where $J(x)$ is the Jacobian matrix that consists of partial derivatives of error values $e(x)$ with respect to x and is given by [40] [41]:

$$\begin{bmatrix} \frac{\partial e_1(x)}{\partial x_1} & \frac{\partial e_1(x)}{\partial x_2} & \cdots & \frac{\partial e_1(x)}{\partial x_n} \\ \frac{\partial e_2(x)}{\partial x_1} & \frac{\partial e_2(x)}{\partial x_2} & \cdots & \frac{\partial e_2(x)}{\partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial e_N(x)}{\partial x_1} & \frac{\partial e_N(x)}{\partial x_2} & \cdots & \frac{\partial e_N(x)}{\partial x_n} \end{bmatrix} \qquad [4.13]$$

Gauss-Newton method considers $S(x)$ to be approximately equal to zero, so equation 4.10, after substitution becomes:

$$\Delta x = -[J^T(x) J(x)]^{-1} J^T(x) e(x) \qquad [4.14]$$

Now, the LM algorithm is essentially, a modification to the above equation. The process of modifying the diagonal elements of $J^T(x)J(x)$ matrix is called damping and the equation above consists of a damping term μ which is always positive. With damping term added, equation 4.14 modifies to become [40] [41]:

$$\Delta x = -[J^T(x) J(x) + \mu I]^{-1} J^T(x) e(x) \qquad [4.15]$$

For very small positive values of μ, the term μI in the above equation can be ignored and the algorithm becomes Gauss-Newton. Larger values of μ make the algorithm function as Gradient Descent. The algorithm is thus capable of alternating between a slow descent approach, when sum of squares function is far from a pre-defined minimum value and a fast convergence approach, when the value is in close vicinity to the minimum [42].

Once Δx is computed, the new sum of squares is calculated using $x + \Delta x$. If this new function is greater than the original sum of squares function given by $E(x)$ in equation 4.9, then the damping term μ is increased by a parameter β, else reduced by β and the process repeats iteratively. Convergence criterion for the algorithm is reached, when the norm of $\nabla E(x)$ evaluates to be less than some pre-determined value or when sum of squares function is minimized to some specified value.

Chapter 5

Design Implementation

This chapter begins by showing the transistor level hardware implementation of the synapse and the activation function circuit using CMOS technology. Subsequently, the implementation of a single layer of neuron circuit which is simply an interconnection of a single synapse and activation function element, has been shown. The last section manifests the implementation of the feedforward multilayer network for solving the XOR classification problem.

Section 3.2 of chapter 3 discussed various electronic circuits required for the implementation of a multilayer feedforward neural network in the silicon domain. Synapse and Activation Function circuits form the basic building blocks in the CMOS design of artificial neural networks. This chapter demonstrates the actual implementation of the synapse and activation function elements, first as a standalone entity to verify the individual functionality, and then as a part of a multilayer network implementation.

The circuits shown in the upcoming sections are designed using Cadence OrCAD Capture version 16.6, and the simulation is performed using Cadence PSpice A/D. The layout of the circuits is accomplished using Electric VLSI Design System version 9.06 with MOCMOS technology on a 200 nm design scale. A Network Consistency Check (NCC), also known as

Layout Vs Schematic (LVS) check, is performed at the individual circuit and overall structural level to verify the design functionality from a fabrication standpoint.

5.1 Synapse Circuit

The synapse circuit for realizing analog multiplication is implemented using 11 MOSFETs arranged to form a Gilbert Multiplier Cell, that performs a four quadrant multiplication operation. All MOSFETs are biased to operate in the sub-threshold regime and provide a square law relation between the current and the voltage. Fig. 5.1 shows the design of the synapse circuit using Cadence OrCAD Capture version 16.6.

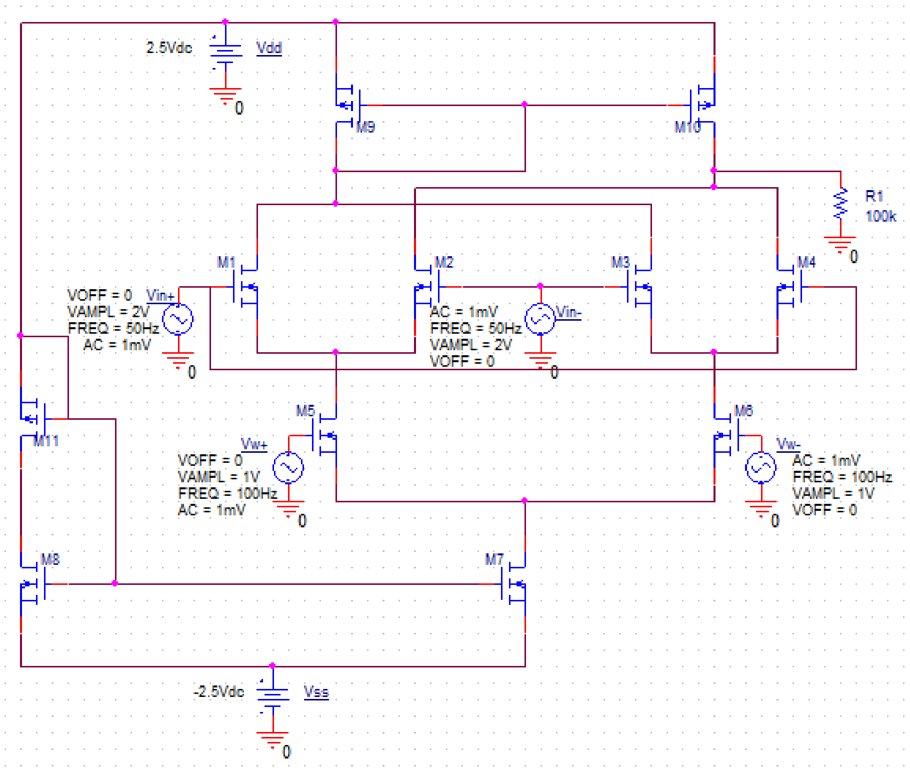
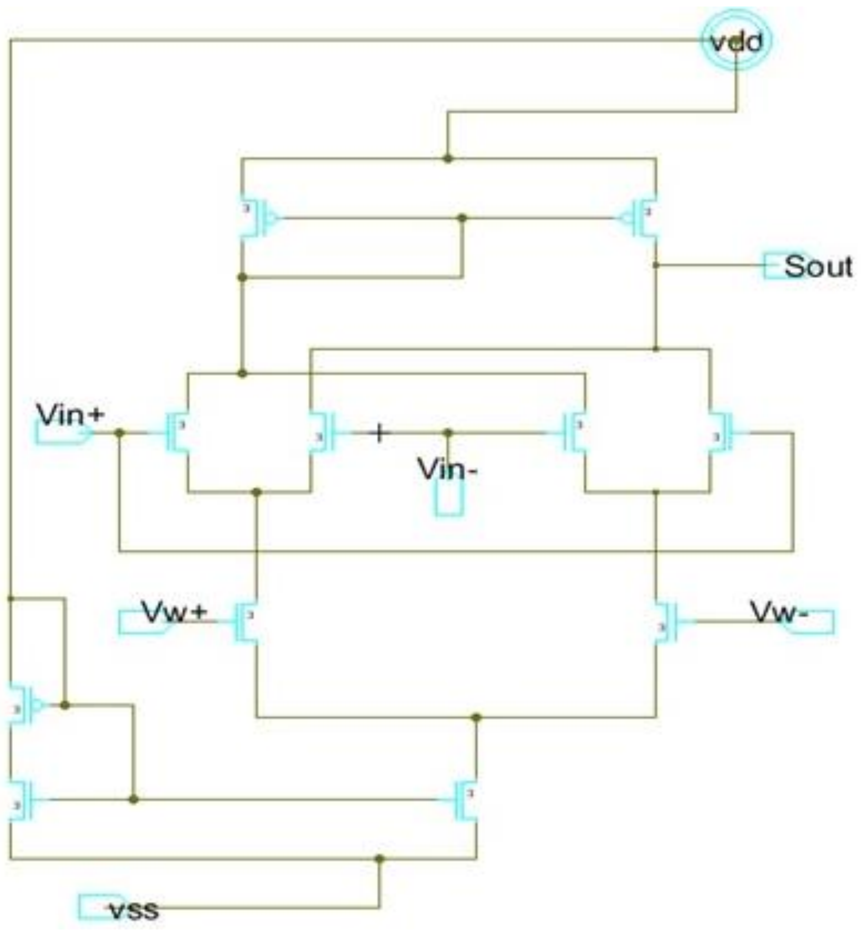


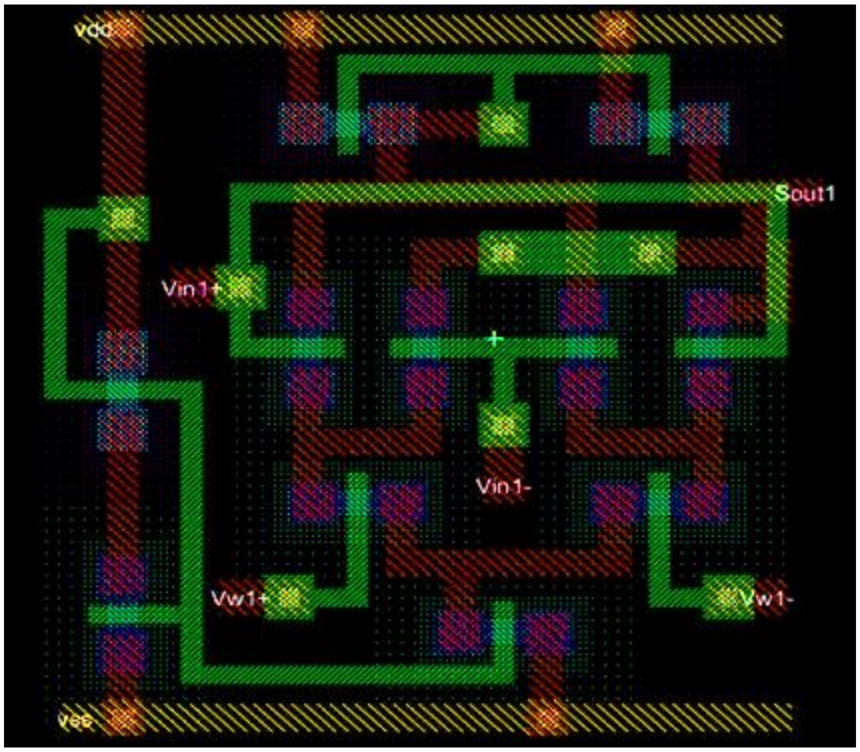
Fig. 5.1 Design of the Synapse Circuit using Cadence OrCAD

Since the synapse circuit computes output in the form of a current, a high value resistor, typically 100k, is added to get an equivalent voltage output.

Fig. 5.2 depicts the schematic and layout of the synapse circuit done using Electric VLSI Design software. All transistors have a width of 3 microns and length of 2 microns. The layout is made as compact as possible and is found to occupy an area of 80 x 85 sq. microns.



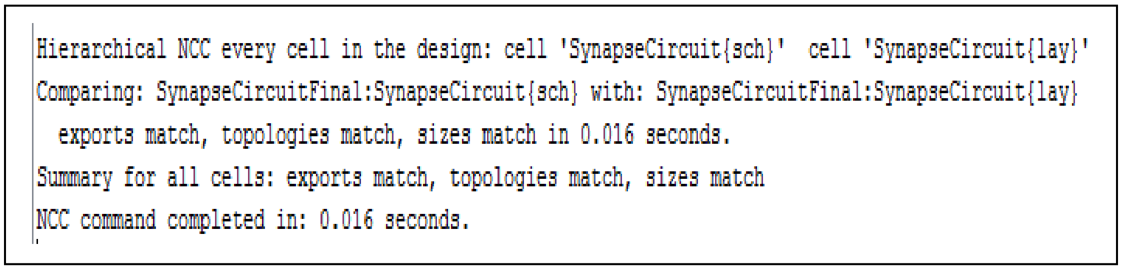
(a)



(b)

Fig. 5.2 (a) Schematic of Synapse Circuit. (b) Layout of Synapse Circuit Using Electric VLSI Design software

NCC is then performed to verify the consistency of the circuit in terms of layout and schematic and the result is depicted in Fig. 5.3 below.



```
Hierarchical NCC every cell in the design: cell 'SynapseCircuit{sch}'  cell 'SynapseCircuit{lay}'
Comparing: SynapseCircuitFinal:SynapseCircuit{sch} with: SynapseCircuitFinal:SynapseCircuit{lay}
  exports match, topologies match, sizes match in 0.016 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.016 seconds.
```

Fig. 5.3 Network Consistency Check Result for the Synapse circuit

The circuit passes the initial LVS check in terms of circuit exports (nets or ports), overall network topology and transistor sizes.

5.2 Activation Function Circuit

The mechanism for providing the non-linear activation is implemented using a CMOS differential amplifier circuit, designed using enhancement mode MOSFETs functioning in the sub-threshold region of operation. Fig. 5.4 below shows the schematic of the circuit designed in Cadence OrCAD.

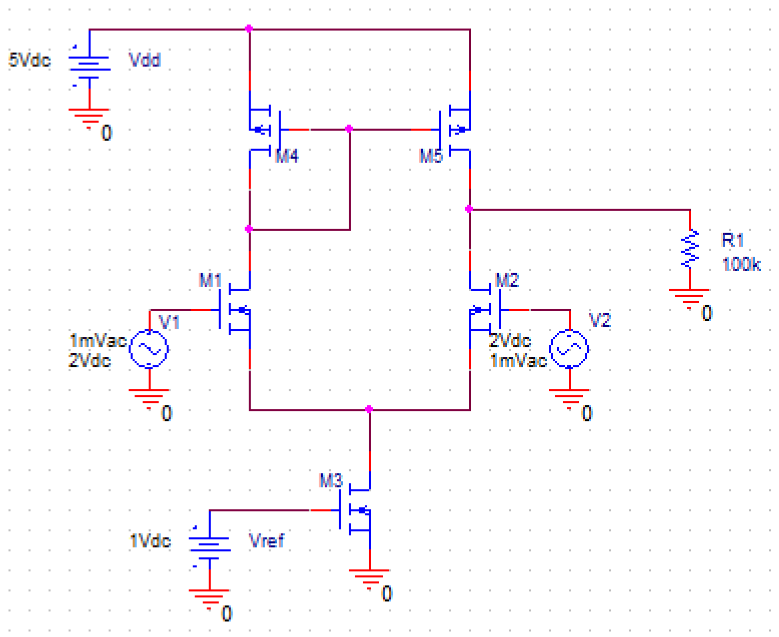
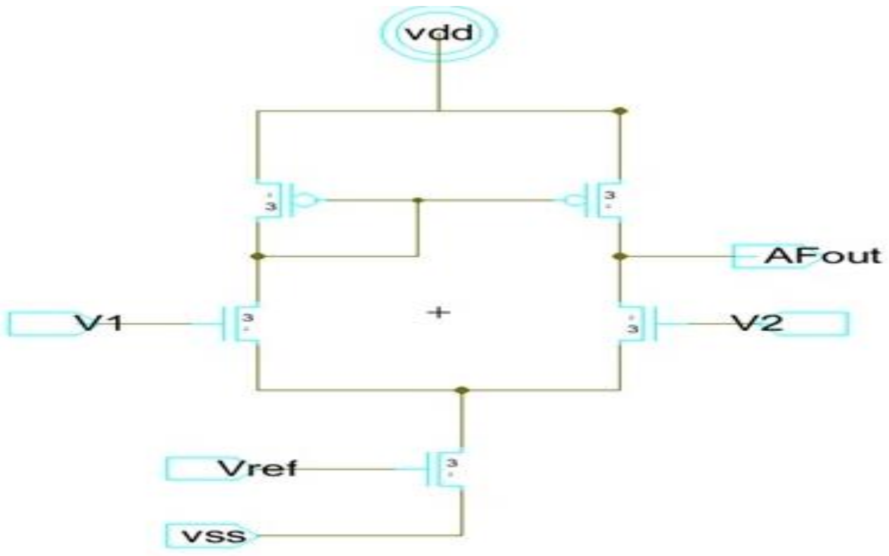
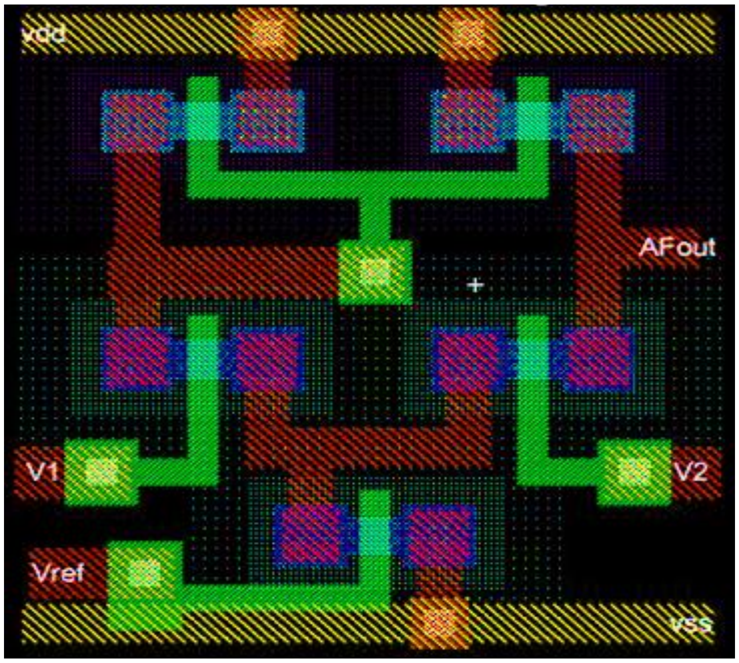


Fig. 5.4. CMOS Differential Amplifier used as a non-linear Activation Circuit designed using Cadence OrCAD

The same circuit is designed in a slightly different way using Electric VLSI Design software, wherein all the enhancement mode MOSFETs are replaced by standard MOSEFTs. All transistors are designed to operate in the sub-threshold region and have a length of 3 microns and width of 2 microns. Fig. 5.5 shows the schematic and layout of the activation function circuit. The layout occupies an area of 49 x 52 microns.



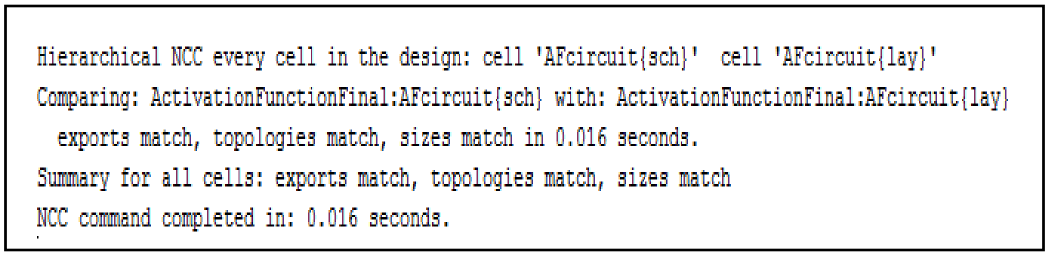
(a)



(b)

Fig. 5.5 (a) Schematic of the Activation circuit. (b) Layout of the Activation circuit. using Electric VLSI Design software

NCC is then performed similar to that done for the synapse circuit and the results are shown in Fig. 5.6.



```
Hierarchical NCC every cell in the design: cell 'AFcircuit{sch}'  cell 'AFcircuit{lay}'
Comparing: ActivationFunctionFinal:AFcircuit{sch} with: ActivationFunctionFinal:AFcircuit{lay}
  exports match, topologies match, sizes match in 0.016 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.016 seconds.
```

Fig. 5.6 Network Consistency Check Result for the Activation Function circuit

The circuit passes the LVS check in terms of circuit exports (nets or ports), overall topology and transistor sizes.

5.3 Single Layer Neuron Circuit

The single layer neuron circuit is simply an interconnection of the standalone synapse and activation function circuits that were discussed in the previous two sub-sections. Fig. 5.7 shows the block diagram of a typical single layer neuron cell. The synapse circuit block computes the multiplication result of input voltage and weight value and the activation function circuit acts on the resulting product to generate the output.

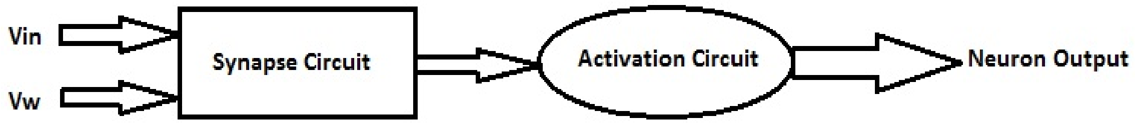


Fig. 5.7 Block Diagram of a Single Layer Neuron Cell

The CMOS implementation of the single layer neuron cell is shown in Fig. 5.8 below. Since the synapse and activation circuits compute the output in the form of a current, two resistors, each with a value of 100k ohms are added to provide a voltage output.

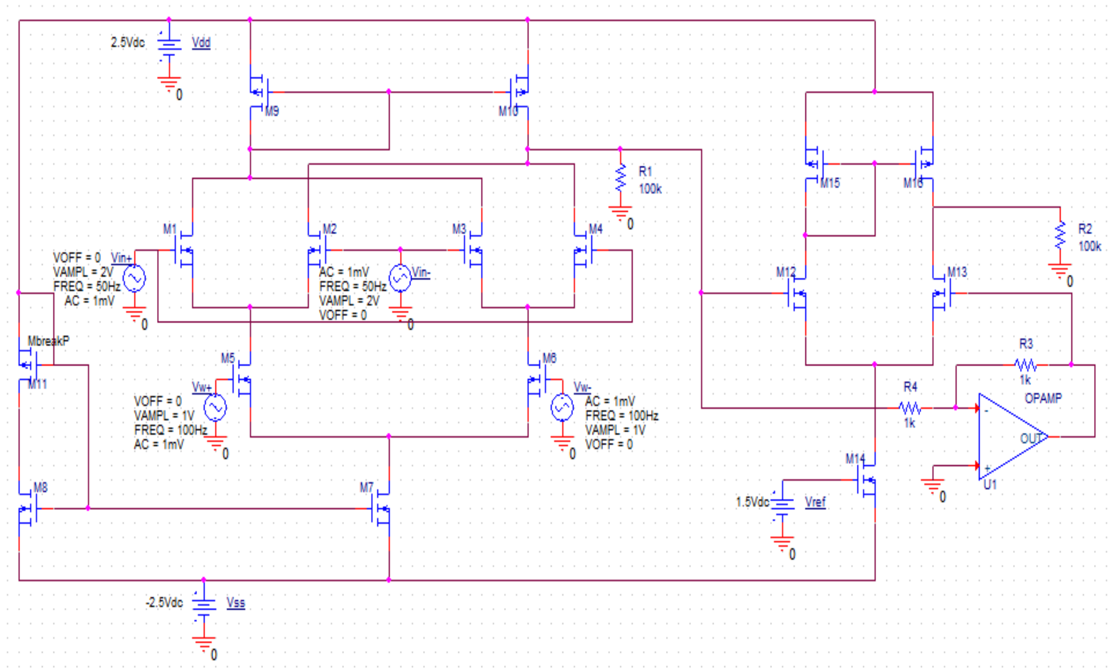
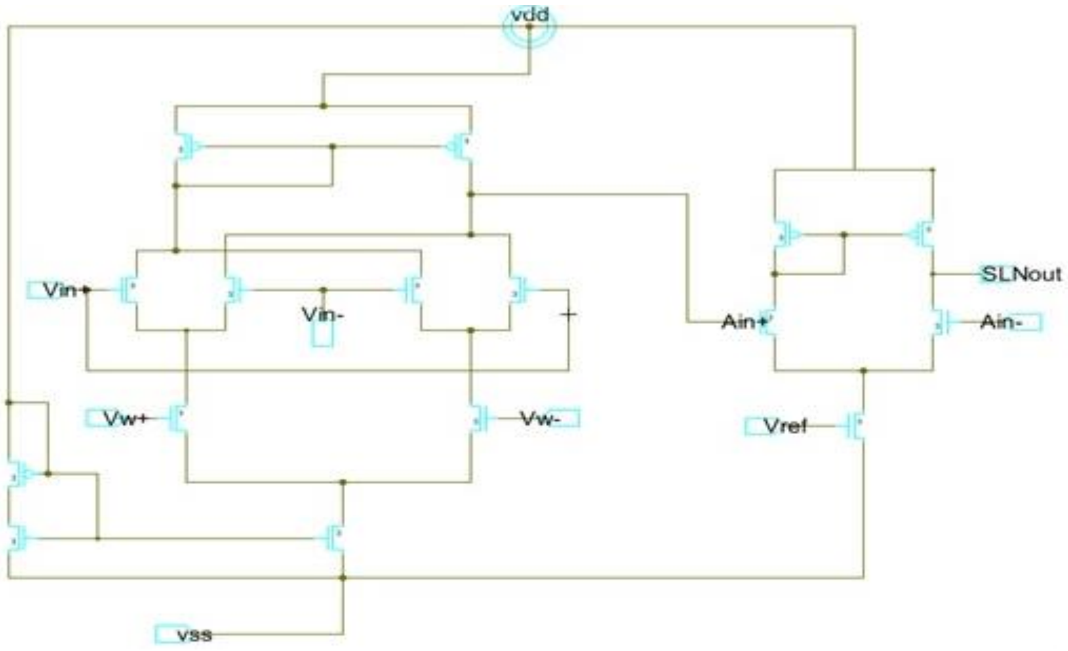


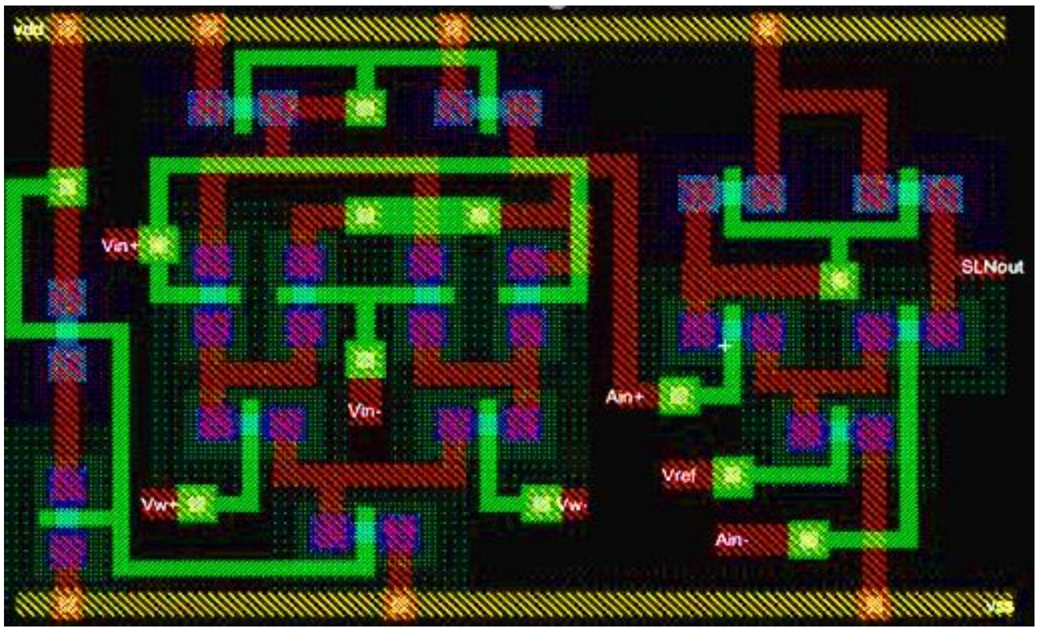
Fig. 5.8 Circuit Design of Single Layer Neuron Circuit using Cadence OrCAD

The Op-Amp shown in figure 5.8, acts as an inverting buffer and computes a simple negation of the voltage applied at its input. The resulting output is fed to transistor M15. The goal is to feed complementary voltages to the gates of transistors M14 and M15. When the network is required to operate in the digital domain with binary inputs, the op-amp can be replaced by a simple CMOS inverter circuit.

Fig. 5.9 depicts the schematic and layout design in Electric VLSI Design software. Each transistor has a length of 3 microns and width of 2 microns. The layout is found to occupy an area of 132 x 85 sq. microns.



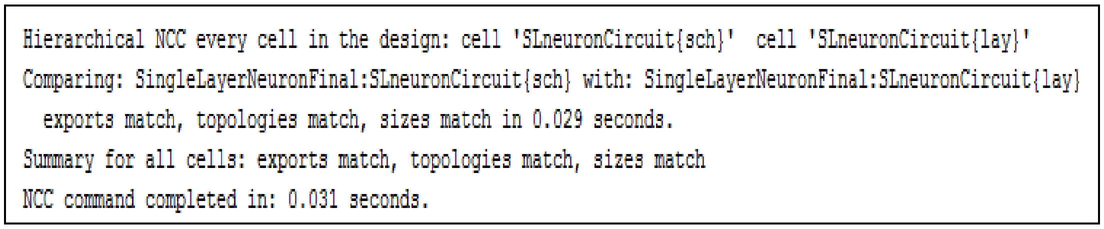
(a)



(b)

Fig. 5.9 (a) Schematic of Single Layer Neuron. (b) Layout of Single Layer Neuron using Electric VLSI Design software

NCC is performed as before to ensure that the design meets all the constraints and the result is shown in fig. 5.10. The circuit as per expectations satisfies the NCC in terms of circuit exports, overall network topology and transistor sizes.



```
Hierarchical NCC every cell in the design: cell 'SLneuronCircuit{sch}'  cell 'SLneuronCircuit{lay}'
Comparing: SingleLayerNeuronFinal:SLneuronCircuit{sch} with: SingleLayerNeuronFinal:SLneuronCircuit{lay}
  exports match, topologies match, sizes match in 0.029 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.031 seconds.
```

Fig. 5.10 Network Consistency Check Result for Single Layer Neuron Circuit

5.4 Multilayer Feedforward Neural Network Implementation

The idea behind the design of a single layer neuron circuit from the previous section can now be extended, to design a structure consisting of multiple layers and having more capability compared to that of a single layer circuit. The network operates in the feedforward mode, meaning that the inputs applied to the network on multiplication with the weight values, wash through the network, layer to layer, until the final output is obtained. The neural architecture designed in this section attempts to solve the XOR classification problem, explained next, having the feature of non-linear separability when the input data is plotted in the x-y plane. The network is comprised of four synapse elements and two activation function elements that form the input layer, another two synapse elements and one activation function element that forms the hidden layer and finally, the output of the latter activation function is taken to be the overall network output forming the output layer. A MATLAB code is developed to train the neural network using the backpropagation algorithm explained in section 4.2.1, to have the network output approximately equal to the XOR output with a certain level of user defined error tolerance.

5.4.1 Example Application: Solving the XOR Classification Problem

The fundamental limitation with single layer neural networks specifically designed to function as classifiers is that, they can only be used to classify problems having a linearly separable nature. The idea of linear separability stems from the basic working principle of McCulloch and Pitts Neuron discussed in section 2.2. Since the output of the neuron can either be zero or one, it can be classified to be in one of the two categories at any given time and input conditions. If the neuron output is recorded for different input and weight values and then plotted in the input plane, a line, or in some cases a plane, can be drawn to correctly classify the output that will always lie on any one side of the line or plane.

A single layer neural network with 'n' input layers and one output layer can correctly classify problems having linearly separable behavior. Digital logic operations such as AND, OR, NOT and other simple pattern classification problems based on these can be easily realized with single layer networks. However, for complex classification applications based on XOR and XNOR operations, single layer networks turn out to be ineffective. To strike a good analogy, consider the need to implement an XOR function in digital logic with two inputs A and B and further assume a hypothetical scenario where a standard XOR gate is not available. In such a case, one way to implement XOR is to have two AND gates to get $A\bar{B}$ and $\bar{A}B$ and then have an OR gate to logically add them together, thereby realizing an XOR function. The two AND gates form the first layer and the OR gate forms the second layer, giving a two-layer implementation.

Similarly, with neural networks, the requirement is to have a multilayer structure as shown in fig. 5.11 [43] with at least one hidden layer capable of correctly classifying the patterns presented at its input.

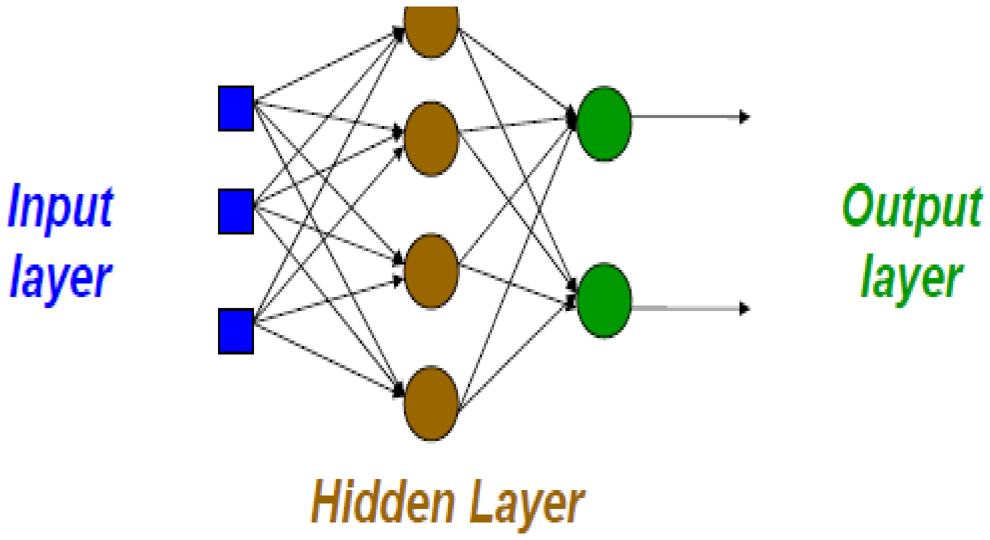


Fig. 5.11 Multilayer Neural Structure with One Hidden Layer [43]

Since XOR forms the basis of several other complex classification problems like face recognition, handwritten digit recognition, etc., the multilayer network designed in this section works towards solving the XOR problem. Table 5-1 shows the truth table for an XOR function with two inputs x_1 and x_2 in both unipolar and bipolar mode.

Table 5-1 Truth table of XOR

(a) Unipolar

x_1	x_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

(b) Bipolar

x_1	x_2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

Considering unipolar mode of operation, let the like inputs be encoded as class B and unlike inputs as class A. The plot of these two classes of data points in the x_1x_2 plane is shown in fig. 5.12 [44]

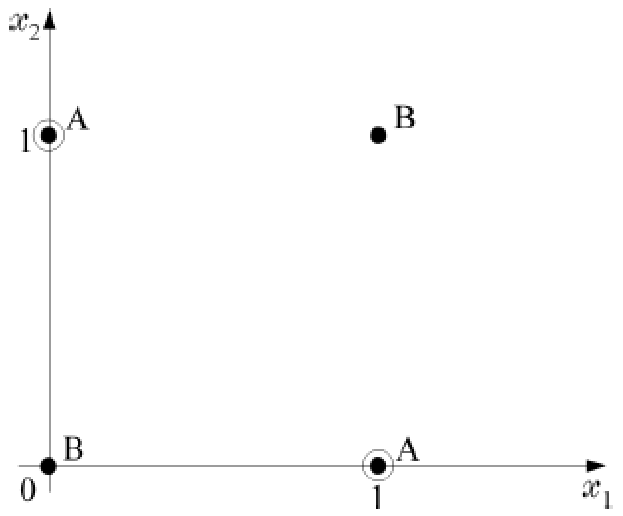


Fig. 5.12 Plot of input data points for XOR classification [44]

It can be seen that no single line of separation can be drawn to classify these points accurately to lie in one of the two classes. There should be atleast two lines or any other planar structures to classify them. With two layer networks, the first layer is used to perform a non-linear mapping of the input data making it linearly separable. The second layer does a logical OR of the linearly mapped data, thereby ensuring proper classification. The output neuron realizes a hyperplane in the transformed input space that classifies data points into two sets or classes. Fig. 5.13 shows one possible way of classifying data points from fig. 5.12 [44].

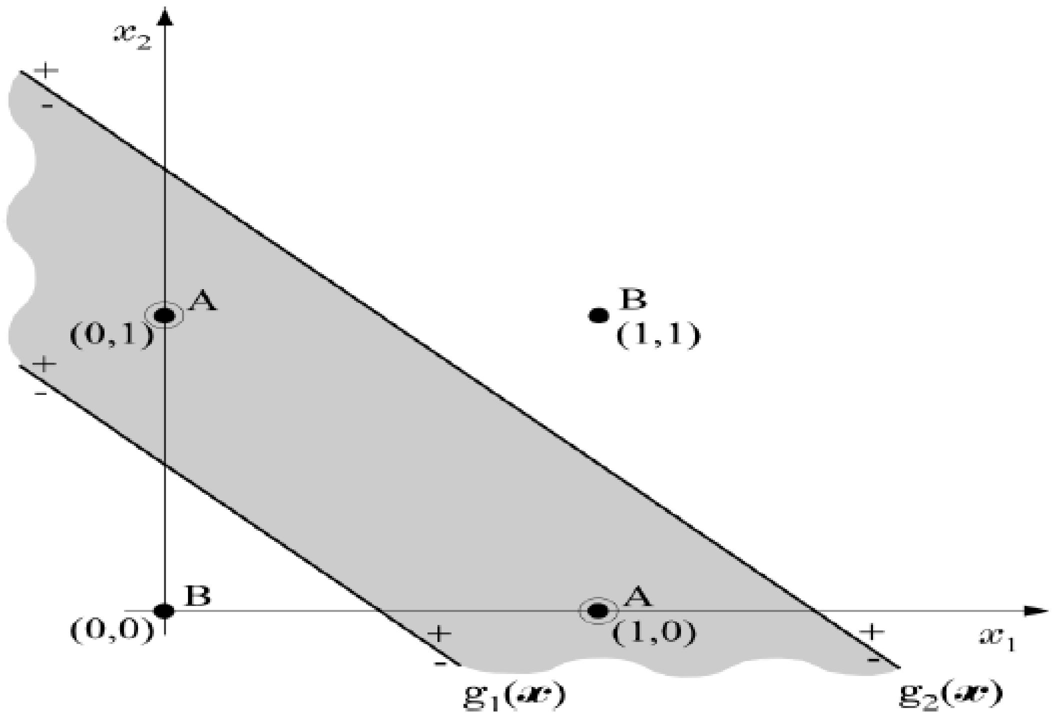


Fig. 5.13 One possible solution of XOR classification [44]

$g_1(x)$ and $g_2(x)$ are any two functions that divide the input plane into two parts. The portion below $g_1(x)$ and above $g_2(x)$ corresponds to class B and the portion in between corresponds to class A. This XOR classification process can also be generalized by having clusters of data, encoding them to be in two classes and then using a multilayer neural network to properly classify them. One key advantage of using multilayer networks is that the presence of the hidden layer leads to efficient error minimization, providing network output very close to the target one. The multilayer network can be designed to have more than one hidden layer depending on the amount data that requires classification. Multiple hidden layers enable quicker error minimization and faster convergence, albeit at the cost of increased component count and overall structural complexity.

Fig. 5.14 shows the block diagram of the overall network proposed in this work to serve the example application.

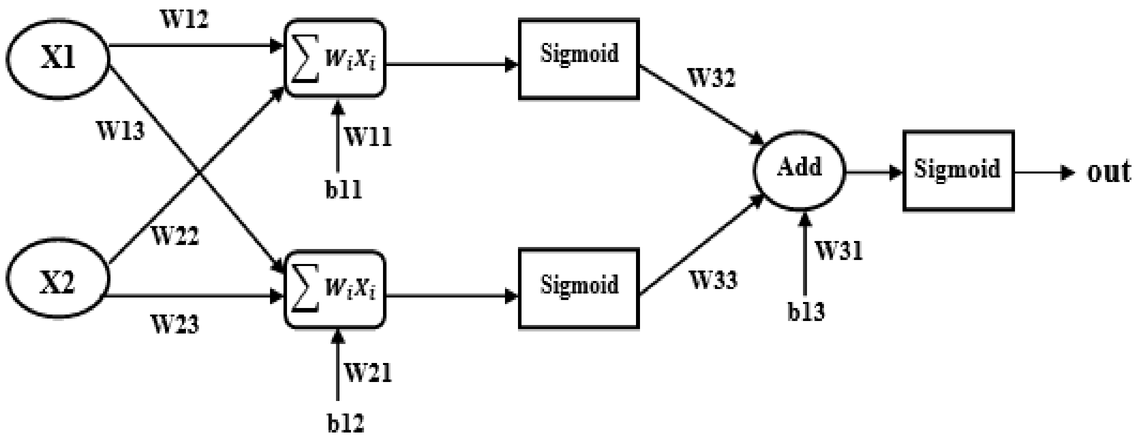


Fig. 5.14 Block Diagram of the XOR Multilayer Feedforward Network

X1 and X2 are the inputs that get multiplied with the weights (assigned randomly) and then pass through the activation function block labelled as ‘Sigmoid’ in the figure above. b11, b12 and b13 are the optional bias inputs that make the synapse output stronger or weaker. The ‘Add’ block essentially adds the synaptic outputs from the hidden layer with the bias term b13, to calculate the overall network output after the squashing function acts on it. On basis of the block diagram shown above, the schematic design is carried out in Cadence OrCAD and Electric VLSI Design software just as in the case of previously designed circuits. Fig. 5.15 depicts the schematic of the network in Cadence OrCAD.

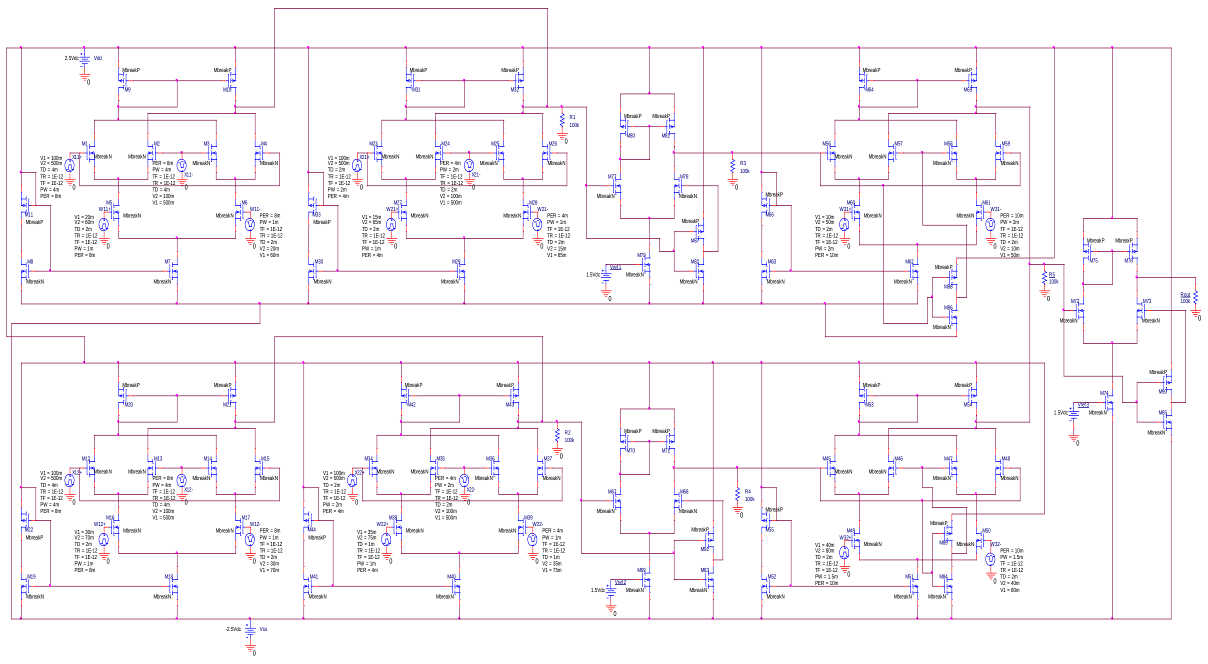
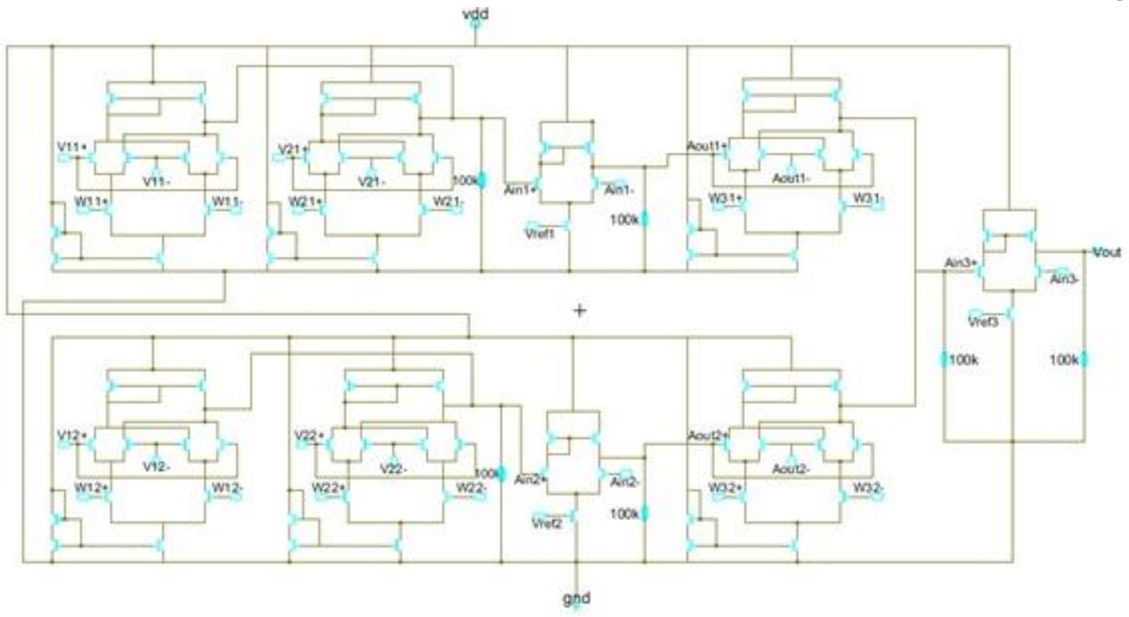
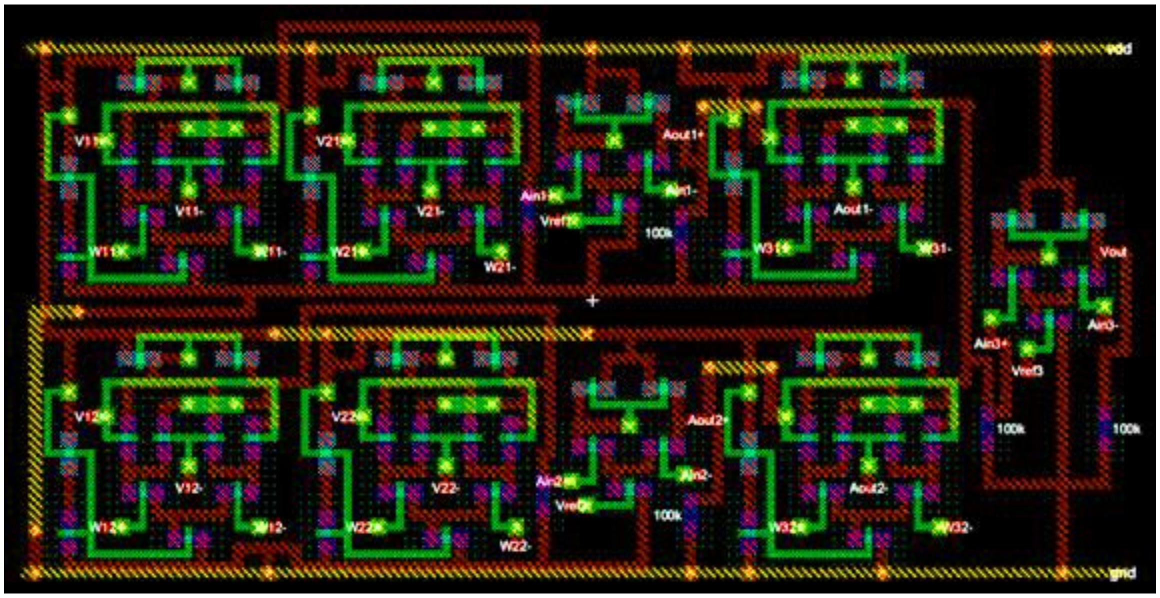


Fig. 5.15 Schematic of the Proposed Multilayer Network in Cadence OrCAD

The two synapse elements on the extreme left (top and bottom) are encoded as input X1 and the two elements adjacent to them as class X2. Fig. 5.16 shows the schematic and layout of the network designed using Electric VLSI Design software. The same schematic is shown in appendix C as three separated parts rotated horizontally for more clarity. For all transistors, the aspect ratio is kept constant to 1.5 (width of 3 microns and length of 2 microns) with operation in the sub-threshold region to ensure consistency with the designs done in the previous sections. The layout for the overall network is found to occupy an area of 0.065 sq. mm.



(a)

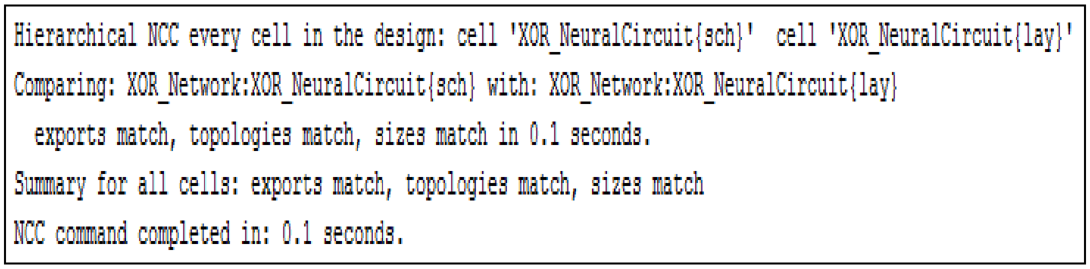


(b)

Fig. 5.16 (a) Schematic of Multilayer Neural Network to solve the XOR problem using Electric VLSI Design software

(b) Layout of Multilayer Neural Network to solve the XOR problem using Electric VLSI Design software

A final NCC is performed to verify the equivalency of schematic and layout and also to ensure that all design constraints are being satisfied, with a potential scope towards fabrication. Fig. 5.17 depicts the results of this check.



```
Hierarchical NCC every cell in the design: cell 'XOR_NeuralCircuit{sch}'  cell 'XOR_NeuralCircuit{lay}'
Comparing: XOR_Network:XOR_NeuralCircuit{sch} with: XOR_Network:XOR_NeuralCircuit{lay}
  exports match, topologies match, sizes match in 0.1 seconds.
Summary for all cells: exports match, topologies match, sizes match
NCC command completed in: 0.1 seconds.
```

Fig. 5.17 Network Consistency Check Result for the Multilayer Neural Network

Chapter 6

Results and Discussions

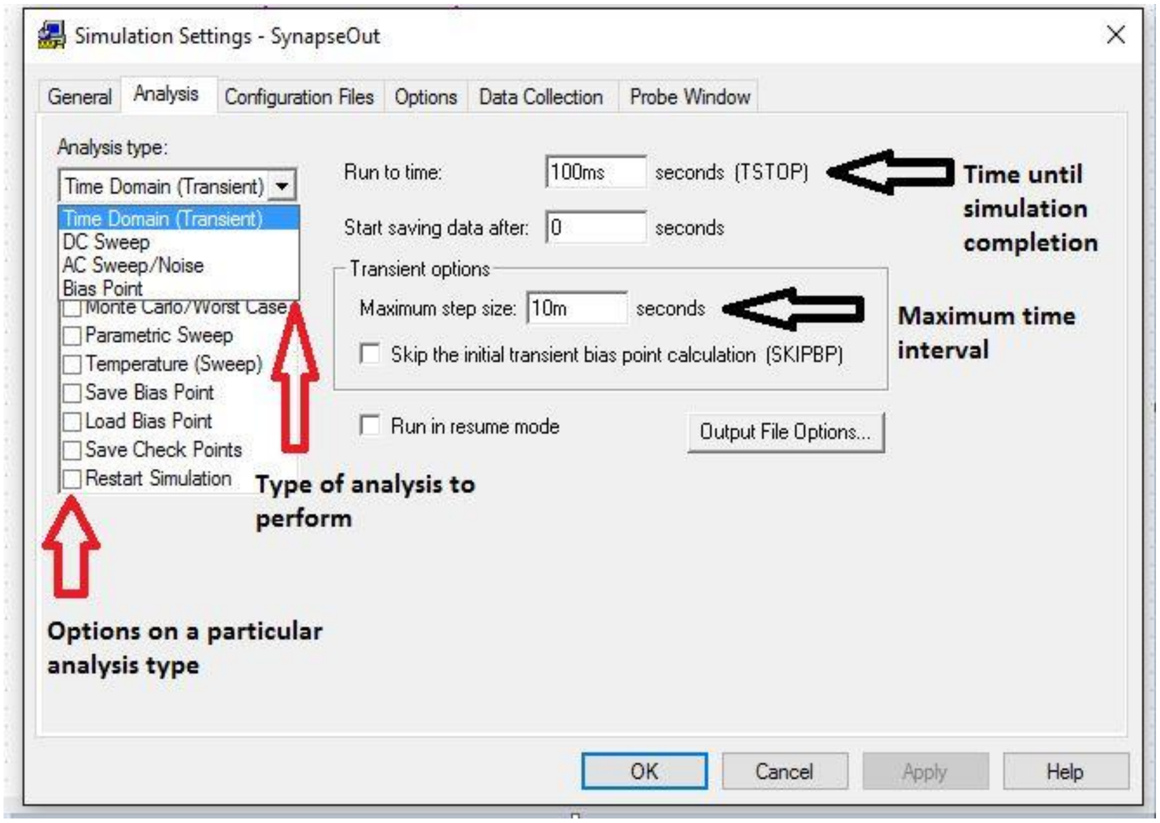
This chapter outlines the results obtained on simulating the designs implemented in the previous chapter. The chapter begins by displaying the simulation results of the synapse and the activation function circuit respectively. The subsequent section depicts the results of instantaneous power dissipation from the synapse element, activation function element, single layer circuit and the multilayer network designed. The final section shows the results obtained on training the multilayer network for XOR classification using MATLAB.

6.1 Simulation Results of Synapse and Activation Function Circuit

This section shows the results obtained on simulating the synapse and activation function circuit using Cadence OrCAD PSpice A/D in conjunction with Cadence OrCAD Capture. Firstly, the simulation settings for both the circuits are shown, and then the results are outlined. In OrCAD design, it is important to adjust the design settings in the simulation profile, prior to running PSpice simulations. The profile gives a general idea about the various types of analysis that can be conducted and the parametric values on basis of which the simulation is run. This enables the software to create a netlist header structure for the schematic at hand after checking for design errors, if any. If the existing design has errors, netlist is not generated and the errors are outlined in a separate simulation output file.

6.1.1 Synapse Circuit Simulation

Fig. 6.1 shows the simulation profile setup for the synapse circuit. This profile is common to all circuit schematics designed using Cadence OrCAD capture and the user has the flexibility to input parametric values as desired. The main features on the selection of Analysis type have been labelled as shown:


Simulation Settings - SynapseOut

General | Analysis | Configuration Files | Options | Data Collection | Probe Window

Analysis type: Time Domain (Transient)

Run to time: 100ms seconds (TSTOP)

Start saving data after: 0 seconds

Transient options

Maximum step size: 10m seconds

☐ Skip the initial transient bias point calculation (SKIPBP)

☐ Run in resume mode

☐ Monte Carlo/Worst Case
☐ Parametric Sweep
☐ Temperature (Sweep)
☐ Save Bias Point
☐ Load Bias Point
☐ Save Check Points
☐ Restart Simulation

Fig. 6.1 Simulation Profile of the Synapse Circuit using Cadence OrCAD

The analysis type is selected to be Time Domain (transient) to have the simulation run in real time. The simulation is made to run until a stop time of 100 ms with 10 ms time interval. The input voltage applied to the synapse circuit is a sine wave with 50 Hz frequency and 2 V peak to peak. The weights are applied in the form of a sine wave as well, with 100 Hz frequency and 1 V peak to peak.

Fig. 6.2 shows the result obtained on simulating the circuit with above parameters. The input voltage is shown in ‘red’, weight voltage in ‘blue’ and synapse output in ‘green’.

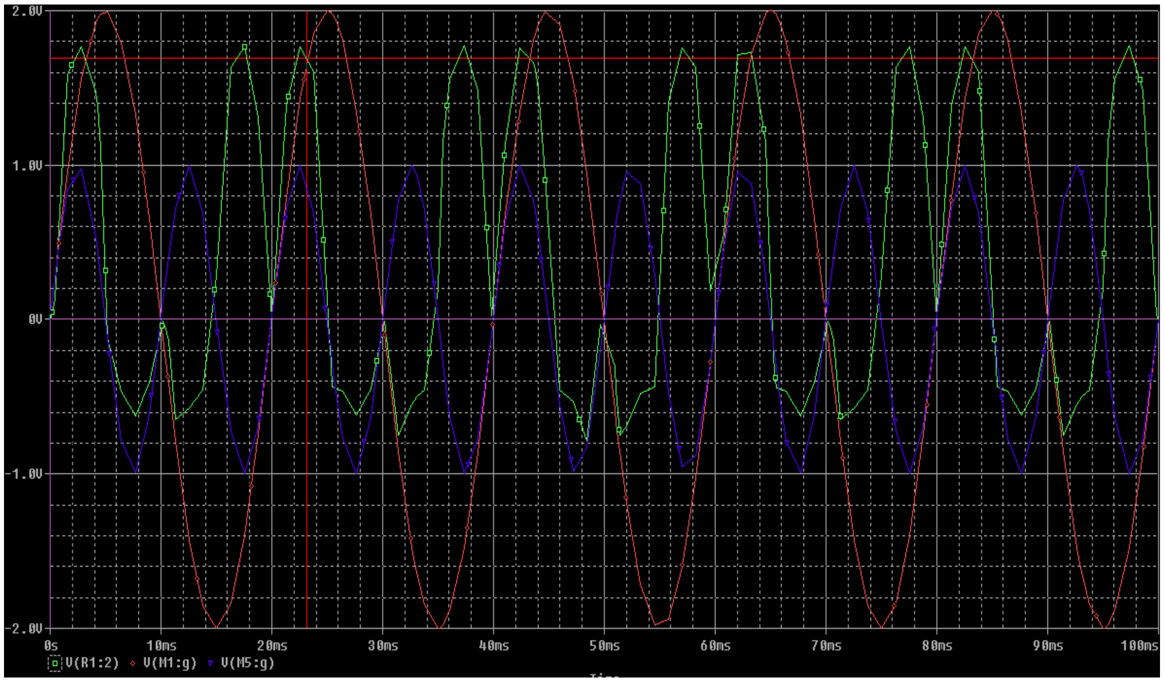
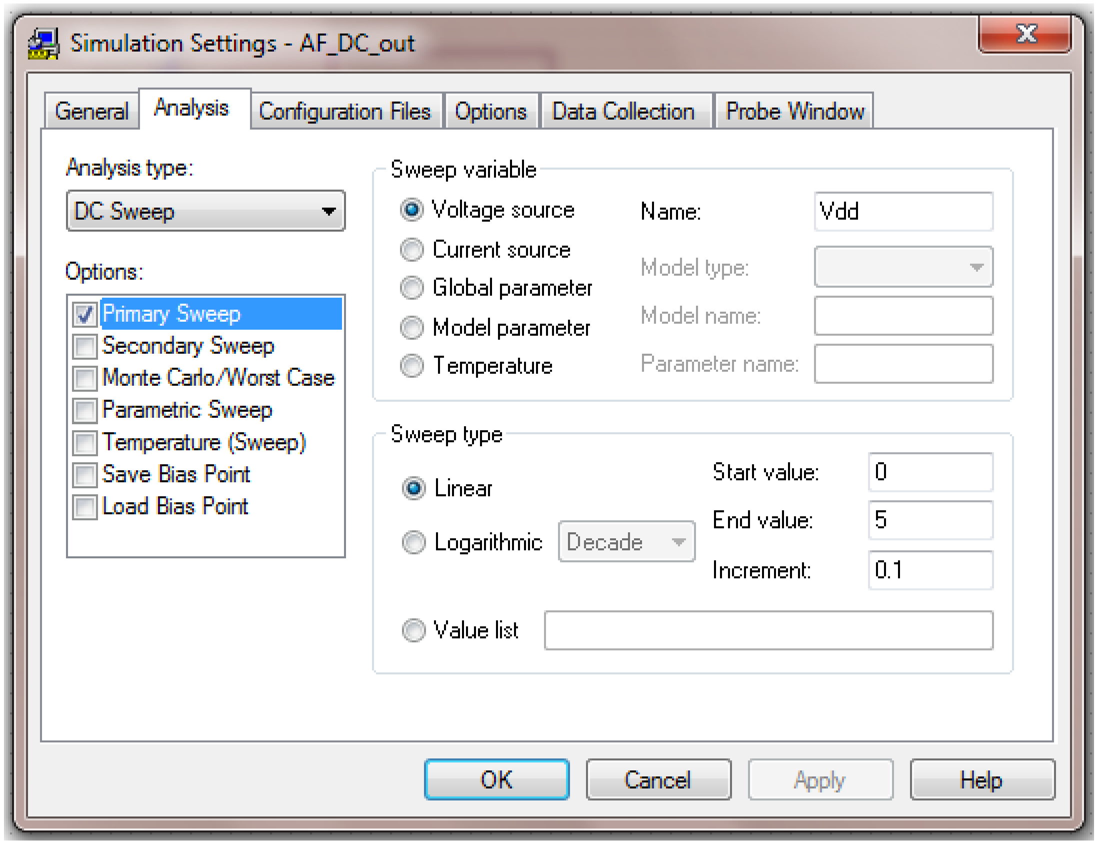


Fig. 6.2 Simulation Result of the Synapse Circuit using PSpice A/D

A four quadrant multiplication of input voltage and weight values is obtained as the synapse output. The output values are not an exact multiplication of inputs and weights, but have a scaling factor associated with them. This is because the output voltage is proportional to the product of input and weight values and that scaling factor approximates proportionality with equality. The output of the synapse element then feeds as an input to the activation function circuit.

6.1.2 Activation Function Circuit Simulation

Fig. 6.3 shows the simulation profile for the activation function circuit. The analysis type selected is DC sweep which is a full rail to rail sweep starting at 0 V and going to 5V.



Analysis type: DC Sweep

Options:
- [✓] Primary Sweep
- [ ] Secondary Sweep
- [ ] Monte Carlo/Worst Case
- [ ] Parametric Sweep
- [ ] Temperature (Sweep)
- [ ] Save Bias Point
- [ ] Load Bias Point

Sweep variable
- [✓ Voltage source] Current source, Global parameter, Model parameter, Temperature
- Name: Vdd
- Model type:
- Model name:
- Parameter name:

Sweep type
- [✓ Linear] Logarithmic (Decade), Value list
- Start value: 0
- End value: 5
- Increment: 0.1

Fig. 6.3 Simulation Profile of the Activation Function Circuit using Cadence OrCAD

The result is a sigmoid (S-shaped) curve with maximum value of approximately 1V. Fig. 6.4 shows the simulation plot of the activation function circuit.

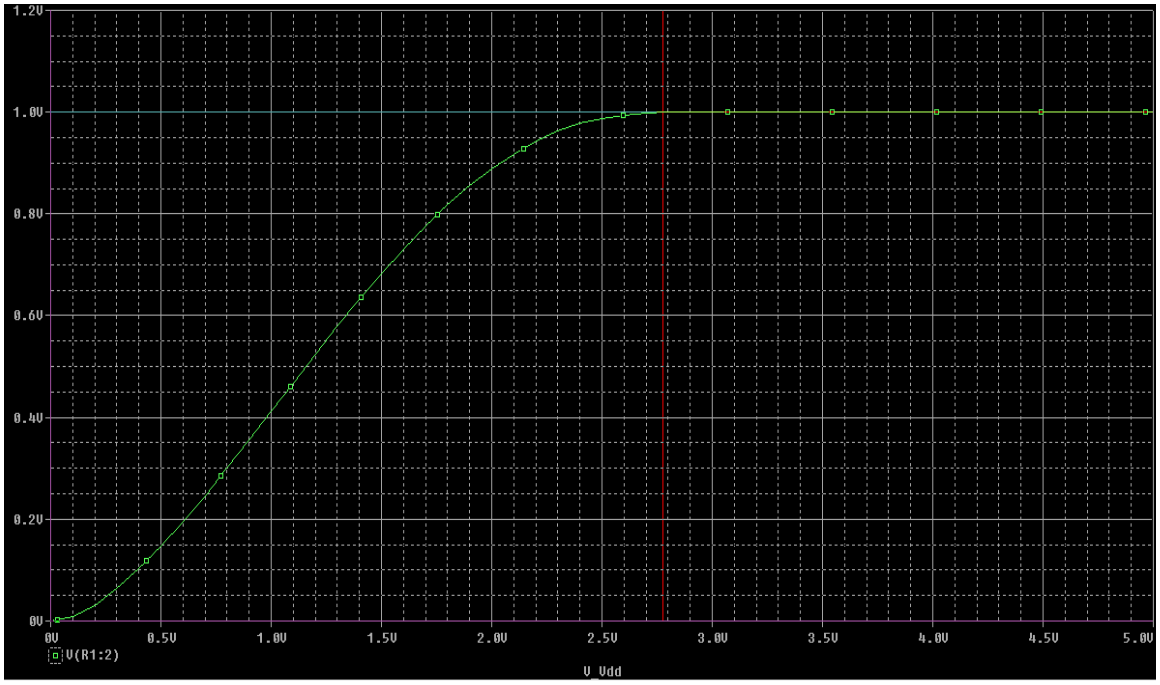


Fig 6.4 DC Sweep Analysis Result of the Activation Function Circuit using PSpice A/D

The activation function circuit shown in fig. 5.4 is then modified to have both positive and negative biasing (2.5V magnitude) and sine wave inputs are given to the gates of transistors M1 and M2. The sine wave has a frequency of 50 Hz with a peak to peak amplitude of 200 mV. Input given to M2 is complementary of that given to M1. A transient analysis is performed with the simulation profile of fig. 6.1 and the result is shown in fig. 6.5 below. It can be seen that differential amplification is provided in response to the applied sine wave input at the same frequency.

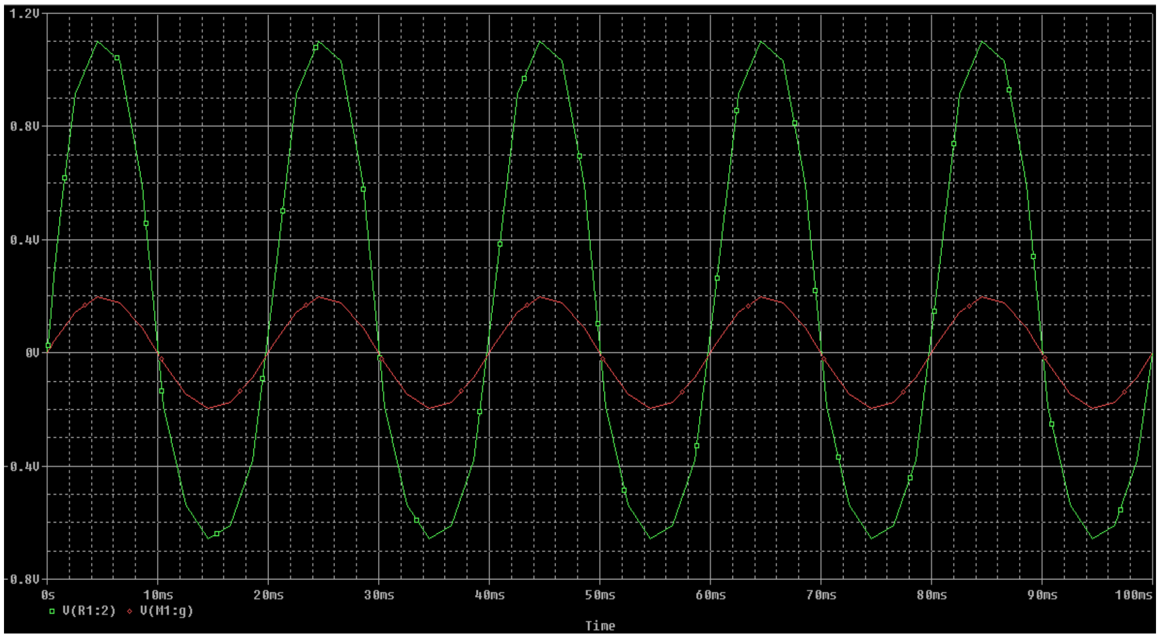


Fig 6.5 Transient Analysis Result of the Activation Function Circuit using PSpice A/D

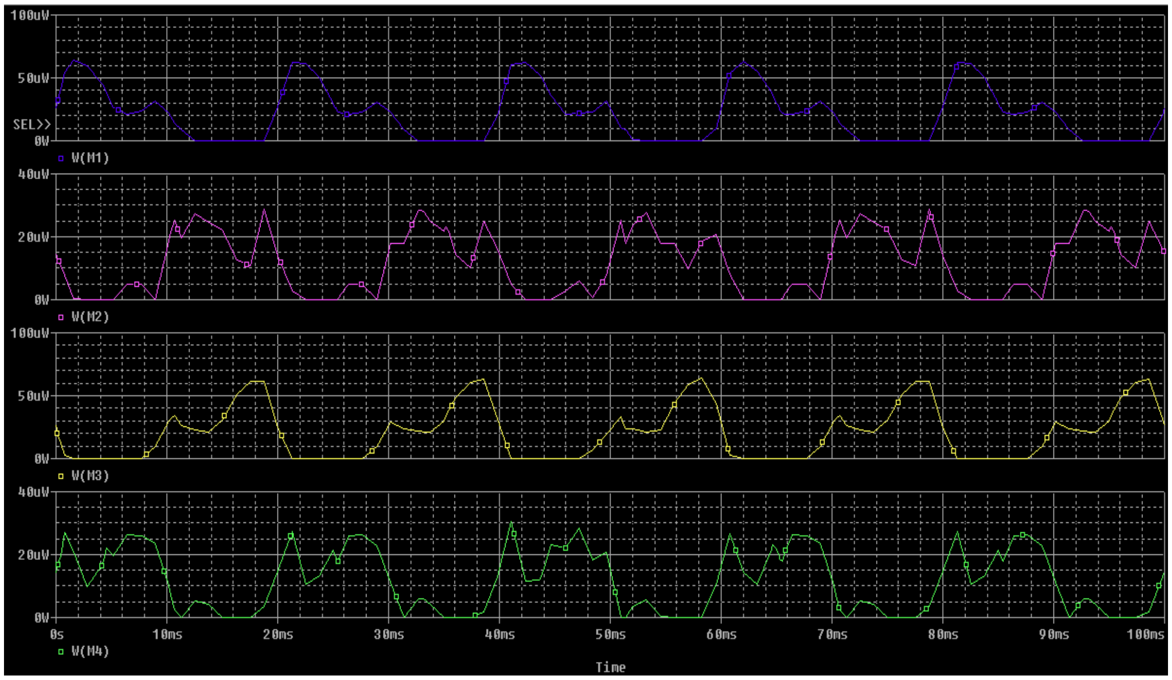
The input voltage is plotted in 'red' and the circuit output in 'green'

6.2 Results for Instantaneous Power Dissipation

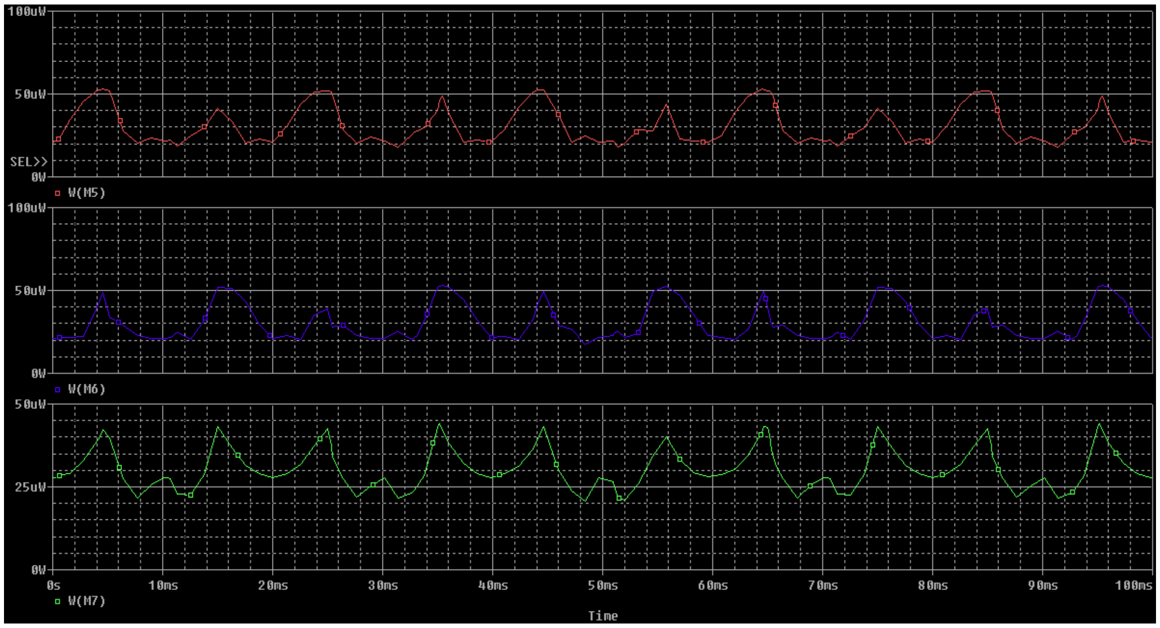
This section demonstrates the results of the instantaneous power dissipation, first from the synapse and the activation function element, and then from the single and multilayer neuron circuit. The analysis is performed specifically at the design level and the requirement is to have as low power dissipation as possible, from each circuit and also the overall network. In the schematic designed for every element using Cadence OrCAD, power probes are connected on the PSpice component whose power dissipation analysis is desired and a transient simulation is run. For the synapse and activation function element, a detailed component by component analysis is done. For the single layer circuit, the primary interest is in knowing the amount of dissipation that occurs when the synapse and the activation function circuit are connected together. With the multilayer circuit, the analysis is performed in a typical layer to layer interconnection scenario to quantify the effects of ‘loading’, when multiple synapse and activation function elements connect together.

6.2.1 Synapse Element

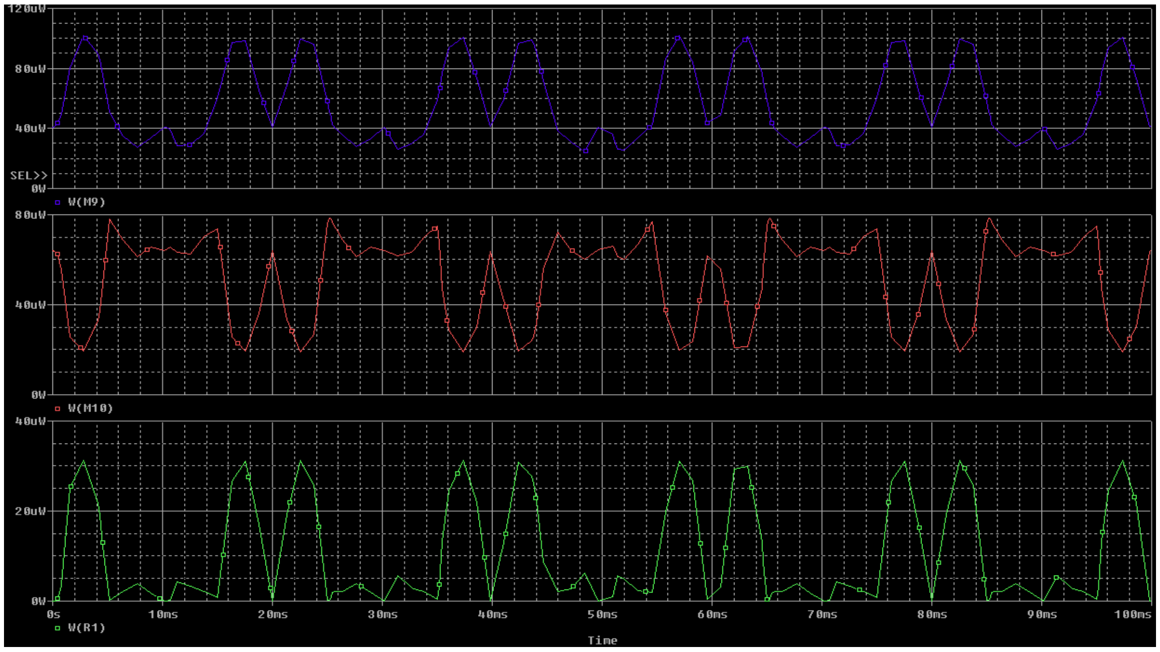
The synapse circuit shown in fig. 5.1 is analyzed for power dissipation by connecting probes on each component in the circuit and a transient simulation is run to view the power waveforms. Fig. 6.6 shows the result of power dissipation from every component in the synapse element.



(a) Waveforms for transistors M1, M2, M3, M4



(b) Waveforms for transistors M5, M6, M7



(c) Waveforms for transistors M9, M10, R1

Fig. 6.6 Instantaneous Power Dissipation Waveforms from Synapse Element using PSpice

Table 6-1 below summarizes the results for minimum and maximum power dissipation from the waveforms shown above.

Table 6-1 Summarized Results of Power Dissipation from Synapse Circuit

Element	**Power Dissipation (uW)**	
	Min.	Max.
M1	0	63.78
M2	0	28.76
M3	0	63.91
M4	0	30.67
M5	17.68	52.82
M6	17.42	52.83
M7	20.68	44
M9	24.65	102.01
M10	19.10	78.35
R1 (Output)	0	31.33

It can be seen that the power dissipation from the synapse element is in the sub-uW range. It is interesting to note that the power dissipation is zero at the output resistor when zero inputs are fed to the circuit.

6.2.2 Activation Function Element

The analysis is now performed on the activation function circuit, similar to that done for the synapse element. The results are as shown in fig. 6.7.

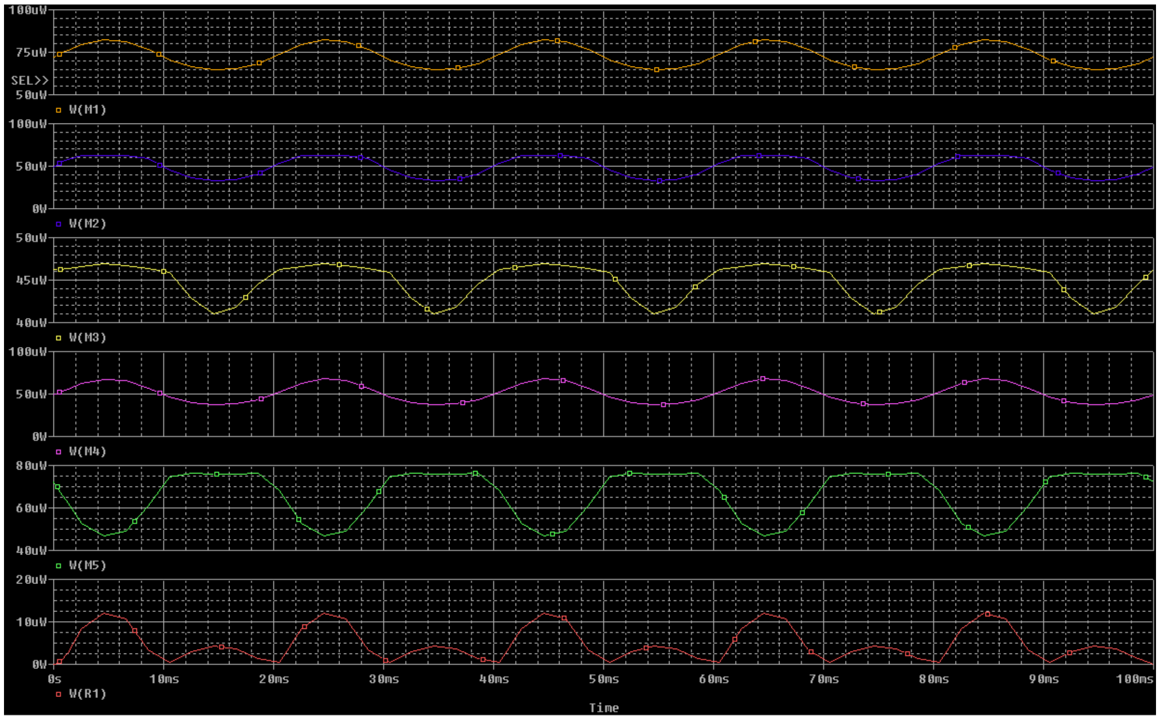


Fig. 6.7 Instantaneous Power Dissipation Result from Activation Function Element using PSpice

Table 6-2 outlines the summary of results for minimum and maximum dissipation based on the above waveforms.

Table 6-2 Summarized Results of Power Dissipation from Activation Function Circuit

Element	**Power Dissipation (uW)**	
	Min.	Max.
M1	64.75	82.31
M2	33.35	62.77
M3	41.06	46.89
M4	37.17	67.61
M5	46.66	76.35
R1 (Output)	0	12.10

It can be seen that the power dissipation for the activation function element is in the range of 33 uW to 82 uW. Similar to the synapse circuit, power dissipation is zero when the input voltage is zero.

6.2.3 Single Layer Neuron Circuit

With a component level analysis done in the previous two sub-sections, this section analyzes the effect of a simple connection of the synapse and activation function element. A wattage probe is connected on the output side of the activation function circuit and a transient simulation is run as before. The result is shown in fig. 6.8 below.

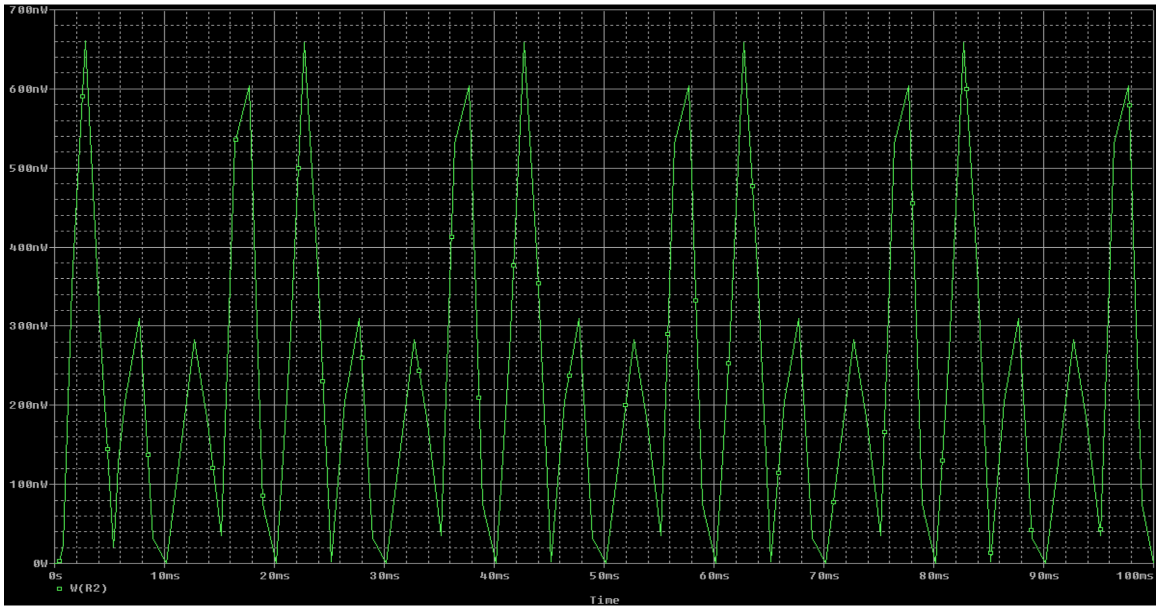


Fig. 6.8 Instantaneous Power Dissipation Result from Single Layer Neuron Circuit using PSpice

6.2.4 Multilayer Neuron Circuit

In this section, the attempt is to analyze the power dissipation when the neural structure under consideration has an interconnection of the synapse and activation function elements. For the multilayer structure shown in fig. 5.15, power dissipation is analyzed, respectively from input to the hidden layer, hidden to the output layer and finally at the overall network output. Fig. 6.9 shows the result.

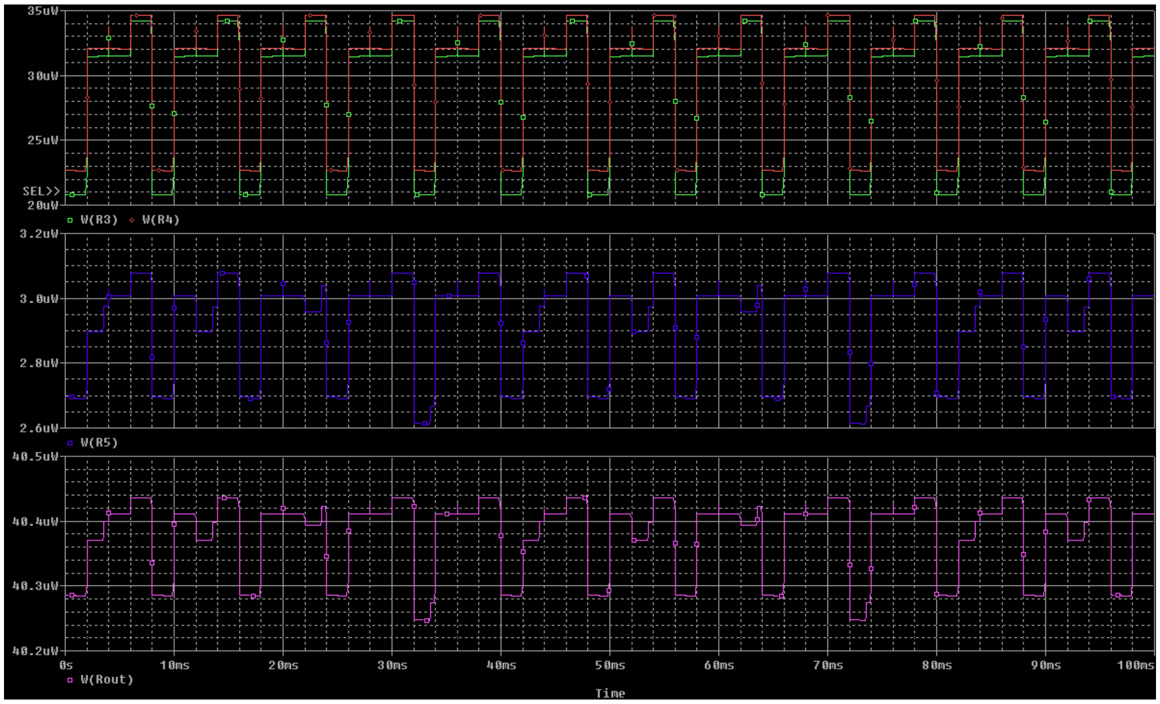


Fig. 6.9 Instantaneous Power Dissipation Result from Multilayer Neuron Circuit using PSpice

Table 6-3 below summarizes the results for minimum, maximum and average power dissipation from the waveforms shown above. It can be seen that the power dissipation values are in the range of 21 uW to 40 uW. Also, there is very little dissipation when transitioning from input layer to the output layer through the hidden layer.

Table 6-3 Summarized Results of Power Dissipation from Multilayer Neuron Circuit

Layer	**Power Dissipation (uW)**	
	Min.	Max.
Input to Hidden	21.70	34.41
Hidden to Output	2.61	3.07
Output	40.24	40.43

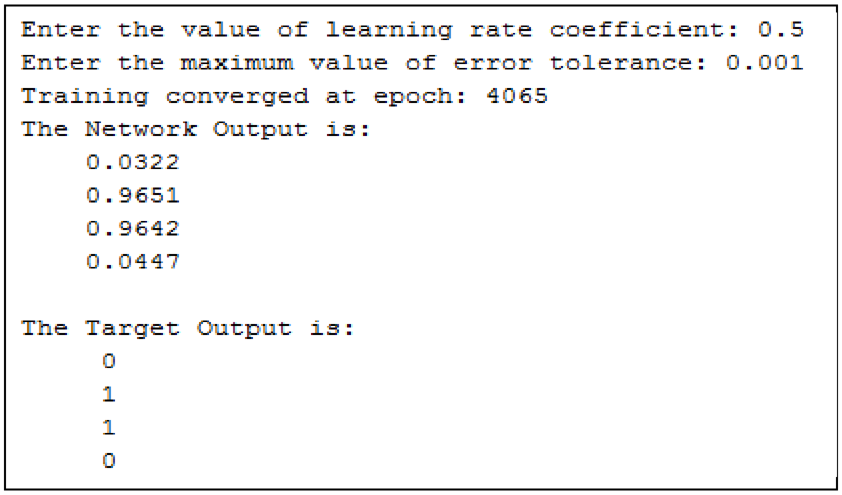
6.3 Results from XOR network training

The multilayer neural network designed in section 5.4 (fig. 5.14) to solve the XOR problem is trained using the back propagation algorithm in MATLAB. The detailed code developed in MATLAB is given in appendix B at the end. A training pattern emulating the XOR truth table is presented to the network at the input and the network is found to correctly generate the output close to the target one, with a small amount of error. Table 6-4 shows a sample test case for training.

Table 6-4 Test Case Parameters for XOR Classification

Parameter	**Value**
Training Input	[0 0; 0 1; 1 0; 1 1]
Target Output	[0; 1; 1; 0]
Learning Rate	0.5
Error Tolerance	<= 0.001

The code runs on this data and the network converges in 4065 epochs. Fig 6.10 shows the result of XOR training based on the input parameters mentioned above.



```
Enter the value of learning rate coefficient: 0.5
Enter the maximum value of error tolerance: 0.001
Training converged at epoch: 4065
The Network Output is:
    0.0322
    0.9651
    0.9642
    0.0447

The Target Output is:
     0
     1
     1
     0
```

Fig. 6.10 Result of XOR training based on the test case from table 6-4

Table 6-5 shows the comparison of network output with the target one, in response to the applied training input pattern. The network output obtained is close to the target one with a certain error.

Table 6-5 Network Output vs Target Output for the XOR Problem

Training Input Pattern		Network Output	Target Output
0	0	0.0322	0
0	1	0.9651	1
1	0	0.9642	1
1	1	0.0447	0

The user has the flexibility to change any or all of these parameters, depending on the problem requirement. Generally, the learning rate is chosen to be between 0.2 and 0.9 and the error tolerance is of the order of 10^{-3}. If the learning rate is taken to be less than or equal to 0.1, an arrangement is made in the code to increase the maximum number of epochs. This is because with lesser learning rate, the network converges slowly and hence requires more epochs.

Further, the network error is recorded for each epoch and after the convergence is achieved, the error is plotted against the number of epochs. This gives the user an understanding of how many iterations it takes for the error to go down to a minimum, given a particular value of learning rate constant and error tolerance. Fig. 6.11 shows the plot of error versus number of epochs based on the result obtained.

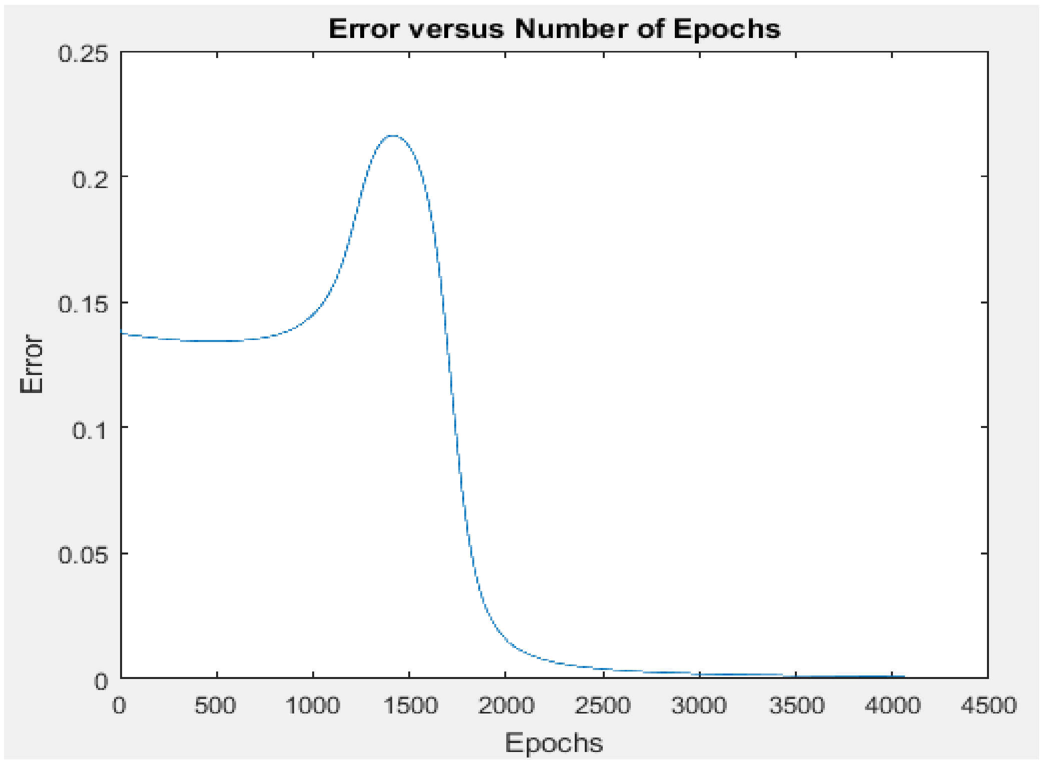


Fig. 6.11 Plot of Network Error versus Number of Epochs

Chapter 7

Conclusion and Future Research Work

7.1 Conclusion

The main objective of the research done in this thesis was to enable the readers to have a general understanding about the fundamentals of artificial neural networks, the way they can be implemented in hardware using CMOS technology and trained using one of the several learning algorithms, commonly used in the field of neural computation. The transistor level implementation of artificial neural networks is primarily governed by proper designing of the synapse and the activation function elements and interconnecting them to form complex neural structures. These structures can then be used to cater, either a specific application or a broad range of applications.

In this thesis, analog implementation of multilayer neural network using CMOS technology was studied, with a view to achieve area and power efficiency. A step by step implementation of a multilayer feedforward neural network was shown. The schematic designs were carried out using Cadence OrCAD Capture and the layouts were done using the Electric VLSI Design System. Subsequently, a backpropagation training algorithm was developed in MATLAB to train the designed network to solve the classical XOR problem. The simulations were done using PSpice A/D to verify the functionality of the individual schematics and also to quantify the results of instantaneous maximum and minimum power dissipation, from every circuit element and also the overall design. For the schematics and layouts done using Electric software, results for the layout vs schematic check were outlined to ensure that all design rules were satisfied. Further, an attempt was made to make the layouts as compact as possible, to make the design area efficient. The multilayer feedforward neural network designed to solve the XOR problem was found to correctly classify sample training input patterns with an error tolerance of 0.1%.

7.2 Future Research Work

The research done in this thesis touched down upon the basics of hardware implementation of an artificial neural network and its training using the backpropagation algorithm. The domain in which the implementation is carried out is governed by the application in which the neural network will be employed. From the implementation standpoint, future research work is envisaged in the area of Mixed Signal (MS) implementation and its eventual System on Chip (SOC) fabrication. Attempts will be made to combine the advantages offered by analog and digital implementations to have a certain level of multi-objective optimization, in terms of area, power and speed of operation. The layouts done using Electric VLSI Design System give rise to the possibility of circuit fabrication, as a separate area of research. In the area of neural network training, future work will involve transistor level implementation of the backpropagation training algorithm using CMOS technology in the analog domain. Another prospect will involve, implementing the neural hardware using Field Programmable Gate Arrays (FPGAs) to have faster convergence speed.

Appendix

A. Circuit Netlists

A.1 Synapse Element:

```
* source SYNAPSECIRCUIT

M_M1        N00237 N00982 N00289 N00289 MbreakN

M_M2        N00249 N00337 N00289 N00289 MbreakN

M_M3        N00237 N00337 N00301 N00301 MbreakN

M_M4        N00249 N00982 N00301 N00301 MbreakN

M_M5        N00289 N001121 N00341 N00341 MbreakN

M_M6        N00301 N001301 N00341 N00341 MbreakN

M_M7        N00341 N00385 N00373 N00373 MbreakN

M_M8        N00365 N00385 N00373 N00373 MbreakN

M_M9        N00237 N00237 N00385 N00385 MbreakP

M_M10         N00249 N00237 N00385 N00385 MbreakP

M_M11         N00365 N00385 N00385 N00385 MbreakP

V_Vdd        N00385 0 2.5Vdc

V_Vss        N00373 0 -2.5Vdc

V_Vin+         N00982 0  AC 1mV

+SIN 0 2V 50Hz 0 0 0
```

```
V_Vin-         0 N00337  AC 1mV

+SIN 0 2V 50Hz 0 0 0

V_Vw-         0 N001301  AC 1mV

+SIN 0 1V 100Hz 0 0 0

V_Vw+         N001121 0  AC 1mV

+SIN 0 1V 100Hz 0 0 0

R_R1         0 N00249  100k TC=0,0
```

A.2 Activation Function Element:

```
* source AF_DC

V_Vref         N04824 0 1Vdc

V_Vdd         N04712 0 5Vdc

M_M1         N04686 N04808 N04698 N04698 MbreakN

M_M5         N04694 N04686 N04712 N04712 MbreakP

M_M3         N04698 N04824 0 0 MbreakN

M_M4         N04686 N04686 N04712 N04712 MbreakP

M_M2         N04694 N04814 N04698 N04698 MbreakN

R_R1         0 N04694  100k TC=0,0

V_V1         N04808 0 DC 2Vdc AC 1mVac

V_V2         0 N04814 DC 2Vdc AC 1mVac
```

A.3 Single Layer Neuron:

```
* source SLNN
M_M11         N03243 N03261 N03261 N03261 MbreakP
R_R1         0 N03135  100k TC=0,0
M_M9         N03099 N03099 N03261 N03261 MbreakP
M_M7         N03221 N03261 N03249 N03249 MbreakN
M_M5         N03171 N029671 N03221 N03221 MbreakN
V_Vin+         N03539 0  AC 1mV
+SIN 0 2V 50Hz 0 0 0
M_M3         N03099 N03215 N03183 N03183 MbreakN
M_M8         N03243 N03261 N03249 N03249 MbreakN
M_M1         N03099 N03539 N03171 N03171 MbreakN
M_M2         N03135 N03215 N03171 N03171 MbreakN
V_Vin-         0 N03215  AC 1mV
+SIN 0 2V 50Hz 0 0 0
V_Vss         N03249 0 -2.5Vdc
V_Vw+         N029671 0  AC 1mV
+SIN 0 1V 100Hz 0 0 0
V_Vw-         0 N029911  AC 1mV
+SIN 0 1V 100Hz 0 0 0
M_M6         N03183 N029911 N03221 N03221 MbreakN
V_Vdd         N03261 0 2.5Vdc
M_M10         N03135 N03099 N03261 N03261 MbreakP
```

```
M_M4         N03135 N03539 N03183 N03183 MbreakN

M_M12         N04018 N03135 N04030 N04030 MbreakN

M_M13         N04198 N04170 N04030 N04030 MbreakN

M_M16         N04198 N04018 N03261 N03261 MbreakP

M_M14         N04030 N04194 N03249 N03249 MbreakN

M_M15         N04018 N04018 N03261 N03261 MbreakP

V_Vref         N04194 0 1.5Vdc

R_R2         0 N04198  100k TC=0,0

E_U1         N04170 0 VALUE {LIMIT(V(0,N04472)*1E6,-15V,+15V)}

R_R3         N04472 N04170  1k TC=0,0

R_R4         N03135 N04472  1k TC=0,0
```

A.4 Multilayer Neuron:

```
source XOR_2

M_M19         N66764 N64064 N64050 N64050 MbreakN

M_M16         N66690 N650060 N66740 N66740 MbreakN

R_R2         0 N66336  100k TC=0,0

M_M45         N69298 N64296 N69408 N69408 MbreakN

M_M44         N68598 N64064 N64064 N64064 MbreakP

M_M7         N66058 N64064 N64050 N64050 MbreakN

M_M23         N67366 N67654 N67474 N67474 MbreakN

V_X11+         N66290 0

+PULSE 100m 500m 4m 1E-12 1E-12 4m 8m
```

```
M_M72         N64538 N64148 N64550 N64550 MbreakN

M_M46         N64148 N69054 N69408 N69408 MbreakN

M_M79         N65442 N65514 N64050 N64050 MbreakN

V_W31+         N669160 0

+PULSE 10m 50m 2m 1E-12 1E-12 2m 10m

M_M77         N65406 N65928 N65442 N65442 MbreakN

R_R1         0 N65928  100k TC=0,0

M_M43         N66336 N68358 N64064 N64064 MbreakP

R_R5         0 N64148  100k TC=0,0

V_W31-         0 N670441

+PULSE 50m 10m 2m 1E-12 1E-12 2m 10m

M_M61         N64020 N670441 N64030 N64030 MbreakN

M_M55         N69474 N64064 N64064 N64064 MbreakP

M_M65         N64148 N64082 N64064 N64064 MbreakP

R_R3         0 N64134  100k TC=0,0

M_M48         N64148 N64296 N69420 N69420 MbreakN

M_M30         N67548 N64064 N64050 N64050 MbreakN

V_W12-         0 N650501

+PULSE 70m 30m 2m 1E-12 1E-12 1m 8m

M_M4         N65928 N66290 N65976 N65976 MbreakN

R_Rout         0 N75391  100k TC=0,0

M_M27         N67474 N657280 N67524 N67524 MbreakN

V_Vdd         N64064 0 2.5Vdc

M_M58         N64082 N64024 N64020 N64020 MbreakN

M_M17         N66702 N650501 N66740 N66740 MbreakN
```

```
M_M39         N68508 N661641 N68574 N68574 MbreakN

V_X22-        0 N65260

+PULSE 500m 100m 2m 1E-12 1E-12 2m 4m

M_M68         N64296 N73722 N64300 N64300 MbreakN

M_M78         N64134 N73483 N65442 N65442 MbreakN

R_R4        0 N64296  100k TC=0,0

M_M40         N68574 N64064 N64050 N64050 MbreakN

M_M63         N64044 N64064 N64050 N64050 MbreakN

V_X21+         N67654 0

+PULSE 100m 500m 2m 1E-12 1E-12 2m 4m

V_W21-         0 N658481

+PULSE 65m 15m 2m 1E-12 1E-12 1m 4m

M_M70         N64264 N64264 N64064 N64064 MbreakP

M_M20         N66618 N66618 N64064 N64064 MbreakP

M_M42         N68358 N68358 N64064 N64064 MbreakP

M_M69         N64300 N64376 N64050 N64050 MbreakN

M_M53         N69298 N69298 N64064 N64064 MbreakP

M_M25         N67366 N65166 N67486 N67486 MbreakN

M_M67         N64264 N66336 N64300 N64300 MbreakN

M_M28         N67486 N658481 N67524 N67524 MbreakN

V_X12+         N66988 0

+PULSE 100m 500m 4m 1E-12 1E-12 4m 8m

M_M80         N65406 N65406 N64064 N64064 MbreakP

M_M18         N66740 N64064 N64050 N64050 MbreakN

V_W11-         0 N648681
```

```
+PULSE 60m 20m 2m 1E-12 1E-12 1m 8m

M_M52         N69474 N64064 N64050 N64050 MbreakN

V_Vref2         N64376 0 1.5Vdc

M_M9         N65824 N65824 N64064 N64064 MbreakP

M_M41         N68598 N64064 N64050 N64050 MbreakN

V_W22+         N660240 0

+PULSE 35m 75m 1m 1E-12 1E-12 1m 4m

M_M66         N64044 N64064 N64064 N64064 MbreakP

M_M38         N68492 N660240 N68574 N68574 MbreakN

V_X12-         0 N64980

+PULSE 500m 100m 4m 1E-12 1E-12 4m 8m

M_M29         N67524 N64064 N64050 N64050 MbreakN

V_Vref1         N65514 0 1.5Vdc

M_M81         N64134 N65406 N64064 N64064 MbreakP

V_Vref3         N65264 0 1.5Vdc

M_M54         N64148 N69298 N64064 N64064 MbreakP

M_M64         N64082 N64082 N64064 N64064 MbreakP

M_M2         N65928 N64798 N65964 N65964 MbreakN

M_M5         N65964 N648240 N66058 N66058 MbreakN

V_W32+         N669640 0

+PULSE 40m 80m 2m 1E-12 1E-12 1.5m 10m

M_M75         N64538 N64538 N64064 N64064 MbreakP

M_M57         N64148 N64024 N69922 N69922 MbreakN

V_Vss         N64050 0 -2.5Vdc

M_M31         N67366 N67366 N64064 N64064 MbreakP
```

```
M_M62         N64030 N64064 N64050 N64050 MbreakN

M_M10         N65928 N65824 N64064 N64064 MbreakP

M_M32         N65928 N67366 N64064 N64064 MbreakP

V_W11+         N648240 0

+PULSE 20m 60m 2m 1E-12 1E-12 1m 8m

M_M76         N75391 N64538 N64064 N64064 MbreakP

M_M33         N67548 N64064 N64064 N64064 MbreakP

M_M60         N69922 N669160 N64030 N64030 MbreakN

M_M35         N66336 N65260 N68492 N68492 MbreakN

M_M59         N64148 N64134 N64020 N64020 MbreakN

M_M21         N66336 N66618 N64064 N64064 MbreakP

M_M50         N69420 N75453 N69256 N69256 MbreakN

M_M34         N68358 N68792 N68492 N68492 MbreakN

M_M51         N69256 N64064 N64050 N64050 MbreakN

M_M8         N66082 N64064 N64050 N64050 MbreakN

M_M11         N66082 N64064 N64064 N64064 MbreakP

M_M15         N66336 N66988 N66702 N66702 MbreakN

M_M12         N66618 N66988 N66690 N66690 MbreakN

M_M1         N65824 N66290 N65964 N65964 MbreakN

M_M56         N64082 N64134 N69922 N69922 MbreakN

M_M49         N69408 N669640 N69256 N69256 MbreakN

V_X21-         0 N65166

+PULSE 500m 100m 2m 1E-12 1E-12 2m 4m

M_M36         N68358 N65260 N68508 N68508 MbreakN

V_W22-         0 N661641
```

```
+PULSE 75m 35m 1m 1E-12 1E-12 1m 4m

M_M71        N64296 N64264 N64064 N64064 MbreakP

M_M3        N65824 N64798 N65976 N65976 MbreakN

M_M73        N75391 N74490 N64550 N64550 MbreakN

V_W21+        N657280 0

+PULSE 15m 65m 2m 1E-12 1E-12 1m 4m

M_M6        N65976 N648681 N66058 N66058 MbreakN

M_M47        N69298 N69054 N69420 N69420 MbreakN

M_M14        N66618 N64980 N66702 N66702 MbreakN

M_M13        N66336 N64980 N66690 N66690 MbreakN

M_M74        N64550 N65264 N64050 N64050 MbreakN

V_X22+        N68792 0

+PULSE 100m 500m 2m 1E-12 1E-12 2m 4m

V_W32-        0 N75453

+PULSE 80m 40m 2m 1E-12 1E-12 1.5m 10m

V_X11-        0 N64798

+PULSE 500m 100m 4m 1E-12 1E-12 4m 8m

M_M26        N65928 N67654 N67486 N67486 MbreakN

M_M37        N66336 N68792 N68508 N68508 MbreakN

M_M22        N66764 N64064 N64064 N64064 MbreakP

M_M24        N65928 N65166 N67474 N67474 MbreakN

V_W12+        N650060 0

+PULSE 30m 70m 2m 1E-12 1E-12 1m 8m

M_M82        N73483 N65928 N64050 N64050 MbreakN

M_M83        N73722 N66336 N64050 N64050 MbreakN
```

M_M84	N69054 N64296 N64050 N64050 MbreakN
M_M85	N74490 N64148 N64050 N64050 MbreakN
M_M86	N64024 N64134 N64050 N64050 MbreakN
M_M87	N73483 N65928 N64064 N64064 MbreakP
M_M88	N64024 N64134 N64064 N64064 MbreakP
M_M89	N69054 N64296 N64064 N64064 MbreakP
M_M90	N74490 N64148 N64064 N64064 MbreakP
M_M91	N73722 N66336 N64064 N64064 MbreakP

B. MATLAB Code for XOR Classification Problem

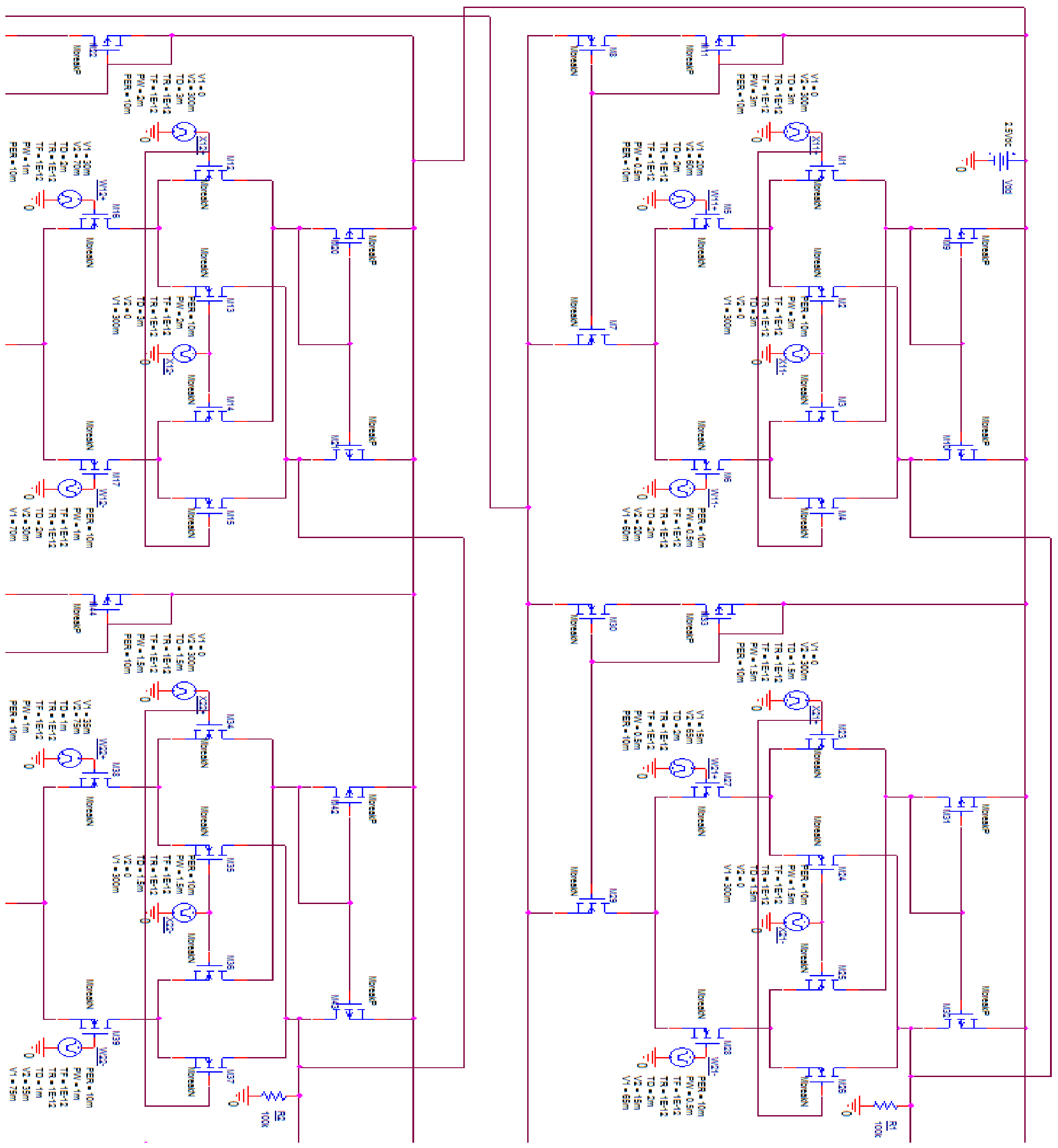
```
% This MATLAB Code shows the backpropagation algorithm acting on the
%proposed   multilayer   feedforward   neural   network   for   iteratively
%minimizing  the  error  between  the  network  output  and  the  target  XOR
response

close all
clearvars
clc

X = [0 0; 0 1; 1 0; 1 1];         % training input pattern

T = [0; 1; 1; 0];                 % target output pattern

% Enter the matrix for storing bias terms

b = input('Enter the bias matrix: ');

% Error condition for mismatching input and target matrices

if size(X) ~= size(T)
```

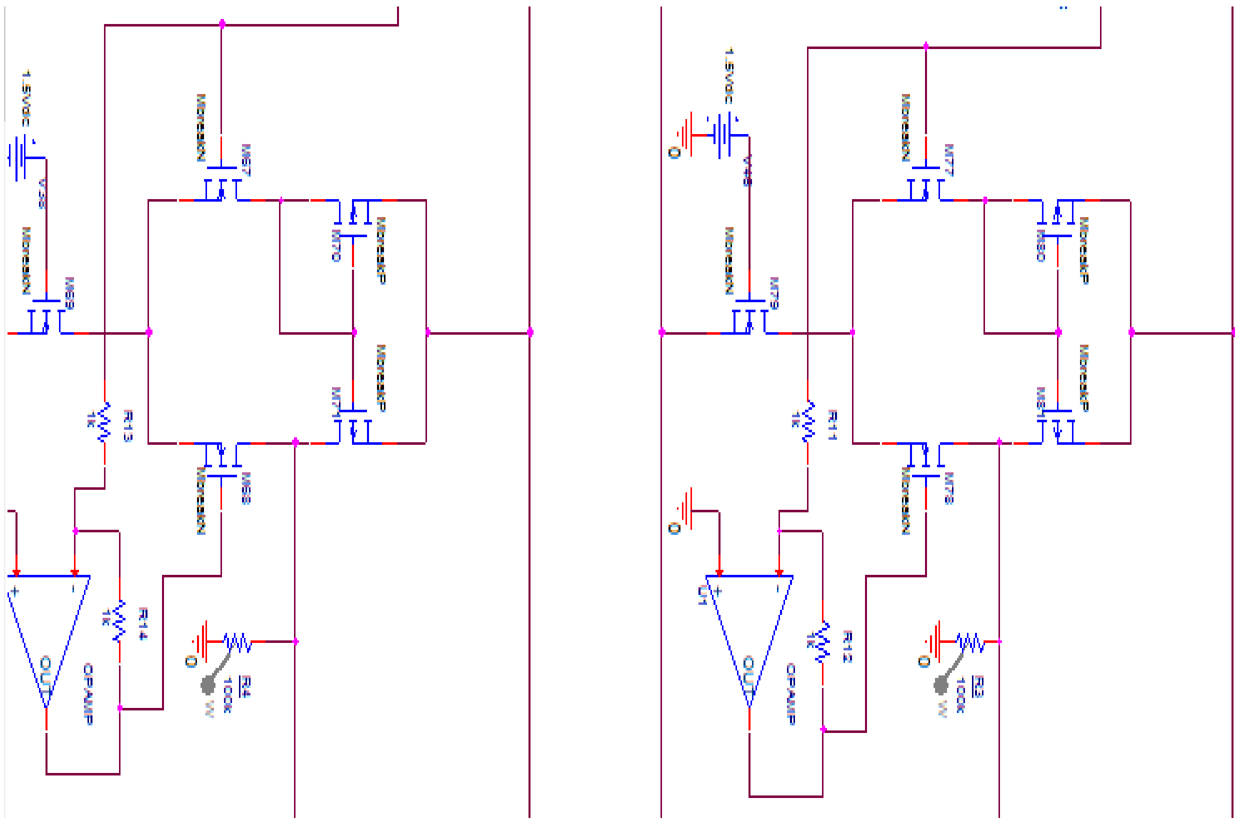
```
disp('***ERROR: Training Input and Target Output matrices do not match in
size***')
return
end

% Enter the learning rate coefficient

alpha = input('Enter the value of learning rate coefficient: ');

% Set the maximum number of epochs

max_epochs = 10000;

% Condition for learning rate values below 0.1

if alpha <= 0.1

max_epochs = max_epochs*10;

end

% Enter the value of error tolerance

error_tolerance = input('Enter the maximum value of error tolerance: ');

numIL = length(X(:,1));          % Calculate the size of input layer

numOL = length(T(:,1));          % Calculate the size of output layer

numHL = 1;                       % Set the number of hidden layer

W = 0.05*rand(3,3);      % Weight matrix initialized to small random values

current_epoch = 0;       % Maintain count for current iteration

% Loop through the maximum iterations

for i = 1:max_epochs

current_epoch = current_epoch + 1;

out = zeros(4,1);    % Output matrix initialized to zero for first pass

% Loop through the input layer

for j = 1:numIL

% Calculate the net sum from the first input to the hidden layer

netH1 = b(1,1).*W(1,1) + X(j,1).*W(1,2) + X(j,2).*W(1,3);

% Calculate the ouput value from the first activation function
% block using the sigmoid function
```

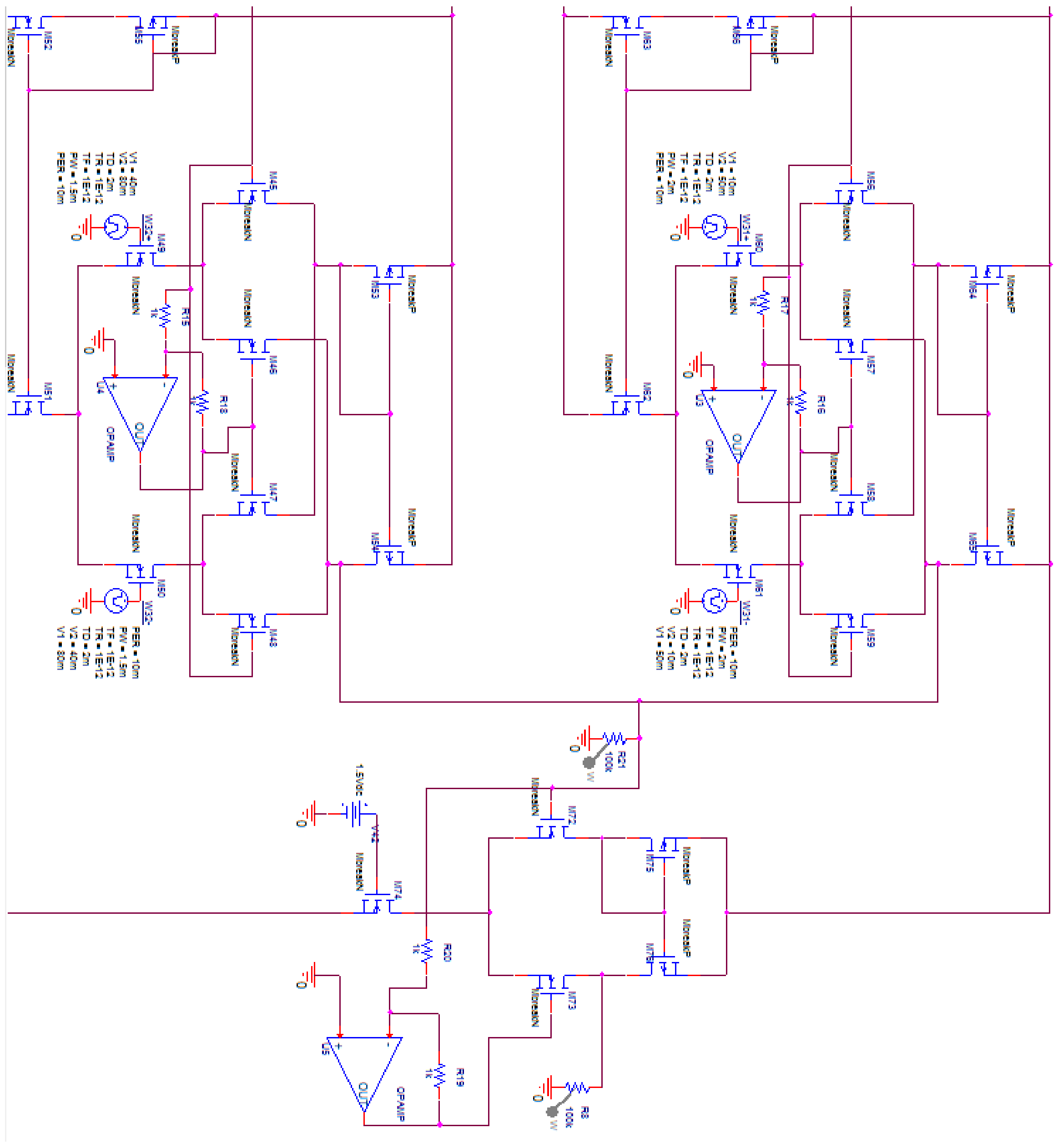
```
out2(1) = 1./(1 + exp(-netH1));

% Calculate the net sum from the second input to the hidden layer

netH2 = b(1,2).*W(2,1) + X(j,1).*W(2,2) + X(j,2).*W(2,3);

% Calculate the ouput value from the second activation function
% block using the sigmoid function

out2(2) = 1./(1 + exp(-netH2));

% Calculate the net sum from the hidden to the output layer

netHO = b(1,3).*W(3,1) + out2(1).*W(3,2) + out2(2).*W(3,3);

% Calculate the overall network output using the sigmoid function

out(j)= 1./(1 + exp(-netHO));

error = T(j) - out(j);                % Calcuate the network error

err(i) = (sum((error).^2)).*0.5;      % Calcualte the MSE

delta = out(j)*(1 - out(j))*error;     % Calculate the output delta error
value

% Progressively compute delta values from the hidden to the output
% layer

del_OH1 = out2(1)*(1-out2(1)).*W(3,2)*delta;

del_OH2 = out2(2)*(1-out2(2)).*W(3,3)*delta;

% Loop through number of output layers minus one

for k = 1:numOL-1

% Condition for updating weights associated with the bias values

if k == 1

W(1,k) = W(1,k) + alpha*b(k,1)*del_OH1;

W(2,k) = W(2,k) + alpha*b(k,2)*del_OH2;

W(3,k) = W(3,k) + alpha*b(k,3)*delta;

else

% Condtion for updating original values of weights using
% learning rate coefficient and previously computed delta values
```

```
W(1,k) = W(1,k) + alpha.*X(j,1)*del_OH1;

W(2,k) = W(2,k) + alpha.*X(j,2)*del_OH2;

W(3,k) = W(3,k) + alpha.*out2(k-1)*delta;

end
end
end

% Condtion for MSE reaching error tolerance value

if err(i) <= error_tolerance

fprintf('Training converged at epoch: %d\n',i);
break
end

end

% Display the values of network and target output after training converges

disp('The Network Output is: ')
disp(out);
disp('The Target Output is: ')
disp(T);

% Plot the chart for Error vs Number of Epochs

figure(2)
plot(err)
xlabel('Epochs')
ylabel('Error')
title('Error versus Number of Epochs')
```

C. Multilayer Neural Network Design using Cadence OrCAD for XOR Classification




1.5Vdc
V38
MbreakN
M67
M68
M69
M70
M71
MbreakP
R13
1k
R14
1k
R4
100k
OPAMP
OUT
0
1.5Vdc
V48
M77
M78
M79
M80
M81
R11
1k
R12
1k
R3
100k
U1
OPAMP
OUT
0



References:

[1] A. Arun, H. Srivastava, S. Mada and M. B. Srinivas, "Analog VLSI Design of Neural Network Architecture for Implementation of Forward Only Computation", Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics (PRIMEASIA), 2012

[2] J. R. Shinde and S. Salankar, "Multi-objective Optimization for VLSI Implementation of Artificial Neural Network", Current Trends in Technology and Science, ISSN 2279-0535, Volume 04, Issue 03, Apr-May 2015

[3] J. Dugger, "Adaptive Analog VLSI Signal Processing and Neural Networks", Thesis submitted to the School of Electrical and Computer Engineering, Georgia Institute of Technology, November 2003

[4] N. Chasta, S. Chouhan and Y. Kumar, "Analog VLSI Implementation of Neural Network Architecture for Signal Processing", International Journal of VLSI design & Communication Systems (VLSICS) Vol.3, No.2, April 2012

[5] S. Hayman, "THE McCULLOCH-PITTS MODEL", International Joint Conference on Neural Networks, 1999

[6] L. Zhang and B. Zhang, "A Geometrical Representation of McCulloch–Pitts Neural Model and Its Applications", IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 10, NO. 4, JULY 1999

[7] “Rosenblatt’s Perceptron”, Catalogue, Pearson Education Canada, Oct 2008, Last accessed May 31, 2017

[8] O. G. Selfridge, Pandemonium: A Paradigm for Learning, Mechanization of Thought Processes: Proceedings of the Symposium Held at the National Physical Laboratory, November 1958, London: HMSO, pp. 513-526

[9] B. Widrow and M. A. Lehr, "Adaptive Neural Networks and their Applications", Information Systems Laboratory, Dept. of Electrical Engineering, Stanford University, California, 1993

[10] M. Minsky and S. Papert, "Perceptrons: An Introduction to Computational Geometry", 1969

[11] P. J. Werbos, "Backpropagation: Past and Future", IEEE International Conference on Neural Networks, 1988

[12] D. B. Parker, "A Comparison of Algorithms for Neuron like Cells", Proc. Second Annual Conference on Neural Networks for Computing, Proceedings Vol. 151, 327-332,
American Institute of Physics, New York, 1987

[13] A. Grubl, "VLSI Implementation of a Spiking Neural Network", Dissertation presented to the Joint Faculties for Natural Sciences and Mathematics, Ruprecht-Karls-Universitat, Heidelberg, Germany, July 2007

[14] M. A. AboEl-soud, R. A. AbdelRassoul, H. H. Soliman and L. M. El-ghanam, "Low-Power CMOS Circuits for Analog VLSI Programmable Neural Networks, Proceedings of the 15th International Conference on Microelectronics, 2003

[15] M. Valle, D.D. Caviglia, and G.M. Bisio, "Design of a CMOS ASIC Chip Featuring Analog Neural Computational Primitives", Procedings of EURO ASIC, 1992

[16] D. Kibler and P. Langley, "Machine learning as an Experimental Science", Proceedings of the Third European Working Session on Learning, London, UK, 1988

[17] K. Du and M. Swamy, "Neural Networks and Statistical Learning", DOI: 10.1007/978-1-4471-5571-3, October 2013

[18] H. B. Barlow, "Unsupervised Learning", Neural Computation, Volume: 1, Issue: 3, September 1989

[19] P. Hasler, C. Diorio, B. A. Minch and C. Mead, "Single Transistor Learning Synapse with Long Term Storage", International Symposium on Circuits and Systems (ISCAS), 1995

[20] P. Patel, "Realization of Logic Gates using CMOS Gilbert Mulitplier Cell", 3rd Internation Advance Computing Conference (IACC), 2013

[21] B. D. Yammenavar, V. R. Gurunaik, R. N. Bevinagidad and V. U. Gandage, "DESIGN AND ANALOG VLSI IMPLEMENTATION OF ARTIFICIAL NEURAL NETWORK", International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.3, July 2011

[22] K. Leboeuf, A. Namin, R. Muscedere, H. Wu and M. Ahmadi, "High Speed VLSI Implementation of the Hyperbolic Tangent Sigmoid Function", Proceedings of 3rd International Conference on Convergence of Hybrid Information Technology, Nov 2008

[23] B. Zamanlooy and M. Mirhassani, "Efficient VLSI Implementation of Neural Networks with Hyperbolic Tangent Activation Function", IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 22, NO. 1, January 2014

[24] S. Liu and R. Mockel, "Temporally learning floating-gate VLSI synapses", IEEE International Symposium on Circuits and Systems, 2008

[25] S. Ramakrishnan, P. E. Hasler and C. Gordon, "Floating Gate Synapses with Spike-Time-Dependent Plasticity", IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS, Vol. 5, No. 3, JUNE 2011

[26] P. Hasler, B. A. Minch and C. Diorio, "An Auto-zeroing Floating Gate Amplifier", IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol 48, Issue 1, 2001

[27] L. Chang, R. K. Montoye, Y. Nakamura, K. A. Batson, R. J. Eickemeyer, R. H. Dennard, W. Haensch and D. Jamsek, "An 8T-SRAM for Variability Tolerance and Low-Voltage Operation in

High-Performance Caches", IEEE JOURNAL OF SOLID-STATE CIRCUITS, Vol. 43, No. 4, April 2008

[28] V. Sivakumar and M. Malathi, "PROGRAMMABLE SYNAPTIC MEMORY WITH SPIKING NEURAL NETWORK IN VLSI", Information Communication and Embedded Systems (ICICES), International Conference 2014

[29] G. Srinivasan, P. Wijesinghe, S. S. Sarwar, A. Jaiswal, and K. Roy, "Significance Driven Hybrid 8T-6T SRAM for Energy-Efficient Synaptic Storage in Artificial Neural Networks", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016

[30] G. Lecerf, J. Tomas and S. Saighi, "Excitatory and Inhibitory Memristive Synapses for Spiking Neural Networks", IEEE International Symposium on Circuits and Systems (ISCAS), 2013

[31] T. Cabaret, L. Fillaud, B. Jousselme, J. O. Klein and V. Derycke, "Electro-grafted organic memristors: properties and prospects for artificial neural networks based on STDP", Proceedings of the 14th IEEE International Conference on Nanotechnology Toronto, Canada, August 18-21, 2014

[32] H. Kim, M. Sah, C. Yang, T. Roska, and L. O. Chua, "Memristor Bridge Synapses", Proceedings of the IEEE, Vol. 100, No. 6, June 2012

[33] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang and L. O. Chua, "A Circuit-Based Learning Architecture for Multilayer Neural Networks with Memristor Bridge Synapses", IEEE

TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, Vol. 62, No. 1, January 2015

[34] M. M. Gupta, L. Jin and N. Homma, "Static Neural Networks", John Wiley and Sons, ISBN: 0-471-21948-7, 2003

[35] J. M. Keller, "Multilayer Neural Networks and Backpropagation", Journal: Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation, John Wiley & Sons, 2016

[36] An. S. Shafie, I. A. Mohtar, S. Masrom and N. Ahmad, "Backpropagation Neural Network with New Improved Error Function and Activation Function for Classification Problem", IEEE Symposium on Humanities, Science and Engineering Research, 2012

[37] C. Yu, Y. Tang and B. Liu, "An adaptive activation function for multilayer feedforward neural networks", TENCON '02. Proceedings, IEEE Region 10 Conference on Computers, Communications, Control and Power Engineering, Vol. 1, Year 2002

[38] X.G. Wang, Z. Tang, H. Tamura and M. Ishii, "A Modified Error Function for The Backpropagation Algorithm", Neuro-computing, Vol. 57, 2004

[39] M. Lourakis, "A Brief Description of the Levenberg-Marquardt Algorithm Implemented by levmar", Institute of Computer Science Foundation for Research and Technology - Hellas (FORTH), February 11, 2005

[40] M. A. Costa, A. P. Braga, B. Rodrigues de Menezes, "Improved generalization learning with Sliding Mode Control and the Levenberg-Marquadt Algorithm", Proceedings of the VII Brazilian Symposium on Neural Networks (SBRN’02), 2002

[41] M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm", IEEE TRANSACTIONS ON NEURAL NETWORKS, Vol. 5, No. 6, November 1994

[42] P. de Chazal and M. D. McDonnell, "Efficient computation of the Levenberg-Marquardt algorithm for feedforward networks with linear outputs", International Joint Conference on Neural Networks (IJCNN), 2016

[43] R. Cancelliere, "Feedforward Neural Networks", Dept. of Computer Science, University of Turin, Italy

[44] J. Elder, "Multilayer Perceptrons", Computer Science Engineering (CSE) 4404/5327 Introduction to Machine Learning and Pattern Recognition, November 2012