7-31-2023

Design of a Current Mode Circuit for Zeroth-Order Optimization of Memristor-Based Neuromorphic Systems

Shery George Mathews
sgm3401@rit.edu

Design of a Current Mode Circuit for Zeroth-Order Optimization of Memristor-Based Neuromorphic Systems

Shery George Mathews

Design of a Current Mode Circuit for Zeroth-Order Optimization of Memristor-Based Neuromorphic Systems

Shery George Mathews

July 31, 2023

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

RIT | Kate Gleason College of Engineering

Department of Computer Engineering

Design of a Current Mode Circuit for Zeroth-Order Optimization of Memristor-Based Neuromorphic Systems

Shery George Mathews

Committee Approval:

Dr. Cory E. Merkel (Advisor) Date
Assistant Professor, Department of Computer Engineering

Dr. Marcin Lukowiak Date
Professor, Department of Computer Engineering

Dr. Hussin Ketout Date
Lecturer, Department of Computer Engineering

Acknowledgments

This work has reached this outcome only because of the countless help and feedback from my faculty advisor Dr. Cory Merkel. I would like to thank my graduate advisor Dr. Cory Merkel for all the help, guidance, support, and motivation offered to me that helped this work reach this far. I am also grateful to all my friends, relatives, co-workers, and RIT computer engineering faculty members for all the support and resources provided to me.

Dedication

To my friends, family, relatives, co-workers, and teachers, for all your love, motivation, feedback, and support. I am expressing my sincere gratitude to my parents: Mathews George Kattunilathu and Sosamma Mathews, my sister, Sherin Mathews, and my wife Sandra Mathews for all the support they provided to me who kept me going forward to finishing this research work.

Abstract

Artificial neural networks (ANNs) have become widely used over the past decade due to their excellent performance in applications such as object recognition and classification, time series forecasting, machine translation, and more. Key factors contributing to the success of ANNs in these domains include the availability of large datasets and constant improvements in powerful parallel processing architectures such as graphics processing units (GPUs). However, modern ANN models' growing size and complexity preclude their deployment on resource-constrained hardware for edge applications. Neuromorphic computing seeks to address this challenge by using brain-inspired principles to co-design the ANN hardware and algorithms for improved size, weight, and power (SWaP) consumption. Unfortunately, though, standard gradient-based ANN training algorithms like backpropagation incur a large overhead when they are implemented in neuromorphic hardware. In addition, the gradient calculations required for the backpropagation algorithm are especially challenging to implement in neuromorphic hardware, which typically has low precision, functional discontinuities, non-linear weights, and several other non-ideal behaviors.

This thesis proposes a novel circuit design for training memristor-based neuromorphic systems using the weighted sum simultaneous perturbation algorithm (WSSPA). WSSPA is a zeroth-order ANN training algorithm that estimates loss gradients using perturbations of neuron inputs, avoiding large-overhead circuitry needed for backpropagation. The training circuit optimizes the ANN by adjusting the conductance of the memristor-based weights using the loss gradient estimates. Current mode design techniques, including the translinear

principle, are used to significantly reduce the number of transistors required for existing implementations, thereby improving the SWaP efficiency. The proposed circuit consists of a sample and hold circuit, an error calculation circuit, a memristor threshold voltage selector, and synapse input selector circuits. The design was implemented in LTSpice, using 45 nm predictive technology MOSFET models and a simple linear memristor model fit to published experimental data. Demonstration of the proposed hardware was carried out by training an ANN to perform Boolean logic functions such as XOR.

Results show that the current mode circuit design of WSSPA is able to converge on the correct functionality within 16 training iterations. After this, the ANN output current oscillates around the target. Areas for future work include making the feedback adjustment voltage value dependent on the error magnitude, rather than just the sign. This will help the error to converge to 0 after training, eliminating output oscillations. In addition, future studies should confirm the scalability of the proposed design for larger neural networks.

Contents

List of Figures

Chapter 1

Introduction

The concept of neuromorphic computing was developed by Carver Mead in 1980 [29]. It involved the usage of VLSI systems containing analog circuits, mimicking neurobiological architectures present in the nervous system [24]. Neuromorphic computing architecture is inspired by the functioning of biological neural networks such as the human brain. The human brain is very efficient and does not transfer data between a control unit and a memory unit as it does not have separate memory and control unit blocks [24]. The key question behind the inspiration of thoughts that led to neuromorphic computing is "why can't we have the memory and processor in the same place as the human brain works?" [24]. The human brain is significantly better in energy consumption than any other existing computer architecture. Human brains can solve problems like face recognition and situation analysis in a fraction of a second where it would take countless hours to train a computer to do so [24]. This is because of the different architecture that a human brain has compared to a computer. The basic working unit in the human brain is called a neuron. Neurons are cells in the nervous system that transmit information to other nerve cells. There are small spaces between the neurons called synapses that can pass messages between neurons. There may be thousands of synapses in a single neuron. An attempt to create an architecture for computational purposes that models biological neural network architecture resulted in artificial neural networks and different training algorithms that make neural networks adapt and learn.

1.1 Biological Neural Networks

The modeling of neural networks from animal and human brains has largely influenced the concept of neural networks in computing [24]. Computational principles used by the nervous system are key ideas to adhere to neuro-biological architectures. Individual neurons embody the following principles: transmitting information by frequency of spikes, integration of simultaneous information, stability to errors, and memory preservation [24]. A single neuron can be connected to an extensive number of neurons in the network. These connections, synapses, usually consist of axons to dendrites that handle neurotransmitter signaling, along with other types of signaling. Neurons can "hear" all connections although only a small portion of them are active through these spikes. Not only does the strength of a connection matter, but also the location of these connections. Spiking neurons are gauged as the fundamental processing unit of the central nervous system [24]. Spiking neural networks (SNN) can model the central nervous system of biological organisms and are used to study neural circuits. SNNs can be described by the integrate-and-fire model investigated by Louis Lapicque [30].

$$I(t) = C\,\frac{dV(t)}{dt} \tag{1.1}$$

Equation (1.1) simply refers to the time derivative law of capacitance. Note this does not consider a refractory period which would limit the firing rate of the neurons. The leaky integrate-and-fire model considers a leak in the diffusion of ions through the membrane.

$$C_m\left(\frac{dV_m(t)}{dt}\right) = I(t) - V_m\,\frac{(t)}{R_m} \tag{1.2}$$

Vm and Rm are the voltage across the cell membrane and membrane resistance, respectively. The adaptive integrate-and-fire model accounts for different firing patterns from bursting and adaptation.

$$\tau_m \left(\frac{dV_m(t)}{dt}\right) = RI(t) - [V_m(t) - E_m] - R\sum_k w_k \quad \textbf{(1.3)}$$

τm is the membrane time constant, wk is the adaptation current number, with index k, Tk is the time constant of adaptation current wk, and Em is the resting potential. These various neuron models explain neurological firing patterns, acting as a transparent framework to describe important considerations when modeling other types of neural networks.

1.2 Artificial Neuron Model

Artificial neural networks (ANNs) loosely model the neurons in biological brains, essentially acting as an interconnected group of nodes. Similarly, to the SNN model, artificial neurons receive a signal, process it, and signal neurons are connected to it. However, each connection is assigned a weight that represents some degree of significance. Figure 1.1 shows the diagram of an artificial neuron model. Taking inputs x_1, x_2, x_3, ..., x_n with an associated weight, gives a summed result which is passed through an activation function. This activation function may depend on the type of artificial neural network that is being made. These artificial neurons are organized into multiple layers (MNNs), so that neurons only connect to other neurons in the layer before and after the current layer. This consists of a quintessential deep network, and multilayered input layer, output layer, and hidden layer. MNNs can realize large scale learning

tasks effectively when large amounts of computational power are available. Here, MNN can classify linearly invisible data. This concept of having multiple hidden processing layers is utilized heavily in deep learning.

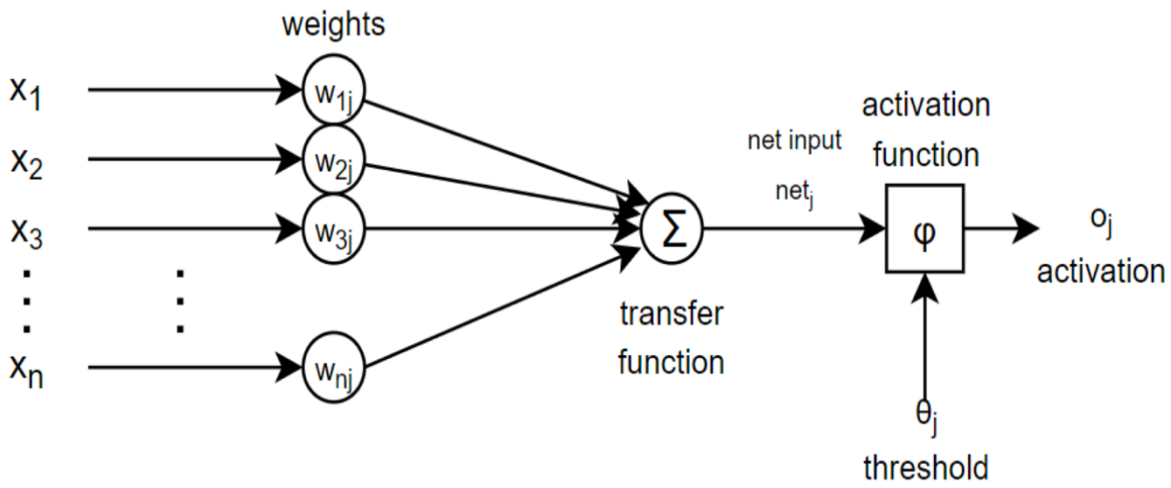


Figure 1.1: Artificial Neuron Model

By considering sample observations, learning can adapt the network to minimize errors, handling tasks better. This involved adjusting the weights and thresholds between groups of artificial neurons and layers. Biological neural networks (BNNs) basic building block is the biological neuron compared to the artificial neuron in artificial neural networks. BNNs are slower in processing information and can allocate dynamic storage, adjusting interconnection strengths [28]. An advantage of ANNs is that they can learn irrelevant of the type of linear or non-linear data. However, it can be difficult to explain the behavior of the network. On the other hand, BNNs can process very complex parallel inputs.

Previously mentioned MNNs may be able to satisfy some of the ANNs drawbacks. These

quintessential deep networks have an advantage of being better than the single-layer perceptron overcomes the weaknesses that the perceptron cannot classify linearly indivisible data [27]. To realize large scale learning tasks, MNNs can perform impressively well and produce state-of-the-art results when massive computational power is available [27].

A neural network needs to be trained to perform certain tasks. An untrained neural network is like a newborn baby that is unaware about the outside world. A baby learns about the world from experiences and directions it gets as it grows. Similarly, Neural networks need guidance to make them aware about how to respond to different inputs. Common training techniques such as backpropagation to train a neural network includes complex computations to calculate loss functions which includes calculating derivatives. When implementing these techniques in hardware, the design complexity gets higher. This thesis novel proposes a circuit design for training an analog neural network by utilizing Zeroth order optimization which in turn reduces the hardware complexity for the training circuit as opposed to the complex derivative calculations involved in training algorithms such as back propagation.

1.3 Summary

Through inspiration from modelling biological neural networks such as the brain, neural networks are applied to wide scale applications ranging from financial market predictions to artificial intelligence, machine learning, and signal processing. Three main characteristic features of a human brain are low power consumption, fault tolerance, and lack of need to be programmed. These ideas led to the development of the concept of neuromorphic computing.

Artificial neurons in an artificial neural network are modeled after biological neurons in a biological neural network such as an animal brain. Each neuron can be connected to many other neurons in the previous and next layer to form a neural network. Each neuron in a given layer of the neural network accepts inputs from the previous layer and creates a weighted sum. This weighted sum will be passed through an activation function to create the neuron output. The output is then passed as input to the neurons in the next layer. The weights associated with each input to a neuron determine the importance of each incoming input to a neuron. The outputs of each neuron depend on its inputs, bias, weights, and the different functions used for activation and propagation. This thesis novel proposes a circuit design for training an analog neural network by utilizing zeroth order optimization which in turn reduces the hardware complexity for the training circuit as opposed to the complex derivative calculations involved in training algorithms such as back propagation.

Chapter 2

Background and Related Work

Artificial neurons in an artificial neural network are modeled after biological neurons in a biological neural network. Each neuron in each layer of the neural network accepts inputs from the previous layer and creates a weighted sum. This weighted sum will be passed through an activation function to create the neuron output. The output is then passed as input to the neurons in the next layer. The weights associated with each input to a neuron determine the importance of each incoming input to a neuron. Different propagation and activation functions are used to train a neural network to do a certain task such as image processing, text recognition, and classification. A propagation function uses a set of incoming inputs to generate a weighted sum of the inputs; moreover, an activation function uses the weighted sum input and generates the output of a neuron. The human brain can be thought of as an electrical device when compared to a computer. The brain works by controlling the ions (charged atoms) that flow through neurons and thus setting the state of a neuron. Neurons and synapses can be considered as electrical devices for controlling electrical current due to ions. This is exactly how a computer works. The CPU controls where the electrons go, setting the state of the CMOS logic circuit within the control unit. Neuromorphic-based computer architecture mimics the human brain neural network. In other words, neuromorphic-based computer architecture is essentially the human brain architecture put together into solid state. To achieve this, there needs to be an artificial synapse. An

electrical device with memory can be called an artificial synapse. The memristor, a two-terminal nano solid state nonvolatile resistive switching component, can provide energy-efficient neuromorphic computing with its synaptic behavior.

2.1 Propagation and Activation Functions

Different propagation and activation functions are used to train a neural network to do a certain task such as image processing, text recognition, speech recognition [7], [8], object detection and classification [3], [4], [5], [6], disease detections [9], [10], [11], automation of vehicles [14], [15], weather forecast [16], [17], biometric scanners [12], [13], and so on. A propagation function uses a set of incoming inputs to generate a weighted sum of the inputs; moreover, an activation function uses the weighted sum input and generates the output of a neuron. Some of the most common activation functions are the rectified linear unit (ReLU), logistic sigmoid function, linear activation function, tanh activation function, threshold activation function, saturating linear activation function, etc. Two of the most common activation functions and the ones that are of interest in designing the analog neural network are the rectified linear unit (ReLU) and the logistic sigmoid function.

2.2 ReLU Activation Function

The ReLU activation function is widely used in deep learning applications ever since it was introduced by Nair and Hinton in 2010 [18]. ReLU is almost a linear function and hence preserves the linear properties which make it easier to optimize with gradient descent methods and offers better performance than the sigmoid

activation function [19], [20], [21]. The ReLU activation function outputs the weighted sum of the inputs if it is within the output thresholds. If the weighted sum is greater than zero, the output will be the weighted sum itself. If the weighted sum is less than or equal to zero, the output will be zero. The plot generated by the ReLU activation function is shown in Figure 2.1. One main advantage of the ReLU activation function over other activations functions is that it guarantees faster computation as it does not involve any complex computations such as exponents or any trigonometric functions [19]. ReLU can result in sparsity in the activations, where a significant portion of the neurons output zero. This sparsity can make the network more efficient during both forward and backward passes, reducing the computational load. The equation for ReLU is shown in (2.1).

$$V_o = \begin{cases} V_{in}, & V_{in} \geq 0 \\ 0, & V_{in} < 0 \end{cases} \quad \textbf{(2.1)}$$

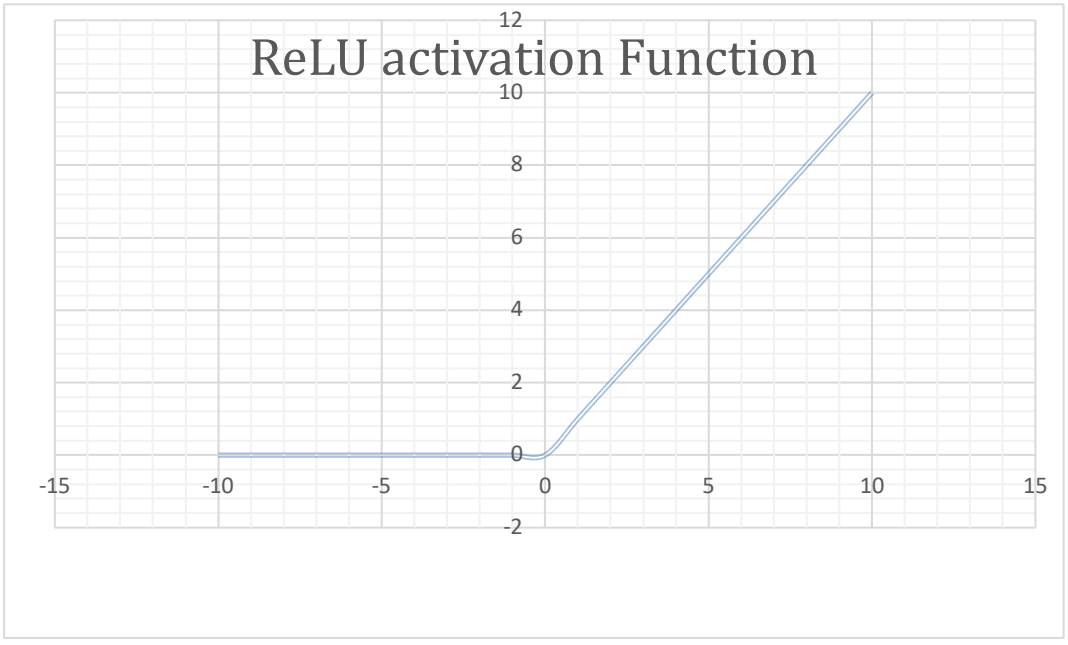


Figure 2.1: ReLU activation function

The neurons with output 0 can be considered as dead neurons. The 0 threshold that prevents the outputs from going to a negative value can be useful when the neuron in the next layer uses these outputs from the previous layer to create a weighted sum for the activation functions. If one of the inputs to a neuron was a large negative value and the weighted sum results in a comparatively similar negative value, depending on all the other inputs, all or many of the neurons in the successive layers will be negative and the neural network may be of no use. The 0 threshold in the ReLU activation function prevents this limitation by creating dead neurons if the weighted sum is less than or equal to 0. However, despite the advantages of this approach, a dead neuron could be an issue sometimes. A dead neuron causes the gradients to die which in turn causes the weight updates not to activate for future data points and thereby results in low efficiency of the learning process as a dead neuron gives no activation [21]. While ReLU can mitigate the vanishing gradient problem, it might lead to an exploding gradient problem for large positive inputs. This can make the training process unstable and slow down convergence. A modified version of the ReLU activation function known as the leaky ReLU (LReLU) was proposed in 2013 to introduce some negative slope to the ReLU preventing a dead neuron and thereby keeping the weight updates alive during the training process [22]. The equation for LReLU is shown in (2.2).

$$V_o = \begin{cases} V_{in}, & V_{in} > 0 \\ \alpha V_{in}, & V_{in} \leq 0 \end{cases} \quad \textbf{(2.2)}$$

The α eliminates the dead neuron problems such that the gradients will not be zero at any time during training. The LReLU computes the gradient with a very small constant value for the negative gradient α. ReLU has an output range only on the positive side of the axis, which can lead to uncentered activations. This can complicate training in some cases, as the network's behavior is influenced by the input range. Poor initialization can lead to a large fraction of neurons remaining inactive in the beginning, affecting learning.

2.3 Logistic Sigmoid Activation Function

The sigmoid activation function is often referred to as logistic sigmoid activation function. The logistic sigmoid activation function uses (2.3) to calculate the output based on the given weighted sum input. It maps the incoming weighted sum inputs to a value between 0 and 1. The shape of the curve generated by a logistic sigmoid function is shown in Figure 2.2.

$$S(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

A sigmoid function maps any real value between zero and 1. This can be generalized by taking the limit of the sigmoid function as the limit tends to -∞ and +∞. The limits of logistic sigmoid function as the limit tends to -∞ and +∞ are shown in (2.4) and (2.5) respectively.

$$\lim_{n\to-\infty} S(n) = \frac{1}{1+e^{-n}} = 0 \tag{2.4}$$

$$\lim_{n\to\infty} S(n) = \frac{1}{1+e^{-n}} = 1 \quad \textbf{(2.5)}$$

A logistic sigmoid activation function can be used to convert the outputs of a neuron into a probability score. Linear passive circuit elements are not capable of remembering a state based on an input. Therefore, a non-linear circuit device known as a Memristor can be used to develop a neural network that can change the weights associated with each input by utilizing some training algorithm.

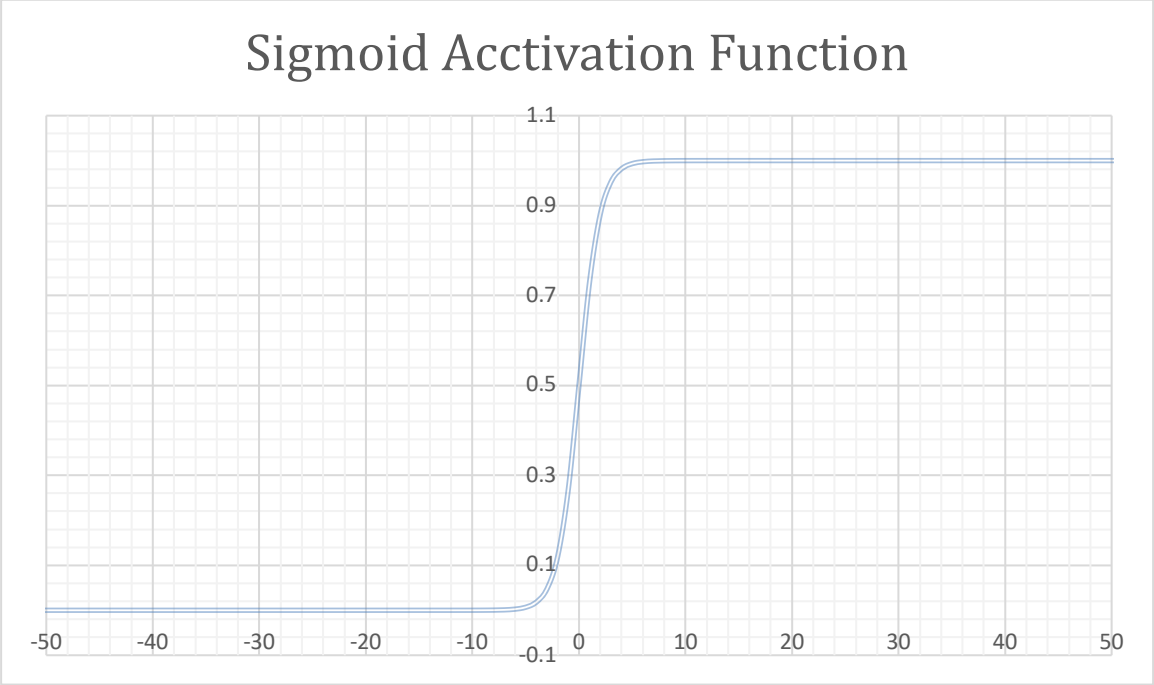


Figure 2.2. Sigmoid activation function

It is suggested to avoid sigmoid activation function when initializing a neural network with random small weights [23]. Sigmoid activation function has some drawbacks such as sharp damp gradients during backpropagation, gradient saturation, slow convergence and non-zero centered output which in turn causes the gradient updates to propagate in different directions.

2.4 Memristor

A memristor is a non-linear two-terminal electrical component relating electric charge and magnetic flux linkage. Until 1971, three fundamental circuit elements ruled the circuit theory. They are Resistors, Capacitors and Inductors. Resistors give the relationship between voltage and current in (2.6).

$$R = \frac{dv}{di} \tag{2.6}$$

Capacitors give relationship between voltage and charge in (2.7).

$$C = \frac{dq}{dv} \tag{2.7}$$

Inductors give the relationship between current and magnetic flux in (2.8).

$$L = \frac{d\phi}{di} \tag{2.8}$$

Resistors, capacitors, and inductors are linked between charge, current, voltage and magnetic flux. This relationship is shown in Figure 4. There is a link missing between magnetic flux and charge. Prof. Leon Chua in 1971 bridged the missing link between magnetic flux and charge based on logical symmetry reasoning [26]. Prof. L Chua named this connection “memristor.” A memristor is basically a resistor with memory. Prof. L Chua mathematically derived the relationship between magnetic flux and charge, shown in (2.9).

$$M = \frac{d\phi}{dq} \tag{2.9}$$

A memristor is a non-linear, passive resistor with no ability to store energy. Memristors can "remember" the last voltage or current that was applied to it. At zero crossing, there would not be any phase shift. A memristor is a non-linear two-terminal electrical component relating to electric charge and magnetic flux linkage. In recent years, memristors have been researched to be used in a variety of applications and many applications are for helping model neural networks. The memristor can be seen as the fourth basic circuit element and due to its ability to be easily scalable, high density, and low power usage, it makes it well-suited to model neural networks and store the synaptic weights. Memristance is applied to circuits to represent variations in synaptic weights. The memristance of the device will only change if the applied input signal exceeds the threshold of the device. Thus, a memristor can be used to create a neuron model. The expressions of the model are described in (2.10) and (2.11).

$$R(t) = R_{on}\frac{\omega(t)}{D} + R_{off}(1 - \frac{\omega(t)}{D} \tag{2.10}$$

$$d\frac{\omega(t)}{dt} = \begin{cases} \mu_v * \frac{R_{on}}{D} * \frac{i_{off}}{i_{on}} * f\big(\omega(t)\big), & v(t) < V_{T-} < 0 \\ 0, & V_{T+} \geq v(t) \geq V_{T-} \\ \mu_v * \frac{R_{on}}{D} * \frac{i_{off}}{i(t)-i_o} * f\big(\omega(t)\big), & v(t) > V_{T+} > 0 \end{cases} \tag{2.11}$$

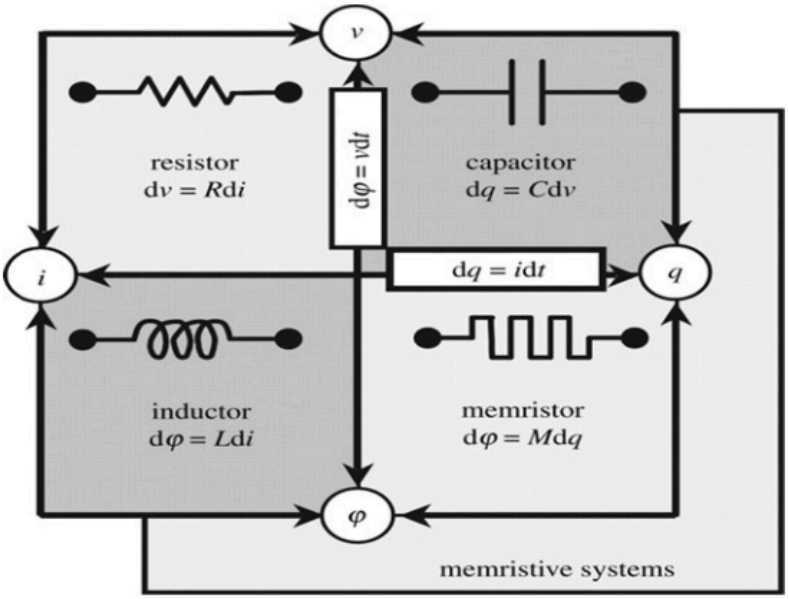


Figure 2.3. Quadrant RLC Relationship [24]

In (2.10) and (2.11), $\omega(t)$ denotes the width of the doped region, μ_v denotes the average ion mobility, i_0, i_{off}, and i_{on} are constants, V_{T+} and V_{T-} are negative and positive threshold voltages. The IV characteristic curve of a memristor results in a pinched hysteresis loop based on (2.10) and (2.11). The IV characteristic curve of a memristor is shown in Figure 2.4.

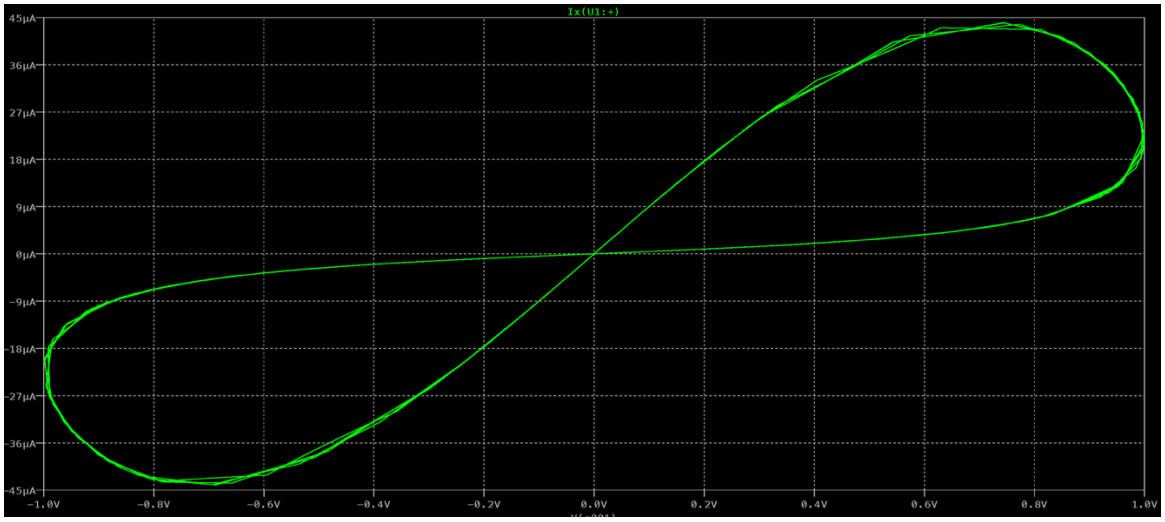


Figure 2.4: IV characteristic curve of a memristor

Memristance of a memristor is used to represent the synaptic weight in the neuron design. The synaptic weight is implemented using a pair of memristors [1]. A memristor's ability to be easily scalable, its high density, and low power usage makes it ideal to model neural networks and store synaptic weights. The synaptic neuron circuit designed by using a pair of memristors can represent negative, zero, and positive synaptic weights [1].

2.5 Neuron Model

There are many neuron models in existence now that utilize memristors to make up a synapse. The current-based training circuit design discussed later in this work requires a neural network to be trained to show the efficiency of the training circuit. The neural network uses neuron model composed of memristors and op-amps. Each synapse is composed of two memristors. The neuron model described in the paper: "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications" was created using op-amps, memristors, and resistors. Synapses consist of two memristors to represent the positive, negative, and zero weights [2]. The output range of the neuron has three output cases as shown in (2.12). The polarity of the two memristors are opposite to each other, so the change in the state will be different for both [2]. The neuron model and the output range based on (2.12) are shown in Figure 2.5 a and b respectively.

$$V_0 = \begin{cases} V_{ss}, & V_0 < V_{ss} \\ R_f \sum V_i w_i, & V_{ss} < V_0 < V_{cc} \\ V_{cc}, & V_0 > V_{cc} \end{cases} \quad \textbf{(2.12)}$$

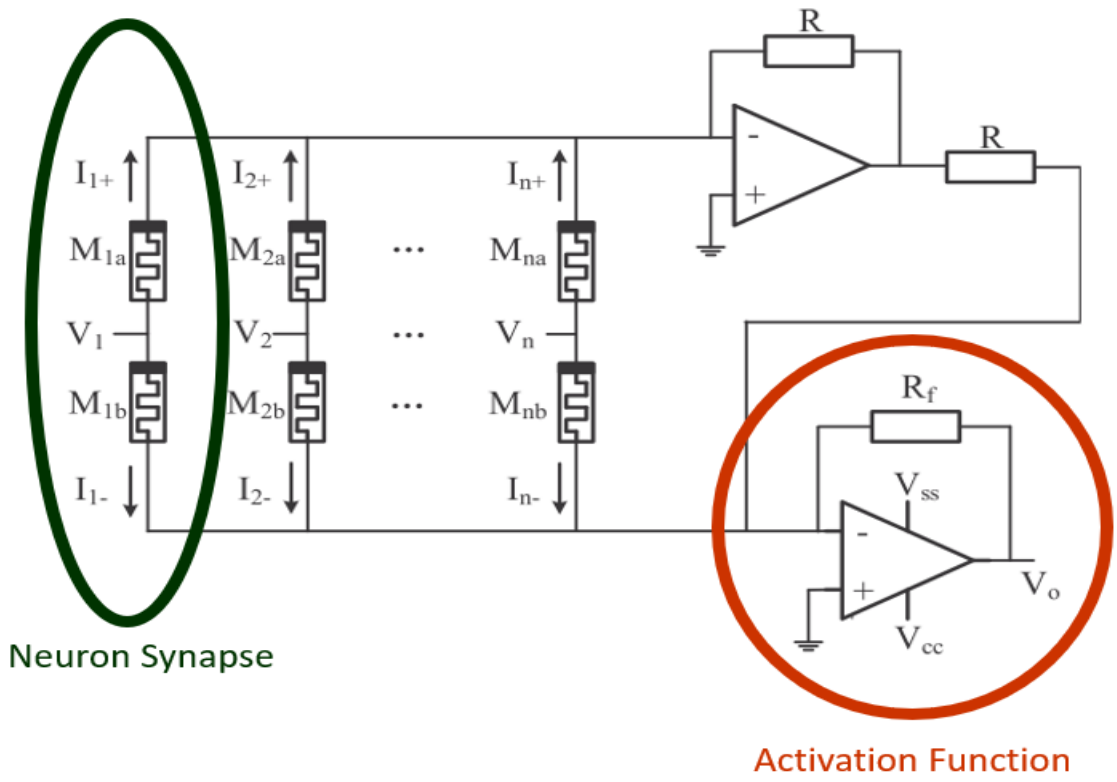


Figure 2.5: a) The neuron circuit [1]

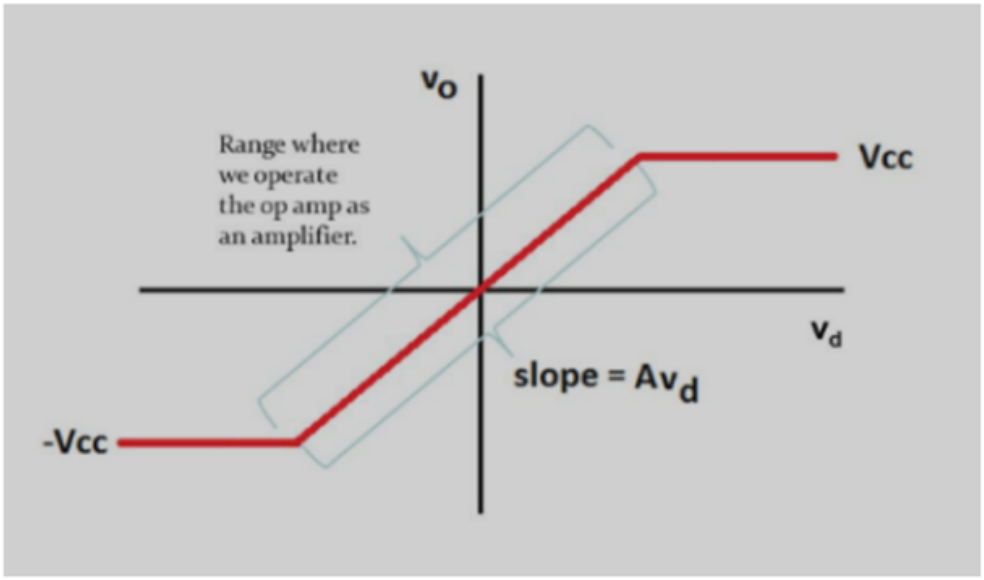


Figure 2.5: b) The neuron output range: the activation function (op-amp characteristic)

In Figure 2.5 a, the memristor pair in each branch determines the weight distribution associated with each input. The weight can be positive or negative based on which memristor was the strongest which in turn decides the net current flow path [2]. V_o represents the output of the neuron after going through the activation function as the op-amp acts as the activation function [2]. In Figure 2.5 b, the activation function clips the output value between -vcc and vcc. This was done by utilizing the op-amp characteristic by setting the positive rail to vcc and the negative rail to -vcc. The op-amp characteristics introduce the hard sigmoid activation function into the neuron circuit. The hard sigmoid function is an activation function that is designed to approximate the sigmoid function while being computationally more efficient and easier to implement. It is commonly used in certain hardware implementations of neural networks, where efficiency is a primary concern. The hard sigmoid function takes a linear segment for inputs within a certain range and saturates at the endpoints. Equation of hard sigmoid function is shown in (2.13)

$$f(x) = \max\left(0, \min\left(1, \frac{x+1}{2}\right)\right) \quad \textbf{(2.13)}$$

Hard sigmoid is computationally efficient to compute, especially in hardware implementations, as it only involves simple arithmetic operations. It also reduces computational complexity compared to more complex activation functions like sigmoid or hyperbolic tangent (tanh). This can be advantageous in resource-constrained environments. However, the hard sigmoid approximates the sigmoid function, which means it might not capture the exact characteristics of the sigmoid for all input ranges which in turn affects the network's learning behavior. The linear segment of the hard sigmoid limits its expressiveness compared to the smooth non-linearity of the sigmoid. This could potentially impact the ability of the network to capture complex relationships in the data. It also saturates at the endpoints (0 and 1)

for inputs outside a certain range. This means that activations in these regions will not contribute to the gradient during backpropagation, potentially leading to a "dead neuron" problem. However, the zeroth order optimization does require complex calculations to compute the gradient. It simply approximates the gradient by introducing small constant perturbations into the circuit.

The multiple parallel branches of a pair of memristors in series in Figure 2.5a models the weight associated with each input to the neuron and number of branches will be equal to the number of total inputs to the neuron including the bias if any [2]. The input to a neuron is connected to each branch in between the two memristors in series. The input voltage is connected between the memristors such that the strength of the on and off resistance in the memristor determines which way more current flows [2]. This enables the implementation of negative weights depending on the current flow (one direction implies positive and the other will be negative) [2]. The output of the neuron based on the input and resistance weight distribution is calculated by (2.14).

$$V_o = v_1\left(R_f\left(\frac{1}{R_{1b}} - \frac{1}{R_{1a}}\right)\right) + v_2\left(R_f\left(\frac{1}{R_{2b}} - \frac{1}{R_{2a}}\right)\right) + \cdots + v_n\left(R_f\left(\frac{1}{R_{nb}} - \frac{1}{R_{na}}\right)\right)$$

$$V_o = \sum_{i=1}^{n} v_i\left(R_f\left(\frac{1}{R_{ib}} - \frac{1}{R_{ia}}\right)\right) \quad \textbf{(2.14)}$$

In (2.14), R_f is the feedback resistance connected to the op-amp generating the neuron output, R_b and R_a are the top and bottom memristance respectively in each input branches of the neuron. The op-amp that generates the neuron output in Figure 2.5a acts as an activation function. The rails of the op-amps are limited to 0V and 1V

[2]. Therefore, any weighted sum of the input voltage over 1V will be clipped at 1V and any voltage under 0V will be clipped at 0V [2]. Any weighted sum voltage within the range 0-1V will be passed on by the op-amp as is. This will be the output of the neuron.

2.6 Weight Distribution

The neuron model from Figure 2.5a could implement positive, negative and zero weight based on the strength of the top and bottom memristors. The strength of the on and off resistances in the memristors will determine the path of flow of the net current through each input branch. For a better understanding of the concept of weight distribution in each branch, let's replace the memristors with resistors and evaluate the new configuration. The weight distribution of each input to the neuron is implemented with a pair of resistors in each input branch of the neuron. (2.14) can be modified to (2.15), where w_i (i=1, 2,…,n) is the weight associated with each input and v_i denotes each input to the neuron [2].

$$\boldsymbol{V_o = v_1 w_1 + v_2 w_2 + \cdots + v_n w_n} \tag{2.15}$$

From (2.14) and (2.15), weight distribution can be expressed by (2.16).

$$\boldsymbol{w_i = \left(R_f \left(\frac{1}{R_{ib}} - \frac{1}{R_{ia}} \right) \right)} \tag{2.16}$$

The weight distribution technique can implement any desired weight simply by using different combinations of R_f, R_b, and R_a values for each branch [2]. However, R_f once set remains the same for all branches as this will be a single feedback resistance connected to the output op-amp of the neuron circuit. Figure 2.6 shows some sample weight calculation when the feedback resistance R_f is 10kΩ [2].

If the top and bottom resistance values are the same in an input branch, the resulting weight at that specific branch will be '0' based on (2.16) [2]. This is also shown in Figure 2.6. Another important aspect of this neuron design is the implementation of negative weights [2]. If the top resistor in an input branch is smaller than the bottom resistor, the resulting weight will be positive. Similarly, if the top resistor is greater than the bottom resistor, the resultant weight in that branch will be negative [2]. This can be seen in Figure 2.6 and (2.16).

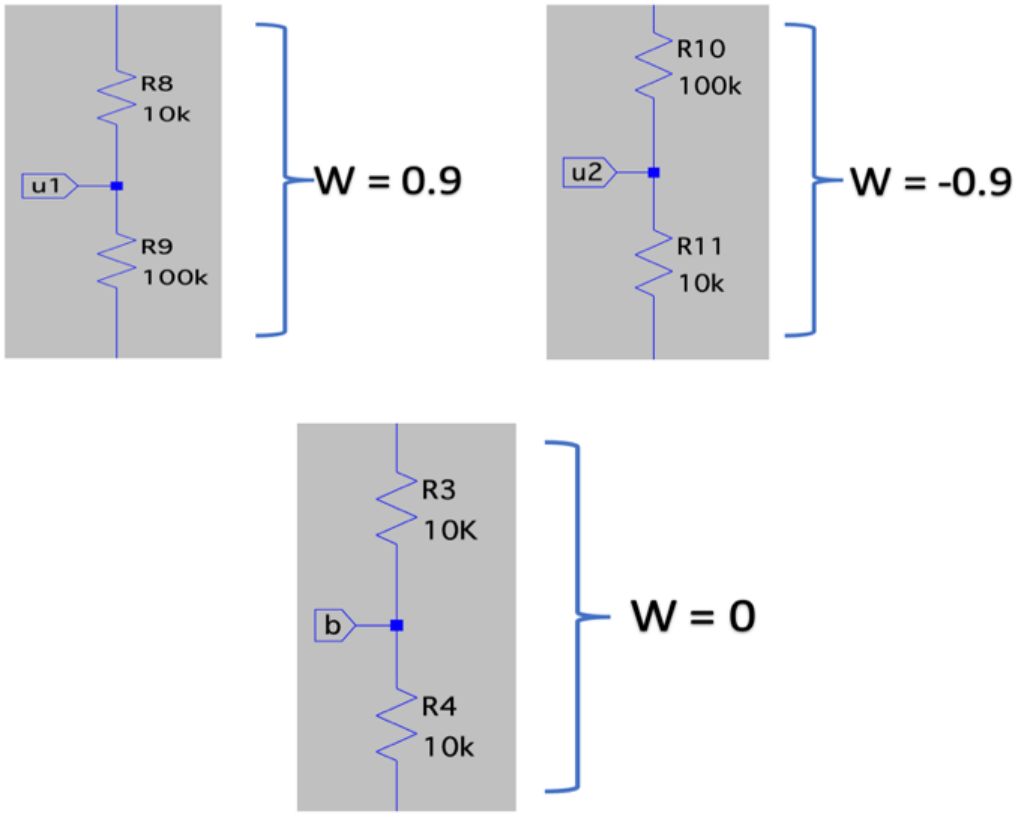


Figure 2.6: Weight distribution examples [2]

If the resistors in Figure 2.6 is replaced with memristors, then the neuron circuit can be trained to perform many tasks such as image processing, text recognition, object recognition and classification, weather forecast, machine translation and so on by utilizing a memristors ability to remember how to respond to different inputs. Most common training techniques used to train a neural network are gradient descent, back

propagation, Euler's method and so on. These techniques for training a neural network includes complex computations to calculate loss functions which includes calculating derivatives. When implementing these techniques in hardware, the design complexity gets higher. Zeroth order optimization technique can be used to overcome this issue.

2.7 Zeroth Order Optimization Technique

The idea of zeroth order optimization to train the analog neural network is to provide a random noise (perturbation) into the neuron output and to compare the noise added neuron output and the actual output to update the weight associated with each input to a neuron to minimize the error in the output based on the target output. The advantage of using zeroth order optimization over existing techniques like backpropagation was the less complexity of hardware architecture and minimal number of transistors used by the zeroth order optimization technique hardware architecture.

Let f(x) be a continuously differentiable objective function on a d-dimension variable $x \in \Re^d$. The one-point gradient estimate of f has the generic form as shown in (2.17).

$$\nabla f(x) = \frac{\phi(d)}{\mu} f(x + \mu u) u \qquad \textbf{(2.17)}$$

In (2.17) u~p is a random direction vector drawn from a certain distribution p, which is typically chosen as either the standard multivariate normal distribution N (0, I) [19] or the multivariate uniform distribution U (S (0, 1)) on a unit sphere centered at zero with a radius of one and where μ is a perturbation radius and z(d) denotes a certain dimension-dependent factor related to the choice of the distribution p [25].

The perturbation is applied only to weight summation. This perturbation influences the output and error. To implement the zeroth order optimization algorithm in a hardware, there are two paths in the circuitry, error with perturbation is calculated in the first path and in the second path error without perturbation is calculated [1]. Finally, the weights are updated according to the difference between the unperturbed and perturbed error function. This algorithm will be applied to every neuron in a neural network by duplicating the necessary circuitry for each neuron.

2.8 Training

Training process in off-chip training or inference operation is implemented in software, then the calculated weights are loaded into the hardware. That could potentially lead to several problems such as mapping conductance value could be stuck on low resistance or high resistance state [2]. The full block diagram of the neuron circuit and the training WSSP algorithm using a current based design is shown in Figure 2.7. Block diagram of the neuron circuit and training based on the WSSP algorithm shown in Figure 2.7 doesn't involve the complex derivative calculation as the backpropagation algorithm [2]. The training is divided into two processes, forward path, and feedback adjustment for memristor value [2]. These two operations are executed alternatively controlled by a signal V_c in the circuit until the algorithm converges [2]. The switching function is performed by transmission gate switches SW, the used operational amplifier is LM741, and the sample and hold are LF398 [2]. The process of forwarding propagation could be summarized as follow when the control signal V_C is at a high level, the inputs are evaluated for error calculation [2]. The input

signals are input to the network for error calculation. The input signal of the neural network is denoted as I_i (i=1, 2,..,n), I_{per} is a small positive current, which is the perturbation signal of the WSSP algorithm. I_i is the output without a perturbation and I_{per} is the output with a perturbation, I_t is the target value. The error E and E_{per} are calculated by (2.18) and (2.19).

$$E_{per} = \left(I_T - I_{per}\right)^2 \tag{2.18}$$

$$E = (I_T - I_o)^2 \tag{2.19}$$

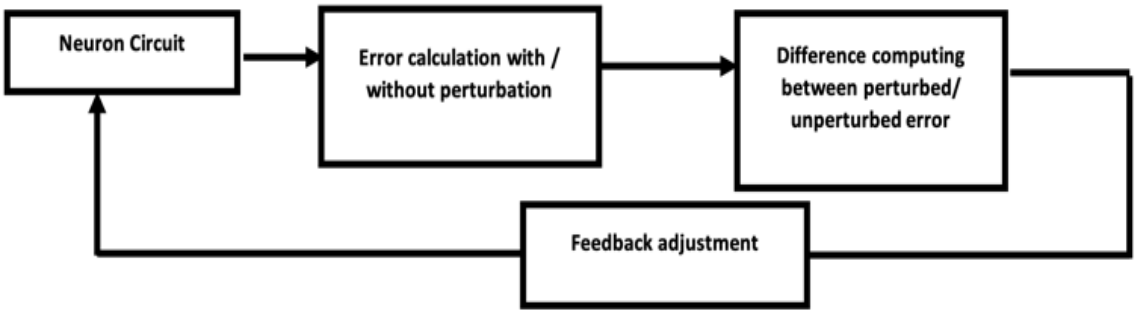


Figure 2.7: Block diagram of the neuron circuit and training based on the WSSP algorithm.

The difference between unperturbed and perturbed error is calculated by taking the difference between E_{per} and E [2]. For the feedback adjustment: when the control v_c is at the low level, V_{fbi} feedback to the network for weight updating is adjusted by (2.20).

$$V_{fbi} = \begin{cases} -(\Delta E + V_{T-}).sign(V_i), & (\Delta E < 0) \\ -(\Delta E + V_{T+}).sign(V_i), & (\Delta E > 0) \end{cases} \tag{2.20}$$

The weight adjustment depends on the error difference, perturbation signal, and input. However, in the hardware, it depends on the error difference and the sign of the input signal [2]. The threshold is added to make sure that the adjusting voltage is greater than the threshold of the memristor.

2.9 Summary

Neural networks are efficient algorithms commonly used for applications such as image processing, text recognition, and classification. Each neuron in a given layer of the neural network accepts inputs from the previous layer and creates a weighted sum. This weighted sum will be passed through an activation function to create the neuron output. The output is then passed as input to the neurons in the next layer. the weights associated with each input to a neuron determines the importance each incoming input to a neuron. Different propagation and activation functions are used to train a neural network to do a certain task such as image processing, text recognition, and classification. A propagation function uses a set of incoming inputs to generate a weighted sum of the inputs; moreover, an activation function uses the weighted sum input and generates the output of a neuron. Two of the most common activation functions and the ones that are of interest in designing the analog neural network are the rectified linear unit (ReLU) and the logistic sigmoid function. Linear passive circuit elements are not capable of remembering a state based on an input. Therefore, a non-linear circuit device known as a Memristor can be used to develop a neural network that is capable of changing the weights associated with each input by utilizing some training algorithm. The memristance of the memristor will only change if the applied input signal exceeds the threshold of the device. Zeroth order optimization for training neural networks as opposed to the hardware complex and time-consuming training algorithms such as back propagation itself reduces the hardware complexity of the training circuit for a neural network. The idea of zeroth order optimization to train the analog neural network is to provide a random noise

(perturbation) into the neuron output and to compare the noise added neuron output and the actual output to update the weight associated with each input to a neuron to minimize the error in the output based on the target output.

The neural network design training design discussed in this chapter was simple as it uses the WSSP algorithm for training. This means there is no complex derivative calculations involved as opposed to training algorithms such as back propagation [1]. Memristance of a memristor was used to represent the synaptic weight in the neuron design. The synaptic weight was implemented using a pair of memristors [1]. A memristor's ability to be easily scalable, its high density, and low power usage makes it ideal to model neural networks and store synaptic weights. The synaptic neuron circuit designed by using a pair of memristors can represent negative, zero, and positive synaptic weights [1]. The memristor pairs in each branch determine the weight distribution associated with each input. The weight can be positive or negative based on which memristor was the strongest which in turn decides the net current flow path [2]. Training process in off-chip training or inference operation is implemented in software, then the calculated weights are loaded into the hardware. That could potentially lead to several problems such as mapping conductance value could be stuck on low resistance or high resistance state. The WSSP algorithm shown doesn't involve the complex derivative calculation as the backpropagation algorithm [2]. The training is divided into two processes, forward path, and feedback adjustment for memristor value. The difference between unperturbed and perturbed error is calculated by taking the difference between E_{per} and E [2]. The weight adjustment depends on the error difference, perturbation signal, and input.

Chapter 3

Similar Work

The Neural network training circuit design discussed in the previous chapter does not require complex circuitry to compute gradient loss as it uses Zeroth order optimization (ZOO) for training purpose instead of complex algorithms such as backpropagation that involves complex derivative calculations. The memristor-based synapse design seems very efficient as it could implement positive, negative and zero weight distribution based on the memristance strength. The objective of this work is to create efficient yet less complex current based hardware to implement neural network training by utilizing the ZOO algorithm. A similar work with a voltage-based training circuit that uses ZOO algorithm is described in the paper: "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications." However, the training part uses complex circuitry with op-amps for implementing the zeroth order optimization. The training circuit is necessary for each neuron to be trained properly and the complexity in terms of the number of transistors used for this purpose grows really fast as the number of transistors in an op-amp circuitry in real life increases. The error calculation circuitry uses a complex two-quadrant multiplier AD633 chip. This has multiple op-amps in it which significantly increases the total number of transistors. Each feedback voltage update circuitry has many two-to-one multiplexers which also contributes to a large sum of transistors. This chapter investigates the limitations of the current training circuit and how it affects the scalability of a neural network. An example of a memristor-based neural network circuit along with the training circuit that involves error calculations using zeroth order optimization is shown in Figure 3.1.

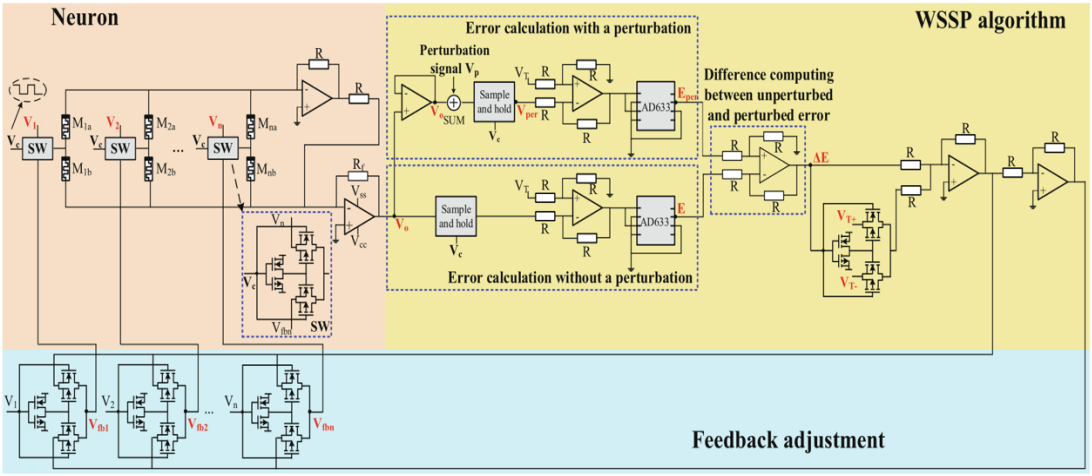


Figure 3.1: Memristor-based neural network circuit with the training circuit [1]

3.1 Error Calculations

The error calculation circuitry uses complex two-quadrant multiplier AD633 chip. This has multiple op-amps in it which in turn increases the total number of transistors used in the training circuit. To implement the zeroth order optimization algorithm, there are two paths in the circuitry, error with perturbation is calculated in the first path and in the second path error without perturbation is calculated. The circuitry that calculates the error with and without perturbation is shown in Figure 3.2. The two circuit paths that calculate the error with and without perturbation were isolated using an op-amp. The difference between the output voltage / perturbed voltage and the target voltage was taken using differential amplifiers in both paths. The differential amplifiers along with the two-quadrant multiplier chip AD633 implement (2.18) and (2.19). Each op-amp consists of at least seven or eight transistors. This means that there will be at least 21 to 24 transistors in training circuit used to implement the differential amplifier to calculate the error with and without perturbation. A common simplified op-amp model, known as the "two-stage Miller compensated op-amp," can be built using approximately seven transistors. This basic model includes two differential input transistors, a current mirror, a compensation

capacitor, and a push-pull output stage. While this model can demonstrate some essential characteristics of an op-amp, it lacks the complexity and sophistication of a real op-amp, and its performance may be limited in terms of gain, bandwidth, and other specifications. The standard 741 Op-amp circuit contains 20 transistors and 11 resistors. Modern op-amps for practical applications and real-world performance are more complex and are built using advanced integrated circuit technology and incorporates hundreds to thousands of transistors. Modifying this circuitry to eliminate the differential amplifiers will save thousands of transistors when implementing the existing design in real life.

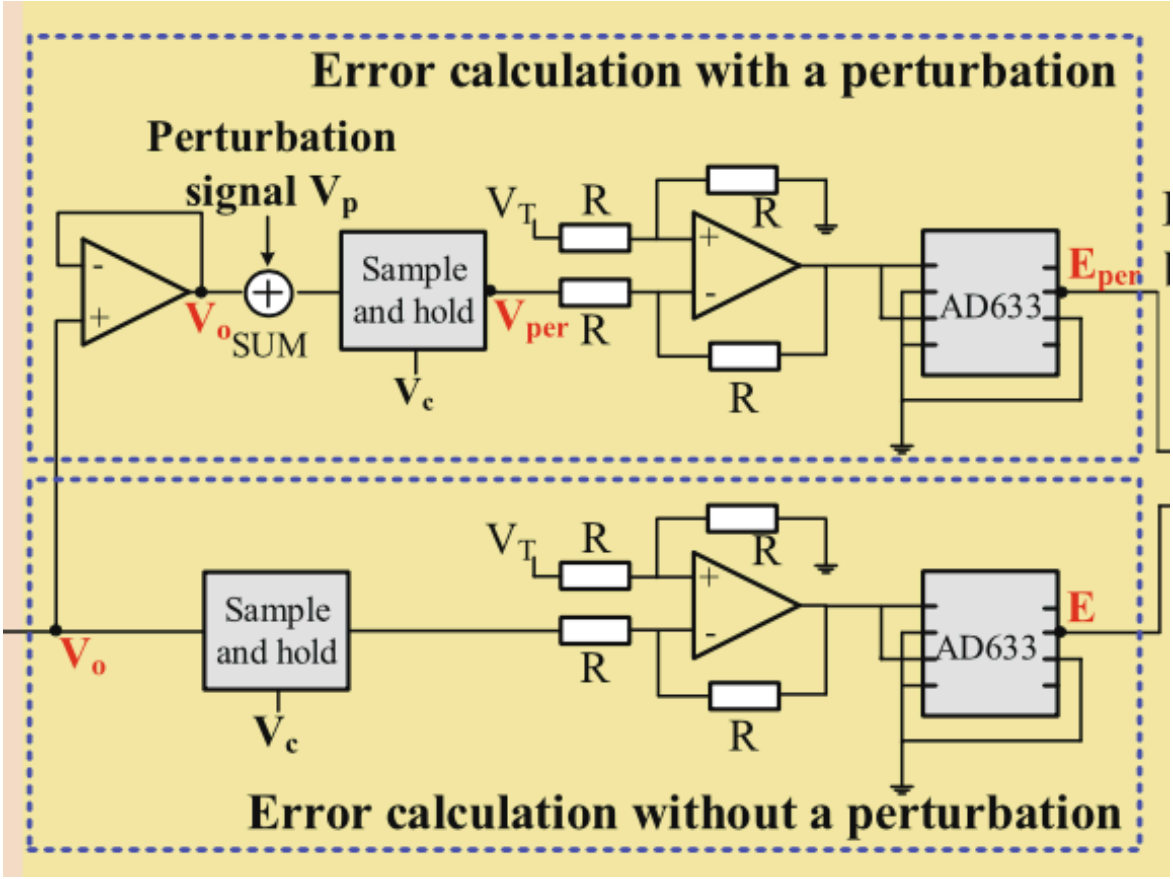
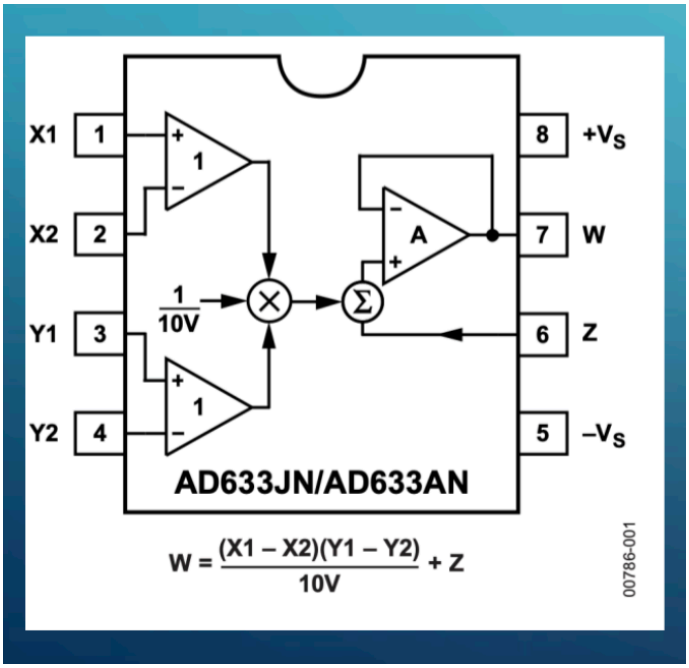


Figure 3.2: Circuitry that calculates the error with and without perturbation [1]

3.2 Two Quadrant Multipliers

The error calculation circuitry uses complex two quadrant multiplier AD633 chip. This chip has multiple op-amps in it and each op-amp is composed of about seven to

eight transistors. Therefore, the usage of AD633 chip for taking the product of voltages significantly increases the total number of transistors required to implement the zeroth order optimization training circuit. The Pinout diagram of the AD633 chip and the circuit setup that calculates the product of the difference between the output voltage / perturbed voltage and the target voltage are shown in Figure 3.3 a) and b) respectively.



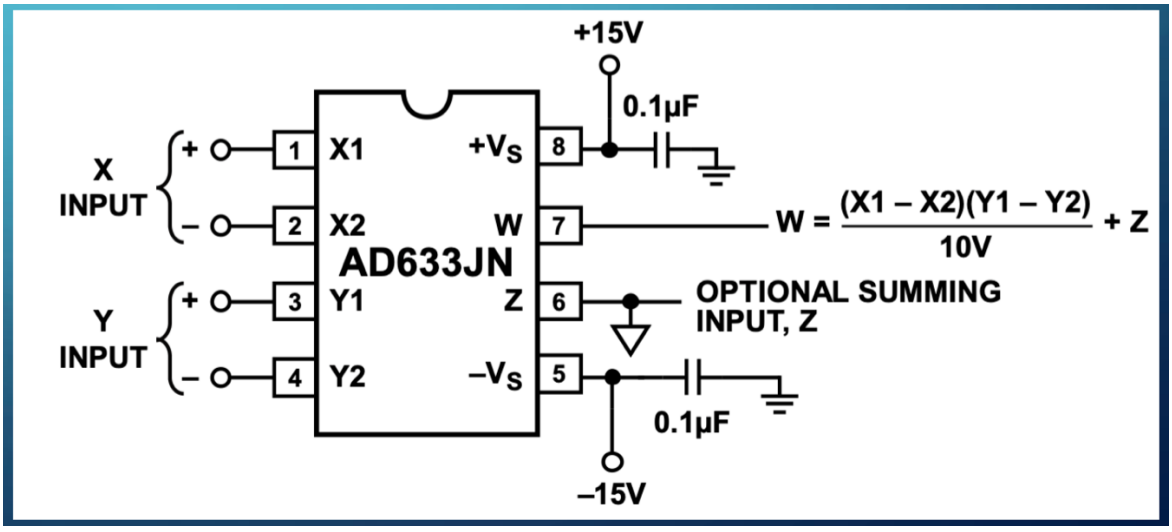


Figure 3.3: a) Pinout Diagram of AD633 chip [31]

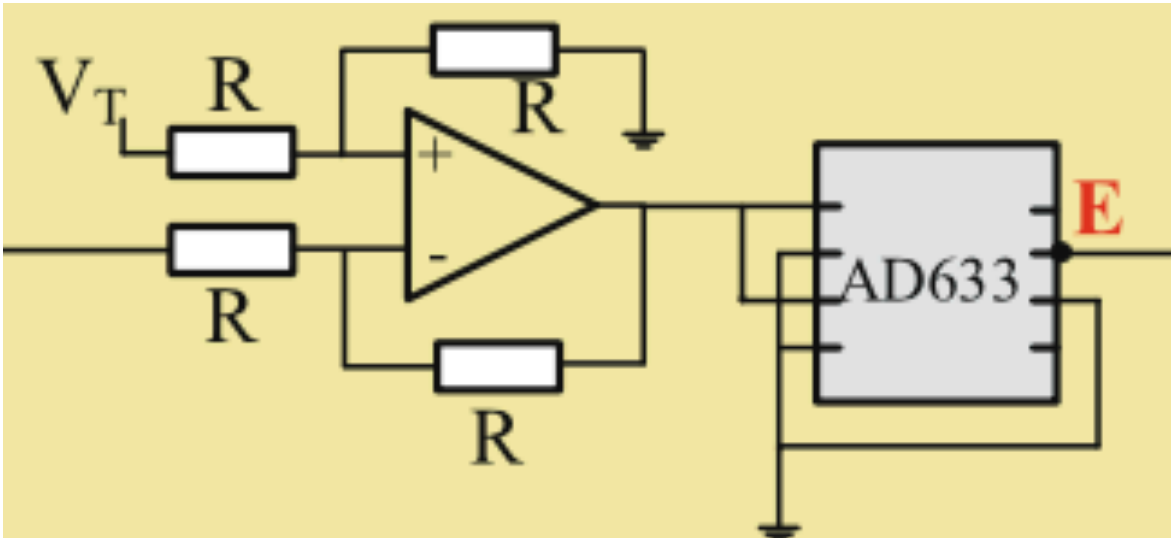


Figure 3.3: b) Circuit setup that calculates the product of the difference between the output voltage / perturbed voltage and the target voltage [1]

In Figure 3.3b, a. differential amplifier was used to take the difference between the output voltage / perturbed voltage and the target voltage. This result was then fed to the two-quadrant multiplier while grounding the other pins to calculate the product of the difference between the output voltage / perturbed voltage and the target voltage. i.e., the error. However, in Figure 3.3a, the AD633 chip can take the difference between given two voltages as well as taking the product of the differences. This means the usage of an extra differential amplifier in Figure 3.3b is unnecessary and introduces an extra 14 to 16 transistors. The AD633 multiplier chip has 3 op-amps in it to perform the two-quadrant multiplication. i.e., about a minimum of 42 to 48 transistors to calculate the difference between the error with and without perturbation. Using a standard 741 Op-amp circuit requires about 120 transistors for this and the number grows significantly as the op-amp circuitry gets complex to increase its performance in terms of gain, bandwidth, and other specifications.

3.3 Two-to-One Multiplexers

The neural network with training circuit basically implies that there will be two working modes for the neural network. They are normal mode and the training mode. During normal mode, the neural network simply performs its intended purpose such as image processing, text recognition, and classification. During training mode, the neural network may update its weight distribution for each neuron to perform a specific task with better accuracy. This mode can be utilized to create a generic neural network that could be trained to do different tasks. To switch between two operational modes, the circuit setup requires a switch. A simple circuit component that can switch between different inputs is multiplexers. Since there are only two operational modes, a two-to-one multiplexer is enough to accomplish the required functionality. The memristor based neural network circuit along with the training circuit is shown in Figure 3.4.

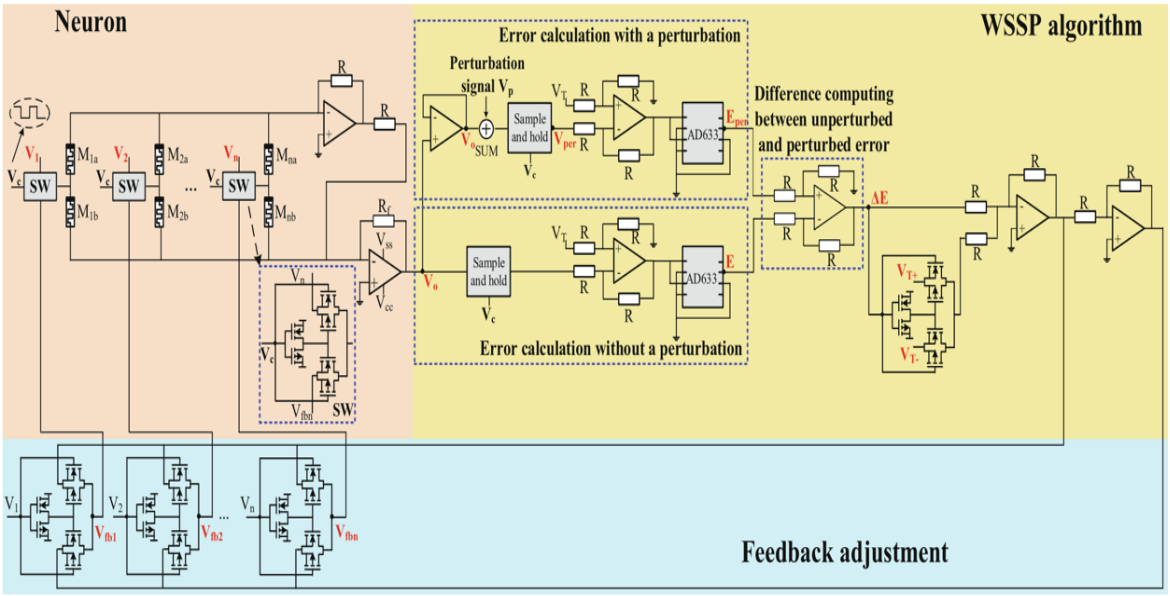


Figure 3.4: Memristor based neural network circuit along with the training circuit [1]

From Figure 3.4, the feedback adjustment circuit that implements the zeroth order optimization technique along with the two required operational modes of the neural network requires a lot of multiplexers. To be precise, a neuron with n inputs requires a total of 2n+1 multiplexers as per the existing design from Figure 3.4. Each multiplexer has six transistors. This implies that for an n input neuron, there will be a total of 12n+6 transistors. Taking the previous example again, for a neural network with 100 layers with a fully connected 100 neurons in each layer. The total number of neurons in such a neural network will be 10,000 neurons and each neuron will have exactly 100 inputs without bias. This implies that a single neuron in such a network will have at least 1206 transistors. Therefore, the total transistors present only in multiplexers in a 100-layer neural network with a fully connected 100 neurons in each layer will be at least 12,060,000 according to the existing design from the paper "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications". This huge number of transistors can be minimized by redesigning the training circuit to deal with current instead of voltage as inputs.

3.4 Summary

The memristor based synapse design seems very efficient as it could implement positive, negative and zero weight distribution based on the memristance strength. However, the training part could be potentially improvised by redesigning some of the circuitry for implementing the zeroth order optimization. The training circuit is necessary for each neuron and the complexity in terms of the number of transistors used for this purpose grows exponentially as the number of neurons and the total number of layers in a neural network increase. The error calculation circuitry uses complex two quadrant multiplier AD633 chip. This has multiple op-amps in it which in turn increases the total number of transistors used in the training circuit. To implement the zeroth order optimization algorithm, there are two paths in the circuitry, error with perturbation is calculated in the first path and in the second path error without perturbation is calculated. Eliminating the usage of complex two-quadrant multiplier chip with a much

simpler circuit setup for calculating the error with and without perturbation will save thousands to ten thousand of transistors when implementing the design in real world as the op-amps may require thousands of transistors to implement correct functionality. To switch between two operational modes, the circuit setup requires a switch. A simple circuit component that can switch between different inputs is multiplexers. Since there are only two operational modes, a two-to-one multiplexer is enough to accomplish the required functionality. The total transistors present only in multiplexers in a 100-layer neural network with a fully connected 100 neurons in each layer will be at least 12,060,000 according to the design described in the paper: “Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications”. A wise redesign of the training circuit for the memristor based neural network design will reduce the hardware complexity and improve the SWaP efficiency.

Chapter 4

Proposed Method

A current-based ANN training circuitry with less circuit components (minimal total number of transistors) that utilizes very minimal to no op-amps in the training circuitry can reduce power consumption as well. The WSSP algorithm that utilizes ZOO optimization requires adding perturbation to all weighted sums and updating the synaptic weights based on the error difference between the error with and without perturbation. To implement simultaneous perturbation to all weighted sums, two isolated branches of the output current is required to calculate the difference between the target current and the actual output squared (error without perturbation) and the error with perturbation. The circuitry to calculate error with and without perturbation will be identical. The only difference is that the perturbed output is used in one branch to calculate the error with perturbation. The training circuitry uses current mirrors to create isolated copies of output currents to process them individually. A Translinear multiplier circuit is used to compute the square of the difference between the target output and the actual output. Using simple circuitry like a Translinear multiplier and thereby avoiding complex two-quadrant multipliers like AD633 will reduce the power consumption and save some transistors. The existing voltage-based training circuit design uses many op-amps and complex two-quadrant multiplier chip AD633 for calculating the product of the difference between the output voltage / perturbed voltage and the target voltage. A clever usage of pulldown circuit with the target output as current

through the pulldown path yields the difference between the output current / perturbed current and the target current as opposed to the usage of differential amplifiers used in the existing design. The proposed design with pulldown circuits to calculate the difference between currents will have very minimal circuitry. The complex two-quadrant multiplier chip can be replaced with the translinear circuit setup to take the product of the difference between the output current / perturbed current and the target current. Thus, the hardware complexity in terms of transistor count of the training circuitry can be reduced significantly. The SWaP efficiency of the redesigned circuit setup will depend on the number of layers and total number of neurons in a neural network. This chapter discusses the main difference between the proposed method of current based training circuit from this work and the voltage-based existing training circuit.

4.1 Error Calculation Circuit

A pulldown circuit was created to operate with currents instead of voltages. The target output current flowing through the pulldown path yields the difference between the output current / perturbed current and the target current. The new proposed circuit that takes the difference between the output current / perturbed current and the target current is shown in Figure 4.1. The output current / perturbed current is the input to the circuit and the pulldown path has a constant current source that flows to the ground. The constant current source in the pulldown network is the actual target output current. Applying KCL on node x on the circuit, the output yields difference between the output current / perturbed current and the target current. This is calculated from (4.1).

$$I_{Error} = I_{in} - I_T \quad \textbf{(4.1)}$$

The current source may be replaced with voltage controlled current source to improve the reliability of the redesigned error calculation circuit by using a target voltage that determines the target current thereby allows the flexibility of keeping input to neuron being voltages even though the training part uses current to update weights.

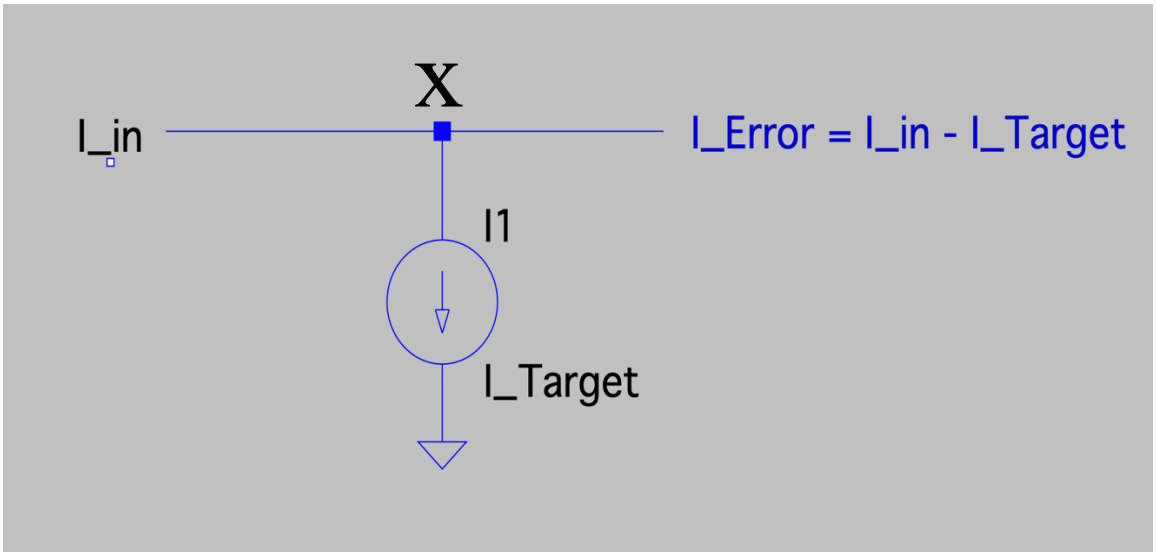


Figure 4.1: Proposed Error calculation circuitry that computes the difference between the output current / perturbed current and the target current

4.2 Translinear Multiplier

The error calculation circuitry in the existing design uses complex two quadrant multiplier AD633 chip. This chip has multiple op-amps in it and each op-amp is composed of about seven to eight transistors. Thus, the usage of AD633 chip for taking the product of voltages significantly increases the total number of transistors

required to implement the zeroth order optimization training circuit. The current based training circuit designed in this work reduces the hardware complexity. This implies that the current based training circuit uses less complex circuitry for error calculation, feedback adjustment and so on. Error calculation circuit requires a circuitry capable of computing the square of the difference between the actual output and the target output and must be capable of working with current inputs. Translinear property comes in handy for this purpose. The translinear multiplier circuit setup uses a very minimal number of transistors as opposed to that of the complex two quadrant multiplier AD633 chip. The translinear multiplier circuit setup is shown in Figure 4.2.

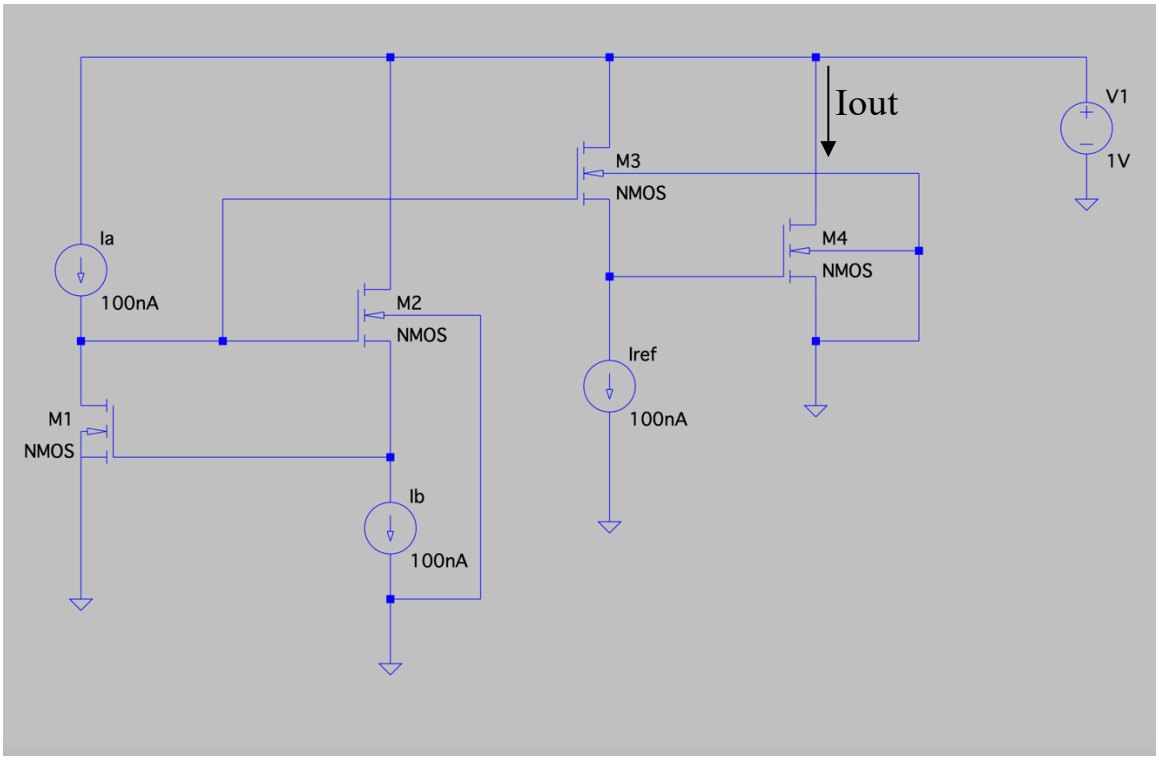


Figure 4.2: Translinear two-quadrant multiplier circuit

In Figure 4.2, the inputs to the translinear two-quadrant multiplier circuit is I_a, I_b, and I_{ref}. The output of the circuit is I_{out}. Unlike the complex two-quadrant multiplier AD633 chip, translinear two-quadrant multiplier circuit only contains four transistors whereas the AD633 chip has 21 to 24 transistors. Thus, the translinear two-quadrant multiplier circuit saves about 17 to 20 transistors in one path. The current training circuit has two paths to calculate the error with and without perturbation. This implies that the translinear two-quadrant multiplier circuit saves at least 34 to 40 transistors. The output current I_{out} can be computed from (4.2).

$$I_{out} = \frac{I_a I_b}{I_{ref}} \tag{4.2}$$

The translinear multiplier has a major drawback that it only accepts positive currents. This implies that the current cannot go backwards, and this could cause an issue when taking the square of the difference between the target current and the output current if the difference results in a negative value. One potential solution for this issue to keep the translinear multiplier to take the product of the difference between the target current and the output current is to introduce a current mode absolute value circuit that overcomes the negative current issue since the square of a negative or positive value results in a positive output. Other potential techniques for computing square of the difference between the target current and the output current will also be explored.

If I_{ref} is set to a constant current throughout the circuit, for example let I_{ref} be one Amp, the output current I_{out} will then be the product of the input currents I_a and I_b. The reference current in the translinear multiplier can be utilized to model the learning rate and thereby no additional circuitry is required to introduce a learning rate to the system as shown in (4.3).

$$if\ I_a = I_b = I_T - I_{per},$$

$$then\ I_{out} = \frac{\left(I_T - I_{per}\right)^2}{I_{ref}}$$

$$but\ E_{per} = \left(I_T - I_{per}\right)^2$$

$$then,\ \frac{\left(I_T - I_{per}\right)^2}{I_{ref}} = \frac{E_{per}}{I_{ref}}$$

$$\frac{1}{I_{ref}} = \alpha \Rightarrow I_{out} = \alpha . E_{per} \quad \textbf{(4.3)}$$

The input current I_{ref} can also be used to scale down or scale up the resultant current output.

4.3 Two-to-One Current Multiplexer

The current circuit setup for training the neural network using zeroth order optimization technique uses many two-to-one voltage multiplexers to switch between the normal operation mode and the training mode. From chapter 3, the total transistors present only in multiplexers in a 100-layer neural network with a fully connected 100 neurons in each layer will be at least 12,060,000 according to the existing design from the paper "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications". This was because a single two-to-one voltage multiplexer has six transistors in it and an n input neuron will have a total of 12n+6 transistors according to the existing design. A redesign of two-to-one voltage multiplexer to a two-to-one current multiplexer may save up some transistors. A proposed two-to-one current multiplexer design is shown in Figure 4.3.

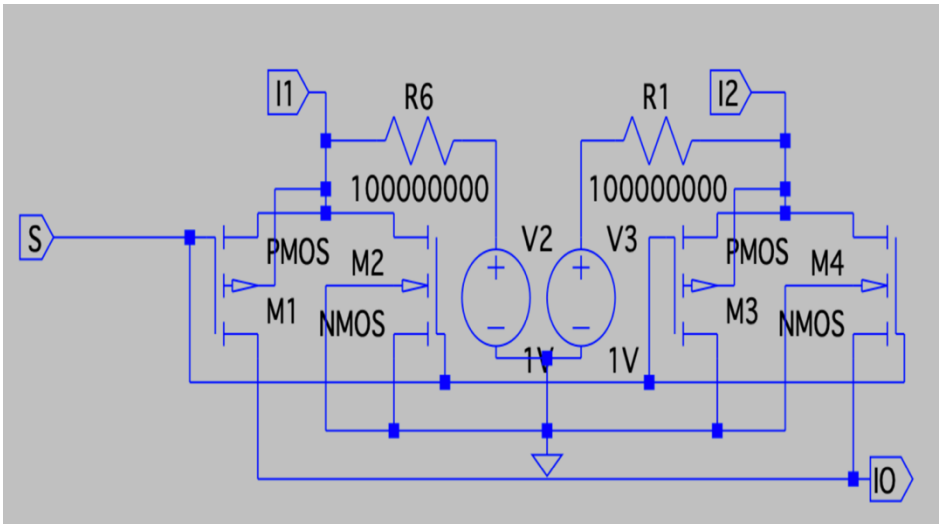


Figure 4.3: Two-to-one current multiplexer

In Figure 4.3, I_1 and I_2 are the two input currents to the multiplexer and S was the select signal that determines which of the two inputs is to be passed as the output I_o. The two large resistors R_1 and R_6 prevent any sneak path current as the input currents are in the range of Milliamps or Microamps. The two-to-one current multiplexer only has four transistors in it as opposed to six transistors in a two-to-one voltage multiplexer. Thus, two transistors are saved per multiplexers. A neuron with n inputs requires a total of 2n+1 multiplexers for the training circuit as per the existing design from Figure 3.4. Other methods will be explored to further minimize the usage of transistors to allow scalability for the neural network design. Each multiplexer has four transistors. This implies that for an n input neuron, there will be a total of 8n+4 transistors. Taking the previous example again, for a neural network with 100 layers with a fully connected 100 neurons in each layer. The total number of neurons in such a neural network will be 10,000 neurons and each neuron will have exactly 100 inputs without bias. This implies that a single neuron in such a network will have at least 804 transistors. Therefore, the total transistors present only in multiplexers in a 100-layer neural network with a fully connected 100 neurons in each layer will be at least 8,040,000. The existing

design for such a neural network will have at least 12,060,000 transistors. This means the redesigned two-to-one current multiplexer saves 4,020,000 transistors. i.e., the hardware complexity is reduced by almost 25%.

4.4 Summary

A closer look on the neural network circuit design discussed in the paper "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications" revealed some potential area for improvements in the training circuit that could reduce the hardware complexity in terms of total number of transistors used in the current circuit that implements zeroth order optimization. A current-based ANN training circuitry with less complex circuit components that utilizes very minimal to no op-amps in the training circuitry can reduce power consumption as well. The existing voltage-based training circuit design uses many op-amps and complex two-quadrant multiplier chip AD633 for calculating the product of the difference between the output voltage / perturbed voltage and the target voltage. The proposed design with pulldown circuits to calculate difference between currents will have very minimal circuitry. The complex two-quadrant multiplier chip can be replaced with the translinear circuit setup to take the product of the difference between the output current / perturbed current and the target current. Thus, the hardware complexity of the training circuitry can be reduced significantly. A multiplexer redesign will reduce the transistor count by 25% compared to existing design.

Chapter 5

Current mode training circuit

The training circuit trains the neural network by adjusting the memristance of the memristors in a synapse by computing the error based on the neuron output and the target output. The training circuit is mainly made up of three sub-components; the error calculation circuit, the memristor threshold select signal generator and a multiplexer to select positive or negative memristor threshold voltage. The error calculation circuit calculates the error between the output current / perturbed current and the target current. The memristance of the memristors is adjusted based on the sign of the error. i.e., if the error is negative, the positive memristor threshold value will be fed into the synapse to adjust the memristance to reduce the error. Similarly, the negative memristor threshold value will be used if the error is positive. The error calculation circuit is made up of current mirror circuits, translinear multiplier circuits and absolute current circuits. This design uses the technique shown in Figure 4.1 to eliminate current mode subtraction circuit to reduce the total transistors used in the design and thereby reduces the complexity of the training circuit design. The neural network along with the training circuit performs forward propagation and training process alternatively. A control signal will be controlling the forward propagation and training process. Forward propagation occurs when the control signal is 1V and the feedback adjustment happens when the control signal is -1V. A multiplexer will handle the forward propagation and feedback adjustment by selecting between the inputs to the neural network and the memristance adjustment voltage based on the control signal.

5.1 Error Calculation Circuit

The error calculation circuit calculates the error between the actual output and the target output and adjust the memristance of the memristors during the feedback adjustment process. The error calculation equations are shown in (5.1), (5.2) and (5.3).

$$E_{per} = \left((I_{target} - I_{actual}\right) + I_{perturbation})^2 \quad \textbf{(5.1)}$$

$$E_{out} = \left(I_{target} - I_{actual}\right)^2 \quad \textbf{(5.2)}$$

$$E = \mathrm{E_{per}} - E_{out} \quad \textbf{(5.3)}$$

E_{per} is the error with perturbation, E_{out} is the error without perturbation and E is the resultant error calculated by subtracting the error without perturbation from the error with perturbation. The error calculation circuit implements (5.1), (5.2) and (5.3). The circuit consists of current mirror circuits, translinear multiplier circuits and current mode absolute value circuits. During the feedback adjustment process, current mirrors are used to create copies of the actual output current and the target output current. The output current of the neural network is subtracted from the target current using the technique shown in Figure 4.1 and two branches of the resultant current are created using a current mirror. A small positive constant perturbation current will be added to one branch. The error between the squares of the perturbated and non-perturbated resultant current is used to adjust the memristance of the memristors. A Translinear multiplier is used to take the square of the current values and the reference current is set to 100nA in (4.1). Translinear multiplier circuits only work with positive currents. However, the perturbated and non-perturbated currents may be negative sometimes (flowing in the backward direction). The negative currents

will break the intended functionality of the circuit. This was mitigated using an absolute value current circuit as the square of a positive/negative number will always be a positive number. The difference between the squares of perturbated and non-perturbated current is taken using technique shown in Figure 4.1. The sign of the error along with the sign of the input voltage will determine the feedback adjustment voltage to adjust the memristance. The error calculation circuit is shown in Figure 5.1.

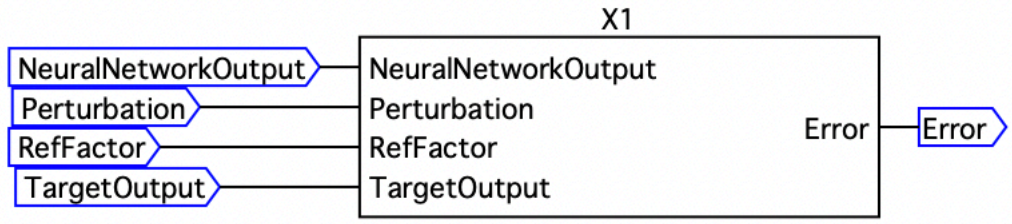


Figure 5.1 a): Error calculation circuit component

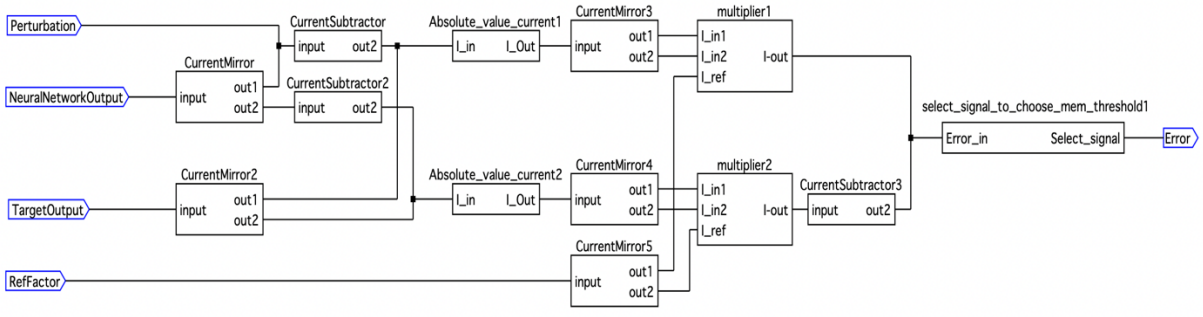


Figure 5.1 b): Error Calculation circuit breakdown

The error calculation circuit accepts actual neural network output, perturbation constant, Reference factor that goes into the multiplier circuit as the divisor and the target output and then calculates the error with magnitude and sign. The training

circuit discussed in this paper do not utilize the magnitude of the error and only use the sign of the error to adjust the memristance of the memristors. Accounting for the error magnitude requires more circuits to make use of the error information. By ignoring the magnitude of the error, the training circuit reduces design complexity while retaining good accuracy in updating the weight of the synapse. This means a constant voltage outside the threshold voltage limit will be applied to the synapse during feedback voltage adjustment process. Thus, the actual output after training for enough cycles will oscillate around the target output. The oscillation swing amplitude can be reduced by adjusting the constant feedback adjustment voltage. A detailed subcomponent level breakdown of the error calculation is shown in Figure 5.1b.

A test simulation was done on the error calculation circuit to test its accuracy. The target output was set to 100nA, the actual output was 0nA, perturbation was set to 10nA, and the reference factor was 100nA. The expected error from (5.1), (5.2) and (5.3) was 21nA whereas the actual error output from the error calculation circuit was 17.6454nA. The error percentage between the actual and expected output is only 15.95%. This data is shown in table I. The simulation waveform of this test case is shown in Figure 5.2.

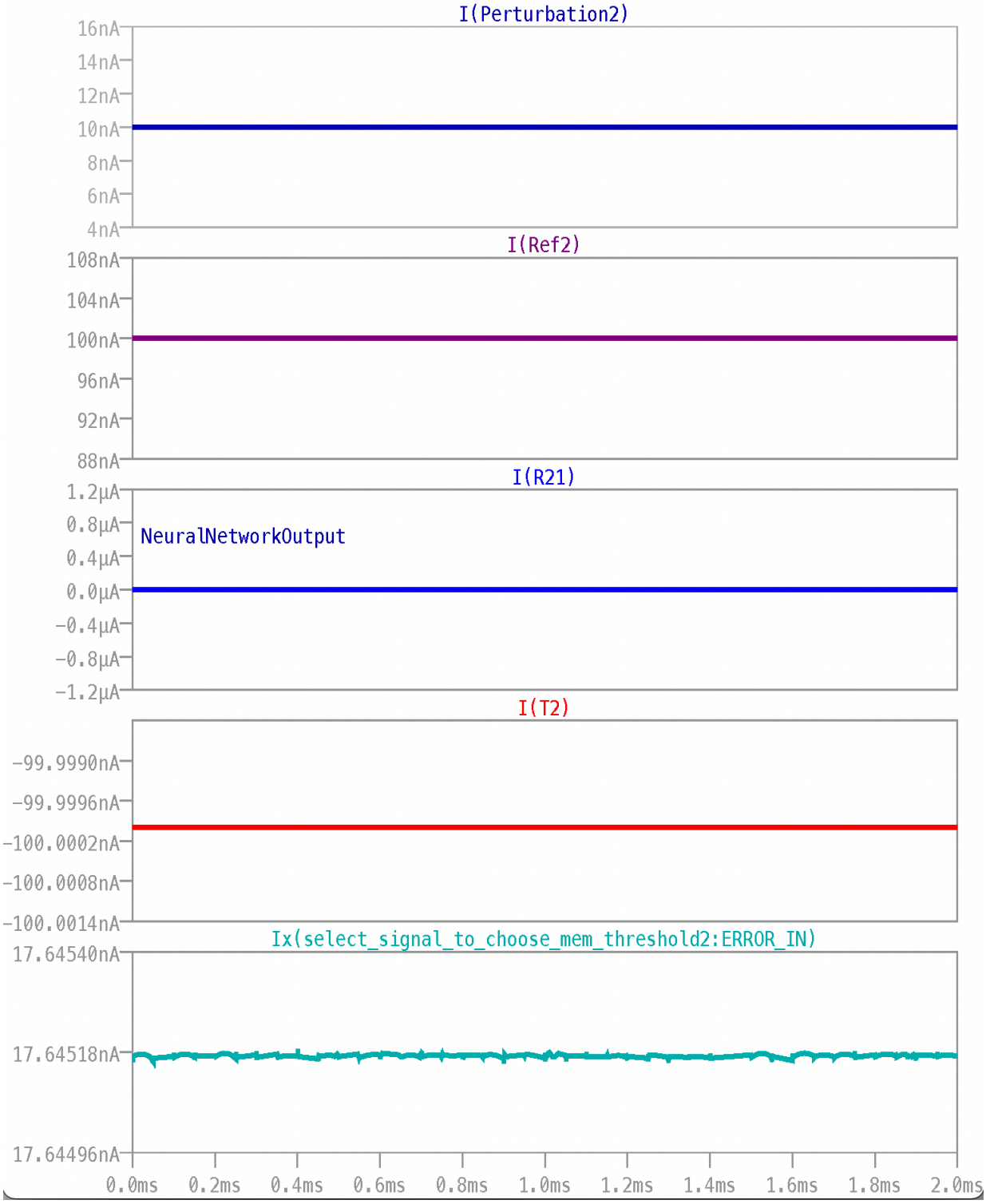


Figure 5.2: Error calculation circuit simulation results

Table I: Error calculation circuit test case data

Target Output	Actual Output	Perturbation	Reference Factor	Error		Error Percentage
100nA	0A	10nA	100nA	**Expected**	**Actual**	**15.95%**
				21nA	17.65nA	

The error calculation test case simulation waveform from Figure 5.2 and the test case data from table I confirms the correct functionality of the error calculation circuit with very minimal error in accuracy.

An error sign generator circuit was created to take in the error current and produce a certain voltage level based on the sign of the error current. This is basically a buffer circuit made of inverters with some feedback loop. In this training circuit, the error sign generator will accept a current input (the error current) and output a -1V if the error current is negative and +1V if the error current is positive. The error sign generator circuit is shown in Figure 5.3. The output of each inverter in the error sign generator is limited to between +1V and -1V by providing a +1V to the source of the PMOS and a -1V to the source of the NMOS. This ensures the final output of the error sign generator circuit after going through a series of inverters is either -1V or +1V based on the sign of the input current to the circuit.

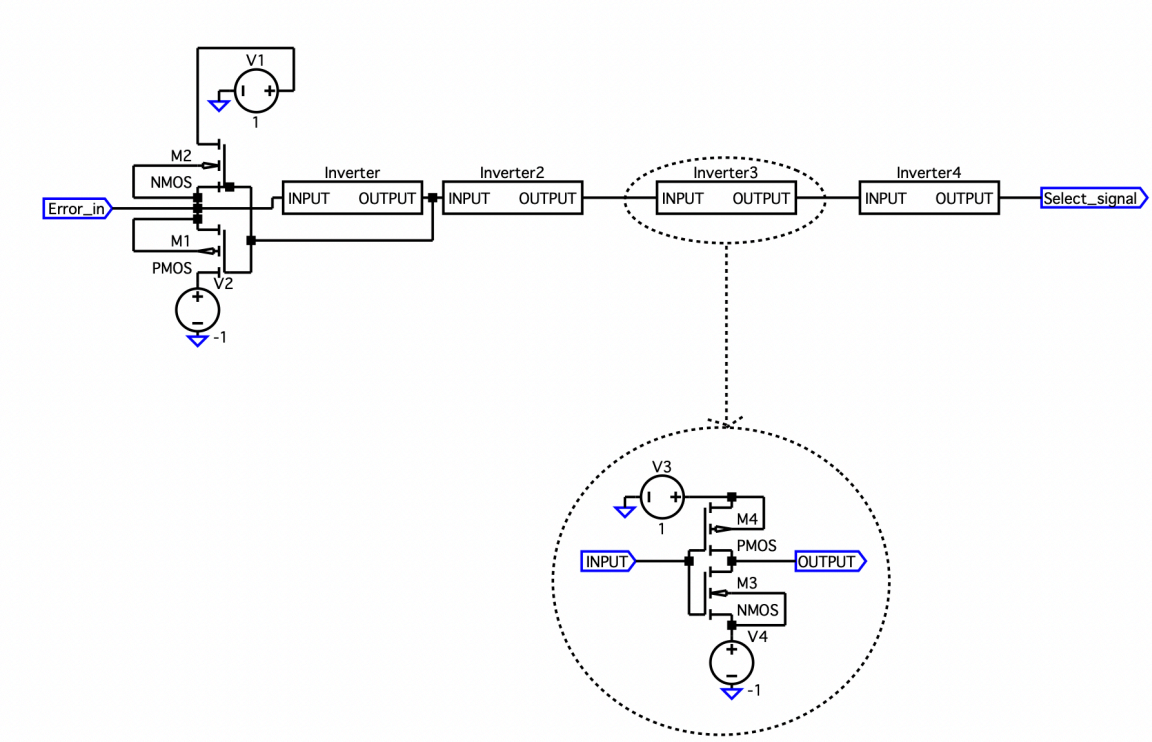


Figure 5.3: Error sign generator circuit

The test simulation waveform of the Error sign generator circuit is shown in Figure 5.4.

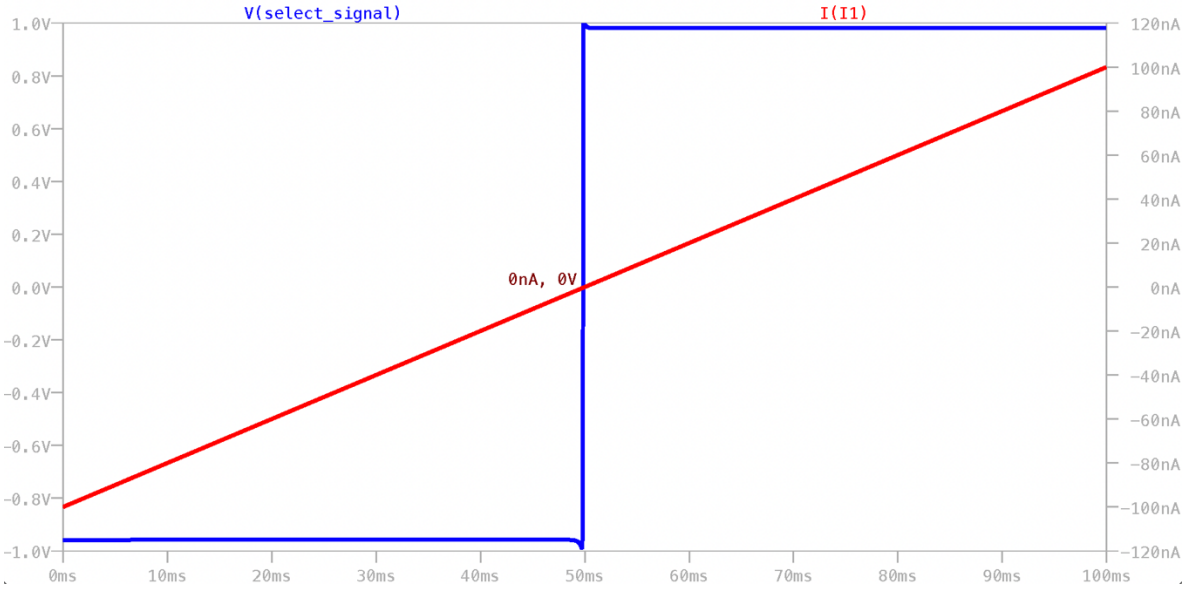


Figure 5.4: Error sign generator circuit simulation waveform

The input current to the error sign generator circuit goes from -100nA to 100nA over a period of 100ms during the test simulation. The output voltage of the error sign generator circuit stays at -1V when the input current was negative and shows a sudden transition to +1V when the input current switches to positive values. This test confirms the correct functionality of the error sign generator circuit. The only drawback of this circuit is when the input current is 0nA. The circuit could go to a metastable state during this transition if the input current ever becomes 0nA. The only possible way for this to happen is to get the error to 0, meaning there's no error. This could never happen in the training circuit due to the perturbation constant. If the perturbation constant is nonzero, there's always a tiny error which gets smaller and smaller after each cycle of training but could never get to 0. The feedback adjustment voltage is a constant in this approach as well. This results in the oscillation of the actual output around the target output, ensuring there will never be a zero-error case. Thus, the error sign generator circuit will function properly with the training circuit.

5.1.1 Voltage to current Converter

The output of the neuron model discussed in Figure 2.5 is a voltage. The current mode training circuit needs an output in terms of current. For this purpose, a simple voltage to current converter circuit shown in Figure 5.5 is used. This circuit consists of an op-amp, an NMOS transistor and a resistor. The positive terminal of the op-amp is connected to the output terminal of the neuron in the output layer of the neural network, the negative terminal of the op-amp is connected to the source of the NMOS transistor, and the output of the op-amp is connected to the gate of the NMOS transistor. The drain of the NMOS transistor is connected to a +1V voltage source and the source is then connected to a resistor. The ideal op-amp behavior sets the negative and positive terminals of an op-amp at equal voltage levels. Thus, the

voltage at the negative terminal of the op-amp will be the output voltage from the neural network. The neural network output voltage is limited to between +1V and -1V. The current flowing through the resistor will be the output current. This current can be manipulated by adjusting the resistance from Ohm's law. This circuit allows isolation of the neural network from the current voltage to current conversion process preventing backflow of current to the neural network when the output is negative.

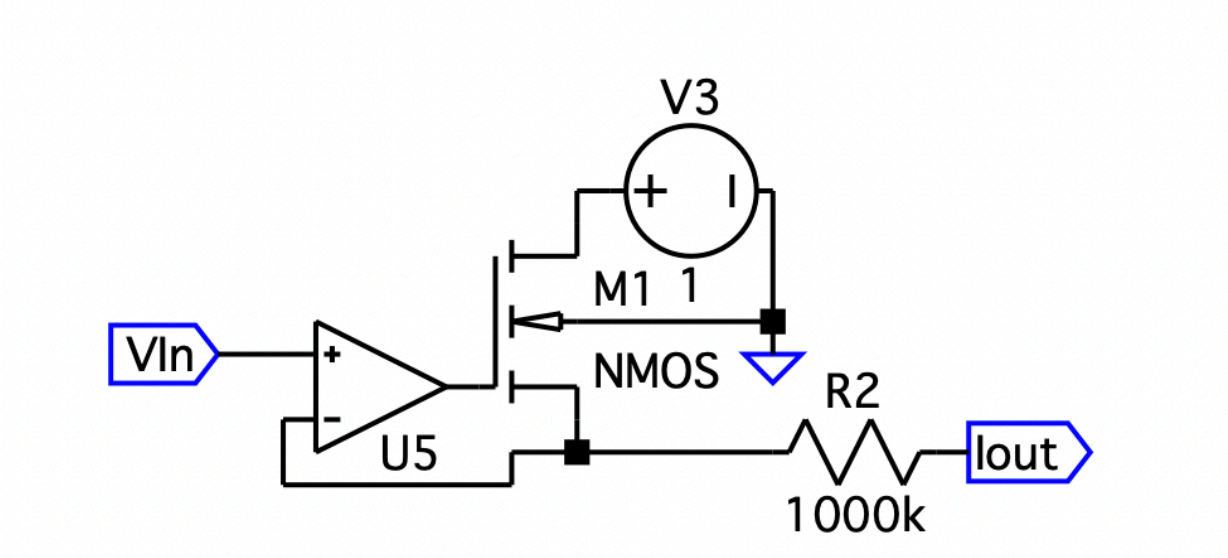


Figure 5.5: Voltage to current converter circuit

The simulation waveform of the voltage to current converter circuit is shown in Figure 5.6. A 10K resistor is used in the test simulation.

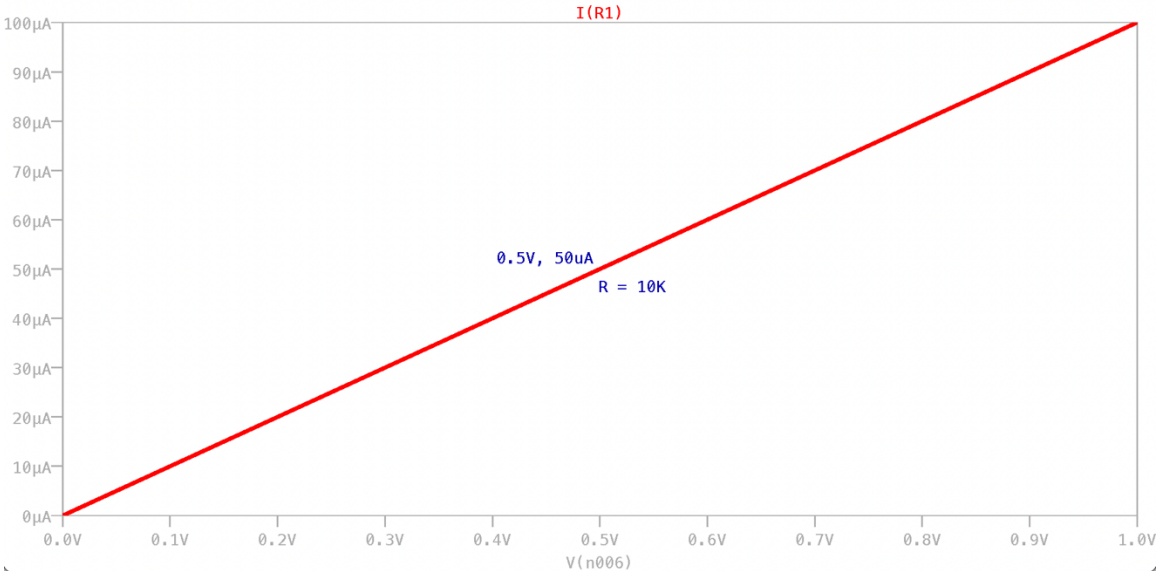


Figure 5.6: Voltage to current converter circuit simulation waveform

The voltage input to the circuit goes from 0V to 1V over a period of 100ms. The expected output of the circuit is from 0uA to 100uA respectively based on Ohm's law. The IV curve of the voltage to current converter shown in Figure 5.6 confirms that the actual output matches expected output. This confirms the correct functionality of the voltage to current converter circuit.

5.1.2 Current Mirror Circuits

The training circuit uses PMOS and NMOS current mirrors to create copies of current from one branch to another. The MOSFETs in the current mirror circuit operate in saturation. An NMOS and PMOS current mirror circuit is shown in fig22. Let the current I1 in be the input current in both current mirror circuits. The output current Iout for the PMOS current mirror flows through M2 and the output current Iout for the NMOS current mirror flows through M3. W and L are the widths and lengths of the MOSFETs respectively. The equation to calculate current through NMOS and PMOS is shown in (5.4) and (5.5) respectively.

$$I_{D_{sat}} = \left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W}{L}\right)(V_{GS} - V_{th})^2(1 + \lambda V_{DS}) \quad (\mathbf{5.4})$$

$$I_{D_{sat}} = \left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W}{L}\right)|(V_{GS} - V_{th})|^2(1 + \lambda V_{DS}) \quad (\mathbf{5.5})$$

Let all the lengths be equal and say L.

NMOS current mirror:

$$\frac{I_{Out}}{I_{in}} = \frac{\left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W_2}{L_2}\right)(V_{GS} - V_{th})^2(1 + \lambda V_{DS})}{\left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W_1}{L_1}\right)(V_{GS} - V_{th})^2(1 + \lambda V_{DS})}$$

$$\frac{I_{out}}{I_{in}} = \frac{\frac{W_2}{L_2}}{\frac{W_1}{L_1}}, but\ L_1 = L_2$$

$$\frac{I_{out}}{I_{in}} = \frac{W_2}{W_1}$$

$$I_{out} = I_{in}\left(\frac{W_2}{W1}\right) \quad (\mathbf{5.6})$$

PMOS current mirror:

$$\frac{I_{Out}}{I_{in}} = \frac{\left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W_2}{L_2}\right)|(V_{GS} - V_{th})|^2(1 + \lambda V_{DS})}{\left(\frac{1}{2}\right)(\mu C_{ox})\left(\frac{W_1}{L_1}\right)|(V_{GS} - V_{th})|^2(1 + \lambda V_{DS})}$$

$$\frac{I_{out}}{I_{in}} = \frac{\frac{W_2}{L_2}}{\frac{W_1}{L_1}}, but\ L_1 = L_2$$

$$\frac{I_{out}}{I_{in}} = \frac{W_2}{W_1}$$

$$I_{out} = I_{in}\left(\frac{W_2}{W1}\right) \tag{5.7}$$

In (5.6) and (5.7), if W_1 and W_2 are equal then I_{out} will be equal to I_{in}. Thus, in the current mirror design shown in Figure 5.7, a replica of the input current can be created on the output branch by simply setting the width and length of the MOSFETs equal. This technique is used in the training circuit to create copies of output and current and the target current to created isolated branches to introduce perturbation. A current mirror is also used to create copies of currents to feed into the multiplier to generate the square of the currents. The Translinear multiplier circuit itself uses multiple current mirror circuits to isolate each current branch.

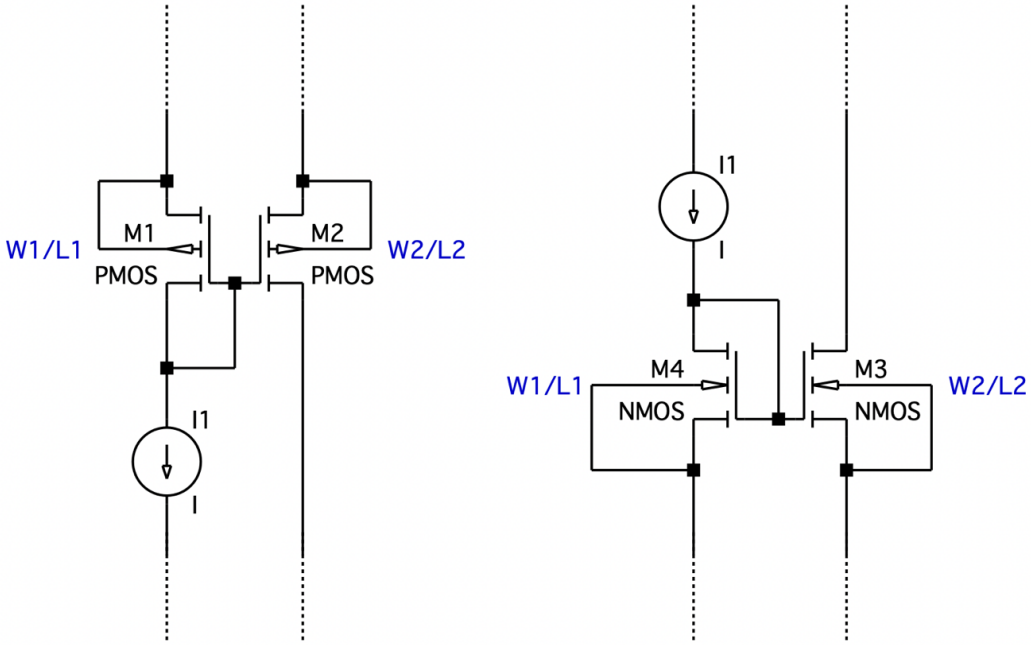


Figure 5.7: PMOS and NMOS current mirror circuit

Current mirrors are a common circuit configuration used in analog electronics to replicate a current from one source to another. While they have several advantages, they also come with a set of disadvantages. Current mirrors can be sensitive to temperature variations, which can lead to changes in the replicated current. This is because the transistors used in the mirror can have varying temperature coefficients that affect their behavior. This can be a significant concern in precision applications. Integrated circuits are subject to manufacturing process variations, which can lead to differences in transistor characteristics even within the same batch. These variations can affect the accuracy and matching of the current mirror, leading to potential performance deviations. Current mirrors often require a certain voltage headroom for proper operation. This means that the voltage difference between the supply voltage and the transistor's threshold voltage must be sufficiently large. This can limit their applicability in low-voltage or low-power circuits. In some current mirror configurations, the output current might not be able to reach the maximum value allowed by the power supply voltage and the transistors' characteristics. This limitation can restrict the usable range of output currents. Transistors used in current mirrors might not exhibit ideal behavior, especially in non-ideal operating conditions. This can lead to deviations from the expected mirrored current, reducing the accuracy of the circuit. Achieving high accuracy and precision in current mirrors often requires additional circuitry for compensation, trimming, or calibration. This can increase the complexity and cost of the design. Some current mirror configurations might have limitations in the range of current ratios that they can accurately replicate. Deviations from the designed current ratio can occur, particularly at extreme ratios. Current mirrors might not perform well in

dynamic conditions or in applications where rapid changes in the mirrored current are required. The transient response of the mirror can be slower due to the inherent capacitances and time constants in the transistor's operation. Despite these disadvantages, current mirrors remain widely used in analog circuit design due to their simplicity, ease of integration, and cost-effectiveness. Designers often balance these drawbacks against the advantages offered by current mirrors in their specific applications.

Transistor matching in current mirrors is a critical concern, as it directly affects the accuracy and performance of the circuit. One of the most effective ways to mitigate transistor matching issues is to use transistors that are fabricated using the same process and are designed to have closely matched characteristics. This can be achieved through techniques like device scaling, layout symmetry, and using transistors from the same wafer. Implement temperature compensation techniques to counteract the temperature sensitivity of current mirrors. This can involve adding temperature sensors to monitor temperature variations and applying compensation schemes to adjust the mirror's operation accordingly. Feedback techniques can be employed to dynamically adjust the current mirror's output based on the actual mirrored current. This can help compensate for any discrepancies caused by transistor mismatch by actively correcting the output. Cascode configuration is another solution. Using a cascode configuration involves connecting an additional transistor in series with the mirror transistors. This can help mitigate the effect of early voltage variations and reduce the dependence on the individual transistor characteristics. Quadrature current mirrors can also help with the accuracy issues as they use two or more branches of mirror transistors that replicate a reference current in parallel.

The output current is then taken as the sum of these replicated currents. This technique can help improve matching by averaging out the mismatch effects. circuit techniques that counteract the effects of process variations can also help with accuracy. This might involve adding extra transistors or components that adjust the mirrored current based on detected process variations. Simulation waveform of the PMOS and NMOS current mirror circuits are shown in Figure 5.8 and Figure 5.9 respectively.

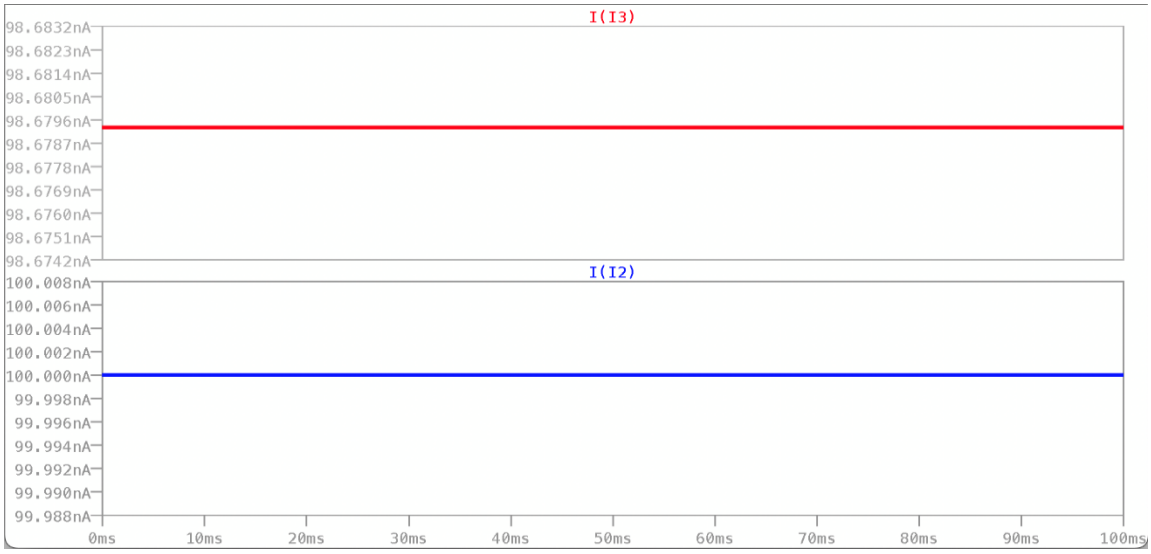


Figure 5.8: PMOS current mirror circuit simulation waveform

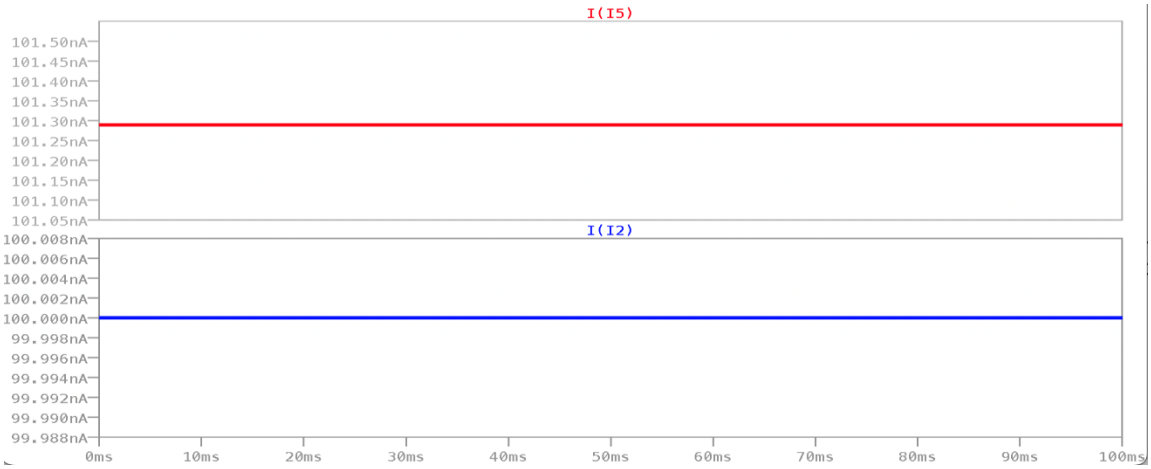


Figure 5.9: NMOS current mirror circuit simulation waveform

The Input current to both PMOS and NMOS current mirror circuits was 100nA and expected output of both current mirror circuits was 100nA. However, the PMOS and NMOS current mirror circuit output was 98.6796nA and 101.29nA respectively. The outputs are not exactly 100nA but very close to 100nA with an error margin of about 1.3%. This negligible error margin is acceptable and thus the current mirrors are used in the training circuit to create isolated replicas of current from one branch to other.

5.1.3 Translinear Multiplier Circuit

To implement (5.1) and (5.2), a translinear multiplier circuit is used to take the square of the currents. The translinear multiplier will accept three inputs and generate an output according to (4.2). The translinear circuit design used in the error calculation circuit is shown in Figure 5.10. The translinear multiplier circuit in Figure 5.10 is a modified version from Figure 4.2. The working principle is the same, but the new design introduces three NMOS current mirrors and two PMOS current mirrors. The design from Figure 4.2 works fine by itself. However, when connecting it with other analog circuit components in the error calculation circuit, the translinear circuits break due to leakage currents. To prevent such leakage currents an NMOS current mirror is added to each input to the translinear multiplier to create isolated replicas of input currents. Similarly, a PMOS current mirror is added to the output current path of the translinear multiplier to create an isolated copy of the output current. These current mirrors prevent leakage current from going in and out of the translinear multiplier circuit and thereby ensuring the correct functionality of the multiplier and other circuits in the error calculation circuit. A side-by-side comparison of the old and new translinear multiplier circuit is shown in Figure 5.11. The current sources in the old design represent the input source to the multiplier circuit.

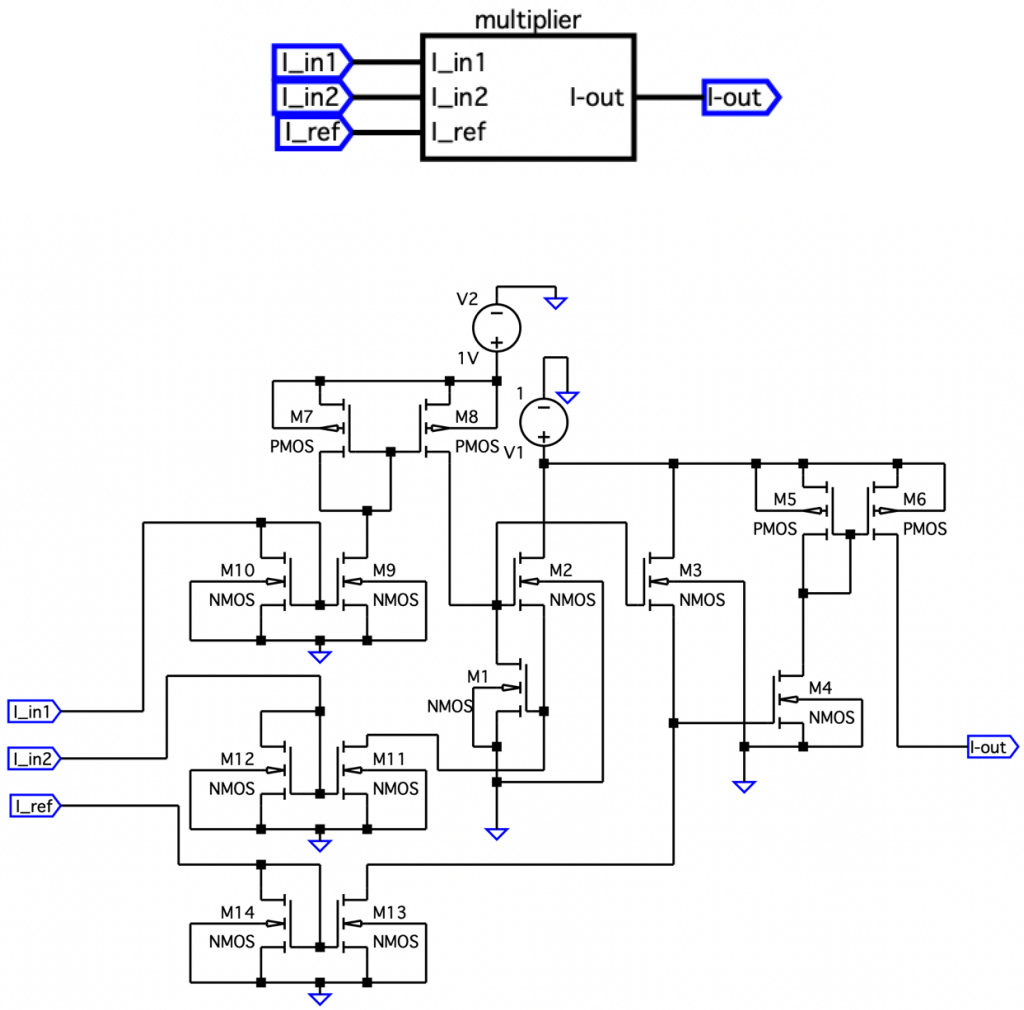


Figure 5.10: Translinear multiplier circuit

The translinear multiplier circuit in Figure 5.10 is a modified version from Figure 4.2. The working principle is the same, but the new design introduces three NMOS current mirrors and two PMOS current mirrors. The design from Figure 4.2 works fine by itself. However, when connecting it with other analog circuit components in the error calculation circuit, the translinear circuits break due to leakage currents. To prevent such leakage currents an NMOS current mirror is added to each input to the translinear multiplier to create isolated replicas of input currents. Similarly, a PMOS current mirror is added to the output current path of the translinear multiplier to create an isolated copy of the output current. These current mirrors prevent leakage current from going in and out of the translinear multiplier

circuit and thereby ensuring the correct functionality of the multiplier and other circuits in the error calculation circuit. A side-by-side comparison of the old and new translinear multiplier circuit is shown in Figure 5.11. The current sources in the old design represent the input source to the multiplier circuit.

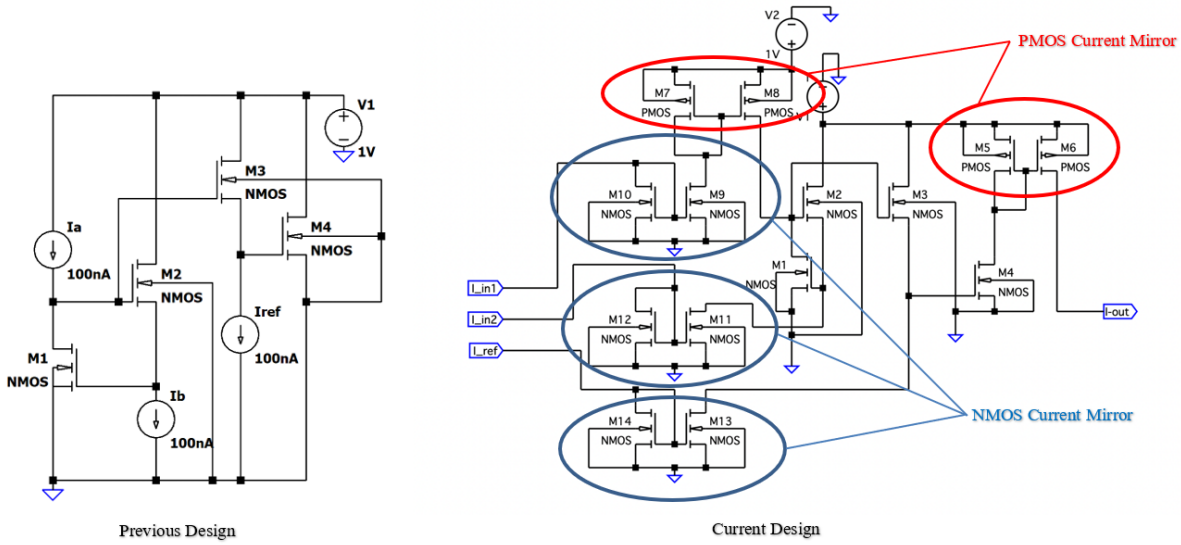


Figure 5.11: Translinear multiplier circuit comparison

A major drawback of this circuit is that it can only handle positive currents. A negative current would break the circuit functionality. This issue was resolved by adding an absolute value current circuit before the multiplier. Thus, the current provided to the multiplier will always be positive. The addition of absolute value current circuit before the multiplier will not break the error calculation computation as the square of a positive and negative number results in a positive number.

The simulation waveform of the translinear multiplier is shown in Figure 5.12. The inputs to the translinear multiplier circuit for this test case were set to 30nA and 50nA, the reference factor was 20nA.

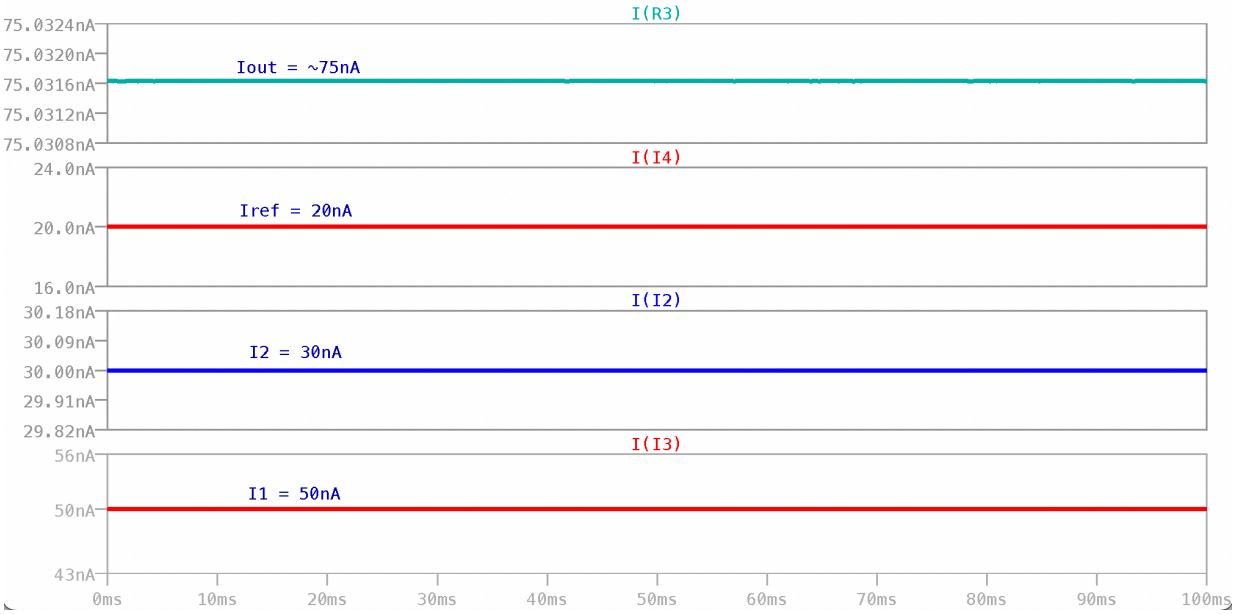


Figure 5.12: Translinear multiplier circuit simulation waveform

The expected output of the translinear multiplier circuit is 75nA (4.2). Figure 5.12 shows that the actual output of the translinear multiplier circuit is 75.0317nA. The actual output is very close to the expected output and the 0.04% error is negligible. Thus proves the new translinear multiplier circuit design with current mirrors works properly as expected.

5.1.4 Absolute Value Current Circuit

The absolute value circuit takes in a current input in flowing in either direction (positive or negative current) and outputs a positive current (current always flows out of the circuit). Current mode absolute value circuit is shown in Figure 5.13.

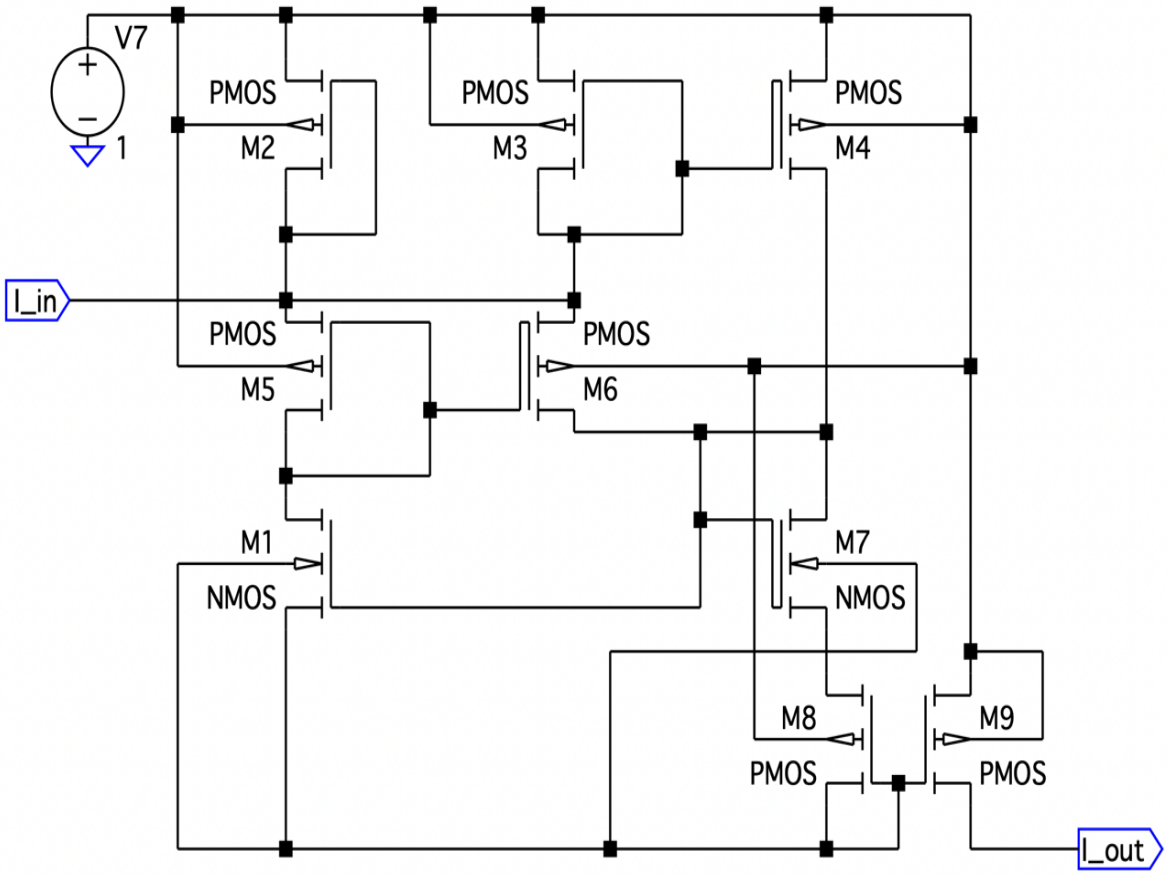


Figure 5.13: Current mode absolute value circuit

The simulation waveform of current mode absolute value circuit is shown in Figure 5.14. the input current ranging from -100nA to 100nA was fed into the current mode absolute value circuit and expected output is the absolute value of the input current at any given time.

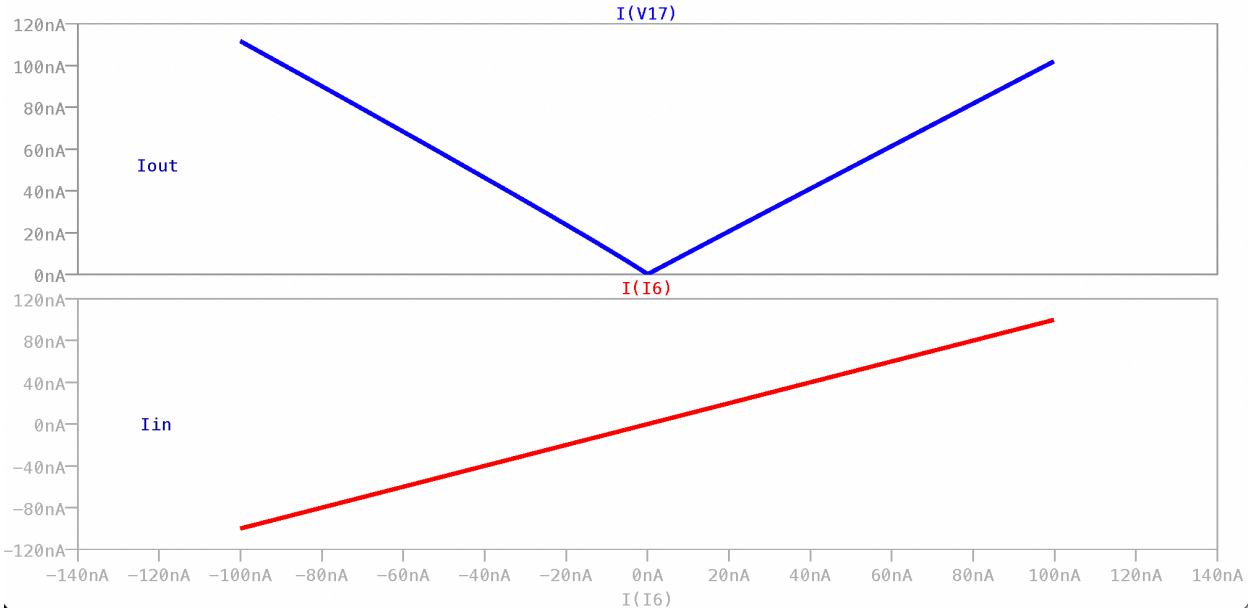


Figure 5.14: Current mode absolute value circuit simulation waveform

From Figure 5.14, as the input current went from -100nA to 0nA, the output went from 100nA to 0nA. the output then went from 0nA to 100nA as the input did. The output current from the current mode absolute value circuit was always positive and hence confirms the correct functionality of the circuit.

5.1.5 Sample and Hold Circuit

A sample and hold circuit is used to create samples of voltages at the output node from the given inputs to the circuit by holding the samples for a definite time. The period during which the circuits create the sample from the input signal is called the sampling time. The Sample and hold circuit design used in the training circuit is a very basic CMOS sample and hold circuit. It basically consists of an NMOS, PMOS and a capacitor that holds the sample voltage. The CMOS switch is controlled by a hold control signal which is a clock pulse signal. The clock pulse in this training circuit is the control signal that determines whether the operation is forward

propagation or feedback voltage adjustment based on the control signal level. +1V indicates forward propagation and a -1V indicates the training process. The sampling phase occurs when the control signal is+1V. During the sampling phase, the CMOS switch will be closed and allowing the capacitor to charge up to the sample input voltage fed into the circuit. The charging time is kept short such that charging can be completed before the end of the sampling control signal. After the sampling phase, the sampling control signal is deactivated, thereby opening the CMOS switch which in turn disconnects the capacitor from the sample circuit. The hold time is also small, and the training process is completed before the capacitor discharges the sample voltage. The sample and hold circuit are shown in Figure 5.15.

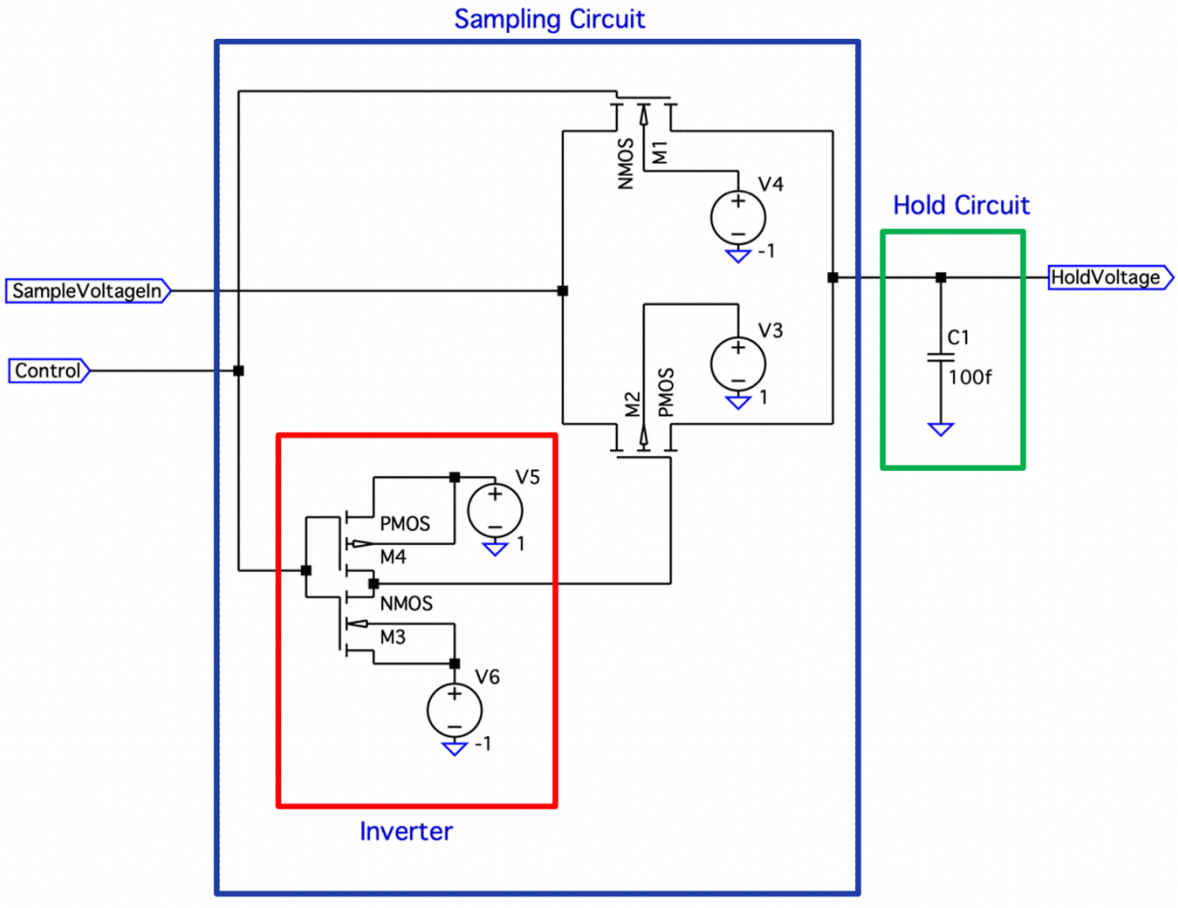


Figure 5.15: CMOS sample and hold circuit.

The simulation waveform of the sample and hold circuit is shown in Figure 5.16. A sine waveform of amplitude 2V is sent to the sample and holds circuit as the input and the control signal is a clock pulse signal. In Figure 5.16, the capacitor output signal (sample and hold output) is charging up during the sampling phase and holds the sample voltage during the hold phase. i.e., the capacitor charges during the sampling phase (when control signal is +1V) to the actual input voltage and holds the last input that it sees before the control signal flips to -1V (hold phase) until the control signal flips back to the sampling phase.

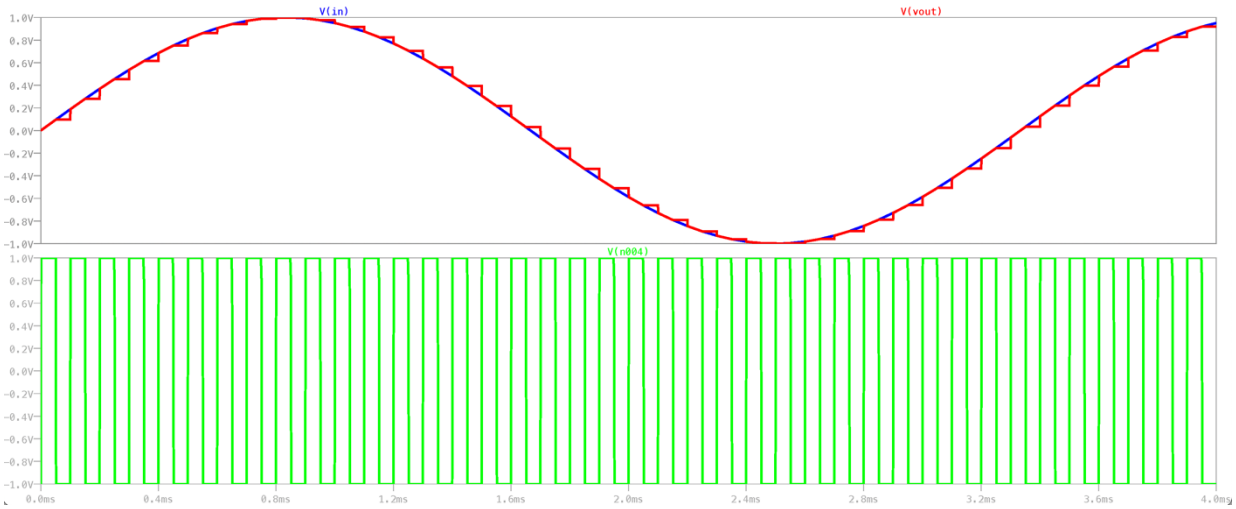


Figure 5.16: CMOS sample and hold circuit simulation waveform.

5.2 The Memristor Threshold Select Signal Generator

In the neural network, the output y can be calculated using (5.8).

$$y = f\sum_{i=1}^{n} \mathrm{w_i} x_i \qquad \textbf{(5.8)}$$

From (5.8), X_i is the input of the network, wi is the weight of the network, f is the nonlinear activation function, and t is the target value of output. In WSSP algorithm, a

small positive constant perturbation signal p is added to the weighted sum s, and the perturbation affects the output and error. The detailed calculation process of WSSP algorithm as follows. First, the error without perturbation is calculated. Second, a perturbation signal is applied to the weighted sum, and the error with a perturbation is calculated. Finally, the weight is updated according to the difference between unperturbed and perturbed error function. As a consequence, the weight update rule is shown in (5.9)

$$\Delta w_i = -\alpha \frac{\partial E}{\partial w_i} = -\alpha \frac{\partial E}{\partial s} \frac{\partial s}{\partial w_i} = -\alpha \frac{\partial E}{\partial s} x_i \quad \textbf{(5.9)}$$

$$w_i(m+1) = w_i(m) + \Delta w_i \quad (\mathbf{5.10})$$

From (5.10),

$$\Delta w_i = -\alpha \frac{\partial E}{\partial s} x_i = -\alpha \frac{\Delta E}{\Delta s} x_i = -\alpha \frac{\Delta E}{p} x_i \quad \textbf{(5.11)}$$

and α is the learning rate, p is a positive constant, E_{per} is the error with a perturbation, E is the error without a perturbation. From (5.11), if the change in error and the input to a synapse has same sign (both +'ve or both – 've), then the change in weight will be negative and the synapse weight must decrease. Similarly, if the change in error and the input to a synapse has opposite sign, then the change in weight will be positive and the synapse weight must increase. Based on the input voltage sign and the error sign, there are four different possible combinations to select the feedback adjustment voltage from. They are shown in Table II.

Table II: Feedback adjustment / Weight update selection

Input voltage sign	Error current sign	Memristance top memristor	Memristance bottom memristor	Feedback adjustment voltage sign	Weight update
-	-	Increase	Decrease	-	Decrease
-	+	Decrease	Increase	+	Increase
+	-	Decrease	Increase	+	Increase
+	+	Increase	Decrease	-	Decrease

A constant feedback adjustment voltage less than the negative threshold voltage of the memristor and a constant feedback adjustment voltage above the positive threshold voltage of the memristor is used for updating weights. An XOR gate can implement the feedback adjustment voltage selection based on the conditions from table II. The input voltages to the neural network range between -1V and +1V. A buffer called input sign generator circuit made up of inverters is used to generate either -1V or +1V based on the sign of the input voltage. The outputs of the error sign generator circuit and the input sign circuit is fed into the XOR gate and the output of the XOR gate will be used as a select signal to a multiplexer that selects positive or negative feedback adjustment voltage based on the select signal. If the select signal is -1V the multiplexer feeds a negative feedback adjustment voltage into the synapse and vice-versa. This feedback adjustment voltage will be higher than the positive threshold voltage or lesser than the negative threshold voltage of the memristor. For this training circuit, it is set to either -1V or +1V and the threshold voltage of the memristor model used in the neural network discussed in this paper is +/- 0.7V. The memristor threshold select signal generator takes the sign of the error produced, the positive and negative threshold voltages of the memristor and the input sign as inputs and gives out the feedback adjustment voltage based on the conditions from table II. If the input conditions demand the feedback adjustment sign to be negative, the memristor threshold select signal generator outputs the negative feedback adjustment voltage (-1V) in this circuit setup and vice-versa. The high-level diagram of the memristor threshold select signal generator is shown in Figure 5.17 and the breakdown of each sub circuit is shown in Figure 5.18, 5.20 and 5.22.

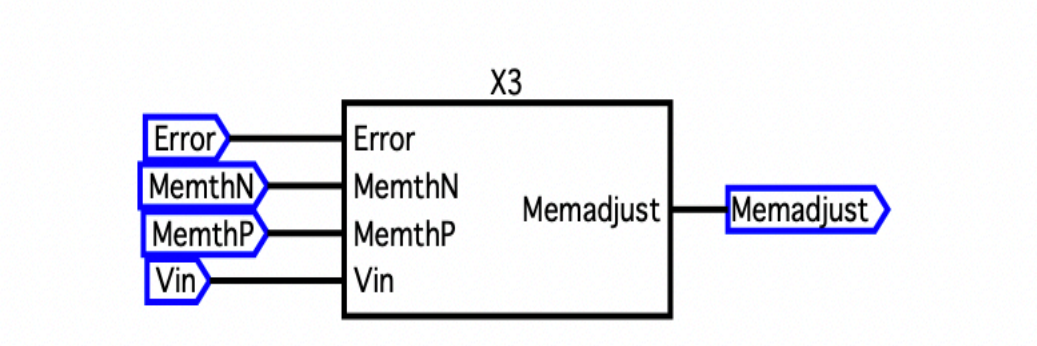


Figure 5.17: memristor threshold select signal generator

The memristor threshold select signal generator consists of input sign generator circuit, Feedback adjustment voltage select signal generator and a multiplexer to choose between the positive and negative memristor threshold voltages.

5.2.1 Input Sign Generator Circuit

The input sign generator circuit is basically a buffer circuit created with two inverters in series. This circuit determines the sign of the input signal by generating a +1V output when the input voltage is positive and a -1V when the input voltage is negative. The circuit works as intended for all input voltages except a 0V input. The circuit will break if a 0V input is sent to it as it may enter the metastable state due to discontinuity in the output at 0V. The input sign generator circuit is shown in Figure 5.18.

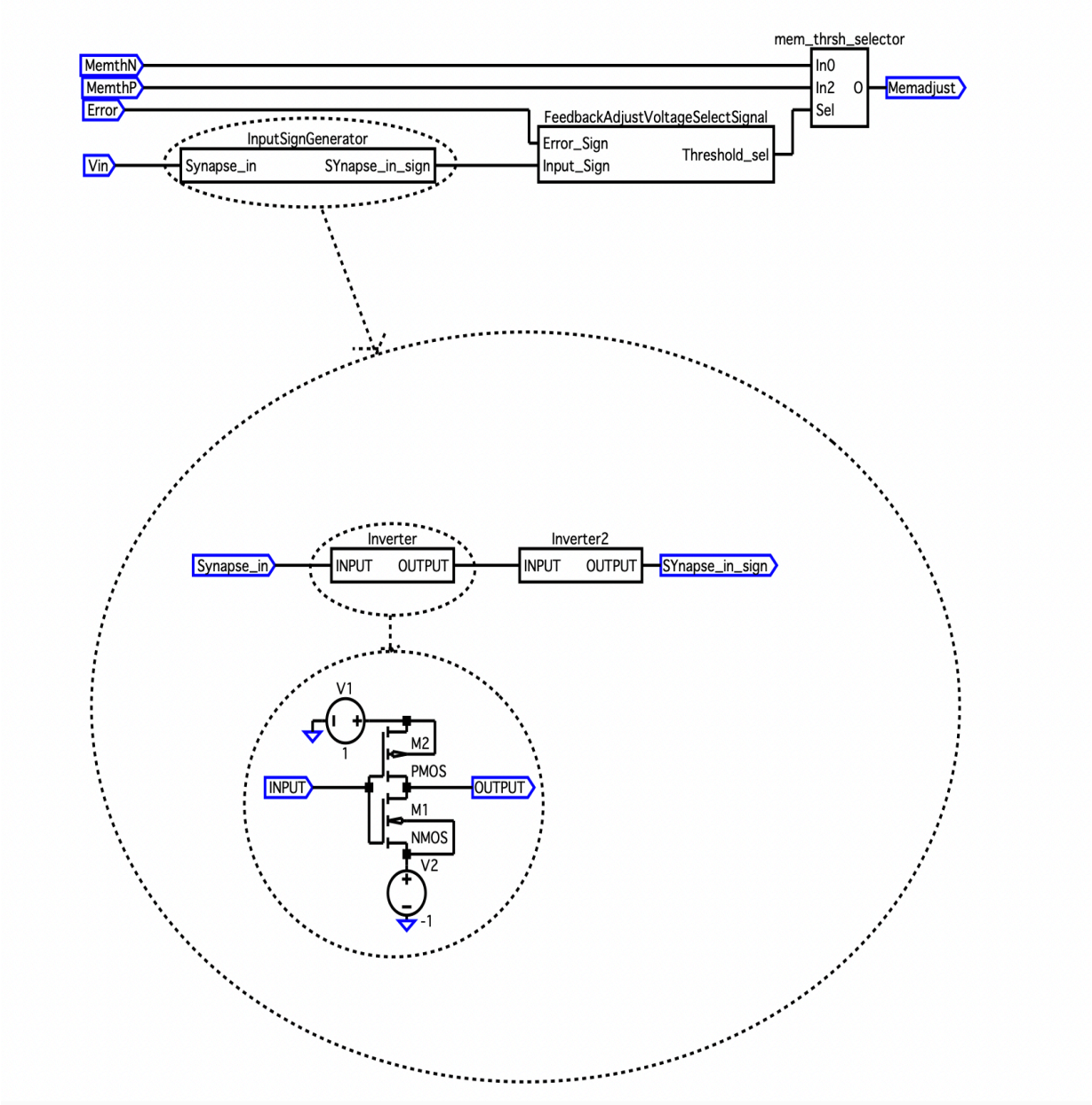


Figure 5.18: Input sign generator circuit

The simulation waveform of the input sign generator circuit is shown in Figure 5.19. An input voltage signal ranging between -1V to +1V is provided to the input terminal of the input sign generator circuit to test its functionality.

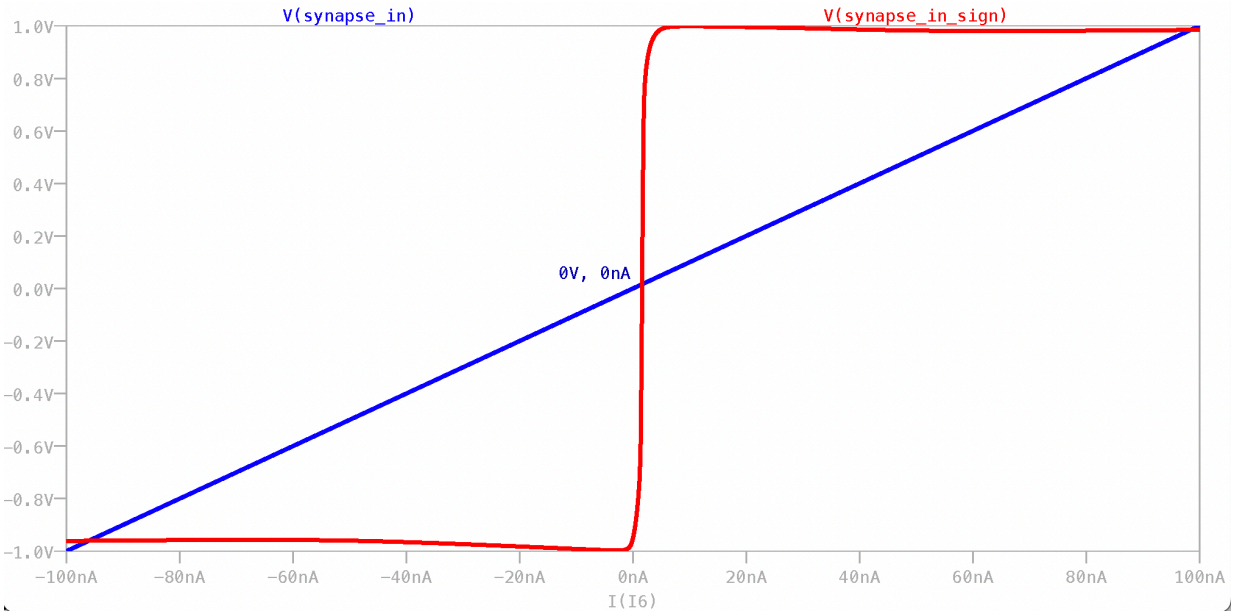


Figure 5.19: Input sign generator circuit simulation waveform

From Figure 5.19, the output voltage of the input sign generator circuit stays at -1 V for all negative voltage inputs and switches to +1V for all positive voltage inputs. Thus, a +1V output from input sign generator circuit indicates that the sign of the input voltage is positive and vice-versa.

5.2.2 Feedback Adjustment Voltage Select Signal Generator

Feedback adjustment voltage select signal generator accepts the error sign and the input sign as inputs and generates a +1V or -1V at the output depending on the input combination based on the conditions from table II. An XOR gate is required to implement the conditions in table II. If the error and input voltage has same sign, the feedback adjustment voltage select signal generator will output a -1V and if the error and input voltage has opposite sign, the output will be a -1V. This output voltage will be used as a select signal to choose between the positive and negative feedback

adjustment voltages. The feedback adjustment voltage select signal generator circuit is shown in Figure 5.20.

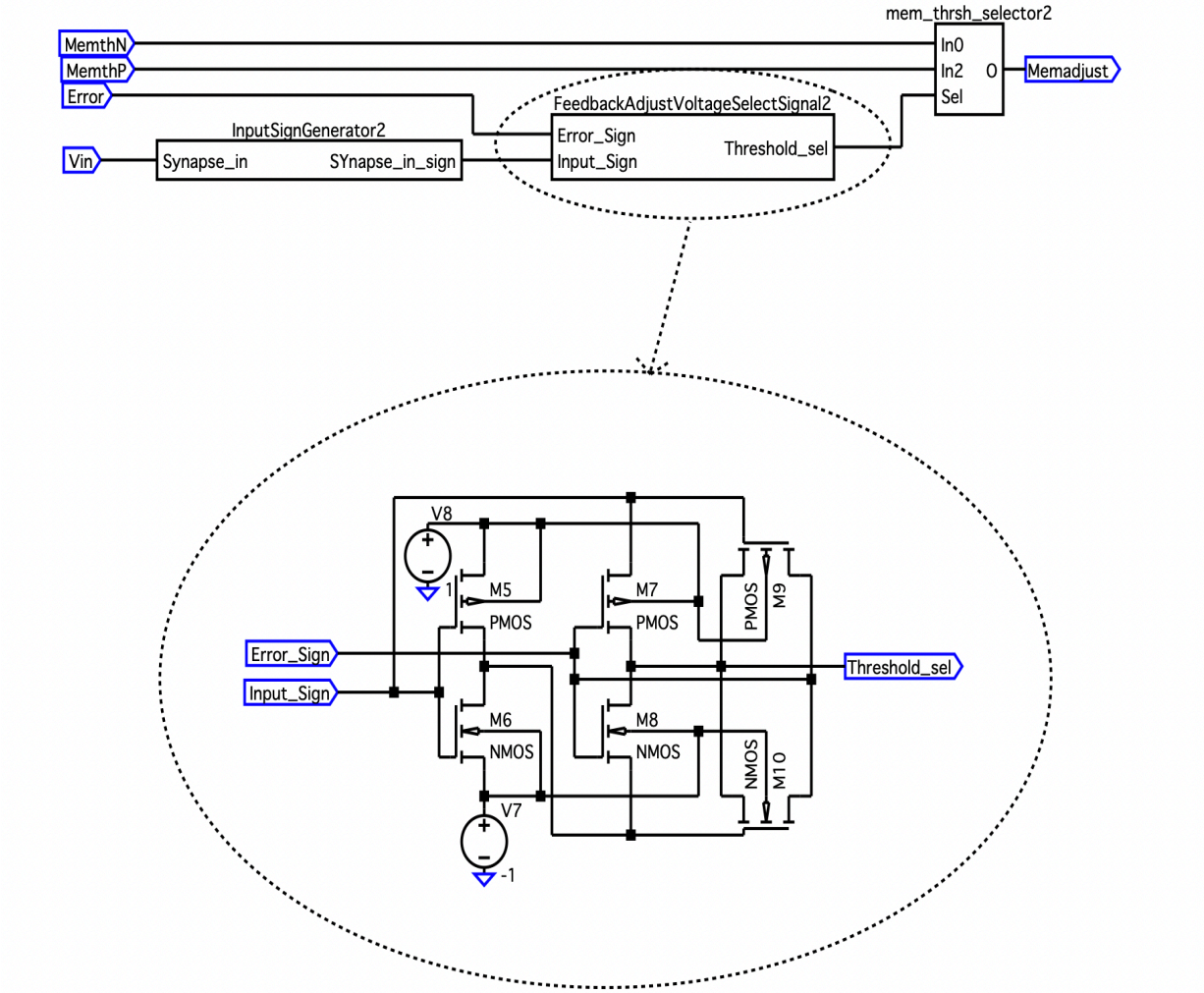


Figure 5.20: feedback adjustment voltage select signal generator circuit

The feedback adjustment voltage select signal generator circuit is tested by providing all possible input combinations of the error sign and input sign as inputs. The simulation waveform of the feedback adjustment voltage select signal generator circuit is shown in Figure 5.21.

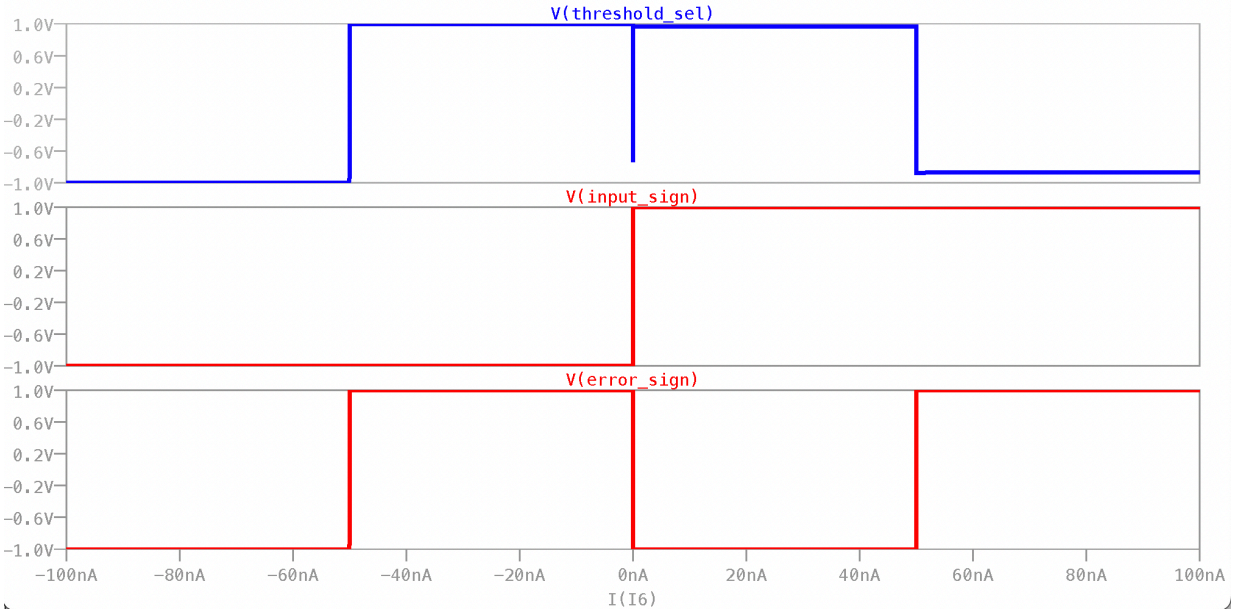


Figure 5.21: Feedback adjustment voltage select signal generator circuit simulation waveform

From Figure 5.21, the output signal (threshold_sel) stays at +1V only when the input and error have opposite signa and the output goes to -1V when input and error have same sign. Thus, shows the correct functionality.

5.2.3 Feedback Adjustment Voltage Selector

The feedback adjustment voltage selector is basically a multiplexer circuit that chooses between the positive and negative feedback adjustment voltages based on the select signal. The select signal input to this circuit is the output from Feedback adjustment voltage select signal generator circuit. The feedback adjustment voltage selector circuits output the positive feedback adjustment voltage if the select signal is +1V and vice versa. The feedback adjustment voltage selector circuit is shown in Figure 5.22.

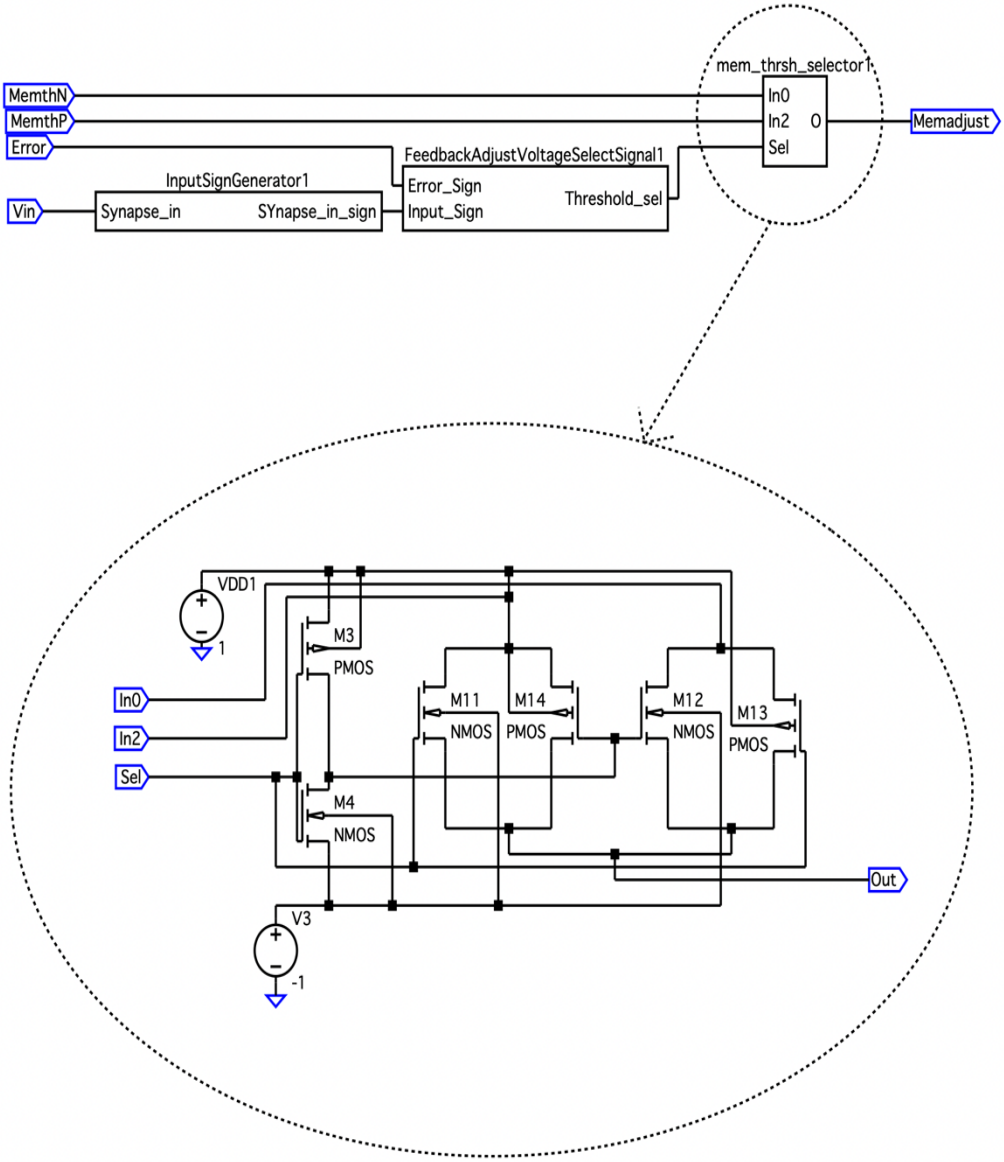


Figure 5.22: The feedback adjustment voltage selector circuit

The feedback adjustment voltage select signal generator circuit is tested by setting feedback adjustment voltages as +0.7V and -0.7V, and the select signal pulse of -1V and +1V. The feedback adjustment voltage selector circuit simulation waveform is shown in Figure 5.23.

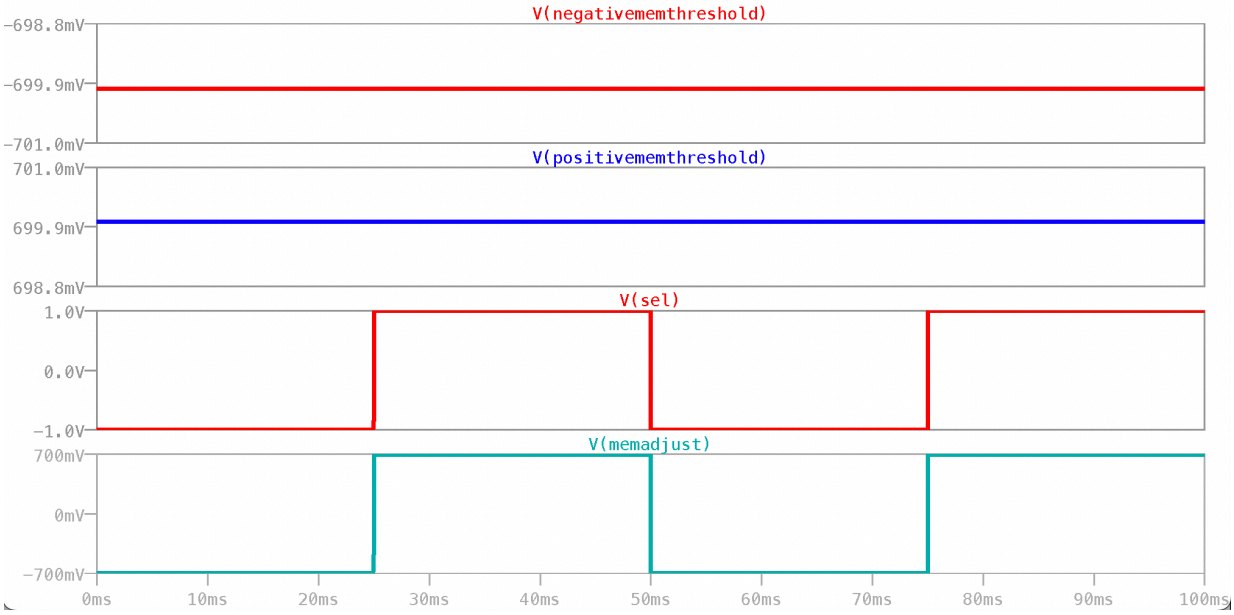


Figure 5.23: The feedback adjustment voltage selector circuit simulation waveform

From Figure 5.23, the output voltage (Vmemadjust) is -0.7V when the select signal is -1V and the +0.7V when the select signal is +1V. This confirms the correct functionality of the feedback adjustment voltage selector circuit.

5.3 Synapse Input select Circuit

The synapse input select circuit is basically a multiplexer circuit that chooses between the feedback adjustment voltage and the actual input to the synapse based on the select signal and determines which signal to pass through to the synapse input. This circuit accepts the feedback adjustment voltage, input voltage to the synapse and the training control signal as input and outputs the appropriate signal based on the control signal. This circuit provides the feedback adjustment voltage to the synapse during the feedback adjustment process (when the training control signal is -1V) and the actual synapse input during the forward propagation (when the training control

signal is +1V). The feedback adjustment voltage selector circuit is reused as the synapse input select circuit as both circuits require the exact same functionality.

5.4 The Training Circuit

The completed training circuit consists of a sample and hold circuit, error calculation circuit, memristor threshold voltage selector circuits and synapse input selector circuits. The sample and hold circuit hold the neural network output and provides it to the error calculation circuit during the feedback voltage adjustment process such that the error calculation circuit calculates the error based on the actual output, target output, perturbation and the reference multiplication factor. The memristor threshold voltage selector circuits use this error sign, the actual synapse input voltage sign and the feedback adjustment voltages to choose the sign of the feedback adjustment voltage to be sent to the synapse. The synapse input selector circuits provide the feedback adjustment voltage to the synapse during the feedback adjustment process and the actual synapse input during the forward propagation. The advantage of this circuit design is that it uses minimal circuit components to implement the zeroth order optimization technique as opposed to the complex circuit design for implementing other training algorithms. The training circuit needs only one sample and holds circuit and error calculation circuit whereas the number of memristor threshold voltage selector circuits and synapse input selector circuits depend on the number of total synapses. As the neural network gets bigger, the training circuit also grows in terms of memristor threshold voltage selector circuits and synapse input selector circuits. The training circuit block diagram is shown in Figure 5.24.

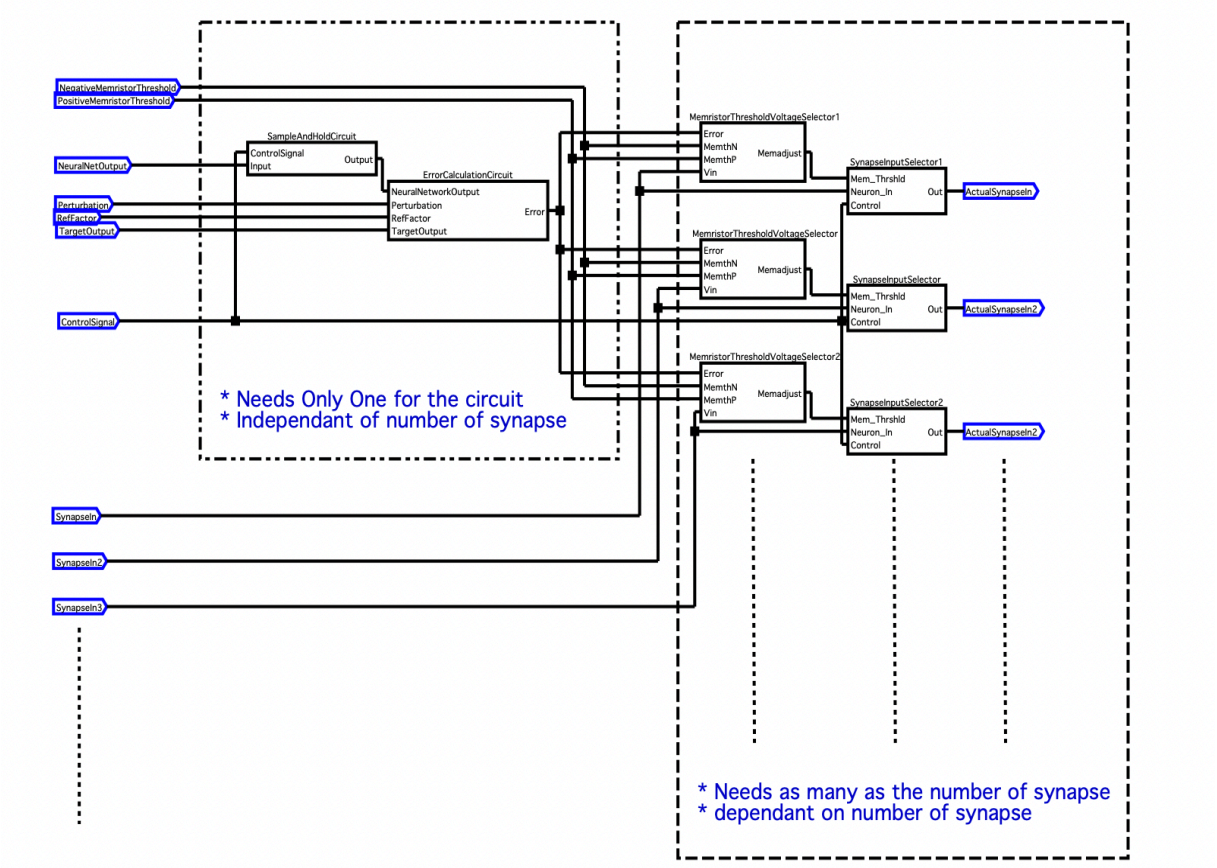


Figure 5.24: The Training Circuit

The functionality of the training circuit is tested by creating all four possible input combination scenarios from table II on a single synapse in the neural network. The simulation waveform of the training circuit is shown in Figure 40. While testing the functionality the input choices were +0.5V and target output choices were -100nA and 100nA. The actual output for all the test cases was 0nA. This is because the testing and data collection was done at the very first forward propagation cycle before any training happens. The constant feedback adjustment voltage was set to -800mV and +800mV which is beyond the +/-700mV threshold voltage of the memristor model used to test the training circuit. The perturbation constant was kept at 10nA for all test cases.

Scenario 1: Negative input and negative error

In this test case, the input to the synapse of interest was -500mV and the target output was 100nA. The error was -16.88nA and hence the expected error sign indicator voltage is -1V. From Figure 5.25a, the actual error sign indicator voltage that goes into the feedback adjustment voltage select signal generator circuit (Verrorsign) is -957mV. Based on table II, the expected feedback adjustment voltage that goes into the synapse input is -800mV and the actual feedback adjustment voltage from Figure 5.25a is -794mV. Thus, satisfying the first case in table II.

Scenario 2: Positive input and negative error

In this test case, the input to the synapse of interest was 500mV and the target output was 100nA. The error was -16.88nA and hence the expected error sign indicator voltage is -1V. From Figure 5.25b, the actual error sign indicator voltage that goes into the feedback adjustment voltage select signal generator circuit (Verrorsign) is -914mV. Based on table II, the expected feedback adjustment voltage that goes into the synapse input is 800mV and the actual feedback adjustment voltage from Figure 5.25b is 796mV. Thus, satisfying the third case in table II.

Scenario 3: Positive input and positive error

In this test case, the input to the synapse of interest was 500mV and the target output was 100nA. The error was 17.74nA and hence the expected error sign indicator voltage is 1V. From Figure 5.25c, the actual error sign indicator voltage that goes into the feedback adjustment voltage select signal generator circuit (Verrorsign) is 959.97mV. Based on table II, the expected feedback adjustment voltage that goes

into the synapse input is -800mV and the actual feedback adjustment voltage from Figure 5.25c is -797mV. Thus, satisfying the fourth case in table II.

Scenario 4: Negative input and positive error

In this test case, the input to the synapse of interest was -500mV and the target output was 100nA. The error was 17.65nA and hence the expected error sign indicator voltage is 1V. From Figure 5.25d, the actual error sign indicator voltage that goes into the feedback adjustment voltage select signal generator circuit (Verrorsign) is 979.95mV. Based on table II, the expected feedback adjustment voltage that goes into the synapse input is 800mV and the actual feedback adjustment voltage from Figure 5.25d is 795mV. Thus, satisfying the second case in table II. Figure 5.25 confirms the correct functionality of the training circuit satisfying all possible scenarios from table II.

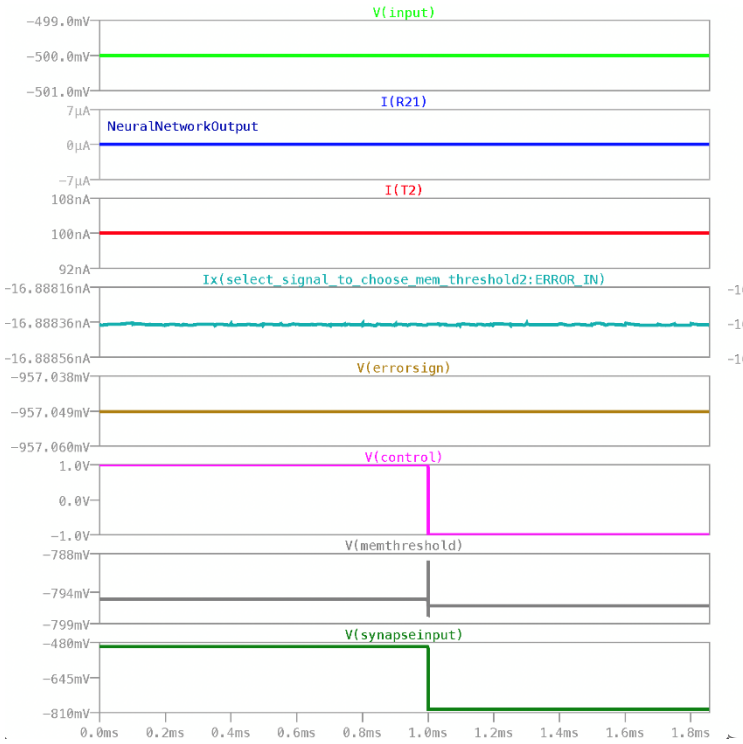


Figure 5.25a: Scenario 1

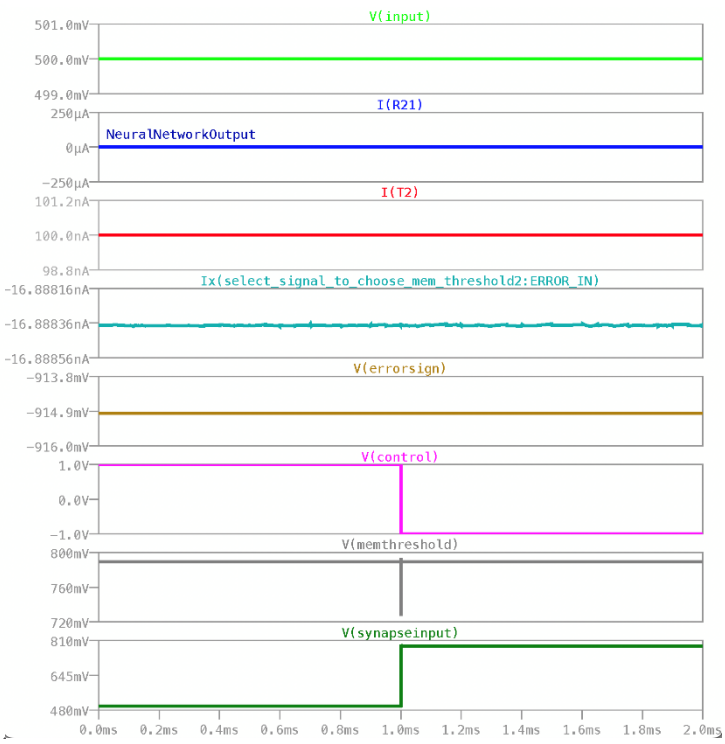


Figure 5.25b: Scenario 2

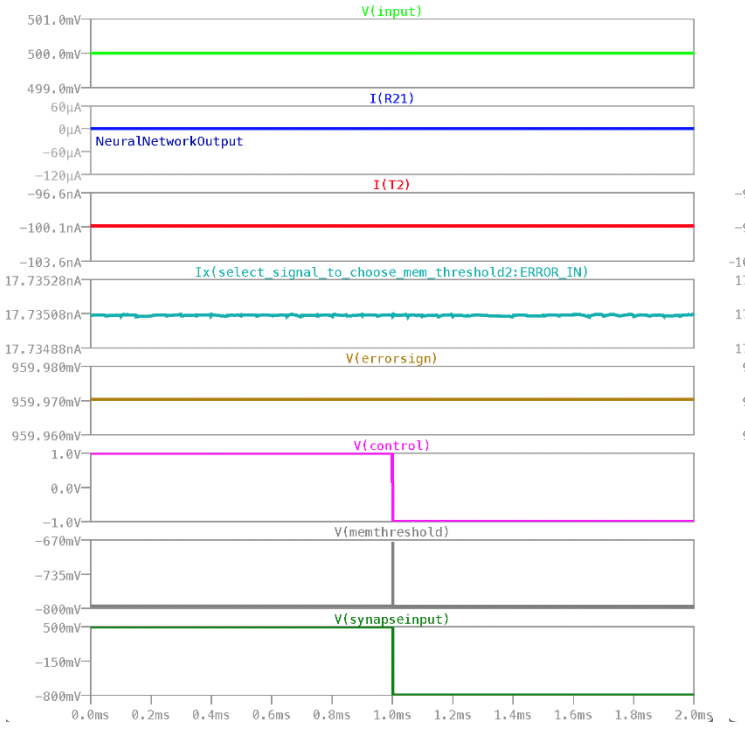


Figure 5.25c: Scenario 3

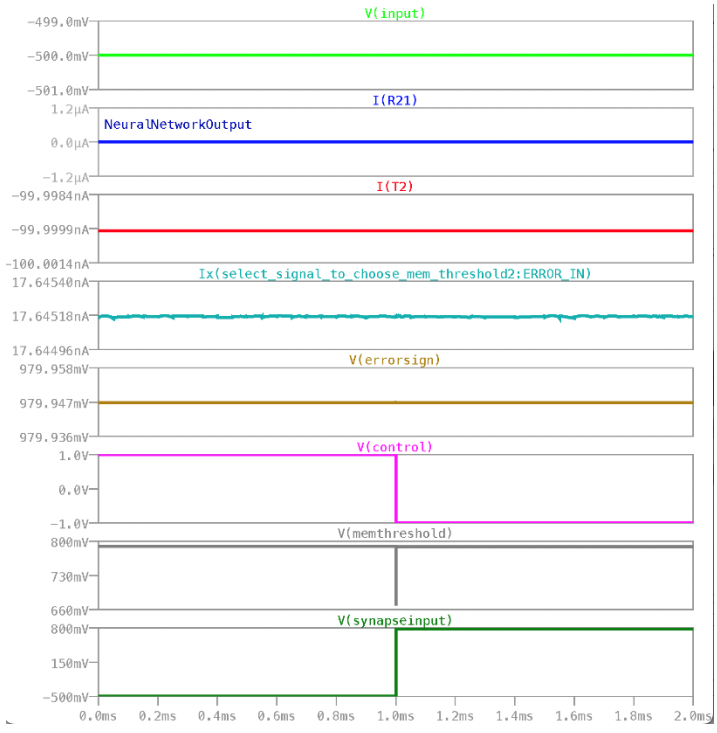


Figure 5.25d: Scenario 4

5.5 Summary

The training circuit trains the neural network by adjusting the memristance of the memristors in a synapse by computing the error based on the neuron output and the target output. The training circuit is mainly made up of three sub-components; the error calculation circuit, the memristor threshold select signal generator and a multiplexer to select positive or negative memristor threshold voltage. The memristance of the memristors in the neural network is adjusted based on the sign of the error. i.e. if the error is negative, the positive memristor threshold value will be fed into the synapse to adjust the memristance to reduce the error. Similarly, the negative memristor threshold value will be used if the error is positive. The completed training circuit consists of a sample and hold circuit, error calculation circuit, memristor threshold voltage selector circuits and synapse input selector circuits. The sample and hold circuit hold the neural network output and provides it to the error calculation circuit during the feedback voltage adjustment process such that the error calculation circuit calculates the error based on the actual output, target output, perturbation, and the reference multiplication factor. The memristor threshold voltage selector circuits use this error sign, the actual synapse input voltage sign, and the feedback adjustment voltages to choose the sign of the feedback adjustment voltage to be sent to the synapse. The synapse input selector circuits provide the feedback adjustment voltage to the synapse during the feedback adjustment process and the actual synapse input during the forward propagation. The advantage of this circuit design is that it uses minimal circuit components to implement the zeroth order optimization technique as opposed to the complex circuit design for implementing other training algorithms. The training circuit needs only one sample and holds circuit and error calculation circuit whereas the number of memristor threshold voltage selector circuits and synapse input selector circuits depend on the

number of total synapses. There are four different scenarios the training circuit must be capable of handling from table II. This chapter gave a brief description of how each individual subcomponent is designed and its working principle. This chapter also showed that the training circuit design is capable of handling all four scenarios mentioned in this chapter. As the neural network gets bigger, the training circuit also grows in terms of memristor threshold voltage selector circuits and synapse input selector circuits. However, this design still uses less circuit components compared to other similar approaches.

Chapter 6

Memristor model, Neural network and Training circuit

Memristor is the fourth fundamental passive circuit element. A memristor demonstrates memory dependent resistance, making it a valuable component in Neuromorphic computing. There are many memristor models proposed over the years for simulating the actual behavior of a memristor. This chapter discusses various memristor models, Neuron model used in validating the training circuit behavior, multilayered neural network and the training circuit. The weighted sum simultaneous perturbation algorithm discussed so far in this paper was based on a single layer neural network. WSP algorithm requires all weights to be perturbed simultaneously. However, the circuit implementation of this approach would be very complex. To minimize the circuit complexity, perturbation is only applied to the weighted sums. A second control signal is introduced to the training circuit to create a perturbed and non-perturbed output for a MNN such that all weighted sum gets perturbed. Training a neural network that solves a linearly inseparable computation using a MNN will prove the scalability of the training circuit that utilizes zeroth order optimization.

6.1 Memristor models

Various memristor models have been proposed to simulate the actual behavior of a memristor. Most common ones are as follows.

6.1.1 Linear Memristor Model

The linear memristor model shows a linear relationship between the charge and across the memristor. In this model, the memristor behaves as a simple resistor in which the resistance change is directly proportional to the volage applied to the memristor terminals. This means the resistance will be proportional to the electric charge that has passed through the memristor.

6.1.2 Window Function Memristor Model

The window functions memristor model is an expanded version of the linear memristor model. This model incorporates non-linear effects by introducing a window function that define the limits within which the resistance of the memristor can change. The non-linear behavior is demonstrated by the memristor when the memristor operates within these limits, enabling a more accurate representation of the device's characteristics.

6.1.3 Teamhp memristor model

The Teamhp memristor model is developed by Team Hewlett-Packard. This memristor model provides a more detailed description of physical memristor behavior. It considers the state dynamics of the memristor, incorporating the internal physical processes that occur during state changes. This memristor model uses drift-diffusion equations to

describe the movement of charged particles within the memristor material, resulting in a more realistic representation of a memristor behavior.

6.1.4 Berkeley memristor model

The Berkeley memristor model is widely used for simulating complex memristor behavior. It aims to capture the nonlinearity and hysteresis effects observed in real memristors. This model incorporates a set of differential equations derived from experimental observations, taking into account factors such as ionic drift, vacancies, and different charge transport mechanisms. By considering these factors, the Berkeley model provides a more comprehensive representation of memristor dynamics.

6.1.5 Memristive Standard Model

The Memristive Standard Model (MSM) is a mathematical framework that offers a unified description of various memristor models. It establishes a common set of equations and parameters that can be utilized to simulate different types of memristors. The MSM allows researchers to compare and analyze different models within a standardized framework, facilitating the understanding and advancement of memristor technology.

6.1.6 Yakopcic memristor model

The Yakopcic memristor model is a well-known model that takes into account the nonlinear dynamics and memory effects observed in memristors. This model incorporates a set of differential equations that describe the behavior of the memristor, capturing various phenomena such as window functions, pinched hysteresis loops, and compliance current. The Yakopcic model provides a more accurate representation of memristor

behavior, especially in capturing the intricate nonlinear dynamics.

6.1.7 Sinh Memristor Model

The Sinh model is a concept related to certain types of memristors, specifically those that exhibit a hyperbolic sine relationship between the charge and flux (or voltage and current) across the device. In the sinh model, the memristor's behavior follows the hyperbolic sine function, resulting in nonlinearity and memory-dependent resistance. This mode is characterized by its ability to retain its resistance state even after the voltage is removed, making it useful for non-volatile memory applications. The sinh model is a distinctive feature of specific memristor types and plays a significant role in their behavior and applications. It provides nonlinearity and memory-dependent resistance, making it suitable for non-volatile memory applications. The sinh model allows the memristor to retain its resistance state even after the voltage is removed. The Neuron model in this paper uses linear ion drift model with a hyperbolic sine relationship between the charge and flux. The window function used for linear ion drift model is shown in (6.1).

$$f_{win}(\gamma) = \begin{cases} e^{-\varepsilon_2^+(\gamma-\varepsilon_3^+)}\frac{1-\gamma}{1-\varepsilon_3^+}, & V_m \geq 0,\ \gamma \geq \varepsilon_3^+ \\ e^{\varepsilon_2^-(\gamma-\varepsilon_3^-)}\frac{\gamma}{\varepsilon_3^-}, & V_m < 0,\ \gamma \leq \varepsilon_3^+ \\ 1, & otherwise \end{cases} \quad \textbf{(6.1)}$$

V_m is the voltage across the memristor, γ is the state variable, ε values are the fitting parameters. The state variable, $\gamma \in [0,1]$ evolution is given by (6.2).

$$\frac{\Delta\gamma}{\Delta t} = \chi\big(V_m(t)\big) = \begin{cases} \varepsilon_4^+ sinh\big(\varepsilon_5^+ V_m(t) - \varepsilon_6^+ V_{tp}\big) f_{win}(\gamma), & V_m > V_{tp} \\ \varepsilon_4^- sinh\big(\varepsilon_5^- V_m(t) - \varepsilon_6^- V_{tp}\big) f_{win}(\gamma), & V_m < V_{tn} \\ 0, & otherwise \end{cases} \quad \textbf{(6.2)}$$

V_{tp} and V_{tn} are the voltage positive and negative memristor threshold voltages. Based on the window function and state variable evolution, the current flowing through the memristor, i_m is given by (6.3).

$$i_m = \begin{cases} \gamma G_{mon} V_m + (1-\gamma) G_{moff} \varepsilon_1^+ \sinh\left(\frac{V_m}{\varepsilon_1^+}\right), & V_m \geq 0 \\ \gamma G_{mon} V_m + (1-\gamma) G_{moff} \varepsilon_1^- \sinh\left(\frac{V_m}{\varepsilon_1^-}\right), & V_m < 0 \end{cases} \quad \textbf{(6.3)}$$

The fitting parameters used in the sinh memristor model is given in table III.

Table III: Sinh Memristor Model Fitting Parameters

Parameter	Value	Bounds
ε_1^+	0.9934	[0,+∞]
ε_2^+	2.5275	[0,+ ∞]
ε_3^+	0.3394	[0,1]
ε_4^+	113.5	[0,+ ∞]
ε_5^+	3.8153	[0,+ ∞]
ε_6^+	-2.0429	[-∞,+∞]
ε_1^-	0.2727	[0,+ ∞]
ε_2^-	4.2894	[0,+ ∞]
ε_3^-	0.4837	[0,1]
ε_4^-	106.2875	[0,+ ∞]
ε_5^-	4.0992	[0, +∞]
ε_6^-	-3.0634	[-∞,+∞]

The fitting parameters from table III along with the window function and state variable is used to calculate the linear model current corresponding to each experimental voltage data.

6.2 Neuron model

The neuron model described in the paper: “Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications” is used to validate the training circuit functionality. It was created using op-amps, memristors, and resistors. Synapses consist of two memristors to represent the positive, negative, and zero weights [2]. The output range of the neuron has three output cases as shown in (2.12). The polarity of the two memristors are opposite to each other, so the change in the state will be different for both [2]. A generic neuron is shown in Figure 6.1. A pair of memristors represent a synapse weight. The weight distribution is calculated from (2.16).

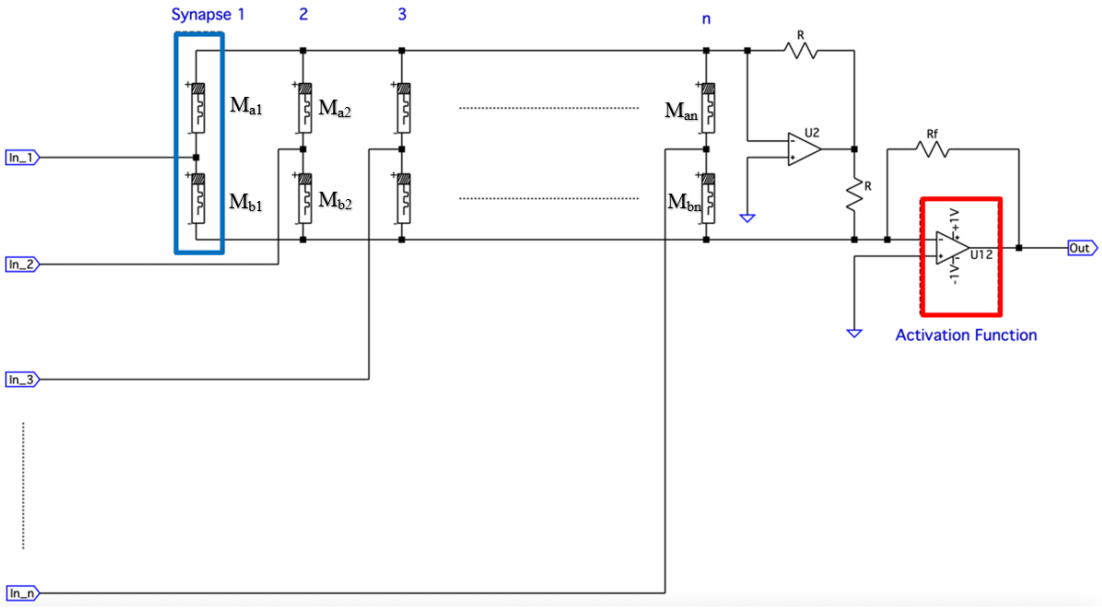


Figure 6.1: Generic neuron with n input synapses.

From (5.11), if the change in error and the input to a synapse has same sign (both +’ve or both

–‘ve), then the change in weight will be negative and the synapse weight must decrease. Similarly, if the change in error and the input to a synapse has opposite sign, then the change in weight will be positive and the synapse weight must increase. The four possible cases based on (5.11) is given in table II. In order to decrease the weight, the memristance of the memristor M_{ai} must increase and M_{bi} must decrease from the synapse i in a given neuron. Feedback adjustment voltage lower than the negative threshold of the memristor is used to achieve this. Similarly, to increase the weight, the memristance of the memristor M_{ai} must decrease and M_{bi} must increase from the synapse i in a given neuron. Feedback adjustment voltage larger than the positive threshold of the memristor is used to achieve this. From (2.16) if the memristance M_{ai} is larger than M_{bi} of synapse i in a a given neuron, the weight associated with the synapse i will be negative and vice versa.

The activation function of the neuron is implemented using an op-amp by limiting its rail voltages to 0.5V and 0V. This is to ensure that during forward propagation and the feedback adjustment process, the neuron output voltage (a synapse input to the next layer) will never go outside the memristor threshold voltages +/-0.7V.

6.3 Multilayered Neural Network

A multi layered Neural Network can is composed of several single layer neural networks. A MNN generally has an input layer, some hidden layer, and an output layer. The architecture of MNN trained by WSSP algorithm is shown in Figure 6.2.

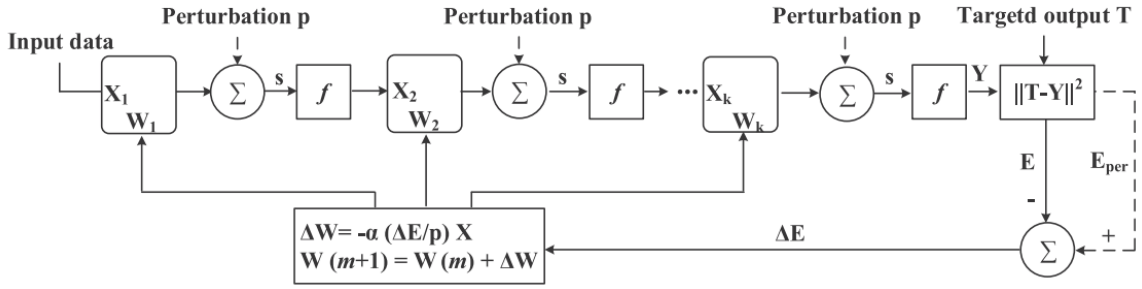


Figure 6.2: The architecture of MNN trained by WSSP algorithm [1]

In WSP algorithm, the perturbation p is applied to all weights simultaneously, but in the circuit implementation, the complexity of circuit will increase with the increased number of synaptic weights. To reduce the circuit complexity, the perturbation p is only applied to weighted sum in WSSP algorithm, which makes the circuit design more concise [1]. The training circuit shown in Figure 5.24 was modified to accommodate simultaneous perturbation to all weighted sums (every neuron output). A second control signal is introduced to the training circuit to create a perturbed and non-perturbed output for a MNN such that all weighted sum gets perturbed simultaneously. A control unit is created that takes in two control signals and outputs the error without perturbation, error with simultaneous perturbation to all weighted sums and a select signal that decides whether the actual synapse input or the feedback voltage adjustment gets forwarded to a synapse. Control signal 1 indicates whether the neural network is in forward propagation or training mode. A +1V control signal indicates forward propagation and a -1V control signal indicates training mode. The period of control signal 2 is half as that of control signal 1. During training mode (Control signal 1 is -1V), when the control signal 2 is high (+1V), a small constant perturbation is added to all the weighted sums in the neural network. During training mode, When the control signal 2

is low (-1V), the error between the perturbed error and the non-perturbed error is calculated and the weights will be adjusted based on the input sign and the error sign according to table II.

A generic neural network is shown in Figure 6.3.

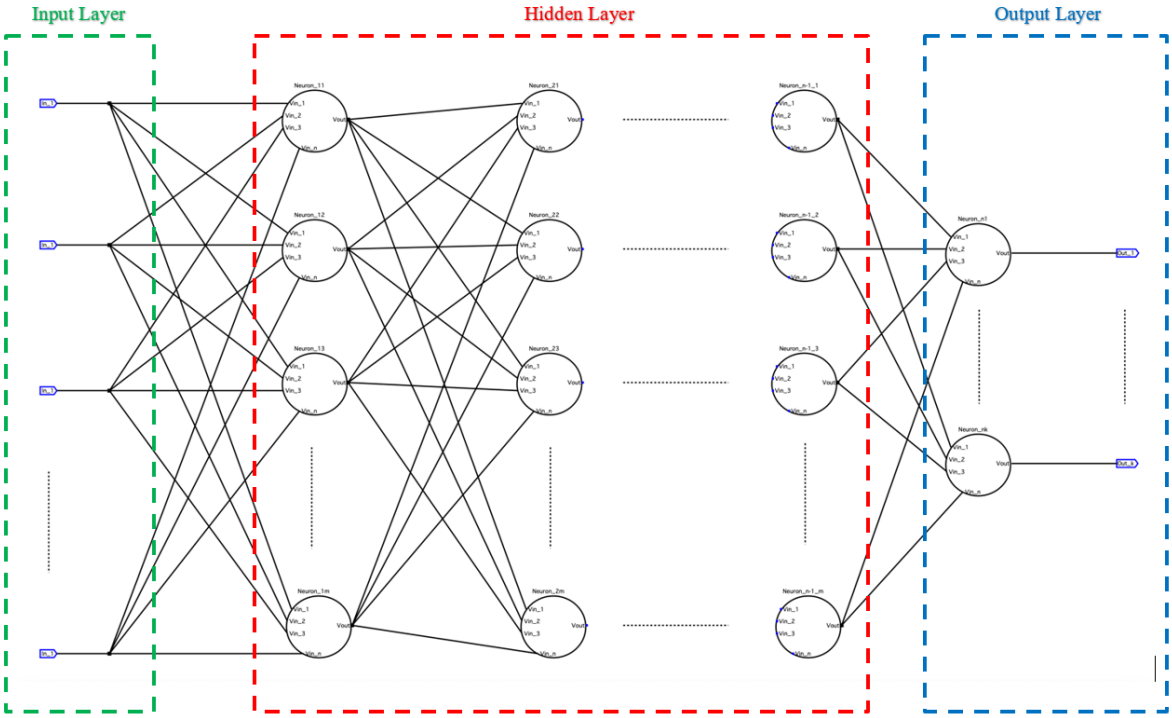


Figure 6.3: A generic Neural Network Layout

6.3.1 Control Unit

A control unit is created that takes in two control signals and outputs the error without perturbation, error with simultaneous perturbation to all weighted sums and a select signal that decides whether the actual synapse input or the feedback voltage adjustment gets forwarded to a synapse. Control signal 1 indicates whether the neural network is in forward propagation or training mode. A +1V control signal indicates forward propagation

and a -1V control signal indicates training mode. The period of control signal 2 is half as that of control signal 1. During training mode (Control signal 1 is -1V), when the control signal 2 is high (+1V), a small constant perturbation is added to all the weighted sums in the neural network. During training mode, When the control signal 2 is low (-1V), the error between the perturbed error and the non-perturbed error is calculated and the weights will be adjusted based on the input sign and the error sign according to table II. The control unit is made up of an inverter, an AND gate and a NOR gate. The control block accepts the two control signals and generates three control outputs. First, the control signal Vc1 (determines forward propagation or training) will be passed through as it is. This signal is then fed as a select signal to a two-to-one MUX that propagates the weighted sum during forward propagation and adds a small positive constant perturbation during training cycle. It also acts as the enable signal for the sample and holds circuit that holds the non-perturbed neural network output. A sample and hold circuit hold the unperturbed neural network output. Second, Vc1 signal gets inverted and then fed to the AND gate as an input along with Vc2 signal. The resultant signal acts as the enable signal for the sample and holds circuit that holds the perturbed neural network output. The second sample and hold circuit thus holds the perturbed neural network output generated when Vc1 is low and Vc2 is high. Third, Vc1 and Vc2 signal are fed into the NOR gate and the resultant output acts as the synapse input select signal in the training circuit. Thus, the feedback adjustment voltage gets fed into the synapse only when both control signals are logic low (-1V). i.e. during the feedback adjustment process. The control unit circuit diagram is shown in Figure 6.4.

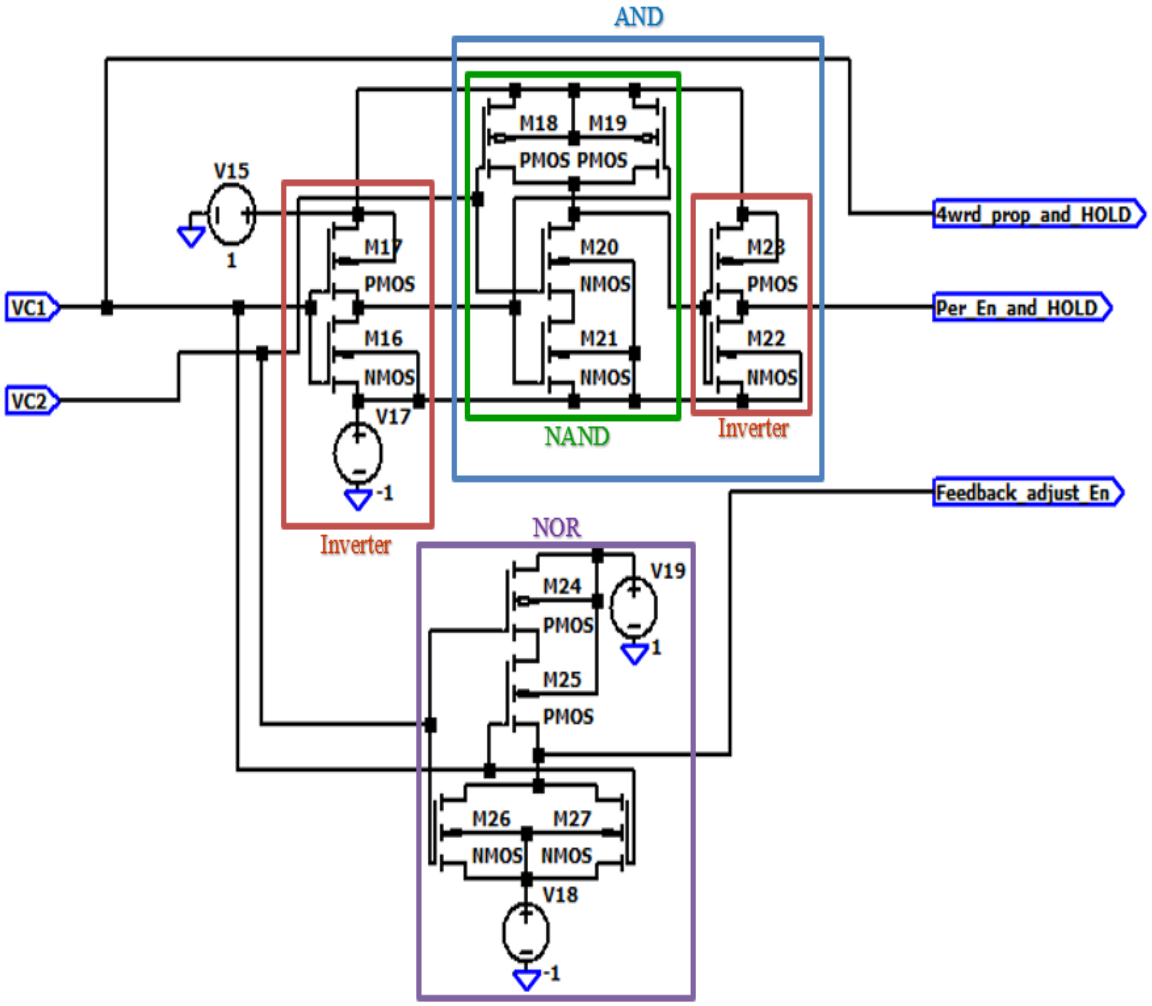


Figure 6.4: Control Unit Circuit Diagram

A simulation scenario to demonstrate the working principle of the Control block updating a synapse weight is shown Figure 6.5.

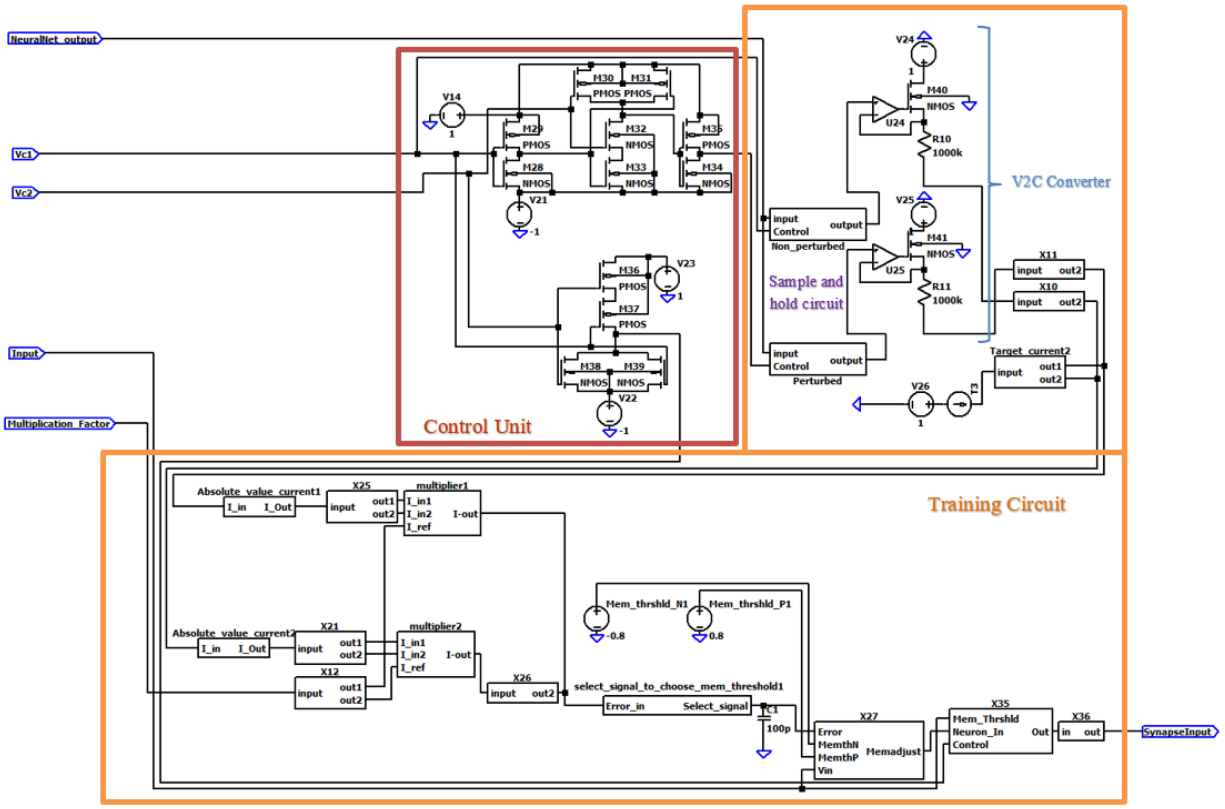


Figure 6.5: Single Synapse Training Circuit with Control Unit

In Figure 6.5, a sin wave is used to mimic the neural network output going into the sample and hold circuit. As the sin wave changes its values continuously, it would be easy to see how the sample and hold circuits hold the non-perturbed and perturbed outputs so that the error calculation can compute the feedback adjustment voltages based on the input and error sign and how the control unit controls the forward propagation and training process. The simulation waveform of the working principle of the Control block holding perturbed and non-perturbed output is shown in Figure 6.6. The period of Vc1 signal is 1ms and Vc2 signal is 0.5ms. the input to a synapse is set to 0.555V and the multiplication factor to

100nA.

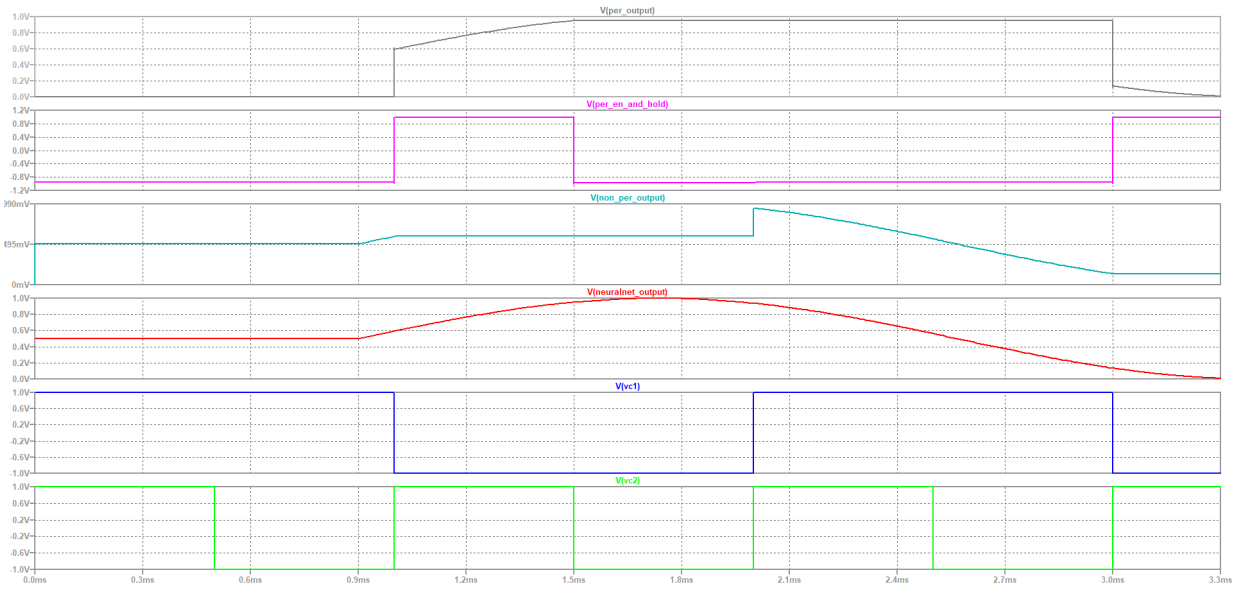


Figure 6.6: Simulation of Control block holding perturbed and non-perturbed outputs.

In Figure 6.6, V(neuralnet_output) represents the output of the neural network, in this scenario, the neural network is in forward propagation mode during 0 to 1ms, adding perturbation during 1 to 1.5ms and feedback voltage updating during 1.5 to 2ms. The V(non_per_output) gets updated only during the forward propagation cycle (when Vc1 is high) and the sample and hold circuit holds the V(non_per_output) constant after the forward propagation throughout the period Vc1 remains an active low (-1V). Similarly, the V(per_output) gets updated only during the perturbation period (when Vc1 is low and Vc2 is high) and the second sample and hold circuit holds the V(per_output) after the perturbation period throughout the feedback voltage output cycle. This makes is easy to calculate the error between the perturbed and non-perturbed output during the feedback voltage update cycle (when both Vc1 and Vc2 stays low) as perturbed and non-perturbed

output remains constant during this period. The simulation waveform of the control block logic selecting the feedback update voltage and hence updating the weight of a synapse is shown in Figure 6.7.

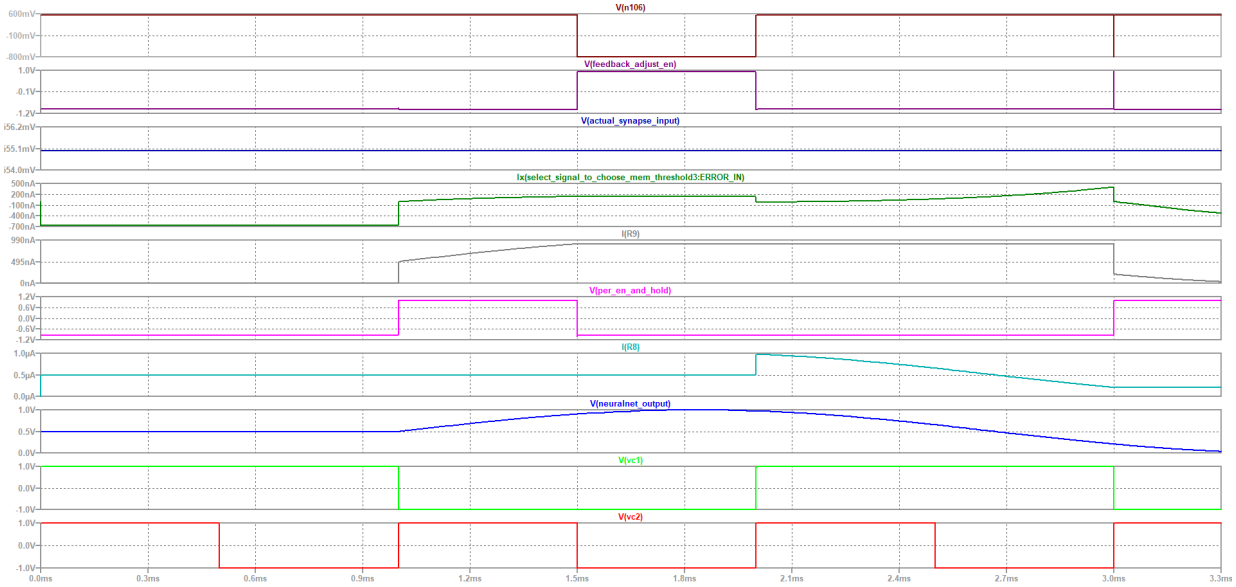


Figure 6.7: The simulation waveform of the control block logic selecting the feedback update voltage.

In Figure 6.7, the feedback adjustment enable signal goes high during the feedback voltage update process (when both Vc1 and Vc2 is active low). The error current Ix is positive during this period as the perturbed output is greater than the non-perturbed output. The input voltage to the synapse remains constant at 555mV throughout the simulation. Since the error and the input to the synapse are both positive, the feedback update voltage must be lower than the negative memristor threshold voltage according to table II. Thus, the expected voltage input to the synapse is -800mV during the feedback voltage adjustment cycle and 555mV during the remainder of the simulation period. The synapse input voltage

is 555mV during the forward propagation and perturbation period and -800mV during the feedback voltage adjustment cycle in Figure 6.7. This confirms the correct functionality of the control unit.

6.3.2 The Complete Circuit

The complete Circuit consists of a neural network and a training circuit. Figure 6.8 shows a complete generic circuit based on ideas discussed in this paper. This circuit consists of a generic neural network with m layers and n neurons in the input and hidden layers and a single neuron in the output layer. This is because this paper describes binary computations. However, this circuit can be scaled into a complex circuit that has multiple output neurons which would be capable of solving classification problems and so on in the future. The training circuit has two control signals to determine the mode of operation of the neural network at a given time, two sample and hold circuit that holds the perturbed and non-perturbed outputs, error calculation circuit and switches that performs the training process. The biggest advantage of this circuit design is that it does not use complex circuit components two quadrant multiplier, uses very minimal number of op amps and uses relatively simple circuit design to implement zeroth order optimization as opposed to complex algorithms such as back propagation. This circuit does not require a switch for every single synapse. However, the number of switches required in layer n to select between the actual synapse input and the feedback update voltage will be equal to the number of neurons in layer n-1. A neural network with m layers and n neurons in each layer (the number of neurons in the output layer does not matter). The total number of switches N_{Sw}, required for a neural network is given in (6.4).

$$N_{Sw} = 2.m.n \quad \textbf{(6.4)}$$

Factor 2 in (6.4) comes from the fact that a switch is required to choose between the positive and negative memristor threshold voltage and to choose between actual synapse input and the feedback update voltage. If a neural network has a different number of neurons in each layer (6.4) will be modified to (6.5). The total number of switches required for such a neural network is calculated by (6.5).

$$N_{Sw} = 2(Total\ \ neurons\ in\ neural\ network - \ neurons\ in\ the\ ouput\ layer) \quad \textbf{(6.5)}$$

Equation (6.5) always holds true whereas (6.4) holds true only when all the hidden layers and the input layers have same number of neurons. The number of XOR logic circuits required to perform the logic from table II can also be calculated using (6.4) or (6.5) depending on the neural network design.

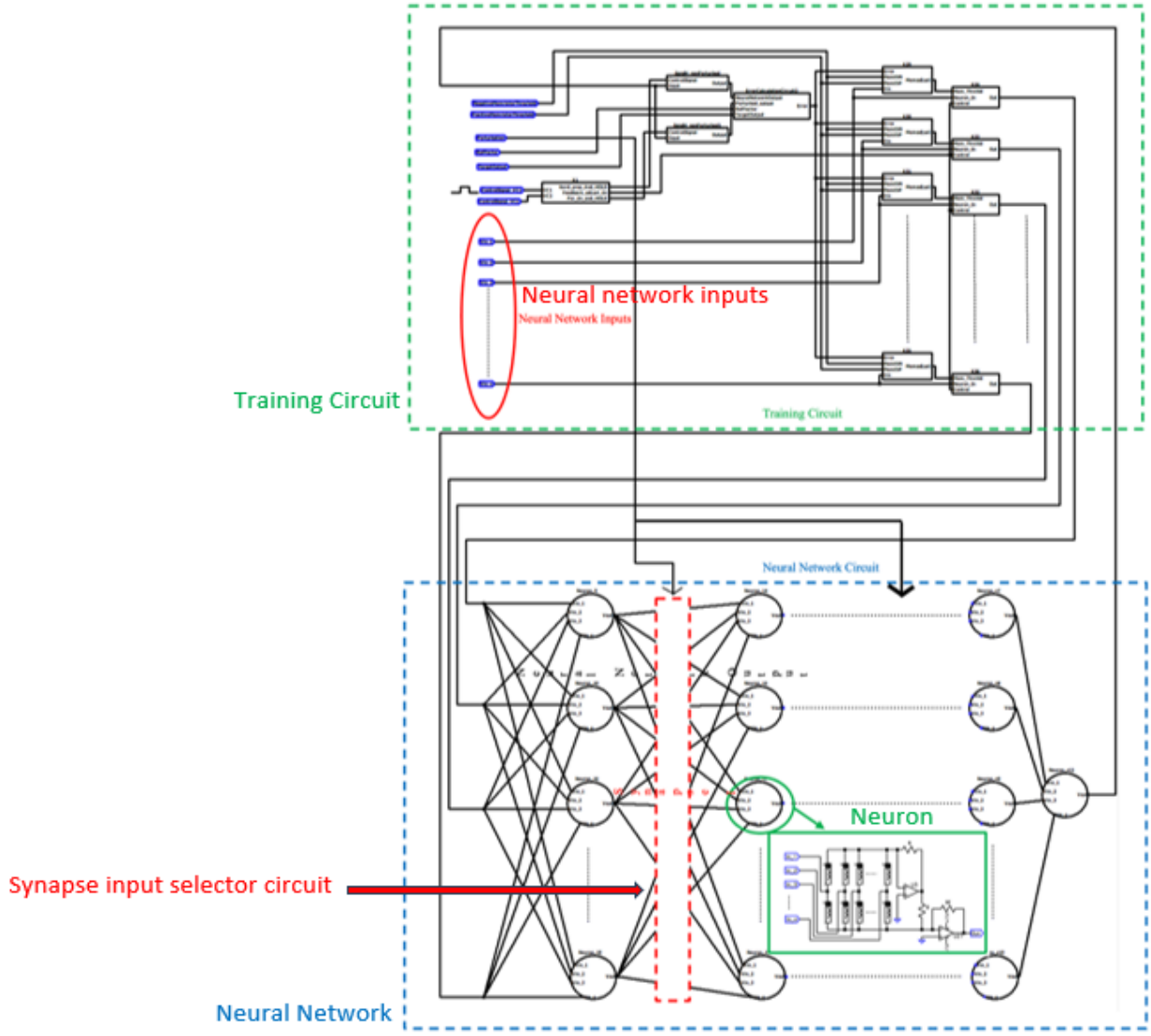


Figure 6.8: The Complete Circuit

The complete Circuit design used in LtSpice for simulating the neural network and the straining process is shown in Figure 6.9. The total number of op-amps required for the complete cicuit can be calculated from(6.6). Every neuron requires two op-amps to calculate the weighted sum and to implement the activation function. Both sample and hold circuit in the training circuit requires n op-amp followed by it in the voltage to current converter circuit to implement current mode training operation.

$$N_{Op\ amp} = 2(Number\ of\ neurons) + 2 \quad \textbf{(6.6)}$$

The neural network in Figure 6.9 has an input layer, a hidden layer and an ouput layer. Both input layer and hidden layer has 3 neurons each having 3 synapses and the output layer has one neuron. The weighted sum of each neuron is going through a series of switches that determine whether the output from previous layer or the feedback update voltage gets passed on to the next layer based on the two control signals. The final neural network output is fed back into the training circuit to compute error inorder to update the weight of each synapse.

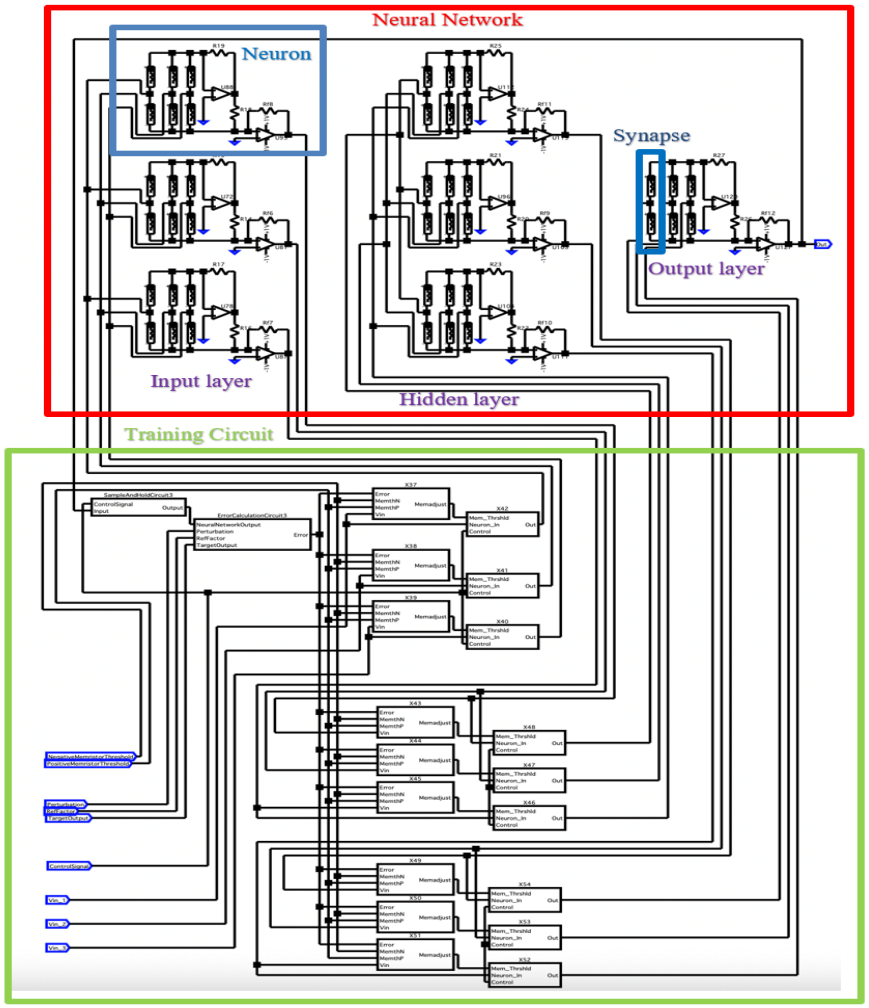


Figure 6.9: The complete Circuit design used in LtSpice for simulation

6.4 Summary

This chapter discussed various memristor models, Neuron model used in validating the training circuit behavior, multilayered neural network, and the training circuit. The weighted sum simultaneous perturbation algorithm discussed so far in this paper was based on a single layer neural network. WSP algorithm requires all weights to be perturbed simultaneously. However, the circuit implementation of this approach would be very complex. To minimize the circuit complexity, perturbation is only applied to the weighted sums. A second control signal is introduced to the training circuit to create a perturbed and non-perturbed output for a MNN such that all weighted sum gets perturbed. Training a neural network that solves a linearly inseparable computation using a MNN will prove the scalability of the training circuit that utilizes zeroth order optimization. A control unit is created that takes in two control signals and outputs the error without perturbation, error with simultaneous perturbation to all weighted sums and a select signal that decides whether the actual synapse input or the feedback voltage adjustment gets forwarded to a synapse. Control signal 1 indicates whether the neural network is in forward propagation or training mode. A +1V control signal indicates forward propagation and a -1V control signal indicates training mode. The period of control signal 2 is half as that of control signal 1. The complete circuit consists of a generic neural network with m layers and n neurons in the input and hidden layers and a single neuron in the output layer. This is because this paper describes binary computations. However, this circuit can be scaled into a complex circuit that has multiple output neurons which would be capable of solving classification problems and so on in the future.

Chapter 7

Results and discussion

This chapter demonstrates and discusses the simulation results of the neural network and the training circuit. This chapter demonstrates how the training circuit can be scaled to train a neural network by showing that it can successfully train a perceptron. The results and discussion involve demonstration of perceptron training, memristance change during synaptic weight update, error calculation and how all the circuit components work together to train the perceptron. The training circuit trains the neural network by adjusting the memristance of the memristors in a synapse by computing the error based on the neuron output and the target output. The neural network along with the training circuit performs forward propagation and training process alternatively. The training circuit calculates the error and updates the synaptic weight based on the error and the synaptic input until the error goes down to 0 and thereby making the neural network output close to equal to the target output.

7.1 Single Neuron Training

A single neuron with two synapses was undergone training in LTspice to confirm the functionality of the training circuit. The perceptron training was simulated for 80ms. The target output was set to -100nA for the first 40ms and then kept at 100nA during 40ms to 80ms. All memristors have the same properties and the weights are not randomly initialized, thus the output is expected to start at 0nA and reach the target output

eventually after the training is completed. All memristors start with same memristance and are expected to have different memristance after training. The perceptron and the training circuit used to verify the functionality of the training circuit is shown in Figure 7.1.

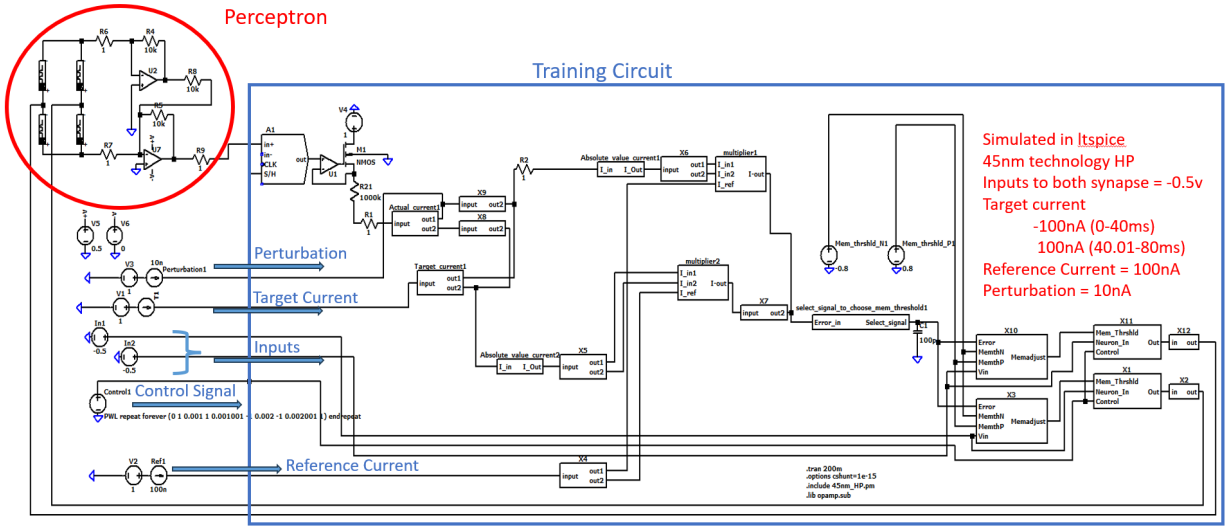


Figure 7.1: Perceptron and training Circuit

In Figure 7.1, the Perceptron has two synapses, the input to each synapse was set to -500mV throughout the simulation. The small constant perturbation current was set to 10nA, the reference factor was 100nA and the memristor positive and negative thresholds were 700mV and -700mV respectively. The neural network output is expected to start at 0nA initially and reach -100nA sometime before 40ms. Once the output reaches the target current (-100nA), it is expected to stay around the target current till 40ms. From 40ms to 80ms the target current switches from -100nA to 100nA. Therefore, the output current is expected to go from -100nA to 100nA sometime before 80ms and stay around 100nA till 80ms. The simulation waveform of the perceptron output current is shown in Figure 7.2.

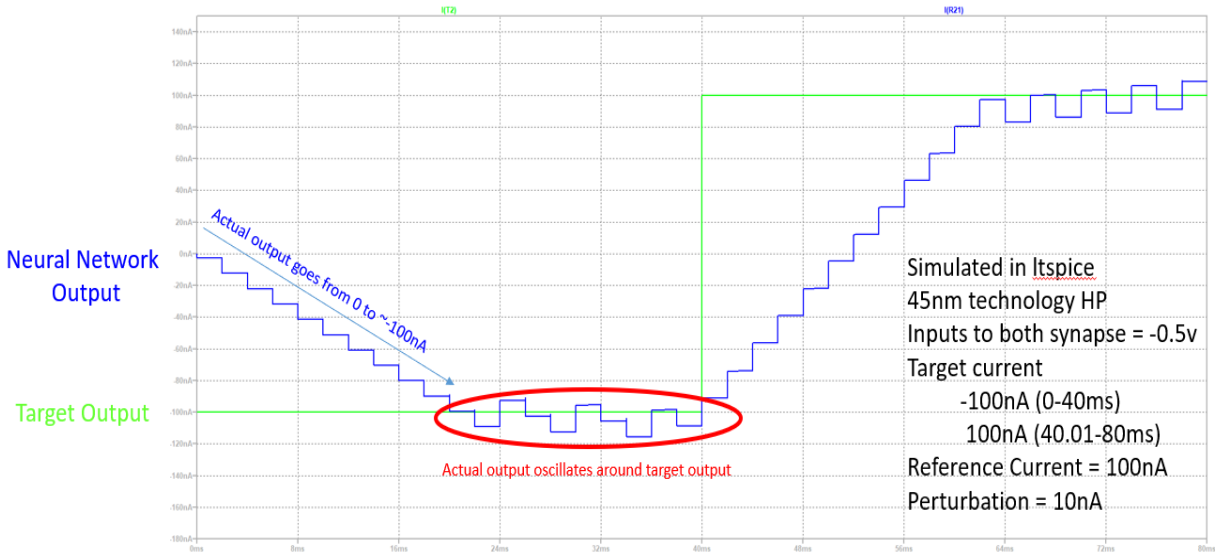


Figure 7.2: The simulation waveform of the perceptron output current

In Figure 7.2, the perceptron output current starts at 0nA at 0ms and reaches -100nA at around 20ms. The output current oscillates around the target current from 20ms to 40ms. The target current switches to 100nA at 40ms and the output current starts climbing to 100nA from -100nA and reaches the target current around 64ns. The output current oscillates around the target current from 64ms to 80ms. The training circuit was able to successfully train the perceptron to provide the target output current. This confirms the correct functionality of the training circuit. The output never settles at the target current but always oscillates around the target current. This is because the feedback update voltage does not vary depending on the error margin but stays constant. Thus, once the output current goes below the target current from above the target current, the training circuit senses the error sign has flipped and hence switches the feedback update voltage sign as well which in turn brings the output current close to target current or above the target current. This process continues and hence the output current never converges to the target current but oscillates around it. The circuit can be modified to change the

magnitude of the feedback update voltage depending on the error margin to converge the output current to the target current. However, this requires more circuit components and logic blocks. To keep the circuit complexity low, the feedback adjustment voltage is set constant and still provides accurate results will very low error margin.

7.2 Memristance change (Synapse weight updating)

A single synapse is selected from the perceptron and the memristance change throughout the simulation is observed. Both memristors have the same memristance initially and is expected to change during the simulation. Due to two different training cycles in the opposite directions, the memristors are expected to have different memristance values by the end of the simulation. The simulation waveform of the memristance change in a synapse is shown in Figure 7.3.

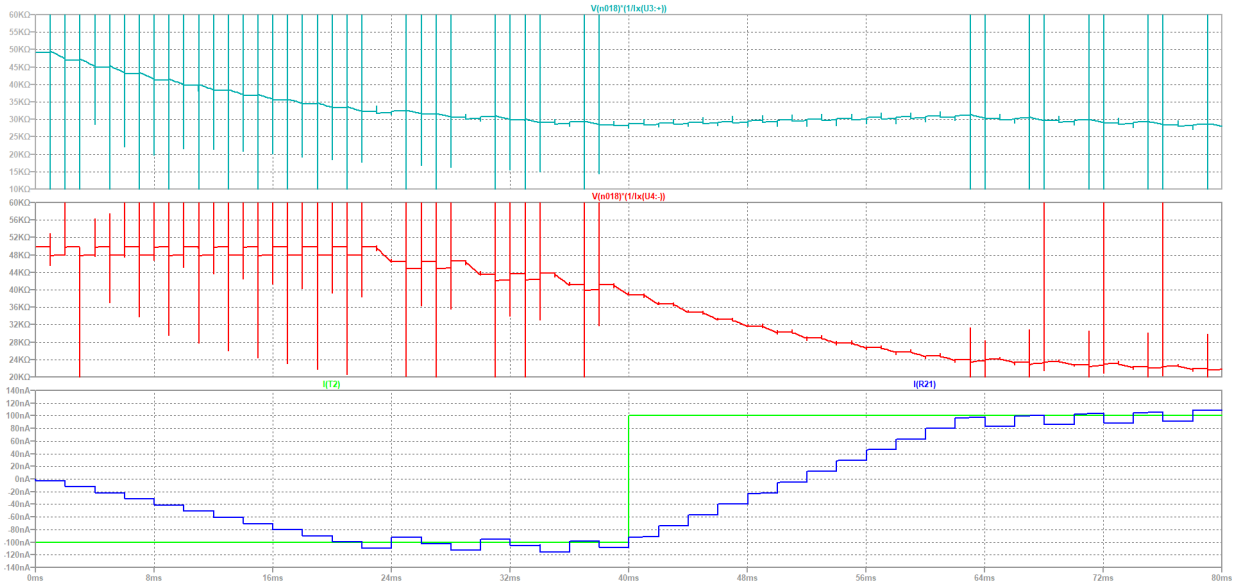


Figure 7.3: The simulation waveform of the memristance change.

Both memristors in the synapse has memristance around 50KΩ initially. In Figure 7.3,

Memristance of memristor a, Ma starts at 50KΩ and decreases until the output current reaches the target current and stays relatively constant till 40ms. Memristance of memristor b, Mb on the other hand oscillates between 48-50K Ω throughout the 40ms. However, it shows a slight decrease when the output current reaches the target current. This is because the output current oscillates around the target current when it reaches the target current. In this scenario the net current/resultant current through the synapse flows through Memristor a. Based on the above observation, during 40-80ms when the target current switches to 100nA, the net current/resultant current through the synapse must flow through the memristor b and hence Mb must decrease while Ma remains constant. In Figure 7.3, Mb starts to decrease until the output current reaches the target current and stays relatively constant till 80ms while Ma remains constant during the 50-80ms period. The simulation meets the expectation on the memristance updating in a synapse. After the complete simulation Ma is at 28kΩ and Mb is at 22KΩ. The Input to each synapse is set to -500mV throughout the simulation and as mentioned earlier, the weights are not initially randomized. Thus, both synapses have the same weight and all memristors have the same memristance. This implies the initial weight configuration of the synapses is 0 due to equal memristance in a synapse as mentioned in Figure 2.6. From 0ms to 40ms, the target current is set to -100nA. From (2.12), the output is the weighted sum of the inputs. Therefore, the weight of the synapse must increase 0 to a higher value in the positive direction such that the output can reach -100nA since the input is already negative (-500mV). Therefore, the weight is expected to increase from 0 to a positive value during the 0-20ms (time taken by output to reach the target) period and stay constant till 40ms. Similarly, it is expected to start decreasing at 40ms until 64ms (output reaches target at 64ms) and remain constant till 80ms. The weight, however, cannot be constant because the output will start oscillating around the target current once it reaches the target current. The simulation waveform of the synaptic weight update is shown in Figure 7.4.

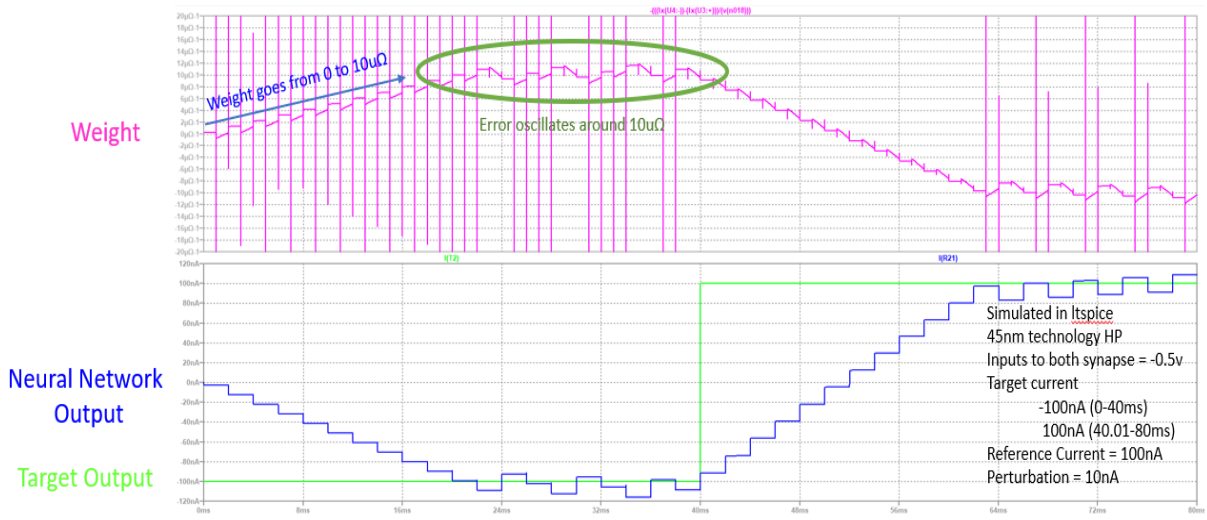


Figure 7.4: The simulation waveform of the synaptic weight update

In Figure 7.4, the weight increases from 0 to a $10\mu\Omega^{-1}$ during the 0-20ms (time taken by output to reach the target) period and stay oscillating around $10\mu\Omega^{-1}$ till 40ms. Similarly, it starts decreasing from $10\mu\Omega^{-1}$ at 40ms and reaches $-10\mu\Omega^{-1}$ at 64ms (output reaches target at 64ms) and remain oscillating around $-10\mu\Omega^{-1}$ till 80ms. During the 40ms to 80ms period while the output goes to 100nA from -100nA, the output crosses 0nA at 52ms. The input to the synapse remains constant at -500mV. Thus, the weight must be 0 at 52ms for the output to be 0nA. In Figure 7.4, the weight is $0\mu\Omega^{-1}$ at 52ms. Thus, the weight update waveform meets the expected behavior.

Let's assume that the output current converges to the target current. The weight of a single synapse when the output current reaches the target current of -100nA is $10\mu\Omega^{-1}$, the input stays constant at -500mV for both synapse and both synapse in the perceptron have equal weights. The output voltage of the perceptron calculated from (7.1) is shown in (7.2).

$$V_O = R_F(V_{in1}w_1 + V_{in2}W_2), \quad \text{From (2.12)} \tag{7.1}$$

$$V_0 = 10k\Omega(-500\text{mV}.10\mu\Omega^{-1} + -500\text{mV}.10\mu\Omega^{-1})$$

$$V_o = 10000(-500.10^{-3}.10^{-5} - 500.10^{-3}.10^{-5})V$$

$$V_o = 10000(-1000.10^{-3}.10^{-5})V$$

$$V_o = -10^{-1}V = -100mV \tag{7.2}$$

The Perceptron output voltage is converted to current using a simple voltage to current converter that has 1MΩ resistance. This can be seen in Figure 7.1. The output current can be calculated using Ohm's law and is given in (7.3).

$$I_o = \frac{V_o}{R_o} = \frac{100mV}{1M\Omega} = -100.10^{-9}A = -100nA \tag{7.3}$$

The output current calculated in (7.3) matches the target current and the actual output current when the perceptron converges to the target current.

7.3 Error Calculation

The error calculation is a key part of the training circuit as it determines the feedback update voltage to update the synaptic weight based on the sign of the error. In the perceptron training simulation, the target current was set to -100nA for the first half period of the simulation. The synaptic weight at the beginning of the simulation is 0 and hence the output is also 0nA. the perturbation constant was set to 10nA. The expected error based

on these conditions can be calculated from (5.1), (5.2) and (5.3) and is shown in (7.4).

From (5.1), (5.2) and (5.3)

$$E = \frac{\left(\left(I_T - (I_{out} + I_{per})\right)^2 - (I_T - I_{out})^2\right)}{I_{ref}} \quad \textbf{(7.4)}$$

The I_{ref} is the reference scaling factor that comes from the translinear multiplier reference input. Plugging in the initial conditions into (7.4),

$$E = \frac{\left(\left(-100nA - (0nA + 10nA)\right)^2 - (-100nA - 0nA)^2\right)}{100nA}$$

$$E = \frac{((-110nA)^2 - (-100nA)^2)}{100nA}$$

$$E = \frac{((-110.10^{-9})^2 - (-100.10^{-9})^2)}{100.10^{-9}} A$$

$$E = \frac{((-110)^2 - (-100)^2)}{100} (10^{-9}) A$$

$$E = \frac{(12100 - 10000)}{100} (10^{-9}) A$$

$$E = 21.10^{-9} A = 21nA \quad \textbf{(7.5)}$$

From (7.5), the Error is expected to be at around 21nA at 0ms and to reach close to 0nA around 20ms because at 20ms the output current reaches the target current. As the output current never converges but oscillates around the target current, the error is expected to oscillate around 0nA from 20ms to 40ms. From 40ms to 80ms, the target current changes to 100nA and the current output is around -100nA. However, the current output is -90nA

from the simulation waveforms as it never converges. Plugging in these conditions into (7.4),

$$E = \frac{\left(\left(100nA - (-90nA + 10nA)\right)^2 - (100nA - -90nA)^2\right)}{100nA}$$

$$E = \frac{((180nA)^2 - (190nA)^2)}{100nA}$$

$$E = \frac{((180.10^{-9})^2 - (190.10^{-9})^2)}{100.10^{-9}}A$$

$$E = \frac{((180)^2 - (190)^2)}{100}(10^{-9})A$$

$$E = \frac{(32400 - 36100)}{100}(10^{-9})A$$

$$E = -37.10^{-9}A = -37nA \quad \textbf{(7.6)}$$

From (7.6), the Error is expected to be at around -37nA at 40ms and to reach close to 0nA around 64ms because at 64ms the output current reaches the target current. As the output current never converges but oscillates around the target current, the error is expected to oscillate around 0nA from 64ms to 80ms. The simulation waveform of the error calculation during the training cycle is shown in Figure 7.5.

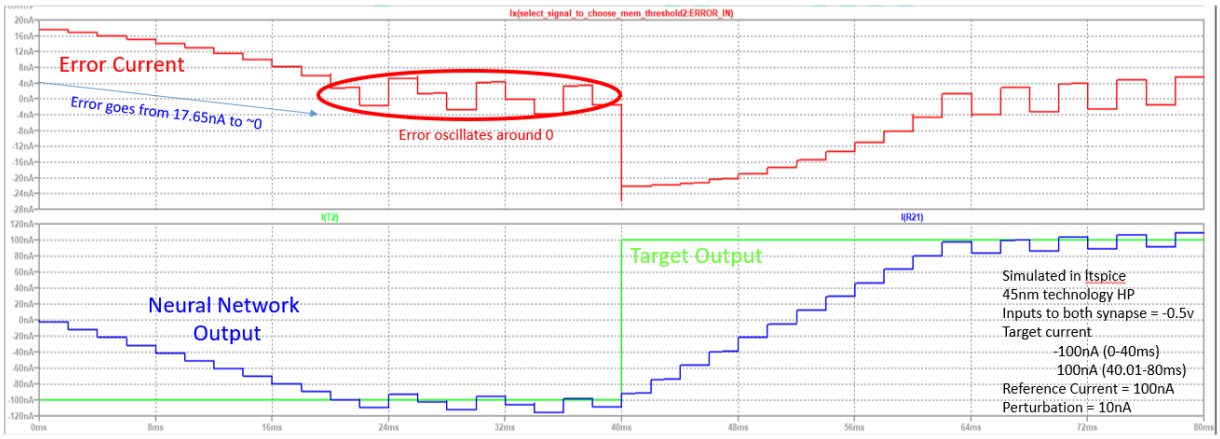


Figure 7.5: The simulation waveform of the Error Calculation

In Figure 7.5, the initial output is 0nA and the target current is -100nA. The error as expected starts at a positive value and steps down to 0nA by 20ms and oscillates around 0nA (stays within -3nA and 4nA) from 20ms to 40ms. The error was expected to start at 21nA. However, the actual error was about 18nA from the simulation. This matches with the data in table III. Similarly, the error as expected starts at a negative value at 40ms and steps up to 0nA by 64ms and oscillates around 0nA (stays within -4nA and 5nA) from 64ms to 80ms. The error was expected to start at -37nA. However, the actual error was about -27nA from the simulation. This margin between the actual and theoretical error value is due to the accumulation of precision loss from the current mirrors, absolute value current circuits, translinear multiplier and current subtractor circuits. The simulation behavior meets the expected behavior of the error calculation circuit and hence confirms the correct functionality of the circuit. Current mirrors can be sensitive to temperature variations, which can lead to changes in the replicated current. Integrated circuits are subject to manufacturing process variations, which can lead to differences in transistor characteristics even within the same batch. These variations can affect the accuracy and matching of the current mirror, leading to potential performance deviations. Transistors

used in current mirrors might not exhibit ideal behavior, especially in non-ideal operating conditions. This can lead to deviations from the expected mirrored current, reducing the accuracy of the circuit. One of the most effective ways to mitigate transistor matching issues is to use transistors that are fabricated using the same process and are designed to have closely matched characteristics. This can be achieved through techniques like device scaling, layout symmetry, and using transistors from the same wafer. Using a cascode configuration involving connecting an additional transistor in series with the mirror transistors can help to increase the accuracy as well. This can help mitigate the effect of early voltage variations and reduce the dependence on the individual transistor characteristics. The potential accuracy issues can be mitigated by increasing the size to minimize the variations in the transistors or by using extra transistors as mentioned in the cascode method. This will increase the transistor count in the training circuit as the current based design uses a current mirror to create copies of current at multiple stages. There is a trade off between the transistor size, count, and the functionality of the circuit. However, the training circuit can adjust the weights of the neuron synapses based on the observed error until the actual output reaches the target out. The adjustable nature of the training circuit helps the balance between accuracy loss due to transistor dissimilarities and the overall functionality of the system. The actual error always lies within 0 and the theoretical error value. Thus, the error calculated by the error calculation circuit will be closer to 0. This in turn helps in faster convergence of the neural network outputs. The smaller the error, the faster the training will be.

7.4 Complete simulation of a single synapse training

The training circuit computes the error between the perturbated and non-perturbated output based on the target output, perturbation constant and the actual output. The sign of the error calculated by the error calculation circuit along with the sign of the input to a synapse determines the feedback update voltage to the synapse to update the synaptic weight based on table II. If the error and the input have the same sign, the feedback update voltage will be negative, and the synaptic weight will decrease. Similarly, If the error and the input have the opposite sign, the feedback update voltage will be positive, and the synaptic weight will increase. Complete simulation waveform demonstrating the working mechanism of the training circuit is shown in Figure 7.6.

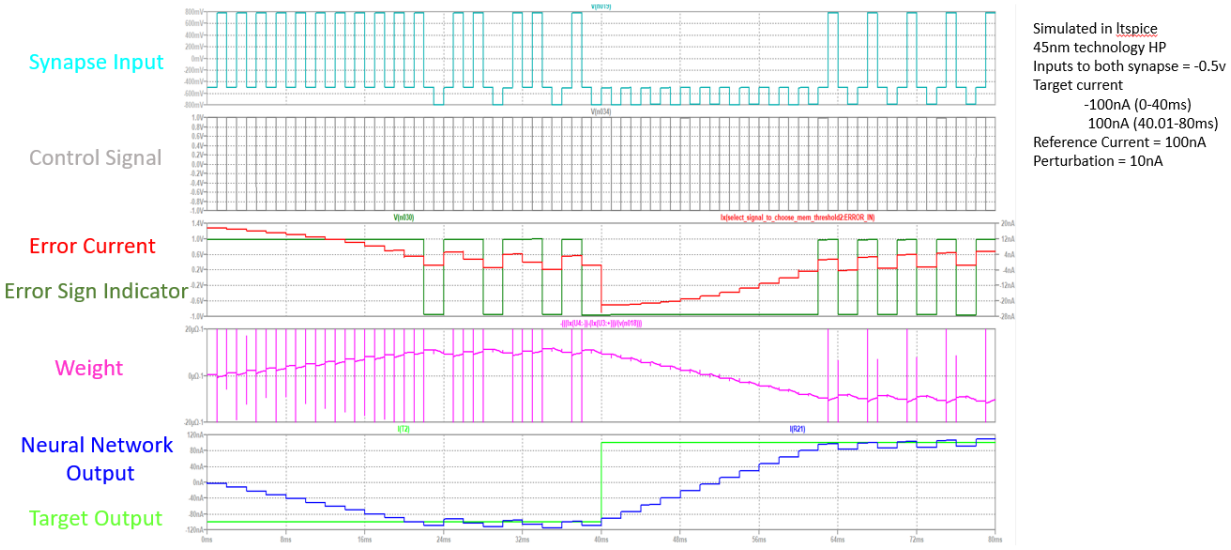


Figure 7.6a: Complete simulation waveform demonstrating the working mechanism of the training circuit.

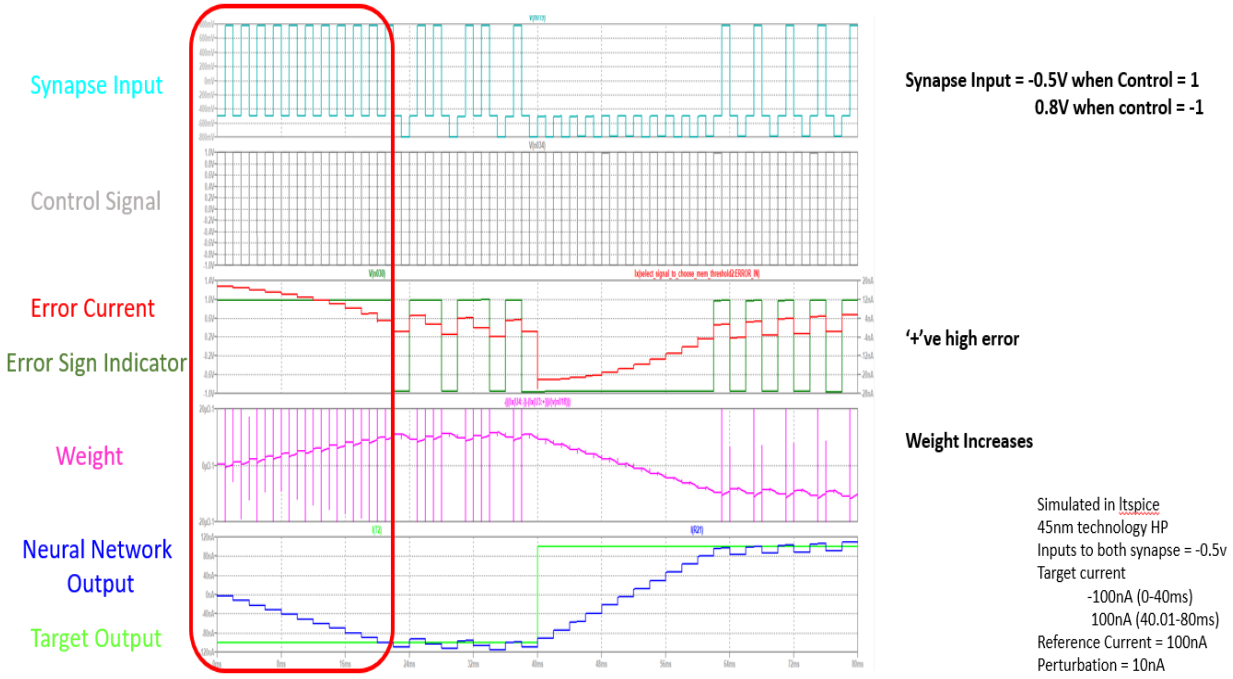


Figure 7.6b: Simulation when Error and Input has opposite sign.

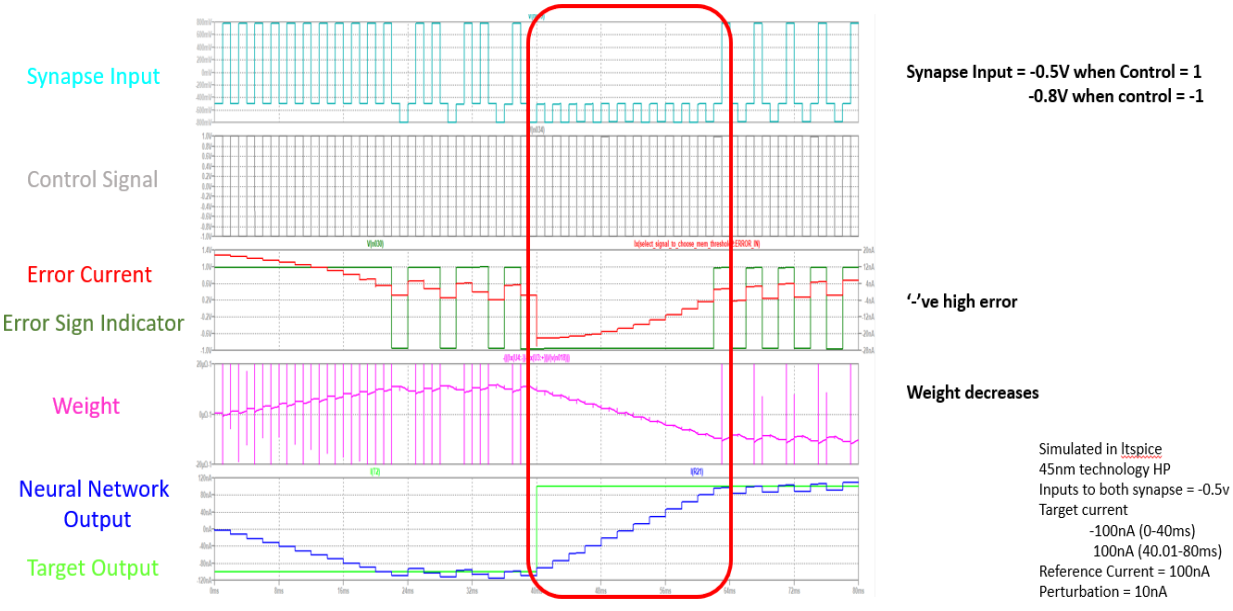


Figure 7.6c: Simulation when Error and Input has same sign.

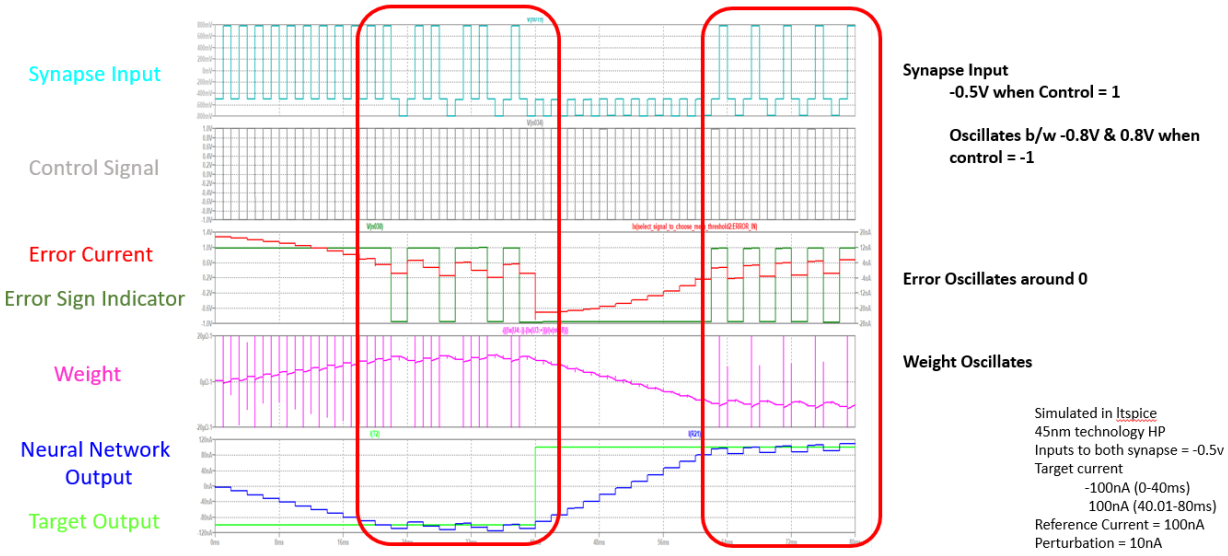


Figure 7.6d: Simulation when Error reaches 0 (actual output reaches target output).

The input voltage to the synapse is set constant at -500mV throughout the simulation and the positive and negative feedback update voltage is set to be 800mV and -800mV respectively (this is outside the memristor threshold voltage +/- 700mV). In Figure 7.6, the error is a positive value from 0ms to 20ms and hence the feedback update voltage fed to the synapse during the training cycle is positive (800mV) as the error and input has opposite sign. The input to the synapse is -500mV during the forward propagation. From 20ms to 40ms the error oscillates around 0nA. The error becomes small positive and negative values during this period and hence in Figure 7.6, the feedback update voltage also oscillates between -800mV and 800mV, 800mV when error stays positive and -800mV when error goes negative. Similarly, the feedback update voltage stays at -800mV from 40ms to 64ms as the error is negative and then oscillates between -800mV and 800mV during the 64ms to 80ms period as the error oscillates between small positive and negative values. The error sign indicator signal stays at +1V when the error is positive

and vice versa. The error sign indicator signal along with the input sign indicator (+1V when the input is positive and vice versa) determines the sign of the feedback update voltage based on table II. The simulation results in Figure 7.6 confirm the correct functionality of the training circuit as it trains the perceptron to produce the target output based on the error and the input to the synapse.

7.5 XOR gate Neural Network simulation results

After verifying the correct functionality of the complete training circuit, an MNN is trained to show the scalability of the design. A two input XOR gate training was done to show that the training circuit can train a non-linearly separable problem. The neural network has a total of three neurons. Two in the hidden layer and one in the output layer. Each neuron has three synapses, two input synapses and one bias synapse. The bias input was always 500mV to all three neurons. All the memristors had same memristance in the beginning of the simulation and hence, each synapse has a zero-weight associated with it. This in turn will make the output of the neural network initially zero and will update the weights gradually to give the correct results, this is shown in Figure 7.7. The inputs to the neural network were 500mV and -500mV representing a logic high and logic low respectively. The expected output (target output) was set to 200nA and -200nA representing a logic high and logic low respectively. The simulation waveform of the XOR gate training is shown in Figure 7.8.

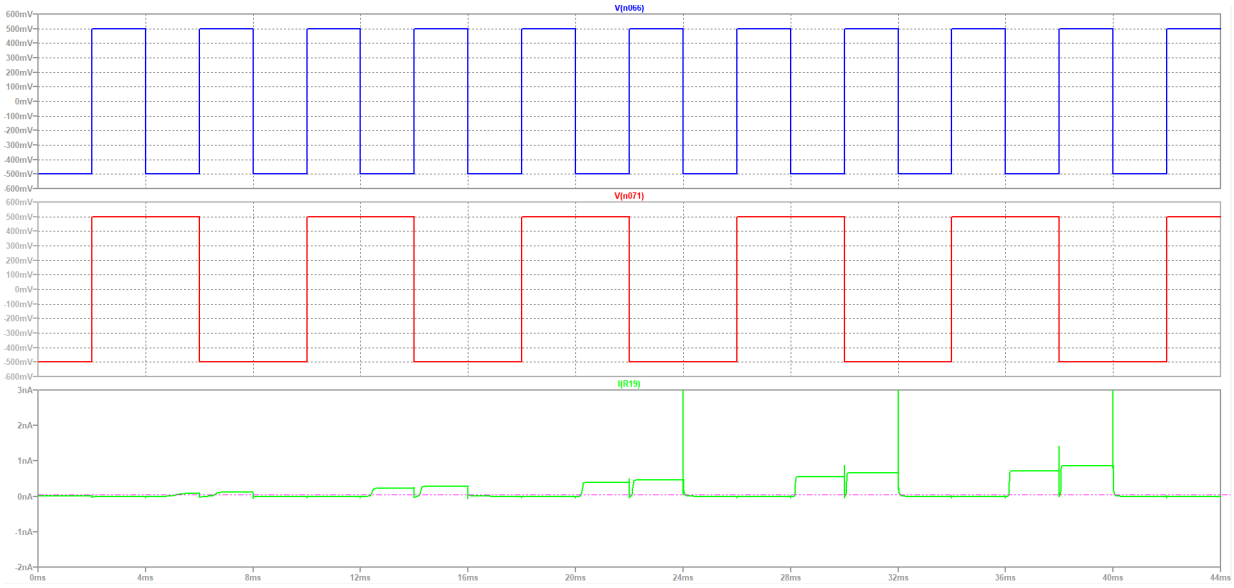


Figure 7.7: Neural Network Output Showing Gradual Learning.

In Figure 7.7, the initial output of the neural network is 0 and it gradually starts to learn the XOR function. This is because all the memristors had same memristance in the beginning of the simulation and hence, each synapse has a zero-weight associated with it.

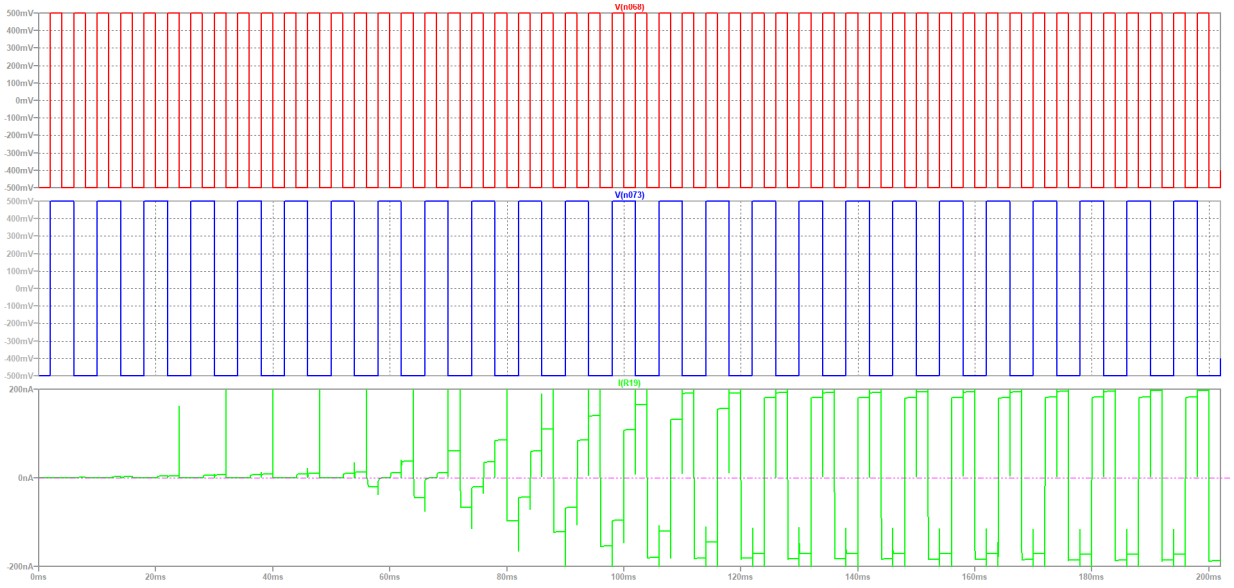


Figure 7.8: Neural Network Output Showing XOR Gate Learning.

In Figure 7.8, the neural network output reaches a steady state by ~120ms. The output is a logic high (200nA) only when both inputs are different and a logic low (-200nA) when both inputs are the same. This confirms the correct functionality of the XOR gate neural network and the training circuit. Once the neural network outputs reach the target outputs (learning is complete), the outputs stay at the expected results forever.

7.6 Area for future improvements

The training circuit functions as expected but there are still areas for improvement. One of the design aspects that could be improved is to make the feedback adjustment voltage value dependent on the error magnitude. This will help the error to converge to 0 after training which in turn helps the output current to converge to the target current instead of oscillating around the target current. One other area for improvement is to reduce the accuracy error in the error calculation circuit by reducing precision loss from the current mirrors, absolute value current circuits, translinear multiplier and current subtractor circuits. Lastly the neuron design could be modified to eliminate the use of op amps in it. This will help with power consumption as well. Currently each neuron has two op amps. Modifying the design to eliminate both or at least one op-amp will make a huge difference in the power consumption as it reduces the total number of transistors required for the entire circuit. For example, currently for a 100-neuron neural network, there will be 200 op-amps in the neural network itself. elimination one op-amp per neuron bring it down to a 100 op-amps. Eliminating both op amps will be even better.

7.7 Transistor Count Comparison

The current based design discussed in this work minimizes the Transistor count of the neural network training circuit as compared to the related design discussed earlier in this work. The related work discussed in the paper "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications" uses an op-amp to create isolated copy of the output branch to add perturbation to it without affecting the normal output branch. Similarly, the existing design uses differential amplifiers to calculate the difference between the target output and the actual output and to calculate the delta error (difference between error with and without perturbation). The AD633 multiplier IC circuit used in the existing design has three Op-amps in it. Op-amps are also used as voltage adders to add memristor threshold and error to calculate the feedback adjustment voltage and as inverting amplifiers. The training circuit without the feedback adjustment part uses 12 Op-amps, two sample and hold circuits and a multiplexer. The current based training circuit utilizes two sample and hold circuits, one voltage to current convertercircuit, seven current mirrors, two absolute value circuits, two translinear multipliers, five inverters, two multiplexers and one XOR gate. Both designs require at least two multiplexers per layer to implement the mode of operation.

Eliminating similar circuitry from existing design and the current based training circuit design, the major drawback of the existing design is the use of 12 op-amps in the training circuit. A translinear multiplier (tm) has 16 transistors used in this design, current mirror (cm) has two transistors, absolute value circuits (abs) have nine transistors, inverters (inv)

have 2 transistors and multiplexers (mux) and XOR gate has six transistors. Total number of transistors when modeling the existing design and the proposed current based design in the real world is shown in (7.7) and (7.8) respectively.

$$Total\ Transistors_{Existing\ design} = 12(transistors\ in\ an\ opamp) + 2(S\&H) + mux$$

$$Total\ Transistors_{Existing\ design} = 12(transistors\ in\ an\ opamp) + 2(4) + 6$$

$$Total\ Transistors_{Existing\ design} = 12(transistors\ in\ an\ opamp) + 14 \quad \textbf{(7.7)}$$

$$Total\ Transistors_{\ Current\ based\ design} = 1(transistors\ in\ an\ opamp) + 2(tm) + 7(cm) + 2(abs) + 5(inv) + 3(mux) + 2(S\&H) + XOR$$

$$Total\ Transistors_{\ Current\ based\ design} = 1(transistors\ in\ an\ opamp) + 2(16) + 7(2) + 2(9) + 5(2) + 3(6) + 2(4) + 6Total$$

$$Transistors_{\ Current\ based\ design} = 1(transistors\ in\ an\ opamp) + 106 \quad \textbf{(7.8)}$$

The total number of transistors required in the proposed current based training circuit calculated from (7.8) assumes the worst-case scenario of 16 transistors per Translinear

multiplier. Best case scenario needs only four transistors per Translinear multiplier and requires only total of 82 transistors for the training circuit. However, the best-case scenario for the existing design is an op-amp design with the minimal possible circuitry. A common simplified op-amp model, known as the "two-stage Miller compensated op-amp," can be built using approximately seven transistors. This basic model includes two differential input transistors, a current mirror, a compensation capacitor, and a push-pull output stage. While this model can demonstrate some essential characteristics of an op-amp, it lacks the complexity and sophistication of a real op-amp, and its performance may be limited in terms of gain, bandwidth, and other specifications. This implies the existing design requires 98 transistors. Modern op-amps for practical applications and real-world performance are more complex and are built using advanced integrated circuit technology and incorporates hundreds to thousands of transistors. Thus, real world model of the existing design may use thousands to ten thousands of transistors whereas the proposed current based training circuit has only one op-amp in it and thus the number of transistors will be significantly low as compared to the existing design. The plot showing the hardware complexity (increasing number of transistors) based on the complexity of real-world op-amp circuitry is shown in Figure 7.9.

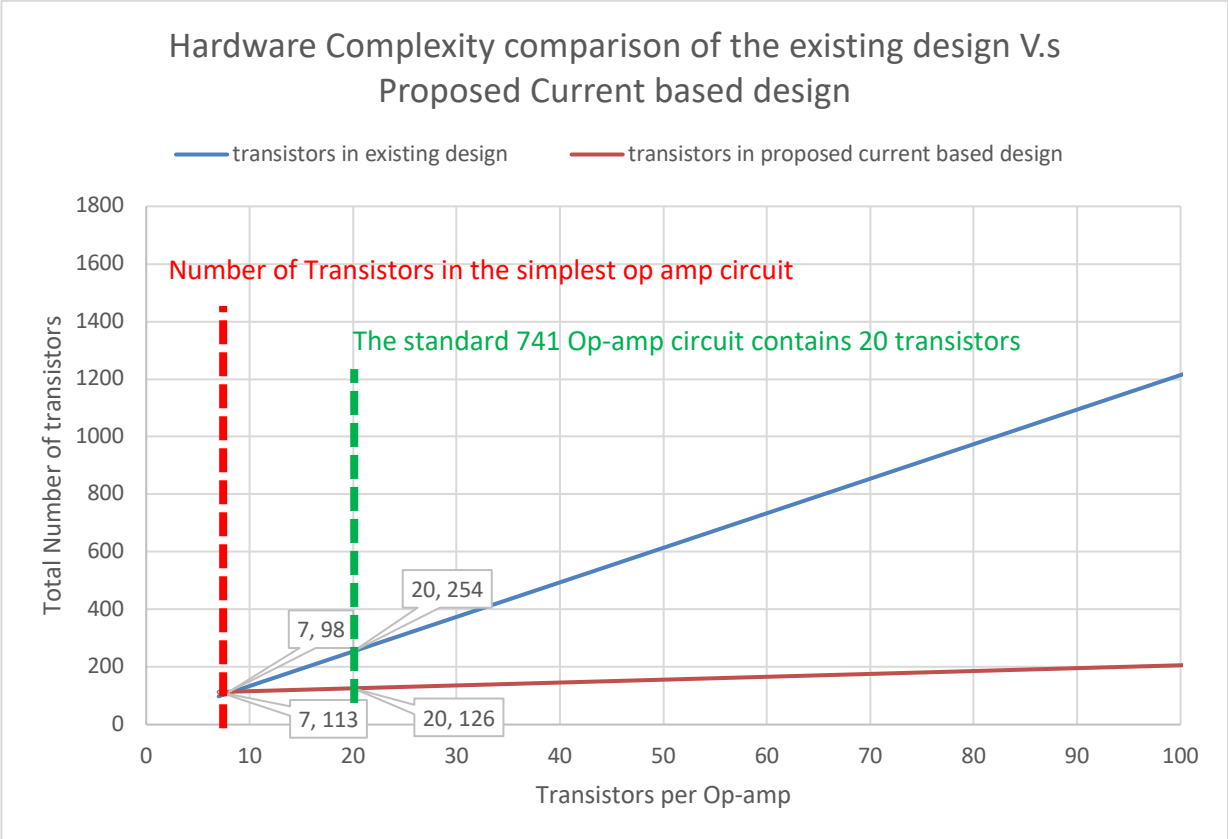


Figure 7.9: Transistor Count Comparison Plot.

From Figure 7.9, if the number of transistors in an op-amp is only seven or eight, the existing design has less transistors compared to the proposed current based design. However, as the complexity of the op-amp circuitry increases, the total number of transistors in the existing design starts increasing significantly as compared to the slow steady growth of hardware complexity in proposed current based design.

7.8 Power Consumption

The power consumption of the neural network and the training circuit was computed by adding the power generated by all the voltage sources in the system from ltspice simulation tool. The MLP circuit used to train XOR functionality along with the current-mode training circuit consumes an average power of 22.26uW. Further circuit analysis is required to compare the power consumption of the current based design and the existing design.

7.9 Summary

This chapter demonstrated how the training circuit can be scaled to train a neural network by showing that it can successfully train an XOR function. The results and discussion involved demonstration of perceptron training, memristance change during synaptic weight update, error calculation and how all the circuit components work together to train the perceptron and finally showing that it can be scaled to train linearly inseparable functions like XOR gate. The training circuit trains the neural network by adjusting the memristance of the memristors in a synapse by computing the error based on the neuron output and the target output. The training circuit calculates the error and updates the synaptic weight based on the error and the synaptic input until the error goes down to 0 and thereby making the neural network output close to equal to the target output. The actual error always lies within 0 and the theoretical error value. Thus, the error calculated by the error calculation circuit will be closer to 0. This in turn helps in faster convergence of the neural network outputs. The smaller the error, the faster the training will be. If the error and the input have the same sign, the feedback update voltage will be negative, and

the synaptic weight will decrease. Similarly, If the error and the input have the opposite sign, the feedback update voltage will be positive, and the synaptic weight will increase. The output never settles at the target current but always oscillates around the target current. This is because the feedback update voltage does not vary depending on the error margin but stays constant. Thus, once the output current goes below the target current from above the target current, the training circuit senses the error sign has flipped and hence switches the feedback update voltage sign as well which in turn brings the output current close to target current or above the target current. This process continues and hence the output current never converges to the target current but oscillates around it. This can be fixed by making the feedback adjustment voltage value dependent on the error magnitude. This will help the error to converge to 0 after training which in turn helps the output current to converge to the target current instead of oscillating around the target current. But this requires more logic and thereby more circuit components are needed which in turn increases the circuit complexity and hence the power consumption.

Bibliography

[1] Cong Xu, Chunhua Wang, Yichuang Sun, Qinghui Hong, Quanli Deng, Haowen Chen, "Memristor-based neural network circuit with weighted sum simultaneous perturbation training and its applications". Neurocomputing, June 27, 2021. https://www.journals.elsevier.com/neurocomputing.

[2] M. Shery, H. Hagar, L. Dylan "An attempt to Minimize Hardware Complexity by Utilizing Zeroth Order Optimization Technique for Training Analog Neural Network," 2022, Neuromorphic Computing Final Project Paper, RIT.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary

[4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions,," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2015. IEEE, 2015, pp. 1–9.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *arXiv*, 2015.

[6] S. M. Noor, J. Ren, S. Marshall, and K. Michael, "Hyperspectral Image Enhancement and Mixture Deep-Learning Classification of Corneal Epithelium Injuries," *Sensors. Multidisciplinary Digital Publishing Institute*, vol. 17, no. 11, p. 2644, 2017.

[7] T. N. Sainath, B. Kingsbury, G. Saon, H. Soltau, A. Mohamed, G. Dahl, and B. Ramabhadran, "Deep Convolutional Neural Networks for Large-scale Speech Tasks," *Neural Networks*, vol. 64, pp. 39–48, 2015. [Online].

[8] A. Graves, A. Mohamed, and undefined G. Hinton, "Speech recognition with deep recurrent neural networks,," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 6645–6649.

[9] S. Albarqouni, C. Baur, F. Achilles, V. Belagiannis, S. Demirci, and N. Navab, "AggNet: Deep Learning From Crowds for Mitosis Detection in Breast Cancer Histology Images,," *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1313–1321, 5 2016.

[10] D. Wang, A. Khosla, R. Gargeya, H. Irshad, and A. H. Beck, "Deep Learning for Identifying Metastatic Breast Cancer," *arXiv*, 2016. [Online].

[11] A. A. Cruz-Roa, J. A. Ovalle, A. Madabhushi, and F. G. Osorio, "A Deep Learning Architecture for Image Representation," *Springer*, pp. 403–410, 2013. [Online].

[12] L. Lazimul and D.Binoy, "Fingerprint liveness detection using convolutional

neural network and fingerprint image enhancement," 2017. [Online].

[13] H. Jung and Y. S. Heo, "Fingerprint liveness map construction using convolutional neural network," *Electronics Letters*, vol. 54, no. 9, pp. 564–566, 2018. [Online].

[14] A. L. Xar, Y. Demir, C. Neyt, and G. Zelis, "Object recognition and detection with deep learning for autonomous driving applications,," *Modeling and Simulation International*, vol. 93, no. 9, pp. 759–769, 2017. [Online].

[15] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, *DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving*, 2015. [Online].

[16] A. Grover, A. Kapoor, and E. Horvitz, "A Deep Hybrid Model for Weather Forecasting." New York, New York, USA: ACM Press, 2015, pp. 379–386.

[17] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu, *Forecasting the weather of Nevada: A deep learning approach*, 2015. [Online].

[18] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," Haifa, 2010, pp. 807–814. [Online].

[19] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, and G. E. Hinton, "On rectified linear units for speech processing," in *International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3517–3521, IEEE. https://doi.org/10.1109/ICASSP.2013.6638312.

[20] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013.

[21] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning." MIT Press, 2016.

[22] A. Maas, A. Hannun, and A. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *International Conference on Machine Learning (icml)*, 2013.

[23]X.GlorotandY.Bengio,"Understandingthedifficultyoftrainingdeepfeedforwardneuralnetworks,"in*InternationalConferenceonArtificialIntelligence and Statistics*, vol. 9. PLMR, 2010, pp. 249–256. [Online].

[24] M. Shery, R. Patil "Neuromorphic Computing: Brain Like Computing Using Memristors," 2021, Computer Architecture paper, RIT.

[25] Sijia Liu, Pin-Yu Chen, Bhavya Kailkhura, Gaoyuan Zhang, Alfred O. Hero III, and Pramod K. Varshney, "A Primer on Zeroth-Order Optimization in Signal Processing and Machine Learning."

[26] L.Chua,"Memristor-The missing circuit element,"in IEEE Transactions on Circuit Theory, vol. 18, no. 5, pp. 507-519, September 1971.

[27] A. Huang, X. Zhang, R. Li, and Y. Chi, “Memristor Neural Network Design,” IntechOpen, 20-Dec-2017. [Online]. Available: https://www.intechopen.com/books/memristor-and-memristive-neural-networks/memristor-neural-network-design.

[28] A. M. Zador, “A critique of pure learning and what artificial neural networks can learn from animal brains,” Nature News, 21-Aug-2019. [Online]. Available: https://www.nature.com/articles/s41467-019-11786- 6.

[29] C. Mead, “Neuromorphic electronic system.”

[30] S. Lu and A. Sengupta, “Exploring the Connection Between Binary and Spiking Neural Networks,“ Frontiers, 30-Apr-2020. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fnins.2020.00535/full.

[31] Low cost analog multiplier AD633 datasheet.