B*asic cell for artificial neural network in CMOS technology*

Givaldo Lino Mendes Vieira Neto

Table of Contents

Table of Figures

Abstract

Artificial Neural Networks are the best-known method to performs learning the way of biological systems do. Neural networks are widely used in many classification applications such as pattern classification, speech recognition etc. In this work we are developing basically neuron cell in CMOS technology for an Artificial Neural Network in analog VLSI. Artificial intelligence through a biological point of view is realized based on mathematical equations and artificial neurons. In the proposed paper it was presented different topologies with different technology to implement a single neuron, after a carefully studied regarding the specification and time limit it was chose the neuron based on Gilbert cell multiplier, adder and neuron activation function to form a basic block of the neuron. The designed neuron is suitable for both Analog and digital applications. The layout design and verification is carried out using cadence virtuoso tool to validate it.

I. Introduction

The artificial neural network (ANN) is gaining notoriety in electronics field. The artificial neural network learns from samples of input and output data that the used learning algorithm is based on multiplier error correction learning and also back propagation. ANN can be trained adjusting the weights to perform a specific task; the weights are constantly updated by the learning algorithm comparing the output of the neural network with the target until output matches the target in the sense that the error function measuring the difference between the target and the output is minimized.

ANN has a lot of applications in different fields; examples are processing of visual information [3] for an autonomous car, telecommunication systems and etc. "*Now, MIT researchers have developed a special-purpose chip that increases the speed of neural-network computations by three to seven times over its predecessors, while reducing power consumption 94 to 95 percent. That could make it practical to run neural networks locally on smartphones or even to embed them in household appliances. "[10] MIT News Officer, Larry Hardesty*

In this work we are going to develop basically neuron cell in CMOS technology in analog VLSI platform,the layout design and verification are carried out using cadence virtuoso tool

II. Principles of Artificial Neural Network

The focus of this project is a single neuron but it is necessary to explain the basic principal of a neural network before going to the implementation of the artificial neuron in VLSI technology, the basic architecture of ANN is composed by three types of layer: input, hidden and output layers (figure 1). A hidden layer in an artificial neural network is a layer in between input layers and output layers, where artificial neurons take in a set of weighted inputs and produce an output through an activation function. It is a typical part of nearly any neural network in which engineers simulate the types of activity that go on in the human brain. [8]

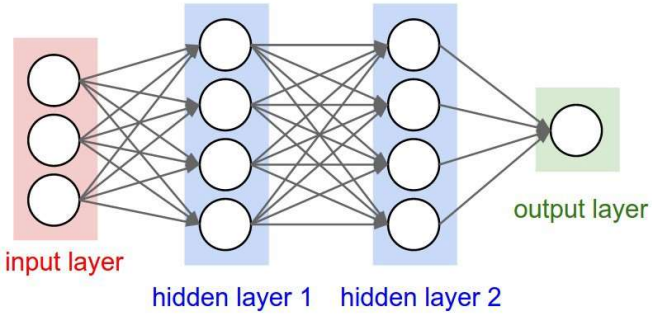

Figure 1: example of an artificial neural network

The first and elemental model of artificial neurons [4] [5] is called *perceptron,* developed in the 1950s and 1960s by the scientist Frank Rosenblatt.

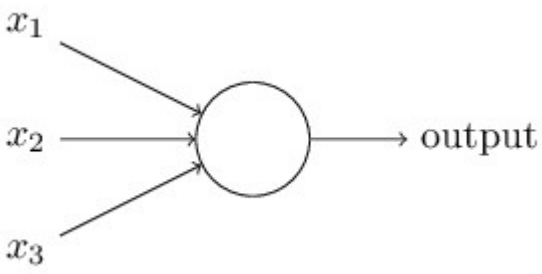

Figure 2: Example of a perceptron neuron with 3 inputs

Rosenblatt proposed a simple rule to compute the output, he introduced the concept of *weights* (w_1, w_2, w_3) in neural network. The weights are real numbers which express the importance of the respective inputs to the output, *figure 2*. The neuron's output is determined by the sum of the relation input and weight ($\sum_j w_j x_j$) and compared to some threshold value ; just like the weights the trhreshold value is a real number which dependes on the neuron's parameter. The algebric terms can be represented by :

$$output = \begin{cases} 0 if \sum_j w_j x_j < threshold \\ 1 if \sum_j w_j x_j > threshold \end{cases} \quad (1)$$

In equation 1, it can also have output equal to -1, if the equation works in a bipolar system. We can also rewrite the *equation 1* moving the threshold to the other side of the inequality, it is called bias point (b = - threshold). So it will be write as:

$$output = \begin{cases} 0 if \sum_j w_j x_j + b < 0 \\ 1 if \sum_j w_j x_j + b > 0 \end{cases} \quad (2)$$

In biological terms, the bias point is a measure of how easy the neuron is going to *fire.* For a perceptron with a really big bias, it's extremely easy for the neuron to output a 1. But if the bias is very negative, then it's difficult for the perceptron to output a 1.

But since we want to implement a learning algorithm, a small change in some weights or bias can cause a drastically change in the output flipping 0 to 1 or vice versa, so it is important that a small change in the weights reflect only a small change in the output (*figure3*).

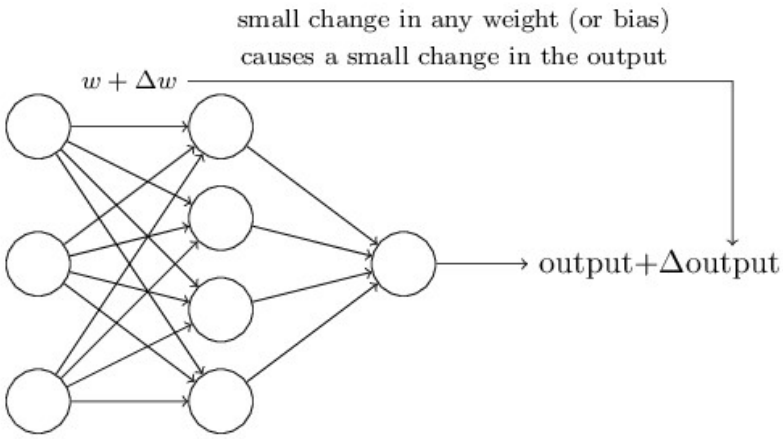

Figure 3: Example of the sigmoid neuron's effect

To overcome this problem, it is necessary to introduce the concept of *sigmoid neuron.* Sigmoid neurons are similar to perceptron neurons but instead of the output being only 0 or 1, it can assume values between 0 and 1 with a sigmoid function, therefore the output of the neuron will be represented by:

$$output = \sigma\left(\sum_j w_j x_j + b\right) \tag{3}$$

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

With σ being the sigmoid function (figure 4), it is important to mention that the shape of the sigmoid function can be smoother depending on the system and the learning process.

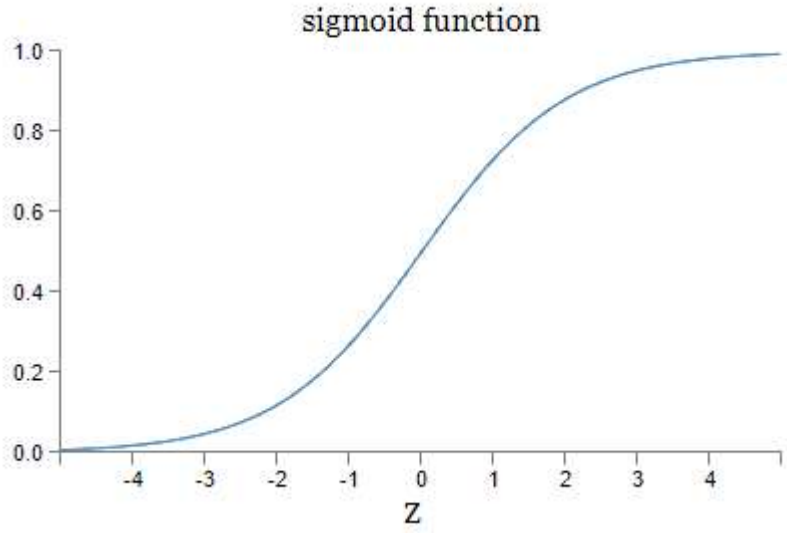

Figure 4: Shape of the sigmoid function for binary system

Information flows through a neural network in two ways. When it's learning (being trained) or operating normally (after being trained), patterns of information are fed into the network via the input units(figure 1), which trigger the layers of hidden units, and these in turn arrive at the output units. This common design is called a *feedforward network*. Not all units "fire" all the time. Each unit receives inputs from the units to its left, and the inputs are multiplied by the weights of the connections they travel along. Every unit adds up all the inputs it receives and the bias point in this way and if the sum is more than a certain value, the unit "fires" and triggers the units it's connected to (those on its right).

The procedure used to carry out the learning process is called training (or learning) strategy. The training strategy is applied to the neural network to in order to obtain the minimum possible minimum loss, without a training strategy neurals network would not have any accuracy of the results. The type of training is determined by the way in which the adjustment of the parameters in the neural network takes place. To ilustrate more about training and neural network implementation in a high level it was developed, in the appendix, a explanation in back propagation and an application in pattern recognition using Matlab/Simulink

III. Methodology

Specification

It is necessary to evaluate our specification and purposes for this project before choosing the right topology and doing simulation. The specification for this work is to use X-FAB 180n m technology in full-custom design, bottom up approach and frequency operation 100k Hz. A symmetric power source of ± 3.3 was chose due to simplicity for designing the system, since this is the first approach in neural network basic cell for a further and improvement work, we are not going to delimited total area and power consumption. The purpose of this work is to give a first approach in artificial neural network and proposed a basic neuron cell that can be further implemented a neural network. The behavioural will be evaluated and the topology will be chosen and test accordantly to the specification above.

Base topology study

In order to choose the right topology, considering the technology already established, it was made a search in different topology and technologies implemented in analog artificial neural network system.

In the article *Analog CMOS-based Resistive Processing Unit for Deep Neural Network Training [11]* they implemented an artifical neural network using CMOS-based resistence processing unit (RPU) showed in figure 5.

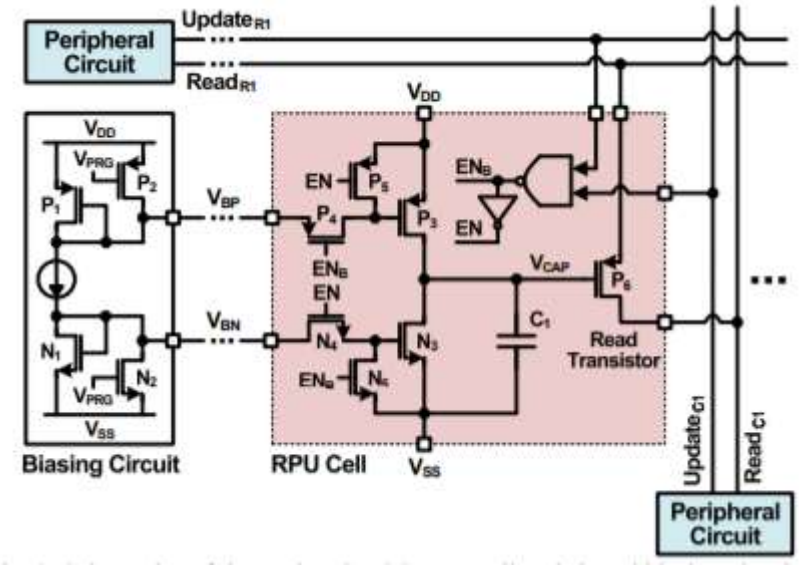

Figure 5 : Schematics of the analog CMOS RPU cell working as neuron.

In this design, the capacitor, C1, serves as a memory element in the cell and stores the weight value in the form of electric charge.The capacitor voltage, Vcap, is directly applied to the gate terminal of the read transistor, P6.Therefore, the charge state stored in the capacitor can be accessed by applying small bias across P6 and measuring the current.

For the weight update, a stochastic computing scheme is used where the local multiplication operation is performed by using a simple coincidence detection method. With the NAND and inverter logics connected with UpdateR1 and UpdateC1(figure 6), the stored weight in a CMOS RPU cell can be updated only when two stochastic pulses from update lines (row and column) are coinciding.

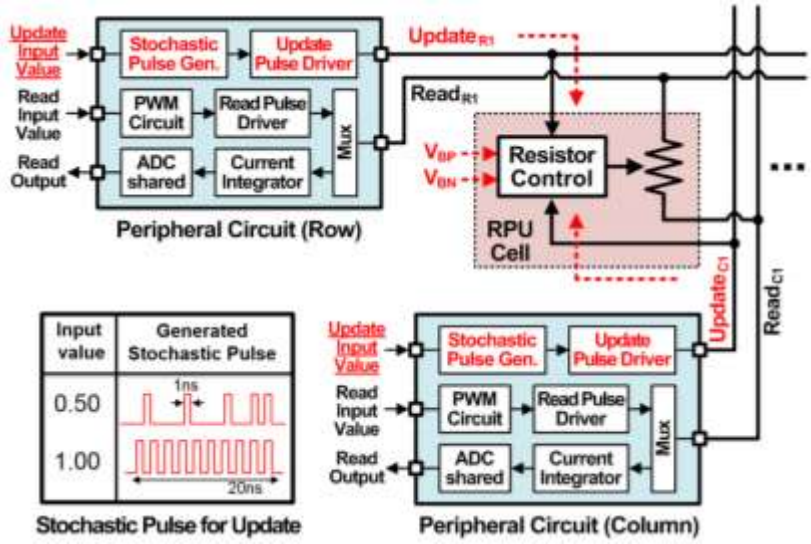

Figure 6 : Block Diagram of the RPU cell and peripheralcircuits to construct a neuron.

Since this article propose a neural network based in a charge storage inside a capacitor it is necessary to implement a peripheral circuit to read and write the storage value, depending on the number of neurons this topology can have a high complexity design.

- This topology requires the studied of all the peripheral illustrated in figure 6 previously and a processor to implement the neuron network as algorithm and stock the weights values in RPU cell. But regarding the technology proposed, high complexity of the system and we desire to implement in a full analog system this topology is not adequate to our project.

The second article, "*Back-Propagation Operation for Analog Neural Network Hardware with Synapse Components Having Hysteresis Characteristics*" [13], makes use of a ferroelectric memristor (FeMEM) and op-amp to construct a neuron (figure 7). R_F is a fixed resistance, whose conductance is G_R. To achieve a synapse

function using an FeMEM, the synaptic circuit modules were devised that consist of inhibitory/excitatory synapse pairs. As the op-amp adder circuit is an inverting amplifier circuit, the inhibitory pairs receive raw input directly and the excitatory ones receive inverted copies of the raw input voltage via a unity gain inverting amplifier. Although this synapse circuit construction needs two FeMEMs, a highly functional neuron circuit can be realized, because the modulation of synapse weight is easier to control individually with two FeMEMs. Here, we denote the channel conductance of FeMEM as G_F.

This topology could be implemented but since the specifications are clear about the technology that need to be used, X-Fab and not a MEM. For specification reason and complexity of the system this topology also is not adequate to our project.

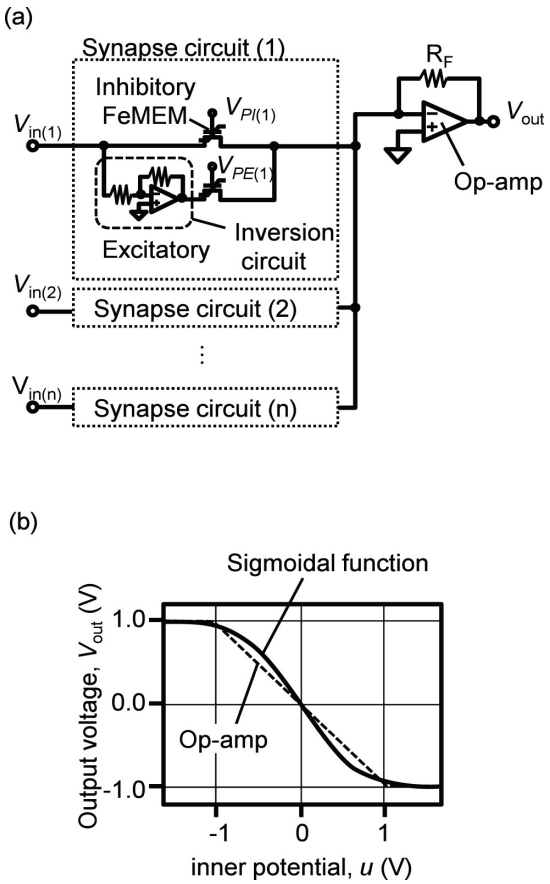

Figure 7 : (a)Topology proposed in article[13] (b)Result obtained Vout in the article [13].

The third topology is an interesting topology is explored in article artificial neuron with switched-capacitor synapses using analog storage of synaptic weights [14] as the title is self-explained they designed an artificial neural network base on capacitive switches, A pseudo-analog electronic or optoelectronic neuron stores synaptic weights as analog quantities, preferably as charges upon capacitors or upon the gates of floating gate transistors.

Multiplication of a stored synaptic weight times a binary pulse-width-modulated synapse input signal periodically produces electrical charge of a first polarity on a first synapse capacitor. Meanwhile a fixed charge of opposite polarity is periodically produced at the same frequency upon another, second, synapse capacitor. The charges on both synapse capacitors at many synapses are periodically accumulated, and integrated, at a single neuron soma in the form of pulse amplitude-modulated charge-encoded signals. This topology has a high complexity since it is necessary to have commands control ϕ_e and ϕ_0 (figure 8) which they are responsible to charge C2 and C1, synapse, and after sent to integration circuit with an op-amp, sum. It also necessary to convert voltage (neuron output) to pulse width modulation (PWM) to work as a command control for the next neuron.

This topology has an even high complexity compare to the previously articles, for complexity reason this topology is not suitable for our work.

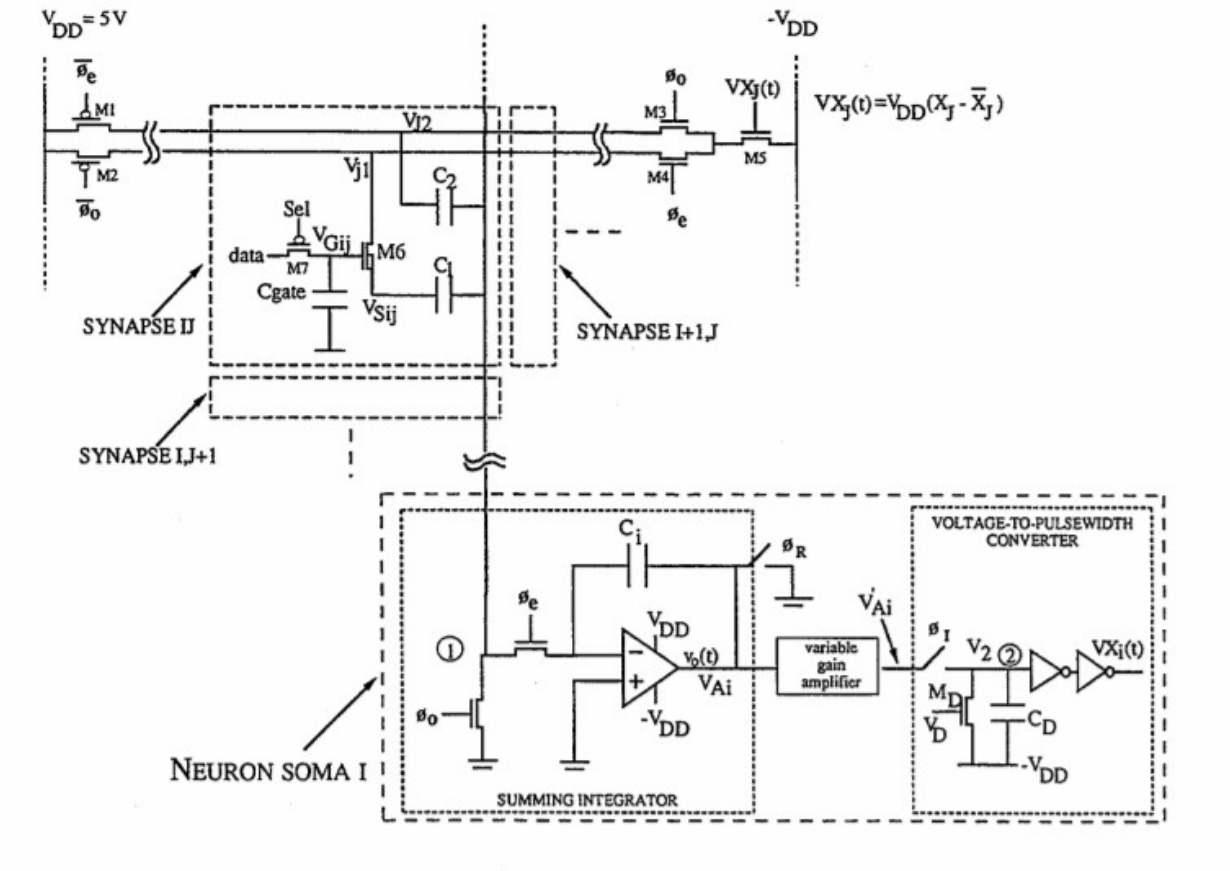

Figure 8 : Topology proposed in article [14]

The fourth topology use a CMOS neural network based on Gilbert Multiplier proposed in the articles "*Analog CMOS Neural Networks Based on Gilbert Multipliers with In-Circuit Learning" [12],* "*Analog VLSI Implementation of Artificial Neural Network*" [1] and "*Analog VLSI Implementation of Neural Network Architecture for Signal Processing"* [2]. This topology makes use of a wide use

ciruit,Gilbert Cell (figure 9), in telecommunication and radio application to work as multiplier operation in the artificial neural network in the [12] they focus on different learning algorithms, which is not the goal in this work however both [1] and [2] work on the design for single neuron using different technologies than can be adapted for our application and specification. Another interesting aspect of this topology is not needed to use a capacitor and for this reason the total area can be decrease compare to other topologies and also the transconductance characteristic simplify the use of a large number of inputs in a neuron.

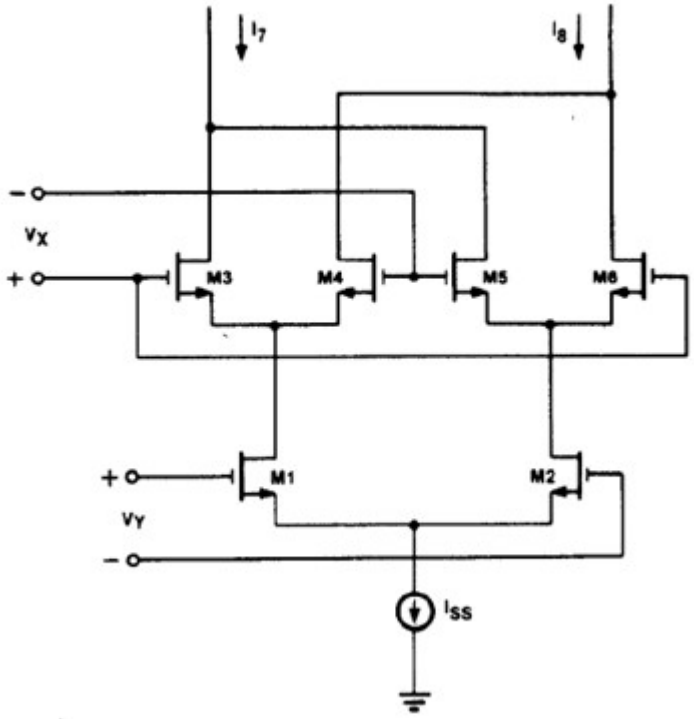

Figure 9: Diagram of a gilbert cell

Looking at the different topologies studied and regarding of time limitation for this internship and, also, simplicity to implement the topology proposed we decided to choose the topology base on Gilbert Cell Multiplier [1] [2] [12] since Gilbert cell has a transconductance characteristic, which reduce the complexity of the system and the simplicity to designed and test each block separately and after test the neuron as a whole. Also, the articles [1] and [2] make use of a technology close to our, so it an opportunity to compare the results in the articles to ours and validate the functionality of the neuron.

IV. Block diagram

For the implementation using analog VLSI, we divided in blocks, as discussed before, the basic operations in a single neuron are multiplication, between input and the respective weight, sum of all the inputs in the neuron and the activation function, sigmoid function, as shown in the figure 10.

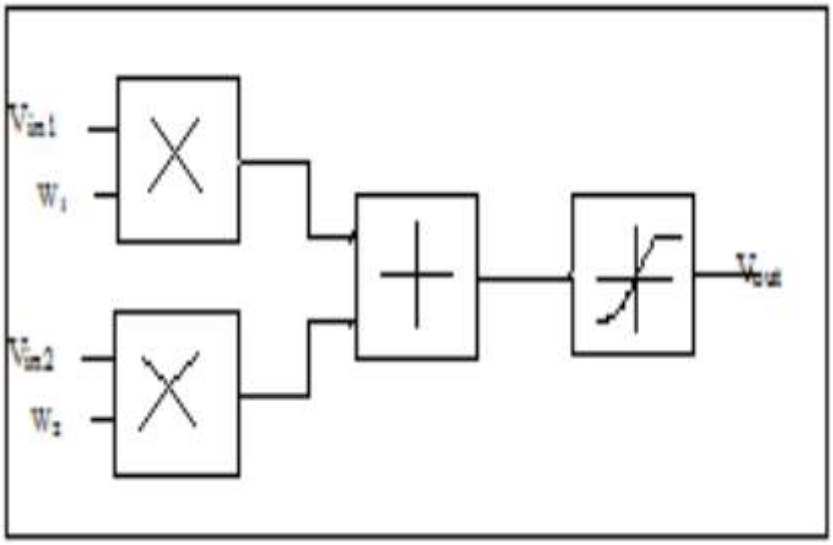

Figure 10: Representation of a single neuron in block diagram for 2 inputs.

The Gilbert cell is a transconductance, input in voltage and output in current, as explained before, the application of the sum block will be a simple wire connecting the two Gilbert cell. For the activation function block input it is necessary to translate the current to voltage (figure 10).

The activation function block will be implemented using a differential amplifier, which we can change the gain, than the shape of the curve, depending on the bias voltage and adjusting the curve to any case.

V. Gilbert Cell

Gilbert cell is a type of mixer; it is widely used for frequency conversion in radio system. It produces an output signal that ais proportional to the product of two input signals, so we are going to implement this concept in our neuron basic cell. The Gilbert cell mixer topology was first used as a mixer by Barrie Gilbert around 1967.

The Gilbert cell mixer essentially comprises two differential transistor pairs whose bias current is controlled by one of the input signals. The other input signal drives the gate electrodes of the differential pair transistors (figure 9), where the gain is controlled by modulating the drain bias current. All transistors work in saturation region, as explained before the big advantage for using a gilbert cell as multiplication block is the transconductance aspect, so if it is necessary to implement a neuron with a large number of inputs we can use the same number of Gilbert cells as there are inputs but for the sum block it will only be necessary to connect the output of all Gilbert Cell since the output will be in current. To obtain a linear circuit without errors in the output it is necessary to studies the parameters of the gilbert cell.

Gain

The conversion gain for a Gilbert Cell consists of three parts, according to Shasanka Sekhar Rout and Kabiraj Sethi [16], the transconductance, the switching gain (A_{sw}) and the output impedance (RL).

$$Gain = g_m \, R_L \, A_{sw}$$

Where, A_{sw} is a function of the shape and the amplitude of the Vx (figure 9) drive and the over-drive voltage of the switching pair ($V_{od,sw}$). If the Vx signal is a square wave with sharp rising and falling edges and its amplitude is larger than $V_{od,sw}$ .If the Vx signal is a sinusoid with a amplitudeV_{Vx} much larger than $V_{od,sw}$ the switching gain is close to that using a square wave; therefore, the switching gain is close to 2/π.The switching gain is proportional to the Vx amplitude when the Vx amplitude is smaller than the over-drive voltage. A larger Vx drive can provide a higher switching gain, too large a Vx drive also degrades the conversion gain because even order harmonics are coupled into the common source of the differential pair, which is also connected to the drain of the Vy input transistors.

A current source is connected to the common source of the differential pair to steal part of DC current from the drain of the input transistor so that there is less DC current flowing through the differential pair and the over-drive voltage becomes

smaller. Only the DC current at the output is reduced while the AC current which contains all the signals remains unchanged.

Noise

There are three major noise sources in a Gilbert Cell: noise generated by the input transistors, the switching noise and by the output load. The first noise, input transistor is composed by the drain thermal noise:

$$i_{d,noise}^2 = 4kT\gamma g_m \tag{11}$$

$$v_{,therm}^2 = \frac{4kT\gamma}{g_m} \tag{12}$$

Another important noise source in the input transistor is the induced gate noise, whose input referred value is given by:

$$i_{g,noise}^2 = 4kT\delta g_g \tag{13}$$

With

$$g_g = \frac{w^2 C_{gs}^2}{5g_{d0}} \tag{14}$$

Where w is the angular frequencyC_{gs} is the gate-to-source capacitance and g_{d0} is the drain conductance when V_{ds} equals to zero, the inductance gate noise is correlated to the drain thermal noise. The total noise in a gilbert cell, especially if we are working in high frequency, can be optimized by use of the proper input impedance. The induced gate noise is partially correlated to the drain thermal noise.

Another switch noise component is flicker noise, which can be modelled as a voltage source in series with the gate. Its time-average power spectral density is given by Behzad Razavi [15], where W of Vx and L of Vx are the witch and length of each switching transistor, respectively.

$$S_{n\,M1M2} = \alpha\, 4KT(R_S + 2\, r_{g\,M1} + \frac{2\gamma}{g_{m\,M1}})g_{m\,M1}^2 \tag{15}$$

$$S_{n\,flicker}(f) = \frac{2K}{WLC_{ox}f} \tag{16}$$

The contribution of the flicker noise to the output noise power spectral density is the same as of parasitic gate resistances. However, since flicker noise is not a white noise and negligible values at high frequencies.

Consider that the load introduces output noise which can be represented by an equivalent noise resistance R_L.

$$\mathrm{S_{n\,load}} = \frac{4\mathrm{kT}}{\mathrm{R_L}} \tag{17}$$

So the total contribution noise inside a gilbert cell could be express by :

$$S_{n\,total} = S_{n\,RF} + S_{n\,flicker}(f) + \mathrm{S_{n\,load}} \tag{18}$$

VI. Activation Function

The activation function block has the purpose to work as sigmoid function, the chosen circuit is a simple differential amplifier with active load. The differential amplifier is probably the most widely used circuit building block in analog integrated circuits, principally op amps. A differential amplifier multiplies the voltage difference between two inputs (V1 – V2) by some constant factor Ad (figure 11), the differential gain. It may have either one output or a pair of outputs where the signal of interest is the voltage difference between the two outputs. A differential amplifier also tends to reject the part of the input signals that are common to both inputs (Vin+ + Vin-)/2. This is referred to as the common mode signal.

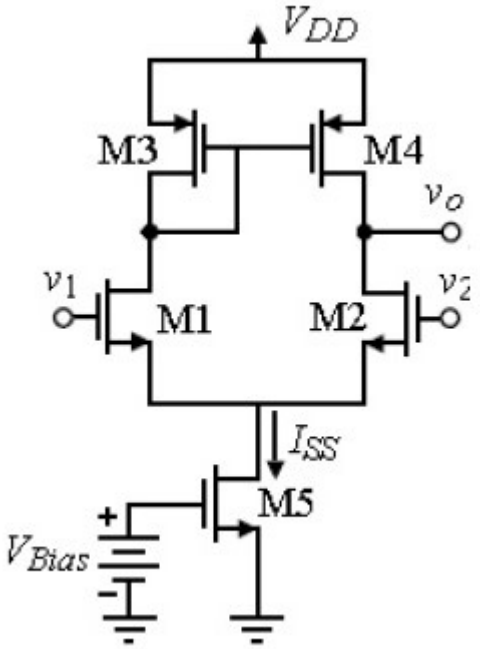

Figure 9: Diagram of a differential amplifier

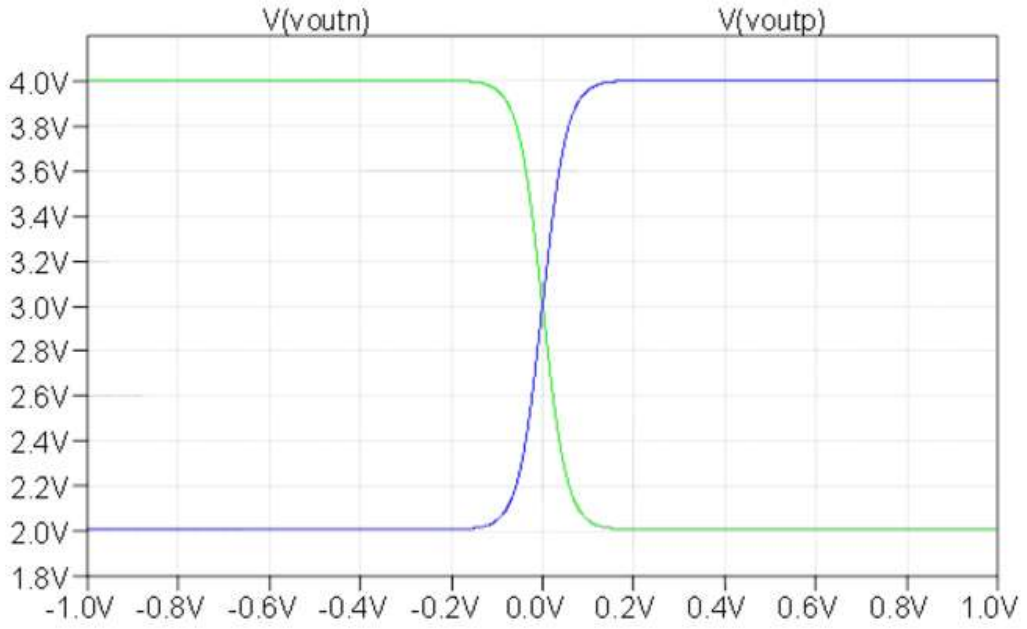

Figure 10: The output of a differential amplifier

As we can see at the figure 12 approximate to a sigmoid function, we are only interested in one output so Vo as showed in figure 12. M1 and M2 are input NMOS devices whose transconductance appears in gain expression. We keep these devices enough wide operating with small overdrive voltage so that they can produce high gain and high Input Common Mode Rejection ratio (ICMR). The PMOS devices M3 and M4 should exhibit high output resistance when we want high gain, thus we keep them enough long. Tail transistor, M5, should have roughly twice overdrive voltage as compare to input devices. The shape of the curve can by changing the $\frac{W}{L}$ and linearity can be improved by decreasing $\frac{W}{L}$ or increasing the tail current, Iss.

Output impedance

From the small signal analysis, it is easy to see that the output impedance is express as follow:

$$R_{out} = r_{o4}||r_{o2} \tag{19}$$

Gain

The gain for differential amplifier can be express as:

$$Av = \frac{V_{out}}{V_{in}} = g_m R_{out} \tag{20}$$

Slew Rate

We can define the slew rate in function of the bias current, Iss, and the total capacitance in the load, we are going to call it C_L.

$$SR = \frac{Iss}{C_L} \tag{21}$$

Gain-Bandwidth

Since we want to the output to be a high impedance node, we can express the gain Bandwidth as follow:

$$GBW = \frac{g_m}{C_L} \tag{22}$$

VII. Results and Simulation

With all the blocks carefully chosen, in the articles [1] and [2] they make use of a system with power source ±1.8 V and operation frequency 10M Hz it was necessary to change the circuit, both Activation function and Gilbert cell, for our specification mentioned in the introduction. To validate the theory using Cadence Virtuoso Analog Design Environment to simulate the behaviour of each block and compare the result with articles [1] and [2].

Activation Function

The activation block (figure 13) was sized in consideration of the highest gain without signal wave getting distortion in the output.

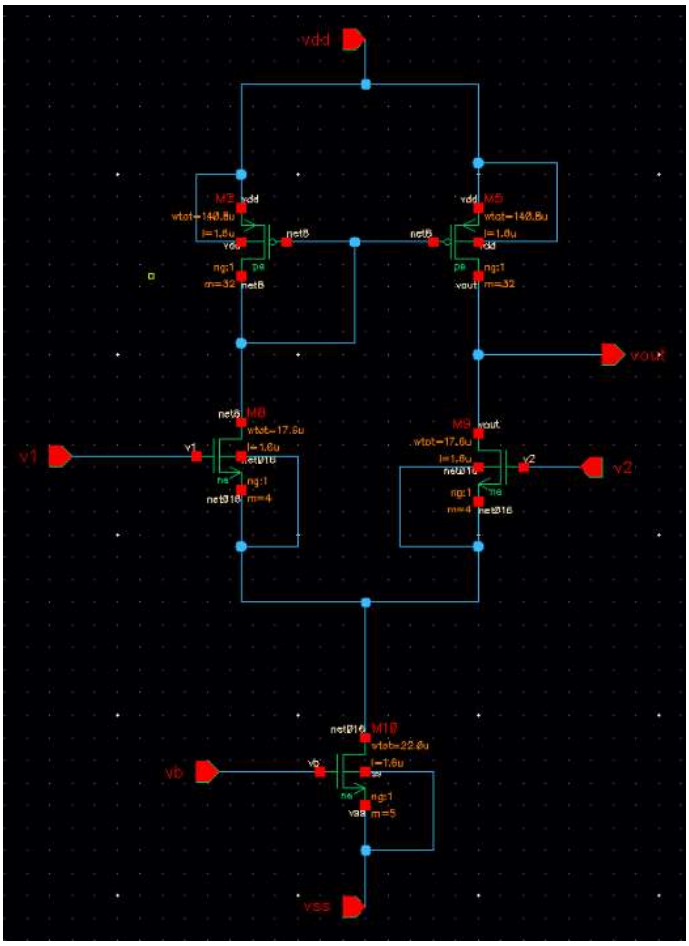

Figure 11: Diagram of the implementation of activation function block in cadence environment.

The slew rate can be measure in figure 14, around 52 V/µs, and we can see that it approaches the sigmoid function's behaviour with a slightly offset (300m V).

We can also see the layout in figure 15, instead of using a big transistor it was use a large number of transistor,32 transistors in parallel for M2 and M5, 4 transistor for M8 and M9 and finally one transistors for M10 (table 1), all of them with the same size (L= 180n W=4.44u), the transistors were sized regarding the highest slew rate and gain possible (high R_{out} see equation 19 and 20). It was chosen a base transistor of length equals to 180n m and width of 4.44u m because it is the smallest transistor used in Activation function and Gilbert cell so it is necessary to designed one transistor and the other transistors are parallels of base transistor, the layout has a total surface of 50x20 µm

Transistor	Total Length(L)	Total Width(W)	Num. of base transistor
M2	180n	140.8 u	32
M5	180n	140.8 u	32
M8	180n	17.6 u	4
M9	180n	17.6 u	4
M10	180n	4.44 u	1

Table 1: Size of transistor in Activation function on figure13.

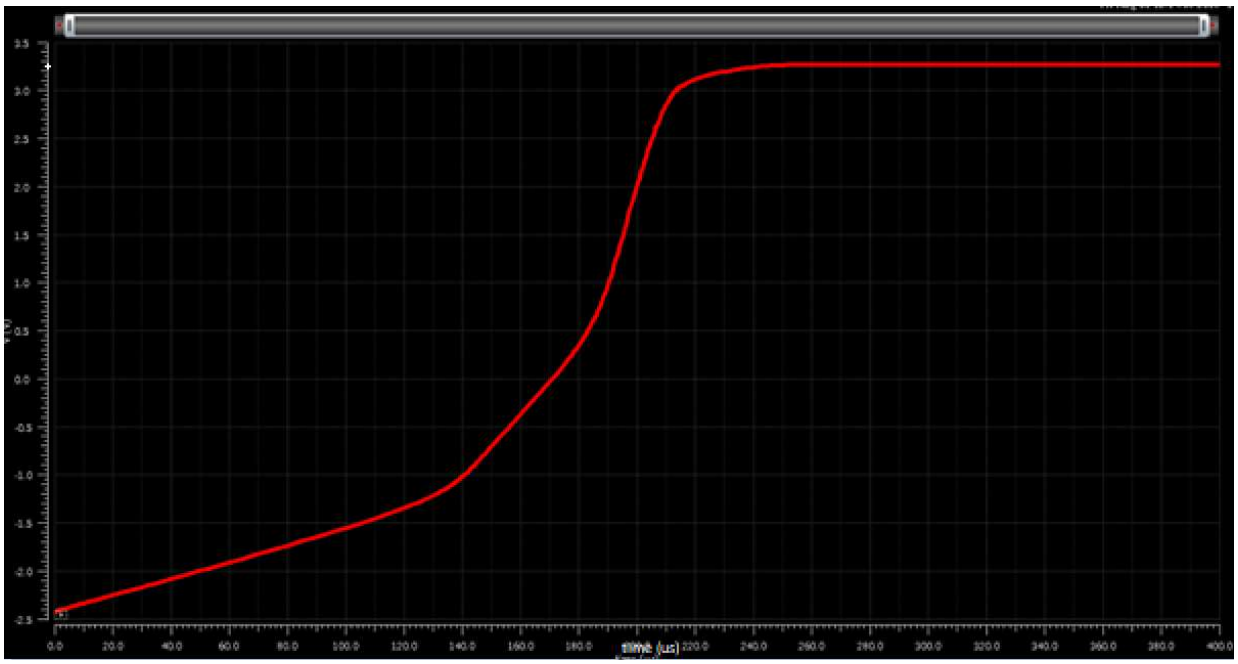

Figure 12: Output wave of activation function block in cadence environment.

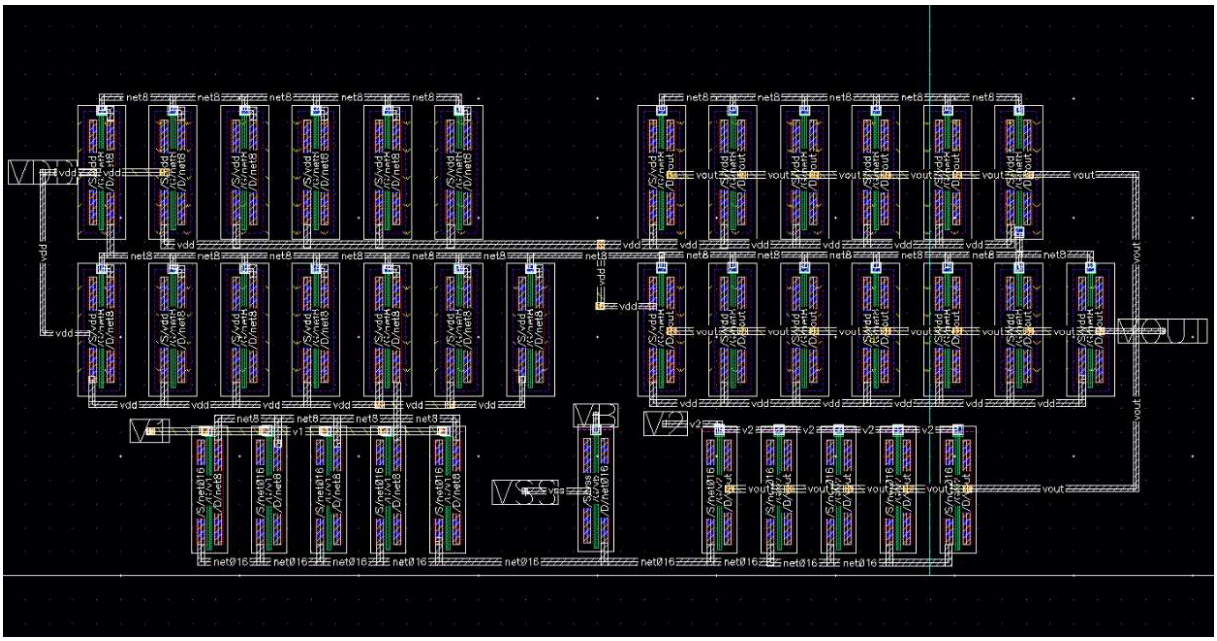

Figure 13: Layout of activation function block in cadence environment.

Using Virtuoso Analog Design Environment, we were able extract the total average power consumption, it was made by simulating the total circuit with ADL simulation and extracting the total power consumption of the circuit via *power_extract function* and then using a calculator to express the value average of *power_extract function*. For this circuit, differential amplifier, we found a total average power consumption equal to 3m W.

Gilbert Cell

For the Gilbert cell (figure16), multiplication block, it was required to make a trade-off between gain and stability. Since the multiplication block needs to work in a frequency defined before (100 K Hz) for that reason the size of the Gilbert cell does not need to be small to work in the frequency desired, but in other hand with low stability we cannot have reliance in the output data, for this reason the inputs voltage in the gilbert cell have limitations parameters, between 100mV and 1V, if the inputs are out of the range the system will not convert, saturating and leading to a error in the output.

Also the output have considerable offset, which can cause error, but to fix this anomaly it is crucial to set the bias input correctly.

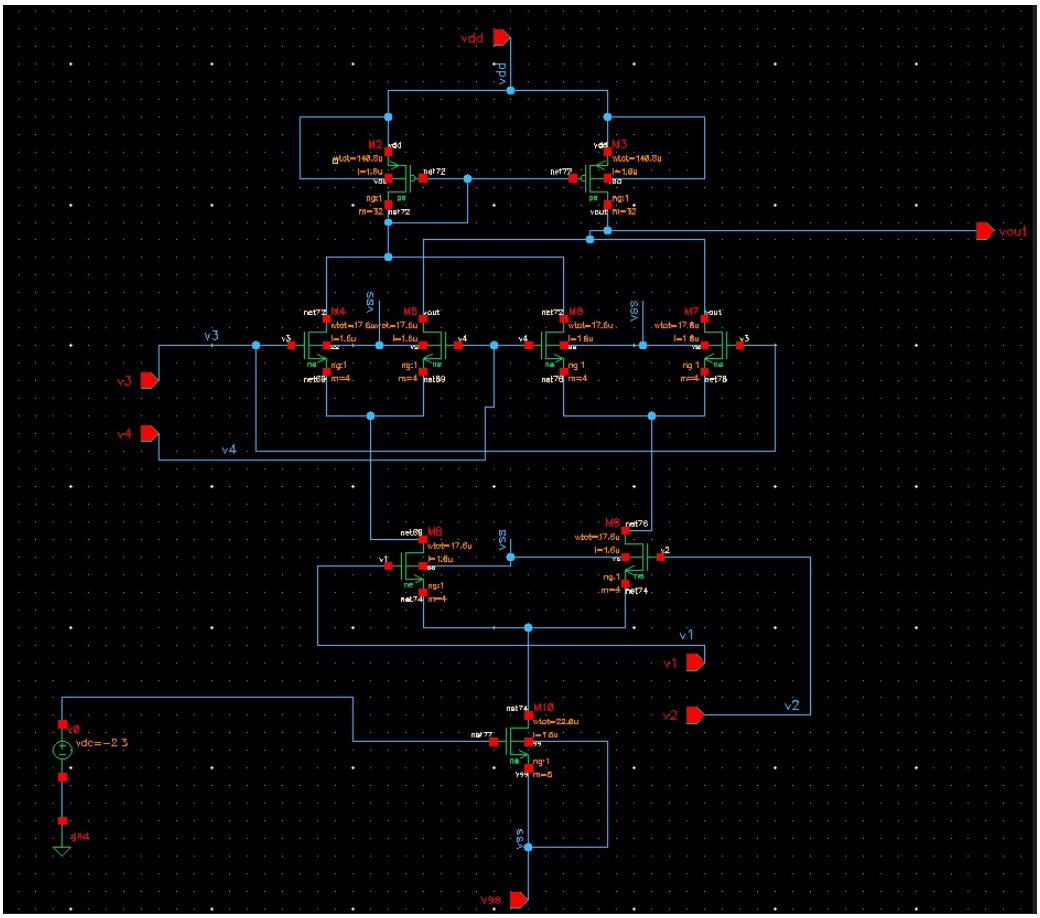

Figure 14: Diagram of the implementation of Multiplication block in cadence environment.

It is showed in figure 17 a test bench of the Gilbert Cell, using ADL Simulation for a total time of 200 μ seconds. It is noticeable that the output is a multiplication between two sines waves with different frequencies (10k and 100k Hz), testbench inside the frequency operation, also remarkable the offset in the output (0.9 V) giving a total offset 1.1 V but the multiplication of both sines is presented.

The layout (figure 18)use the same principal of the activation function block, all of them with the same size (L= 180n W=4.44u) ,32 transistor in parallels for M2 and M3, 4 transistors for M4, M5, M6, M7, M8 and M9 and 5 transistors for M10 (table 2). The layout has a total surface of 96x44 μm.

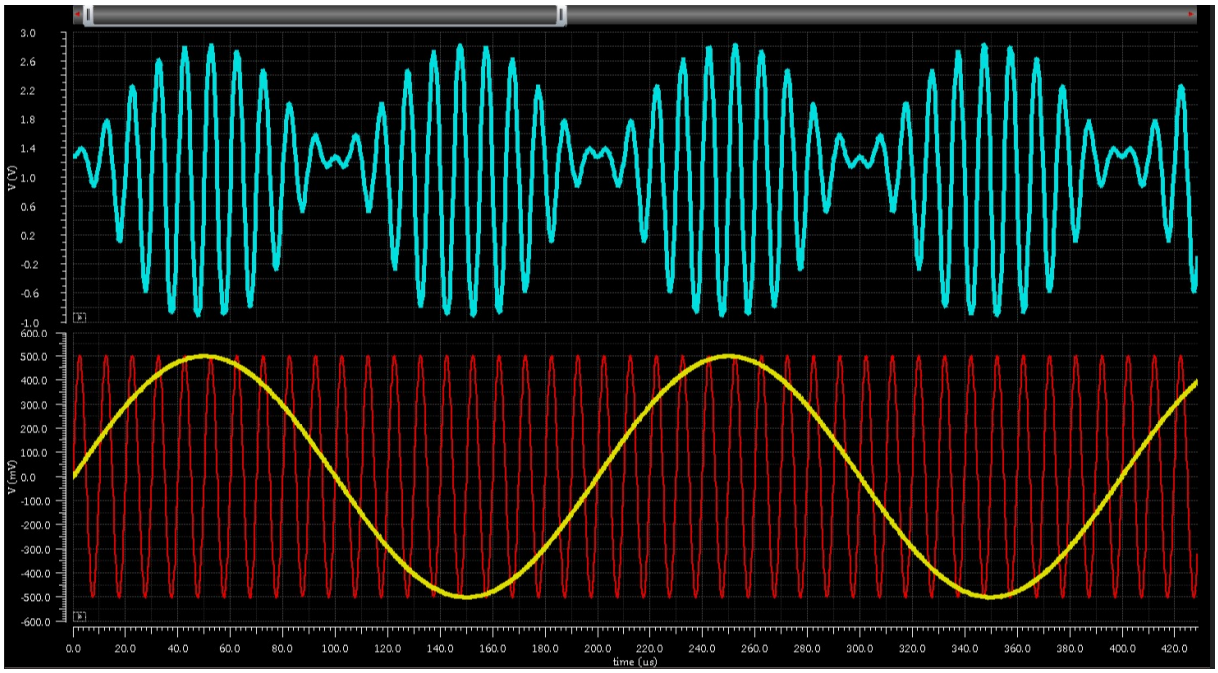

Figure 15: Output wave of multiplication block in cadence environment.

Transistor	Total Length(L)	Total Width(W)	Num. of base transistor
M2, M3	180n	140.8 u	32
M4, M5, M6, M7	180n	17.6 u	4
M8, M9	180n	17.6 u	4
M10	180n	22 u	5

Table 2: Size of transistor in Gilbert cell on figure 16.

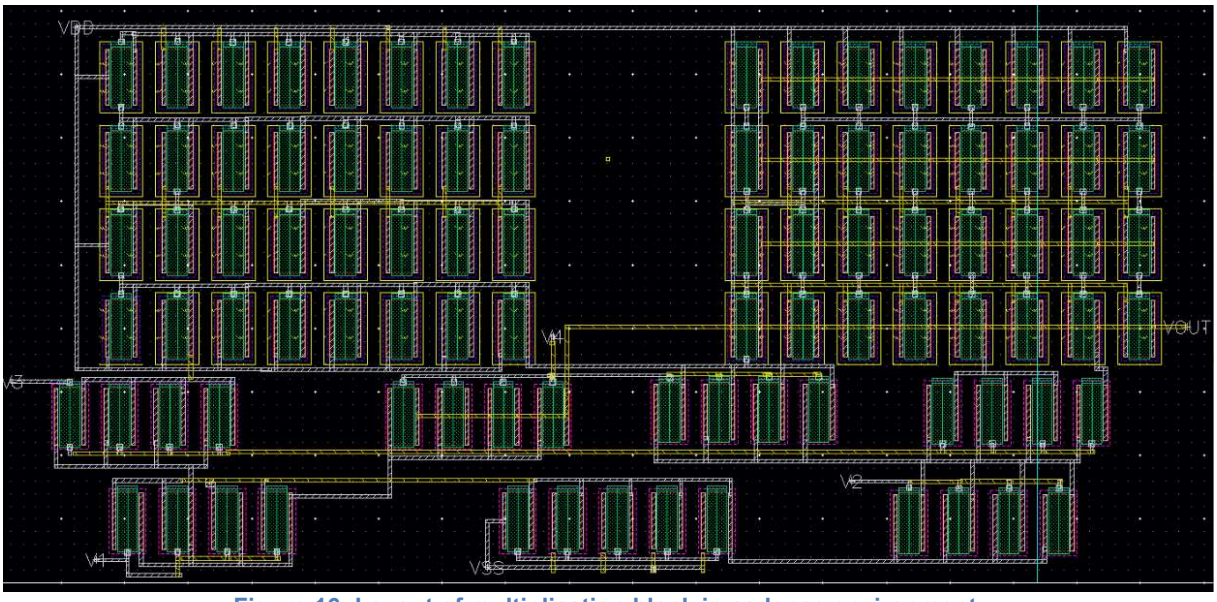

Figure 16: Layout of multiplication block in cadence environment.

For the power consumption in the gilbert cell we used the same evaluation we did for the activation function, it was made by simulating the total circuit with ADL simulation and extracting the total power consumption of the circuit via *power_extract function(figure19)* and then using a calculator to express the value average of *power_extract function*, the average power consumption is 4.14m W.

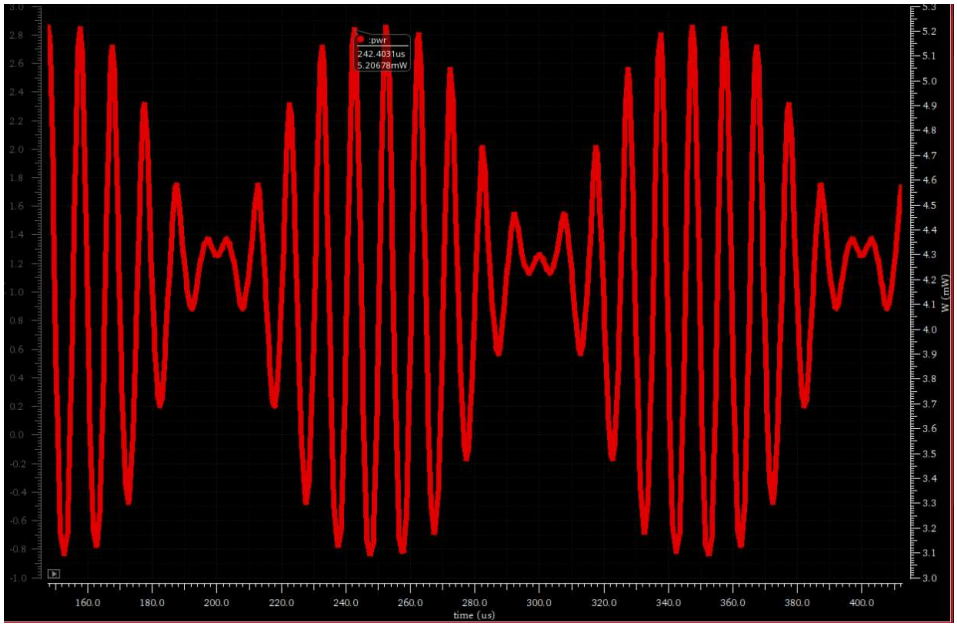

Figure 17 : Power consumption in the multiplication block.

To validate the implementation of those block working together as a neuron, we made a test bench composed by one neuron with two inputs working as an OR logic cell (figure 20), propose in article [1].

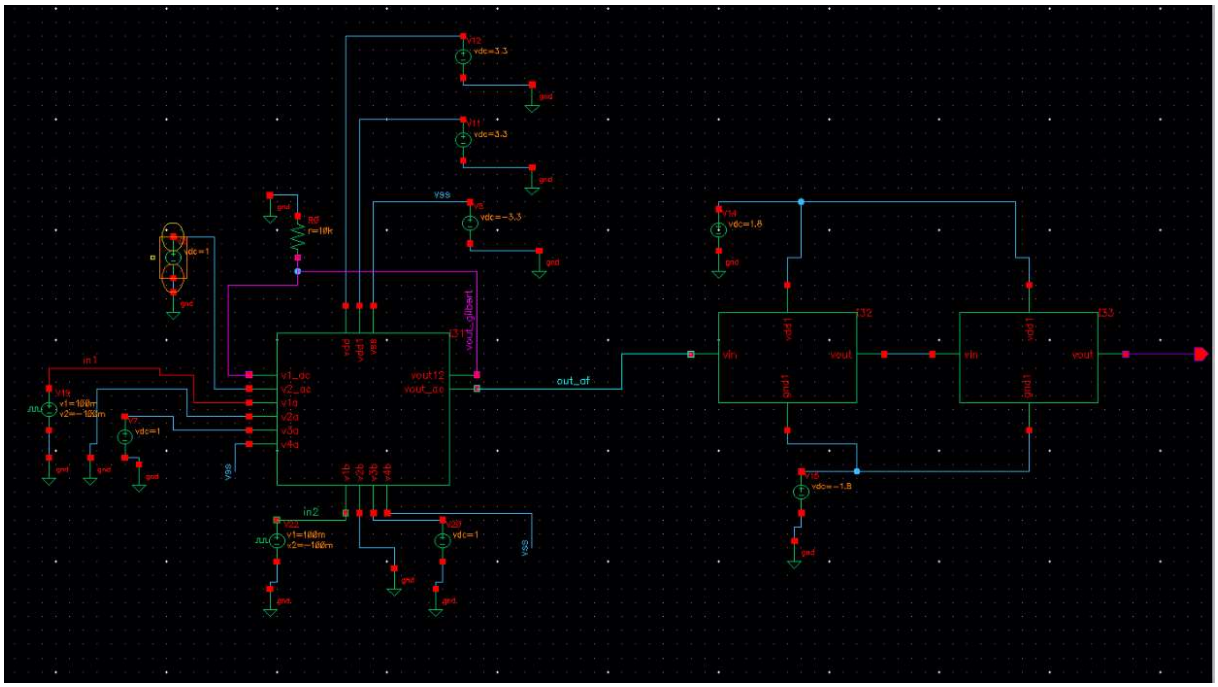

Figure 18: Diagram of the test bench's neuron in cadence environment.

The implementation was composed by two Gilbert cell, for the two inputs, and one Activation function, it was necessary to add a resistance to translate the output of the gilbert cell from current to voltage and then reapply it in the activation function, the input was two square waves with different frequency(2k Hz and 10k Hz) and ± 100mV, the bias input 1V and the weight in both of them are equal, since they have the same "importance" for the output. The power supply is 3.3V for VDD and -3.3 for VSS, it was used a buffer, already present in cadence library analoglib, in the output of the neuron to give us the required amplitude voltage (± 3.3V).

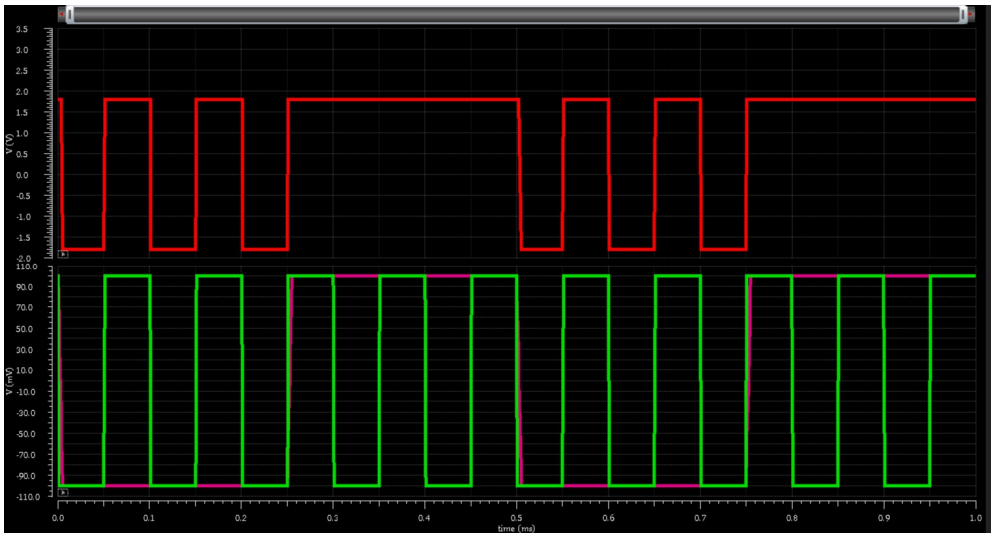

Figure 19: Output wave of the test bench's neuron in cadence environment

We can see at figure 21 that our output signal, red, works exactly as an OR function in spite of the offset voltage in the output we could regulate with the bias input, the bias point can be regulated by changing the input v2 in figure 13 since the total offset in both circuits(Gilbert cell and Activation function) is around 1.1 V we can apply a voltage equal to this value to compensate. We can compare the output results in article [1] and the figure 21 and we can confirm that our system successfully represented an OR function even with the redesign of the transistor and different specifications.

In figure 22 we can see the layout design for the whole neuron (Two Gilbert cell and one Activation function) and it has a total surface of 150x91 µm.

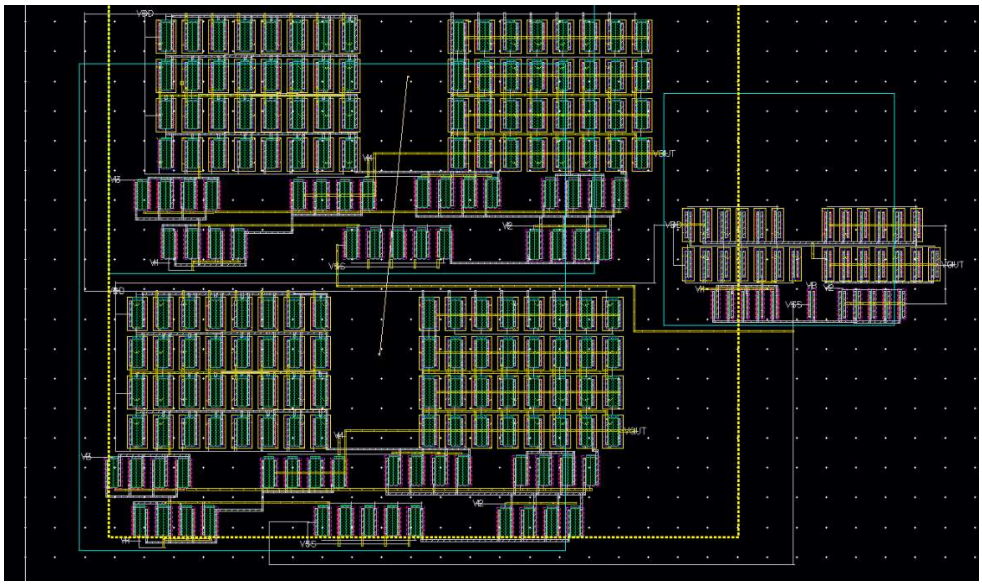

Figure 20: Layout the test bench's neuron in cadence environment.

VIII. Conclusion and future work

This work cited different topologies that can be implemented to make an analog artificial neuron due to time limitation and technology required we carefully chose a neuron base on Gilbert cell which provide us a simple and efficient way make analog artificial neuron. The topology was tested and designed to work in the condition specified in this paper. The schematic was created from the scratch and designing the transistors to electrical specification and technology established. The system gave us a satisfactory result. A test bench was proposed to implement a neuron with two inputs working as OR function such proposition required two Gilbert cells, for each input, and one Activation function, also the OR function was successively demonstrated. This paper successfully implemented a base analog artificial neuron that can be use for future works and improvements in the analog neuron or in neural network applications.

The major issue found in this implementation, that can be studied in future work, was the offset voltage in the output in both blocks that can be fixed resizing the transistor for the right polarization in the system. Another optimization that can be made is the reduction in the total area used in both blocks. In the future work it is planned to implement the complete neural network architecture with hidden layers and trained using back propagation algorithm.

Appendix :

Back Propagation Training Method

The goal with backpropagation [6] [7] method is to update each of the weights in the network so that they cause the actual output to be closer the target output than minimizing the error (equation 5) for each output neuron and the network as a whole.

$$Error_{total} = \sum \frac{1}{2}(target - output)^2 \tag{5}$$

$$\frac{\partial Error_{total}}{\partial w_5} = \frac{\frac{\partial Error_{total}}{\partial output_{o1}} * \partial output_{o1}}{\partial net_{o1}} * \partial net_{o1} \tag{6}$$

Since we want to minimize the error, we can look the effect of one single weight in the total error, for example in the figure 23, if we want to know the change w5 has over the error we use the partial derivative of the error with respect to w5 (equation 6).

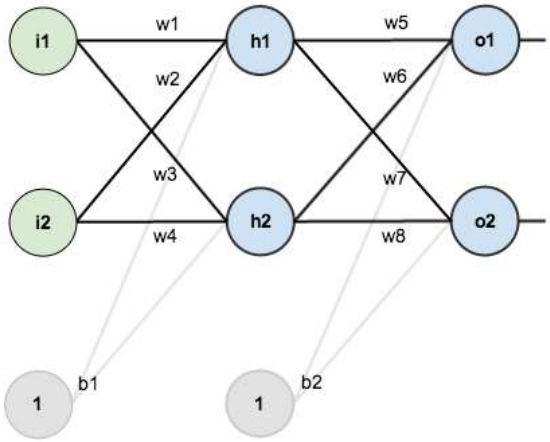

Figure 21: Neural network with two input and

We can rewrite the equation 6 as follow:

$$\frac{\partial Error_{total}}{\partial output_{o1}} = -(target_{o1} - output_{o1}) \quad (7)$$

$$\frac{\partial ouputt_{o1}}{\partial net_{o1}} = output_{o1}(1 - output_{o1}) \quad (8)$$

$$\frac{\partial net_{o1}}{\partial w_5} = output_{h1} \quad (9)$$

So, to decrease the error we need to update a new weight for w5 with the equation 10, α means the learning rate for our system and w_5^+ is the new value for w_5.

$$w_5^+ = w_5 - \frac{\alpha * \partial Error_{total}}{\partial w_5} \quad (10)$$

The same process should be done for all the weights connected with the output neurons ($w_6, w_7 \wedge w_8$). For the weights connected with hidden layer and the input neurons we are going to use a similar process as we did for the output layer, but slightly different to account for the fact that the output of each hidden layer neuron contributes to the output (and therefore error) of multiple output neurons. We know that $output_{h1}$ affects both$output_{o1}$ and $output_{o2}$ therefore the $\frac{\partial Error_{total}}{\partial output_{h1}}$ needs to take into consideration its effect on the both output neurons.

Matlab Simulation

To consolidate all the concepts explained previously, we are going to implement a simple task for a neural network in Matlab [9], a pattern recognition application. Our sample input data is a bunch of pixels in a 28x28 image of a handwritten digit (figure 24).

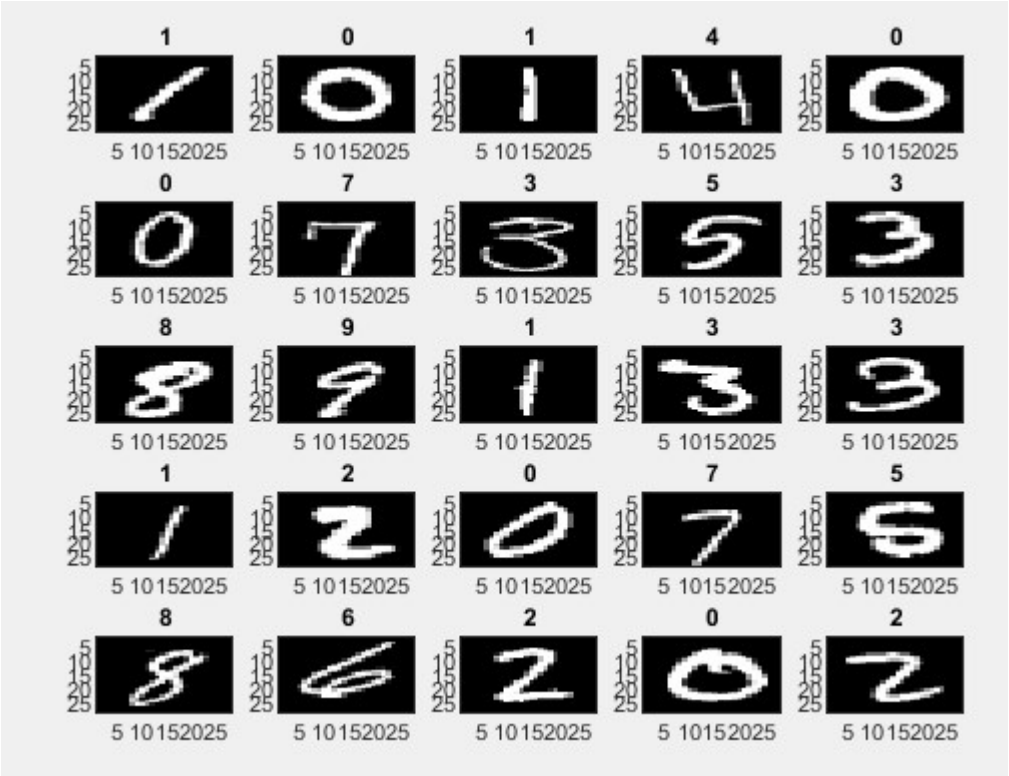

Figure 22: data input sample for the NN

To prepare the data we are going to use nprtool, pattern recognition app from Neural Network Toolbox, which is going to separate each digit in a different block. Our input is a numeric matrix, each column representing the samples and rows the features. This is the scanned images of handwritten digits and our target is a numeric matrix of 0 and 1 that maps to specific labels that images represent. This is also known as a dummy variable. Neural Network Toolbox also expects labels stored in columns, rather than in rows.

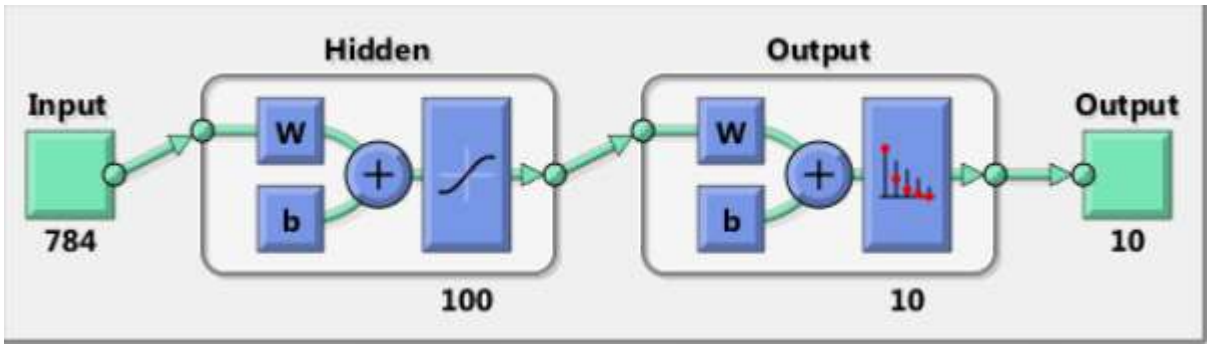

Figure 23: Representation of our Neural Network in Matlab Simulation

As showed in the figure 25 our input that will be 784, each pixel for our 28x28 image, and we have 10 different outputs since there are 10 possible digits. It was necessary 95 iterations to train the neural network and a total time of 27 seconds, as showed in figure 26.

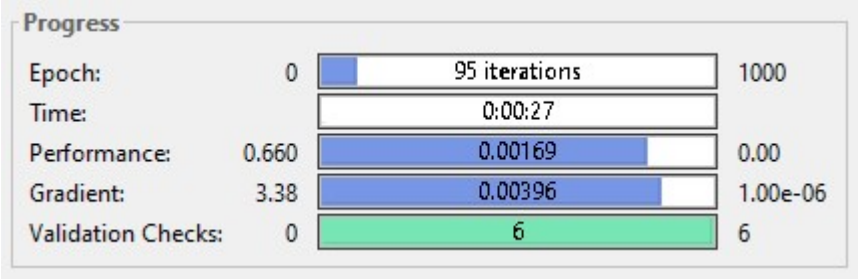

Figure 24: Progress of out neural network in Matlab simulation with 100 hidden layers

The Neural Network Toolbox gives us some extra tools to understand and improve out Neural Network; one of these tools is represented in figure 27 which show us the relation of epoch and best result, when the validation line has derivative equals to zero. An epoch is a measure of the number of times all of the training vectors are used once to update the weights. For batch training all of the training samples pass through the learning algorithm simultaneously in one epoch before weights are updated.

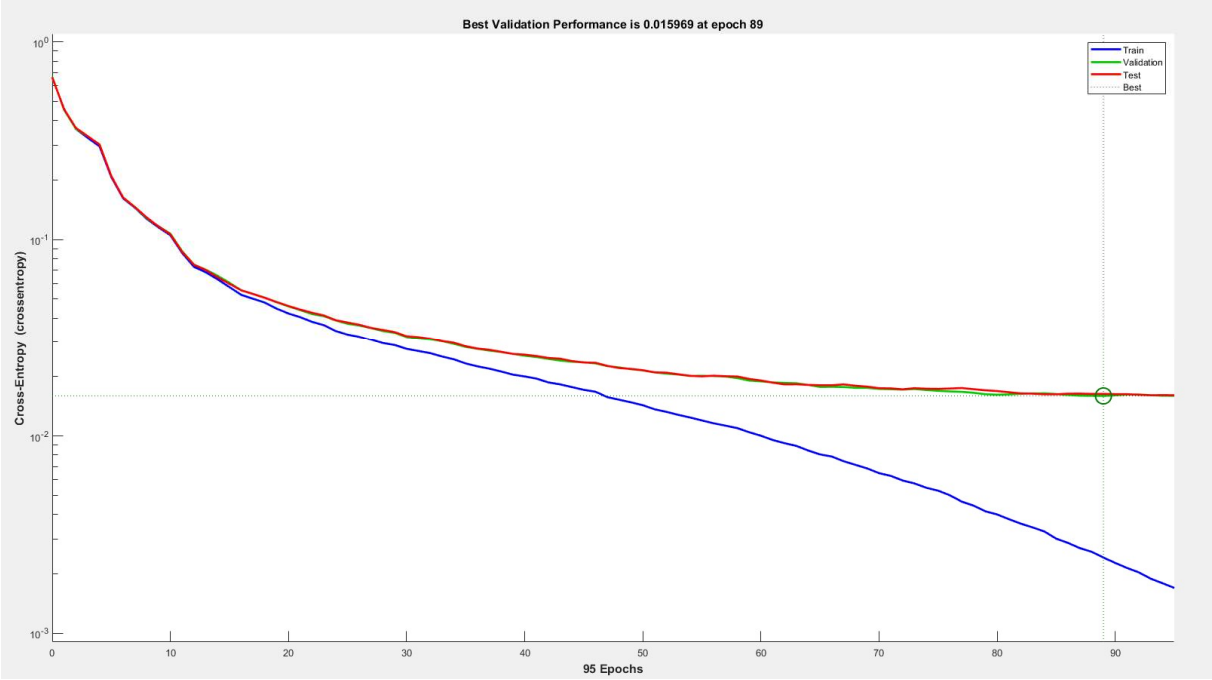

Figure 25: Best validation performance for 100 Hidden layers

Another major concept we can simulate in Matlab is the influential of the number of hidden layers used and how it affects the accuracy of our system, so it started with 10 hidden layers with an increasing step of 50 until it reaches 300 hidden layers. It shows in figure 28 that is possible to improve accuracy increasing the number of hidden layers, but after 100 layers accuracy does not improve that much but the total time and complexity continue to increase. We can see in figure 29 that with 300 layers the total training's time was almost the double compare with 100 layers; As you can see, you gain more accuracy if you increase the number of hidden neurons, but then the accuracy decreases at some point (your result may differ a bit due to random initialization of weights). As you increase the number of neurons, your model will be able to capture more features, but if you capture too many features, then you end up overfitting your model to the training data and it won't do well with unseen data.

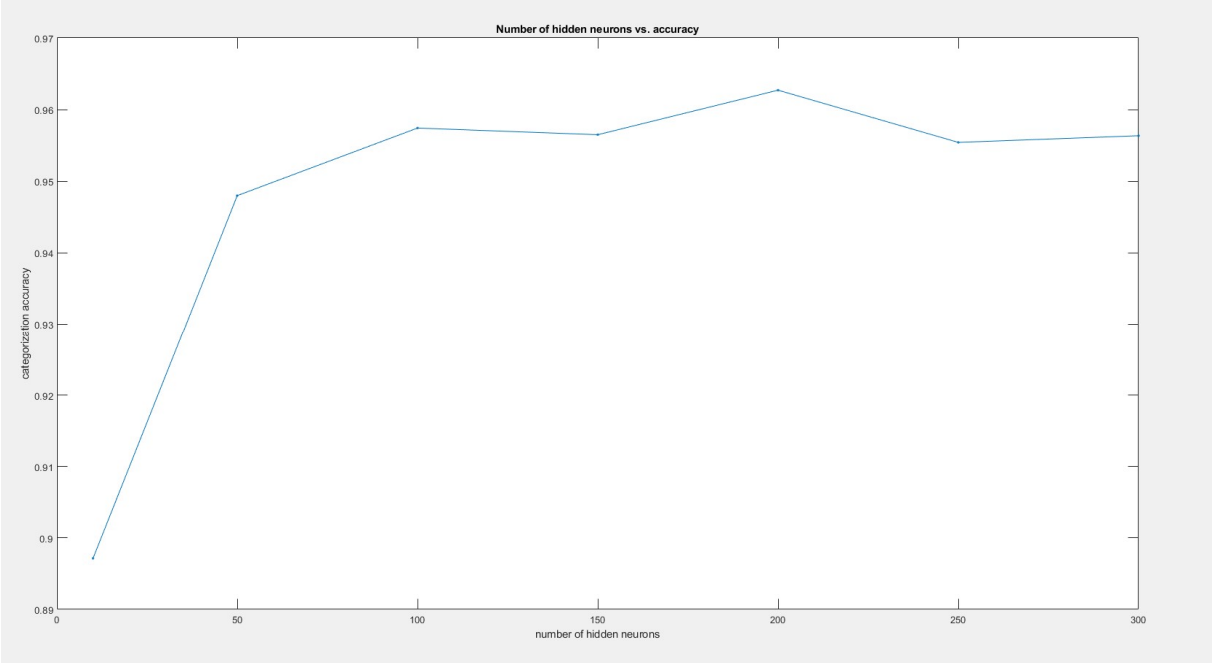

Figure 26: Number of hidden layers x accuracy

.

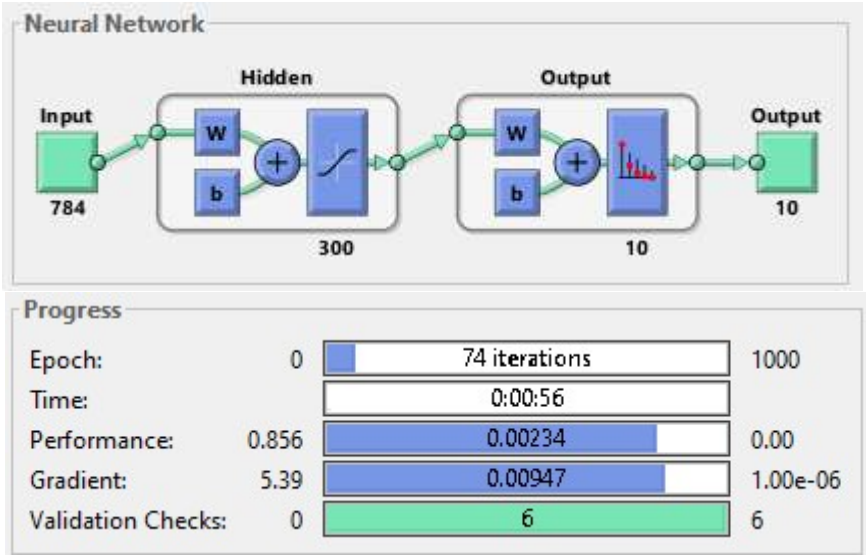

Figure 27: Progress of out neural network in Matlab simulation with 300 hidden layers

Neural Network training code, First NN

Source: Getting started with Neural Network Toolbox [9]

Modified for this work to change the number of hidden layers.

```
close all
clear all

tr = csvread('train.csv', 1, 0);                    % read train.csv
sub = csvread('test.csv', 1, 0);                    % read test.csv
%%

figure                                              % plot images
colormap(gray)                                      % set to grayscale
for i = 1:25                                        % preview first 25 samples
    subplot(5,5,i)                                  % plot them in 6 x 6 grid
    digit = reshape(tr(i, 2:end), [28,28])';        % row = 28 x 28 image
    imagesc(digit)                                  % show the image
    title(num2str(tr(i, 1)))                        % show the label
end

%%

n = size(tr, 1);                    % number of samples in the dataset
targets  = tr(:,1);                 % 1st column is |label|
targets(targets == 0) = 10          % use '10' to present '0'

targetsd = dummyvar(targets);       % convert label into a dummy variable
inputs = tr(:,2:end);               % the rest of columns are predictors

inputs = inputs';                   % transpose input
targets = targets';                 % transpose target
targetsd = targetsd';               % transpose dummy variable

rng(1);                             % for reproducibility
c = cvpartition(n,'Holdout',n/3);   % hold out 1/3 of the dataset

Xtrain = inputs(:, training(c));    % 2/3 of the input for training
Ytrain = targetsd(:, training(c));  % 2/3 of the target for training
Xtest = inputs(:, test(c));         % 1/3 of the input for testing
Ytest = targets(test(c));           % 1/3 of the target for testing
Ytestd = targetsd(:, test(c));      % 1/3 of the dummy variable for testing


%%

%%
```

```
Ypred = myNNfun(Xtest);              % predicts probability for each label
Ypred(:, 1:5)                        % display the first 5 columns
[~, Ypred] = max(Ypred);             % find the indices of max probabilities
sum(Ytest == Ypred) / length(Ytest) % compare the predicted vs. actual

%%


sweep = [10,50:50:300];                 % parameter values to test
scores = zeros(length(sweep), 1);       % pre-allocation
models = cell(length(sweep), 1);        % pre-allocation
x = Xtrain;                             % inputs
t = Ytrain;                             % targets
trainFcn = 'trainscg';                  % scaled conjugate gradient
for i = 1:length(sweep)
    hiddenLayerSize = sweep(i);         % number of hidden layer neurons
    net = patternnet(hiddenLayerSize);  % pattern recognition network
    net.divideParam.trainRatio = 70/100;% 70% of data for training
    net.divideParam.valRatio = 15/100;  % 15% of data for validation
    net.divideParam.testRatio = 15/100; % 15% of data for testing
    net = train(net, x, t);             % train the network
    models{i} = net;                    % store the trained network
    p = net(Xtest);                     % predictions
    [~, p] = max(p);                    % predicted labels
    scores(i) = sum(Ytest == p) /...    % categorization accuracy
        length(Ytest);
end




%%
figure
plot(sweep, scores, '.-')
xlabel('number of hidden neurons')
ylabel('categorization accuracy')
title('Number of hidden neurons vs. accuracy')
%%
net = models{end};                      % restore the last model
W1 = zeros(sweep(end), 28*28);          % pre-allocation
W1(:, x1_step1_keep) = net.IW{1};       % reconstruct the full matrix
figure                                  % plot images
colormap(gray)                          % set to grayscale
for i = 1:25                            % preview first 25 samples
    subplot(5,5,i)                      % plot them in 6 x 6 grid
    digit = reshape(W1(i,:), [28,28])'; % row = 28 x 28 image
    imagesc(digit)                      % show the image
end




%%
n = size(sub, 1);                                % num of samples
sub = sub';                                      % transpose
[~, highest] = max(scores);                      % highest scoring model
```

```
net = models{highest};                                  % restore the model
Ypred = net(sub);                                       % label probabilities
[~, Label] = max(Ypred);                                % predicted labels
Label = Label';                                         % transpose Label
Label(Label == 10) = 0;                                 % change '10' to '0'
ImageId = 1:n; ImageId = ImageId';                      % image ids
writetable(table(ImageId, Label), 'submission.csv');% write to csv
```

REFERENCES

[1] Madhumitha G.B. and Vijayalatha Devadiga (2015) "*Analog VLSI Implementation of Artificial Neural Network*" International Journal of Innovative Research in Computer and Communication Engineering Visvesvaraya Technological University, Karnataka, India

[2] Neeraj Chasta , Sarita Chouhan and Yogesh Kumar (2012) "*Analog VLSI Implementation of Neural Network Architecture for Signal Processing* " International Journal of VLSI design & Communication Systems (VLSICS) Mewar University, Chittorgarh, Rajasthan, India.

[3] Ujwala Q. Kshirsagar (Belorkar) & Ashish E. Bhande (2014) .*VLSI Implementation of Back Propagated Neural Network for Signal Processing.* International Journal of Science and Research (IJSR). Bhusawal, Maharashtra, India

[4] I.A. Basheer, M. Hajmeer (2000) "*Artificial neural networks: fundamentals, computing, design, and application*". Journal of Microbiological Methods. Kansas State University, Manhattan, USA

[5] Laurene Fausett, First edition. (1994). *Fundamentals of Neural Network Architecture, Algorithms and Application*. United of State, New Jersey: Englewood Cliffs

[6] Carver Mead. (1989). *Analog VLSI and Neural Systems*. United of State, Boston: Addison-Wesley

[7] Simon Haykin, Second edition. (1994). *Neural Network a Comprehensive Foundation. United Kingdom, London: Pearson*

[8] Gunther Palm, Ad Aertsen. (1986). *Brain Theory Processing of The First Meeting on Brain Theory.* Germany, Heidelberg: Springer Verlag.

[9] GETTING STARTED WITH NEURAL NETWORK TOOLBOX , Mathwork Matlab documentation, < www.mathworks.com/help/nnet/getting-started-with-neural-network-toolbox.html> last access: 07 Jun 2018

[10] NEURAL NETWORKS EVERYWHERE, Larry Hardesty MIT New Office February 13, 2018 - <www.news.mit.edu/2018/chip-neural-networks-battery-powered-devices-0214> Last access: 03 Jun 2018.

[11] Seyoung Kim, Tayfun Gokmen, Hyung-Min Lee and Wilfried E. Haensch . *Analog CMOS-based Resistive Processing Unit for Deep Neural Network Training.*School of Electrical Engineering, Korea University, Seoul, South Korea

[12] Dean K. McNeill, Christian R. Schneider, Howard C. Card.(1993). *Analog CMOS Neural Networks Based on Gilbert Multipliers with In-Circuit Learning.* Department of Electrical and Computer Engineering University of Manitoba. Winnipeg, Manitoba, Canada

[13] Michihito Ueda Yu Nishitani, Yukihiro Kaneko,Atsushi Omote. Back-Propagation Operation for Analog Neural Network Hardware with Synapse Components Having Hysteresis Characteristics.<www.journals.plos.org/plosone/article?id=10.1371/journal.pone.0112659#authcontrib>Last acess :01 jun 2018

[14] Gökçe Yayla, La Jolla; Ashok V. Krishnamoorthy, San Diego; Sadik C. Esener, Solana Beach.(1994). *Artificial Neuron with Switched-Capactor Synapses Using Analog Storage of Synaptic Weights.* University of California, California United of State.

[15] Behzad Razavi, second edition .(1998). *Design of Analog CMOS Integrated Circuits. New York, United of State :McGraw-Hill Education.*

[16] Shasanka Sekhar Rout and Kabiraj Sethi. (2016). *Design of High Gain and Low Noise CMOS Gilbert Cell Mixer for Receiver Front End Design.* 2016 International Conference on Information Technology. Burla, India