2013

A Mixed-Signal Feed-Forward Neural Network Architecture Using A High-Resolution Multiplying D/A Conversion Method

Farinoush Saffar
University of Windsor

A Mixed-Signal Feed-Forward Neural Network Architecture Using A High-Resolution Multiplying D/A Conversion Method

by

Farinoush Saffar

A Thesis
Submitted to the Faculty of Graduate Studies through the
Department of Electrical and Computer Engineering in Partial Fulfillment
of the Requirements for the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada
2012

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-84859-3

Our file Notre référence
ISBN: 978-0-494-84859-3

A Mixed-Signal Feed-Forward Neural Network Architecture Using A High-Resolution Multiplying D/A Conversion Method

by

Farinoush Saffar

APPROVED BY:

M. Ahmadi, Advisor
Electrical and Computer Engineering, University of Windsor

M. Mirhassani, Advisor
Electrical and Computer Engineering, University of Windsor

M. A. S. Khalid
Electrical and Computer Engineering, University of Windsor

J. Wu
Electrical and Computer Engineering, University of Windsor

A. Jaekel
Computer Science, University of Windsor

December 4, 2012

Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon any one's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from copyright owner(s) to include such material(s) in my thesis and have included copied of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and Graduate Studies office, and that this thesis has not been submitted for a higher degree ti any other University or Institution.

Abstract

Artificial Neural Networks (ANNs) are parallel processors capable of learning from a set of sample data using a specific learning rule. Such systems are commonly used in applications where human brain may surpass conventional computers such as image processing, speech/character recognition, intelligent control and robotics to name a few.

In this thesis, a mixed-signal neural network architecture is proposed employs a high resolution Multiplying Digital to Analog Converter (MDAC) designed using Delta Sigma Modulation (DSM). To reduce chip are, multiplexing is used in addition to analog implementation of arithmetic operations.

This work employs a new method for filtering the high bit-rate signals using neurons nonlinear transfer function already existing in the network. Therefore, a configuration of a few MOS transistors are replacing the large resistors required to implement the low-pass filter in the network. This configuration noticeably decreases the chip area and also makes multiplexing feasible for hardware implementation.

To my parents, for their unending love and support.

Acknowledgments

There are several people I would like to thank for their generous contributions to this project. I would first like to express my sincere gratitude and appreciation to my supervisors, Dr. Mirhassani and Dr. Ahmadi, for their guidance and constant support throughout the course of this thesis.

I also thank my committee members Dr. Wu, Dr. Khalid, and Dr. Jaekel for their participation in my committee, reviewing my thesis, and their valuable comments.

Finally, my deepest gratitude goes to my family for their unconditional love, support and encouragement.

Contents

List of Figures

List of Tables

List of Abbreviations

ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
MDAC	Multiplying Digital to Analog Converter
ANN	Artificial Neural Network
LPF	Low Pass Filter
DSM	Delta Sigma Modulation
PWM	Pulse Width Modulation
PAM	Pulse Amplitude Modulation
PFM	Pulse Frequency Modulation
PPM	Pulse Phase Modulation
MOS	Metal Oxide Semiconductor
CMOS	Complimentary Metal Oxide Semiconductor
FPGA	Field Programmable Gate Array
RAM	Random Access Memory
LSB	Least Significant Bit
OTA	Operational Transconductance Amplifier

Chapter 1

Introduction

1.1 Inspiration: Biological Neural Networks

Biological neural networks are the main inspiration in artificial neural network research. Researchers in this area have always been interested in the way human brain receives information from outside environment, processes those information and at the end outputs their corresponding responses. It has been estimated that the human brain includes 10^{10} neurons and 10^3 - 10^5 synapses. Dendrites in a neural cell act as the receiver and receive the input signals and transfer them to the neurons. The neurons, sum and threshold the upcoming signals and transfer the result to the axon through a long fiber which acts as the output terminal. The point in which the axon of one neural cell meets dendrites of another cell is called synapse. Signals are communicated across these synapses by chemical ion transport and the strength of this communication depends of the density of those ions. Any neural network of any size and configuration is defined by strength of the synapses and the arrangement of

neural cells which depends on its application.

1.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are processors designed based on the operation of human brain and nervous system with massive number of nodes and interconnections but simple computational building blocks.

There are three main computational block in every neural networks: multiplier, adder and nonlinear transfer function. Multiplier in an artificial neural network acts as synapse in biological networks and defines the strength of the signals it receives. Adder and non-linear transfer function play the role of neuron in human brain to sum and threshold the upcoming data.

1.3 A Brief History

The first neural network was introduced by McCulloch and Pitts in 1943 [1] based on neurology concepts. Their proposed network was only capable of solving basic logic problems such as OR and AND functions. In 1958 Rosenblatt succeeded to present Perceptron, a neural network with three layers. Followed by that, in 1960 Widrow and Hoff [2] introduced ADALINE (ADAptive LInear Element) neural network using simple analog blocks and Least Mean Square (LMS) learning rule.

In 1969 Minsky and Papert published a book about limitations on neural network research [3] which caused reductions in funds and interests in this field. However, several researchers kept working in the area which leaded to outstanding results during

70s and 80s. Werbos in 1974 [4] developed the backpropagation algorithm for neural networks which by far was the most common learning rule. Also, Grossberg founded founded a school of thought in 1988 in which resonating algorithms are developed.

Nowadays, neural network research attracts a lot of interest and neural network chips are capable of solving complex problems.

1.4 Hardware-Friendly Learning Algorithms

Backpropagation has been one of the most common learning algorithms for neural network training for a long time. However, despite the fast convergence speed, it may not be a suitable choice for training neural networks implemented in hardware.

The first limitation is the need to design bi-directional paths for the data to flow in forward and backward directions. The need to design the derivative of the nonlinear transfer function is the second reason for not choosing backpropagation for neural network hardwares. Finally, in a neural network trained using this learning algorithm, the parameters can not be adjusted depending on the circuit offsets and non-idealities.

1.4.1 Perturbation Learning Rules

The basic idea in perturbation-based learning rules is to add a slight perturbation to the network and calculate the gradient of the error based on the amount of that perturbation, the actual and the desired output values. Node perturbation and weight perturbation are two common perturbation-based algorithms.

Node Perturbation

Madaline Rule III (MRIII) [5] is the most popular node perturbation training scheme. In this algorithm, weights are initially set to a small random value and the perturbation signal is applied to each node value in the network sequentially. The error is calculated after a forward path and consequently, the weight values are updated. The direction of the weight vectors is opposite of the direction of the gradient of the calculated error in the forward path.

Unlike the backpropagation, this method does not require implementation of backward paths as well as implementation of the derivation of nonlinear activation function. In addition to that, the network is capable of learning the non-idealities existing in the network for online training configurations. However, designing extra circuitries is required for training the network using this algorithm. These additional blocks include multipliers and addressing modules to choose between different nodes in the network. Moreover, the sequential nature of this algorithm makes the training process relatively slow.

Weight Perturbation

This algorithm [6] is similar to MRIII. Weight perturbation is simpler than MRIII in terms of required circuitry for addressing. However, MRIII requires less complex computational circuitry since weight perturbation rule needs additional blocks to create random perturbations which increases overhead area.

1.4.2 Learning Rules With Locally Accessed Information

There is one category of learning rules in which information in the network is processed locally. Unlike the backpropagation algorithm in which the error is accessed globally, this category of algorithms are easier to be implemented in hardware. As an example, in anti-Hebbian learning rule [7], the synaptic weights are updated based on the input and the output of each neuron. Additionally, Brandt in [8] uses only the rate by which the synaptic weights change to estimate the error vector. Hence, the information used as error data are accessed locally within each neuron. However, the calculation of the rate of changes in synaptic weights is complex and requires high accuracy.

1.5 Training Configurations

Training configuration is defined as the way a learning algorithm is implemented in order to train the network based on the existing sample data. There are three main learning configurations for an artificial neural network: Off-line Training, Chip-In-the-Loop and Online training.

Off-line Training

In this training configuration, a host computer runs the learning algorithm and weights are calculated and updated by the host computer. Final values of weights are later downloaded into the chip [9]. An advantage to this configuration is less amount of required circuitry and therefore, smaller chip area. The most important advantage in this training configuration is accurate weight calculations. On the other hand, weight truncation might be required since the non-idealities of the chip are not

taken into consideration for weight calculations and this can lead to poor network performance. This method is not suitable for neural networks used in tasks such as automated control. Such systems, known as adaptive neural networks, have to be able to change their weights depending on the changes in the environment conditions.

Chip-In-The-Loop Training

In this configuration, the host computer calculates and updates the synaptic weights based on the learning algorithm, actual and the targeted outputs. The difference between this configuration and the Off-line configuration is that, the computer is in a loop with the network. Therefore, it receives chip outputs, compares them with the targeted output and readjusts the synaptic weights.

The most important advantage of this method is the ability of the network to learn the offsets and mismatches existing in the network. Additionally, the reacquired chip area is relatively smaller than a network with online training configuration, which will be explained later. The only drawback in this method is the slow training speed due to the periodic weight update [10–12].

Online Training

In this configuration, all of the weight calculations and updates are performed on the neural network chip. Therefore, there is a need for designing an arithmetic block for training calculations. Among implemented networks reported in literature, [13] implemented in analog is used for smart sensing applications, [14] implemented in digital for optical character recognition, [15] implemented in mixed-signal for pixel pattern recognition and [16] and [17] implemented using FPGA for smart olfactory

system and non-linear control, respectively. This method is the most appropriate configuration for adaptive neural systems and the training is relatively fast. However, the required chip area increases due to the training circuities. Moreover, design of the training circuitry adds to the network complexity. This in turn makes the design a challenging task for circuit designers since some learning algorithms, such as backpropagation, are more difficult to be implemented in hardware. Moreover, non-convergence and inaccurate network operations have been reported due to low accuracy of this method for implementing learning algorithms such as backpropagation [18,19]. However, perturbation methods are among the most compatible learning rules for hardware implementation training circuitries.

1.6 Hardware Implementation of Neural Networks

Nowadays, it is more common to implement neural networks in software on a host computer. In this way, the software is basically a neuro-simulator which running on a conventional computer. This method was first introduced by Rochester in 1956 with a Hebbian learning neural network [20]. This type of implementation makes neural network design more flexible and gives the designers the advantage of changing network parameters, such as architecture, learning rule and number of layers, depending on their application. In addition to that, software implementation of neural networks can be done in various user-friendly environments providing the designer with a bright insight to the network architecture and performance.

On the other hand, the parallel nature of computations in neural systems can be best realized using hardware implementation. Moreover, in some applications it is not possible to install a PC/workstation to run the neural network software such as toys and autonomous robots for industrial and exploration applications. Also, running

neural calculations on conventional serial computers is slow and costs a lot in terms of power.

Nowadays, there is a tremendous growth in portable devices with limited battery life which will increase the need for custom low power solutions. Running a neural network software on a portable device like a smart phone consumes a lot of power due to the parallel nature of calculations. Consequently, one option to overcome this problem might be to use a host computer to perform the calculations and transmit the data back to the portable device. This might however, cause data security issues. Moreover, some special applications require the neural network on a small piece of hardware to perform the neural calculations independently, such as neurochips used in silicon retinas [21, 22] and silicon cerebral cortex [23]. In addition to all, hardware neural networks are a more suitable choice for applications where environmental changes require the network to be trained over and over again for a long period of time. In this case, it is worth mentioning that the caculations for a forward path using a regular Pentium takes about $1ms$ which is considered very slow for such applications.

In the next section 4 categoriesof hardware neural network implementation methods are explained briefly including: Analog, Digital, Mixed-Signal and FPGA.

1.6.1 Analog Neural Networks

In Analog Neural Networks (Analog NN) [10, 11, 24–28] there are efficient methods for implementing neurons using simple non-linear analog components. Moreover, addition operation can be performed by simple nodal summation of currents as long as it can drive the next stage of circuits.

On the other hand, accuracy of analog circuits has always been a limiting factor for realization of large size Analog NNs. Although neural networks are capable of recovering from network deficiencies, for off-line training such imperfections cause instability and errors at the output. Additionally, the network needs to store a large number of synaptic weights typically stored on capacitors in analog implementation. The weight values will decay due to capacitor leakage currents and therefore, requires refreshing circuitries. This causes limitation in size and complexity of such Analog NNs. In addition, network performance is affected by the reduced noise margin, variations in power supply and low resolution (maximum 6 to 7 bits) of analog circuitries.

1.6.2 Digital Neural Networks

Digital Neural Networks [29–31] on the other hand, can have extended resolution and higher accuracy. However, generating smooth neural activation function in digital requires complex and large look up tables [32–34]. Another important design issue is the high number of interconnections which in a typical digital neural network, is much higher than an analog neural network. It is also worth mentioning that, in digital implemented neural networks there may be the need to convert the analog signals at the input to digital for processing, and convert them back to analog at the output stage. This can clearly increase the overhead area and circuit complexity.

1.6.3 Mixed-Signal Neural Networks

There has recently been an interest in mixed-signal implementation of neural networks [13, 35–37]. In such systems, advantages from both analog and digital domains are used to overcome the design challenges. Mixed-signal implementation benefits from smaller area, lower power consumption, higher speed and easy activation func-

tion realization which all come form analog domain. On the other hand, there are some advantages in digital domain which can be included in mixed-signal, such as RAM weight storage. Consequently, there is no need to have capacitors and refreshing circuitrie to prevent charge leakage. It is also worth mentioning that due to the susceptibility of analog circuitries to process variations, mixed-signal architectures stand between analog and digital in terms of accuracy. However, this type of implementation is proved to be more flexible for various network sizes.

Pulse Stream

Many signals in biological neural systems are transmitted in the form of electrical spikes [38, 39]. Murray and Smith introduced the first pulse stream implementation of neural networks back in 1987 [39, 40]. In this type of mixed-signal implementation, the network data is coded in digital pulse streams, but processed using analog components. This has raised interest in the area of neural computation because of inaccuracy of analog signals and the fact that digital signals are robust against noise and process variations. In addition to that, small area and high speed of analog computations from one hand, and large area and high power consumption in digital computations on another hand increased the interest in pulse stream implementation methods. There are different methods for encoding information such as Pulse Amplitude Modulation (PAM), Pulse Width Modulation (PWM), Pulse Frequency Modulation (PFM) and Pulse Phase Modulation (PPM).

Using pulse streams in neural network implementation have many advantages such as:

- Low susceptibility to noise and circuit variations: Usually in a hardware implemented neural network the main cause of error is the multiplication operation

and transmission of data.

- Low power consumption
- Simpler multiplexing: In some network architectures it is required to pass different sets of data along the same path. For these cases, pulse stream signals are a better option due to their digital nature rather than analog signals.
- Simpler connection to analog and digital blocks within the chip

Pulse streams in general can have two types of applications in the area of neural networks. The first type of application is to use them in computational blocks to increase the performance of the calculations [41–43]. The second type of application is using pulse streams to implement the neurons and replace pulse modulators with stochastic logic [44–49]. In this thesis, we will focus on the first type of application of pulse streams in neural networks and take advantage of their features in our network architecture.

1.6.4 FPGA

Reconfigurability of FPGAs is its most important advantage as an implementation method for artificial neural networks. In addition to that, these platforms are easily available in market with a wide range of variety and a cost much lower than any fabrication process. However, large area and high power consumption are one of the major drawbacks in this type of implementation. It should be noted that choosing an appropriate neural network model to use optimal hardware resources can always be a challenge.

Among the networks implemented using FPGA, Tisan [16] has proposed an olfaction system using seven gas sensors. Seul in [17] has implemented a cost effective

nonlinear control system by implementing the neural network on a DSP board, and the rest of the subsystems such as control algorithm, counter and PWM signal generator on a FPGA board.

1.7 Real-World Applications

Despite the fact that artificial neural networks implemented in software are more flexible and user-friendly, there are some important neural network hardwares currently working in real-world applications. Table 1.1 shows examples of neural network hardware in real-world applications.

1.8 Objective

In a neural network hardware, multipliers usually consume the largest portion of space on the chip. Multipliers implemented in digital specially require a large amount of circuitry because of the parallel nature of computational operations. Consequently, many designers try to reduce the amount of circuitry in their neural network by reducing the resolution of synaptic weights to 4-6 bits and therefore, reducing the resolution of calculations. For example, binary weighted current mirrors can be used to convert digital synaptic values to analog. This way multiplication is combined with a low-resolution digital to analog conversion scheme in a Multiplying Digital to Analog Converter (MDAC) block [35–37]. It should be noted that, this method limits the resolution of synaptic calculations since the transistors can enter the saturation region quite fast in low power CMOS technologies.

Our objective is to design and implement a feed-forward neural network using

Implementation	Application	Reference
Analog	Image processor	[50]
	Visual collision detector	[51]
	Finger print feature extraction	[52]
	Feedback controller	[53]
Digital	Real time image processing	[54]
	Autonomous robotics	[55]
	Real-time controller	[56]
	Assignment problem solver	[57]
Mixed-Signal	Video Processor	[58]
	Vision system for intelligent vehicles	[59]
	Robotics	[60]
	Robotics	[61]
FPGA	Real-time image processor for mobile robot vision	[62]
	Image segmentation	[63]
	Robotics	[64]
	Real-time hand tracking system	[65]

Table 1.1: Hardware neural networks in real-world applications

high resolution mixed-signal components. Encoding synaptic information in pulse streams and performing neural calculations in pulse domain ensures higher accuracy and high resolution in the network. On the other hand, several techniques, such as multiplexing, a compact filtering and area-efficient analog computational blocks using minimum size MOS transistors has been employed to compensate for the additional area usage due to the high resolution synaptic weights stored in registers. Using these methods reduces the number of interconnections and the amount of circuitries

required for a high resolution neural network. At the same time, we can take advantage of low power consumption of analog components and pulse stream calculations.

The network architecture proposed in this work is trained off-line as the first step to a high resolution neural network. In this work synaptic weights are calculated and updated off the chip. However, high accuracy training is feasible using high resolution synaptic weights and accuracy inherent in pulse stream calculations.

1.9 Thesis Organization

Chapter 2 provides explanations on the proposed architecture and includes general descriptions on the role that each building block plays and the way they are connected in this architecture. Mathematical descriptions and all the parameter calculations for each building block are included in this chapter.

In Chapter 3, a more detailed description of every building block used in the proposed architecture is presented. Each component is discussed in circuit level. Different simulations results, such as DC, transient and Monte Carlo, are included in this chapter as well.

Chapter 4 shows how the proposed neural network architecture is used in an XOR problem. The final network configuration and the corresponding simulation results are included in this chapter as well as general network features. Comparison between the proposed network and other reported networks which describes its distinguishing characteristics is carried out in this chapter as well.

Finally, Chapter 5 includes conclusions, summary and suggested future work.

Chapter 2

Proposed Architecture

2.1 Introduction

In this chapter, an optimized architecture for a high resolution feed-forward neural network is proposed. In order to obtain the desired level of accuracy, the proposed architecture exploits high resolution Multiplying Digital to Analog Converter (MDAC) modules. to achieve this goal, synaptic calculations are performed in pulse stream domain. After a brief overview on the proposed architecture, the system level configuration for one layer of the network is presented. In continuation, each building block in the proposed architecture is presented and discussed in more details.

In general, an Artificial Neural Network (ANN) consists of three main building blocks: multiplier, adder and nonlinear transfer function. In this architecture, multiplication operation between the synaptic weights and the layer inputs is performed in analog domain. However, the weights are stored in registers. Synaptic weights

are converted to high bit rate pulse streams using a delta sigma modulator and then multiplied by layer's input through an analog multiplier. The delta sigma modulator is implemented entirely in digital since it processes digital synaptic values at their input. The adders and neuron nonlinear transfer functions are fully analog.

Multiplexing [66, 67] compensates for the increased chip area due to the increased network resolution. It reduces the number of interconnections as well as required circuitry for synaptic calculations and information processing. Although time multiplexing increases the overall delay of the network, arithmetic operations are performed in a relatively high speed due to the full custom nature of the circuits. System-level configuration of the network is presented in the next section.

2.2 Multiplexing

Figure 2.1 shows how multiplexing can reduce number of modules and interconnections. Diagrams (a) and (b) show general configurations of a $3-3-2$ neural network without and with multiplexing, respectively. Both networks are fully connected and have 3 inputs, 2 outputs and 15 different synaptic weights in their configuration. Basically without multiplexing, 9 and 6 multiplier modules are required for a $3-3-2$ network in its first and second layer, respectively. However, this numbers are reduced to 3 and 3 using the multiplexing method employed in the proposed architecture. As it can be seen, the number of interconnections in the network has also been noticably reduced since the same path of data has been used for different synaptic weights.

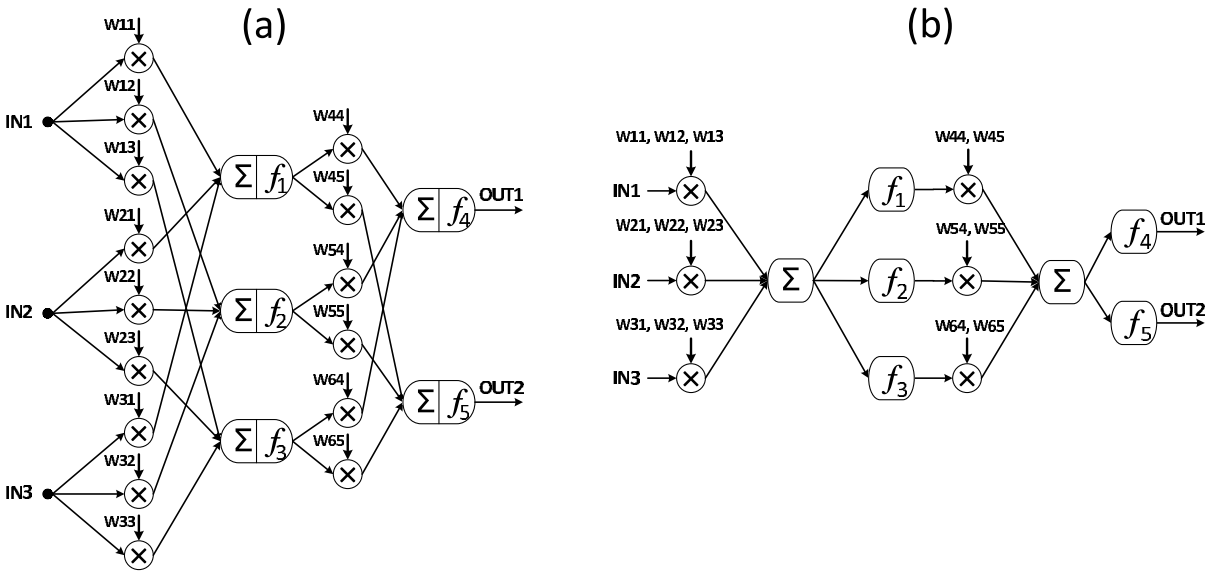

Figure 2.1: Comparison between number of interconnections and multiplier blocks in a sample $3-3-2$ neural network configuration: (a) without multiplexing and (b) with multiplexing

2.3 System-Level Configuration

Figure 2.2 shows the block diagram representation of one layer of the proposed network. The network can be expanded by connecting desired number of this layers in series. High resolution Multiplying Digital to Analog Converter (MDAC) blocks play the main role in this network architecture. Other network components include registers, multiplexers, current switches, tri-state buffers and signal conditioning blocks which operate as the final stage of each layer in the network.

The number of MDAC blocks in each layer is equal to the number of inputs to that layer. For example for a network of size $3-2-1$, there are 3 and 2 MDAC blocks at the first and second layers, respectively. As shown in Figure 2.2, each MDAC block is connected to an input and a memory element through a local data bus. Each memory element consists of two sections: the weight registers (of 12-bit size for a 12-

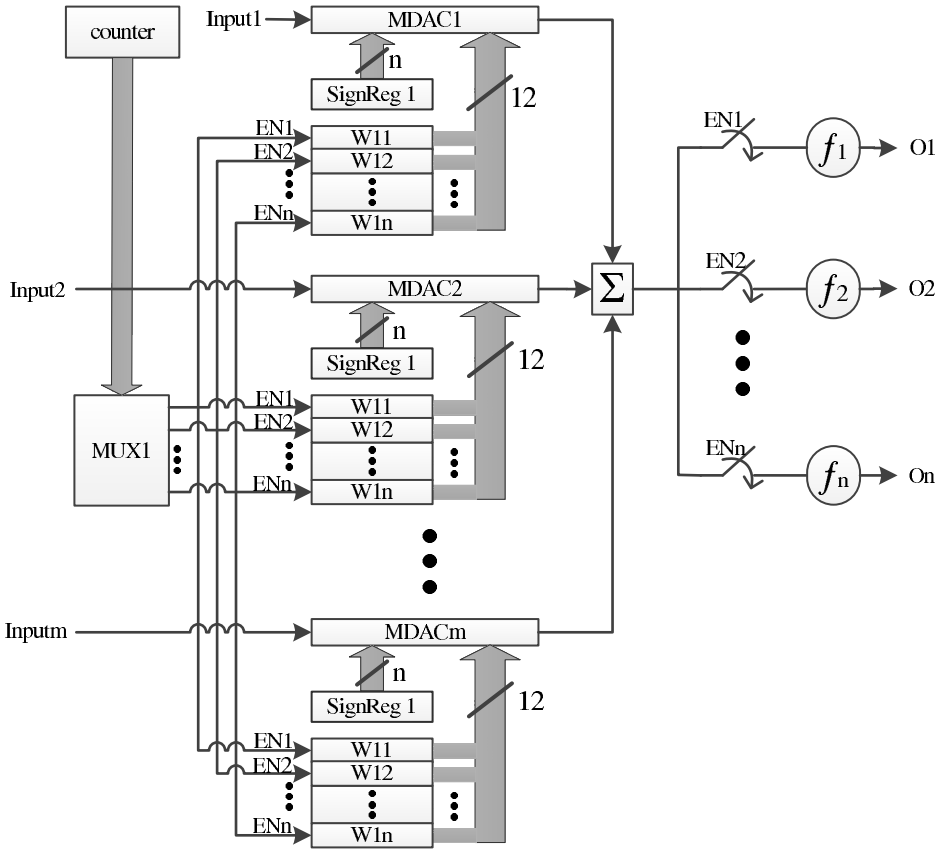

Figure 2.2: The block diagram of the proposed architecture

bit-resolution network) and one sign register. In block diagram of Figure 2.2, weight registers are named as W_{mn}, where m and n represent the number of corresponding neuron and its input, respectively. The sign register is shown as $SignReg_m$, where m refers to the number of its corresponding neuron.

The number of required weight registers in each memory element is equal to the number of neurons in that layer. The weight registers in each memory element are connected to the local data bus through tri-state buffers shared with the MDAC. There is only one sign register for each memory element which is indicative of the synaptic weight sign. Therefore, The number of weight registers in each memory

element is reflective of the number of neurons in that layer.

Each MDAC block can access the network's memory to access synaptic weights are stored in a time-multiplexed fashion. Therefore, each MDAC can read only one set of data from the memory at a time. Multiplexer's delay mainly depends on the clock frequency of the delta sigma modulator used in the MDAC block as well as the propagation delay of its digital components. Although the digital values fed to the MDAC blocks change by time, analog inputs to the MDAC do not change in all times.

In the proposed architecture shown in Figure 2.2, there is one multiplexer in each layer. Access to weight registers is controlled by the multiplexer in that layer. This way, each register can pass its information on the local data bus when its enable signal is triggered by the multiplexer. All of the multiplexers in the network are synchronized and are set when the multiplexer in their previous layer resets. This is to synchronize all of the outputs in a layer (which in another words, are the inputs to the next layer) to be summed together at the same time in the next layer after multiplied by their corresponding weight values.

Initially, the multiplexer, shown as $MUX1$ in Figure 2.2, activates $EN1$ allowing the weight values stored in registers W_{11}, W_{21} and W_{m1} to be available on the local data buses. After the multiplication operation is done by all the m MDACs, the results are added together in a summation block whose output is connected to the first neuron as long as $EN1$ is active. At this time, the result is stored in the first neuron, waiting for the rest of the outputs to be calculated. When $EN2$ is activated a second round of calculations is started and Registers W_{12}, W_{22} and W_{m2} are connected to MDACs. This time, the summation block output passes through the second neuron with $EN2$ active. This process repeats n times until all the inputs to the next layer

are calculated and are ready.

The aim in the proposed architecture is to include the multiplication operation inside the Digital to Analog Converter (DAC) block in order to reduce the number of required MDAC modules. This increases the modularity of the proposed method and makes it easier for this architecture to be adjusted for different network sizes. As it will be discussed later, the MDAC module is designed for high resolution synaptic calculations using pulse stream approach.

2.4 Multiplying Digital to Analog Converter Architecture

In any hardware implementation of neural networks the multiplication module is the dominant building block in terms of size and required circuitries. In particular, parallel addition and multiplication consume large chip areas in digitally implemented neural networks. Hence, reducing the synaptic weights' resolution may seem an acceptable solution for many designers in order to decrease the chip area. However, low resolution synapse values can make it difficult for some network architectures to converge reliably [68].

The Multiplying Digital to Analog Converter (MDAC) in the proposed architecture consists of two main blocks: a Digital to Analog Converter (DAC) that receives the synaptic weights directly from the memory, and an analog multiplier to perform linear multiplication operation between the inputs and the synaptic weights.

The DAC operates based on Delta Sigma Modulation (DSM) in order to convert

synaptic weights to high bi rate delta sigma pulse streams. It has been proved that using pulse streams in neural networks can efficiently improve the accuracy of neural computations due to their increased resolution [39,40]. The low-bit quantizer in delta sigma modulator guarantees the accuracy of computations performed in the network. DSM pulse stream has some similarities to Pulse Width Modulation (PWM) pulse stream. However, several reasons contribute to the fact that DSM technique is used instead of PWM.

- To implement a Pulse Width Modulator, it is required to implement a ramp signal generator and in order to do so, an Operational Transconductance Amplifier (OTA) is required as well as a sample-and-hold block. Faulty OTA design can cause instability problems for the network.

- The output pulse stream in a Pulse Width Modulator has lower frequency than delta sigma pulse stream which makes filter design more difficult for PWM. For a lower output frequency, higher time constant is required to keep the desired voltage ripple at filter's output. A low-pass filter with larger time constant is larger in terms of chip area.

- Unlike Pulse Width Modulation, Delta Sigma Modulation is capable of moving quantization noise to a higher frequency band which is a very important characteristic of this type of modulation. The noise can be removed or attenuated using a low-pass filter.

- Regarding circuit implementation, Delta Sigma Modulation is a better option when the inputs to the system are digital. PWM implementation for digital inputs, particularly for larger word lengths, is a challenging. However, DSM has simpler and less complex circuit implementation since it is entirely in digital domain.

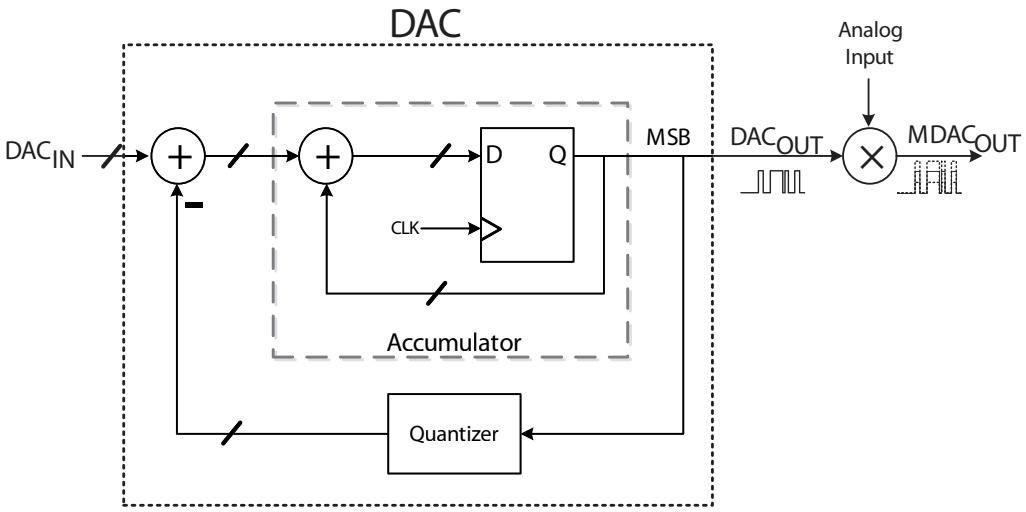

Figure 2.3: The general block diagram representation of the proposed MDAC architecture

The analog multiplier is the final stage in the MDAC module and changes the level of the pulse streams coming from the DSM stage depending on the input values. The most important factor that should be taken into consideration while designing the multiplier, is its linearity. Assume equation $z = Kxy$, where x and y are the multiplier's inputs, z is the multiplication output and K is the multiplication constant. In a linear multiplier, K is constant in all conditions and does not change in time domain. The final output of a layer in the network is obtained by adding the multiplication products of synaptic pulse streams and their analog input and averaging the result in time domain.

Figure 2.3 is a detailed representation of the MDAC block. The MDAC consists of an accumulator, a quantizer block, digital adders and an analog multiplier at the final stage. The first adder computes the difference between the input and the output of the DAC. An accumulator in a delta sigma DAC behaves exactly like an integrator in a delta sigma ADC and accumulates sums of the differences calculated in its previous

stage. This will force the average of the modulator output to follow the input value. The quantizer acts as a comparator and performs quantization by selecting the digital values representing VDD and *ground* depending on DAC_{OUT} signal.

Delta sigma DACs are generally implemented using a modulator and a Low-Pass Filter (LPF) in order to convert the pulse stream resulting from the modulation to a smooth voltage level. Consequently, the digital to analog conversion is usually ended with a LPF stage. However in the proposed architecture, the filtering stage is combined with the non-linear activation function. Hence, the averaging takes place after the multiplication is performed instead of right after the modulation stage. The advantage of this feature is that, it makes the LPF implementation more efficient since it employs the neurons which already exist in the network architecture. A typical RC implementation of LPF requires large capacitor and resistor values which is not desirable due to limited chip area. This filter is feasible to be implemented using neurons in much smaller chip area using few MOS transistors with the same time constant.

Filtering the result in the middle of calculations will cause resolution loss. This is because the multiplication operation is performed after decoding synaptic information using the LPF. The encoded signal can still contain ripples that can affect the multiplication result. In this architecture, all the arithmetic operations are performed in a coded and high resolution fashion and the averaging takes place when all the calculations are done. This will make the operations more accurate which is the second advantage of this method.

2.5 Neuron and Low Pass Filtering

For designing the suitable low pass filter several factors should be taken into consideration. First of all, it should be noted that the maximum allowable voltage drop after filtering the sigma delta pulse stream is 1 LSB which is calculated using Equation 2.1.

$$1LSB = \frac{V_{REF}^{+} - V_{REF}^{-}}{2^N} \tag{2.1}$$

where N is the number of bits of resolution and V_{REF}^{+} and V_{REF}^{-} are equal to $1.8V$ and $0V$ in CMOS $180nm$ Technology, respectively.

Table 2.1 shows maximum acceptable attenuation for different resolutions in terms of Volts and dB, based on Equation 2.1. Hence, as long as the ripple voltage on the output of low-pass filter is less than the values shown in Table 2.1, it will not be interpreted as a different digital value.

Bits of Resolution	Maximum Attenuation (V)	Maximum Attenuation (dB)
8	0.007031	-43
10	0.001757	-55.1
12	0.000439	-67.1
14	0.000109	-79.2
16	0.000027	-91.2

Table 2.1: Maximum acceptable attenuation for different resolutions in terms of Volts and dB

The discrete time representation for a low-pass filter is obtained using equations

2.2, where V_{in} is the input voltage and V_{out} is the output voltage of the system. Appendix B contains more details regarding how these equations are derived.

$$V_{out}(n) = \left(\frac{1}{1+RC}\right)[V_{in}(n) + V_{in}(n-1)] - \left(\frac{1-RC}{1+RC}\right)V_{out}(n-1) \tag{2.2}$$

By making the simple assumption that both input and output, $V_{out}[n]$ and $V_{in}[n]$, were initially zero for small values of n, then repeating equation 2.2, the output signal $V_{out}[n]$ for any given input signal $V_{in}[n]$ can be determined.

Furthermore, the transfer function of a low-pass filter in Laplace domain can be calculated as follows.

$$H(\omega) = \left|\frac{V_{out}}{V_{in}}\right| = \frac{1}{\sqrt{1+(RC\omega)^2}} \tag{2.3}$$

In Equation 2.3, H is the amplitude of the low-pass filter transfer function as a function of frequency ($\omega = 2\pi f$), RC is the filter time constant which is the amount of time required for filter output to settle around its final value.

The value of maximum acceptable voltage drop from Table 2.1 can be replaced with H in Equation 2.3 to calculate the suitable time constant (RC) value for the low-pass filter. In order to do so, Equation 2.3 can be rearranged as follows.

$$RC = \frac{1}{\omega}\sqrt{\frac{1}{H^2} - 1} \tag{2.4}$$

By combining equations 2.4 and 2.1, an expression for time constant in terms of bits of resolution N, V_{FS} and frequency can be derived.

$$RC = \frac{1}{\omega}\sqrt{\frac{2^{2N}}{V_{FS}^2} - 1} \tag{2.5}$$

where, V_{FS} is the full scale voltage which is equal to $V_{REF}^{+} - V_{REF}^{-}$.

Figure 2.4 shows how time constant changes in terms of resolution, assuming that the frequency is constant. The minimum required RC value changes exponentially regarding the number of bits of resolutions. The time constant has a slight change for resolutions between 6 to 11 bits where it starts to have a rapid increase.

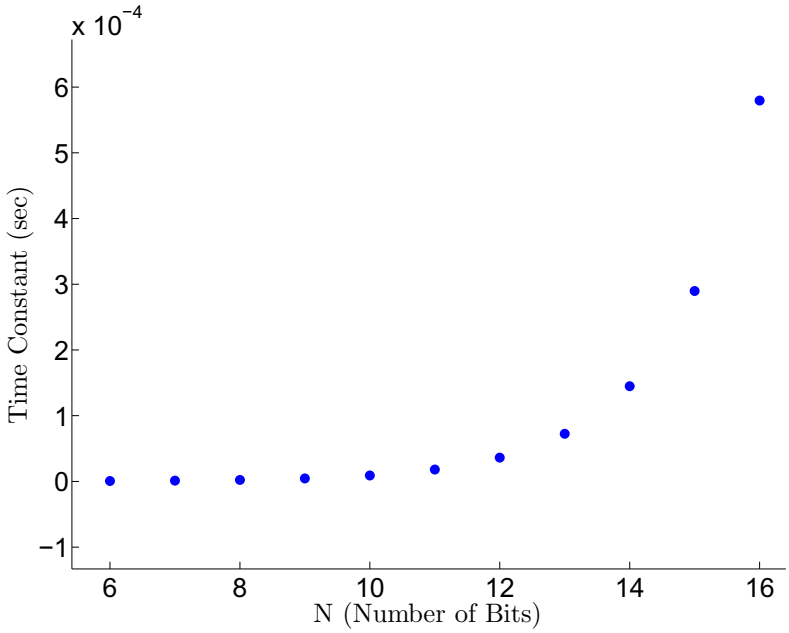

Figure 2.4: Time constant changes in terms of resolution

Table 2.2 is obtained using Equation 2.5 and includes minimum required time constant values for different resolutions and frequencies. As shown in Table 2.2, smaller time constant values are required for higher frequencies with the same resolution which leads to smaller filter designs.

It should also be noted that resolution is the second factor that affects the size of filter. As shown in Table 2.2 for a higher resolution network larger time constant at

the filtering stage is required operating at the same frequency.

Bits of Resolution	Maximum Ripple (mV)	Frequency of operation (MHz)	Minimum Required Time Constant (μs)
10	1.757	10	9.06
10	1.757	100	0.90
12	0.439	10	36.25
12	0.439	100	3.62
14	0.109	10	146.01
14	0.109	100	14.60

Table 2.2: Minimum required time constant for different resolutions and frequencies of operation

It can be concluded from Table 2.2 that the proposed architecture can be adjusted to be used in networks with resolutions higher than 12 bits at some costs. There are two factors that put restrictions on how much the resolution can be increased: the maximum frequency that the network can be designed for and the available chip area. Larger time constant values are required for networks with higher resolution. This leads to larger area occupied by the low-pass filter circuit due to larger capacitors required. However, increasing the frequency of operation can help reduce the size of the filter at the same resolution. Therefore, the extra increase in the size of the filter due to the increase in resolution can be compensated by rising the frequency.

In the proposed architecture, the low-pass filter is implemented using the existing neurons in the network to take advantage of features such as smaller area and easier multiplexing scheme. In order to complete the design of the low-pass filter in this

section, it is required to include the neuron output impedance in the time constant calculations. Hence, the required capacitor value can be easily calculated. In Chapter 3, the neuron circuit will be introduced in details and its output impedance based on MOS transistors small signal model will be calculated.

2.6 Calculations in Pulse Stream Domain

There are two types of calculations performed in pulse stream domain in the proposed neural network architecture:

- Multiplication between a pulse stream and a constant voltage
  In this architecture the synaptic weights are encoded in time domain to maintain high resolution and high accuracy computations. Hence, this information can be decoded by averaging the bit streams in time domain. According to this, after multiplying the bit stream by a constant voltage, the average of the bit stream after filtering is multiplied by that constant voltage.

$$I \int_{\tau} v(t)\,\mathrm{d}t = \int_{\tau} I v(t)\,\mathrm{d}t \tag{2.6}$$

  where, I is the constant input voltage, $v(t)$ is the time varying pulse stream in which the synaptic data are encoded and τ is the time frame in which the the output pulse stream is being averaged.

- Addition between two or more pulse streams
  In general, there are two ways to add two pulse stream signals:

  1. OR operation using digital components. This method can be useful when the network is restricted to include only digital components. The OR technique is based on the fact that for signals that have information encoded in

their time domain, the OR operation can be a good approximation for addition between those signals. However, it has been shown that this addition method is not accurate and is only suitable for small-sized networks [69].

2. Pulse stream addition in analog domain. In general, this type of addition can be performed in current-mode or voltage-mode. Vividly, current-mode addition can most of the times be the prefferd method since it requires simpler circuitries. The resulting pulse stream is no longer a pure delta sigma modulated signal because its amplitude is doubled in some parts as a result of addition between two high signals.

Figure 2.5 shows how addition between pulse streams is performed in current-mode and digital domain using OR operation. It also demonstrates a comparison on how the average values of pulse streams are affected in each of these domains. Graphs (a) and (c) are two input pulse streams with different duty cycles on which the addition operation is being performed. Graphs (b) and (d) are the average values of (a) and (c), respectively, after the filtering stage. (b) settles at $0.55V$ and (d) settles at around $0.8V$. Graph (e) is the addition result between the two pulse streams in current-mode domain. As it can be seen, the amplitude of this signal is doubled when both signals (a) and (c) are high. Graph (f) is the current-mode addition result after passing the filter stage which settles at around $1.35V$. As it can be seen, the average of the addition result is equivalent to adding the averages of the two input pulse streams. Plot (g) shows the result of addition between (a) and (c) in digital domain using OR operation and (h), is the average of (g) over time. As it can be seen, (h) settles at around $1V$ which gives a different result from (f). Hence, current-domain seems to be a more appropriate choice for performing addition between pulse streams.

2.7 Summary

In this chapter, the proposed architecture for a mixed-signal neural network was explained in system level which include MDAC blocks, registers, adders, neurons, multiplexers and current switches. After that, MDAC, neuron and adder were discussed in more details as fundamental building blocks in the proposed neural network architecture. Arithmetic operations are performed in pulse stream domain in order to maintain accurate and high resolution calculations for the network. In addition to all, multiplexing scheme was applied to the network in order to reduce the number of multiplier blocks and interconnections.

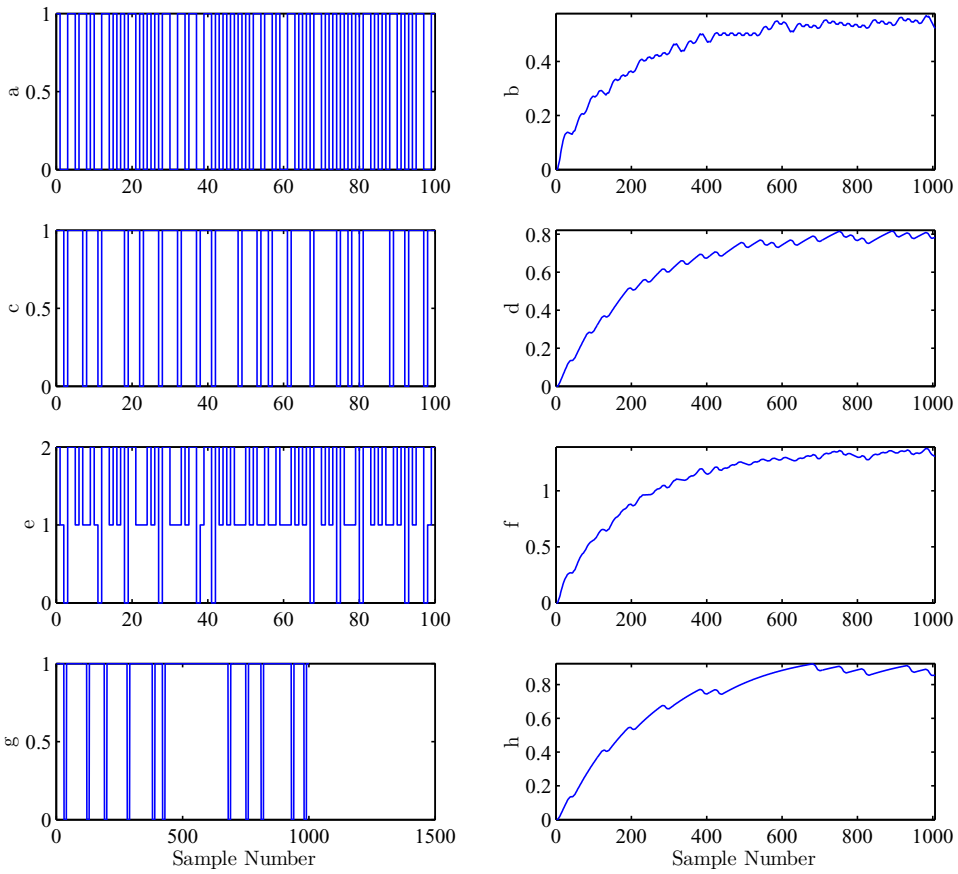

Figure 2.5: Comparison between different pulse stream addition methods: (a) First sample pulse stream, (b) The average value of first sample pulse stream after filtering, (c) Second sample pulse stream, (d) The average value of second sample pulse stream after filtering, (e) The addition result between the first and the second pulse stream using current-mode method, (f) The average of the addition result between the first and the second pulse stream using current-mode method after filtering, (g) The addition result between the first and the second sample pulse streams using OR operation, (h) The average value of the addition result between the first and the second sample pulse streams using OR operation after filtering

Chapter 3

Circuit Designs and Simulation Results

In this chapter, the circuit-level configuration of each building block used in the proposed architecture is presented. Circuit topologies and simulation results including transient, Monte Carlo and corner analysis for each building block are presented and discussed in details. All the simulation results reported in this chapter are done on full custom layouts, presented in Appendix A, and under non-ideal conditions. All the parasitic capacitors are taken into consideration to make sure that all the circuits are tested in real conditions. The circuits are simulated and laid out in TSMC CMOS $180nm$ technology.

3.1 Multiplying Digital to Analog Converter

Multiplying Digital to Analog Converter (MDAC) block plays an essential role in the proposed network architecture. The delta sigma modulator employed in this block encodes 12-bit synaptic weights into time domain which is later multiplied by the network's input using an analog multiplier. Delta sigma modulator is capable of moving quantization noise to a higher frequency band. In addition to that, efficient filtering of the output signal is feasible due to the high frequency pulse stream at the delta sigma modulator. The MDAC module can be divided into two main blocks: the delta sigma modulator and the analog multiplier.

3.1.1 Delta Sigma Modulator

Delta sigma modulator and low-pass filter are essential building blocks in any type of delta sigma data converter, either Digital to Analog Converters (DAC) or Analog to Digital Converters (ADC). The modulator is implemented digitally due to digital input signals, otherwise it is implemented by analog components. The same rule applies to the low-pass filter implementation. The implementation domain of the LPF changes depending on the modulator's output signal, which is digital and analog in ADCs and DACs, respectively.

A delta sigma modulator block is used in a digital to analog conversion scheme for the proposed architecture. That means that inputs to this block are digital coming form 12-bit registers used to store neural network synaptic weights. The modulator outputs a pulse stream which is a 1-bit serial signal with a bit rate much higher than the data rate of the system. In the proposed digital to analog converter, the pulse stream is generated using digital feedback and has the high and low digital values (1

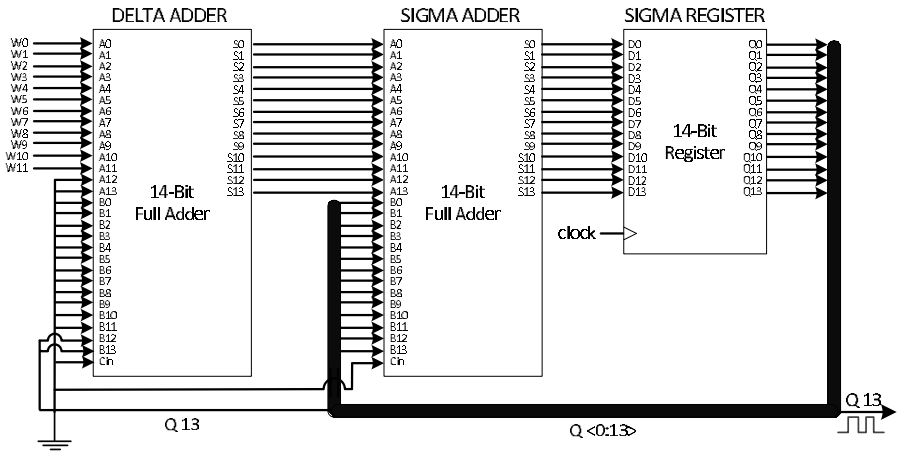

Figure 3.1: delta sigma modulator for a 12-bit system

and 0). The average duty cycle of this pulse stream is proportional to the value of the binary input. This average value is obtained by passing the pulse stream through a low-pass filter. The output of the delta sigma modulator is also known as a Pulse Proportion Modulated (PPM) signal.

Figure 3.1 shows the schematic representation of the delta sigma modulator for a 12-bit resolution system. Two binary adders perform the "Delta" and "Sigma" operation in this implementation. The first adder (Delta Adder) subtracts the digital input (which in our architecture comes from the weight registers) from the modulator's output. The second adder (Sigma Adder) adds its previous value stored in Sigma Register with the output of the first adder. The adder blocks are ripple carry adders and the register is implemented using D flip flops.

As part of the mixed-signal design flow, this implementation was first described in Verilog for system level simulation. The schematic description using library cells was extracted from the Verilog codes using Synopsys toolsets and imported in Cadence. After verification, library cells were replaced with fully custom-designed blocks. Fig-

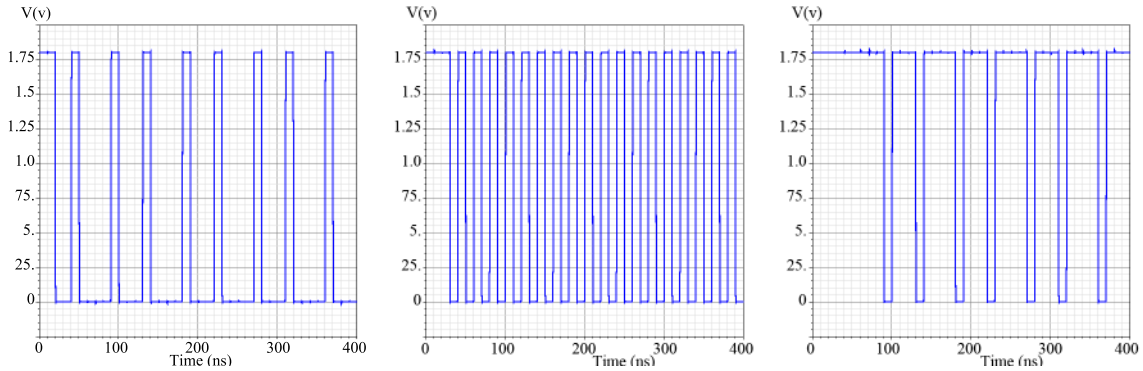

Figure 3.2: Post-layout transient simulation result of the delta sigma modulator

ure 3.2 shows the transient analysis results after running post-layout simulations. The simulations are done in non-ideal conditions and all the inputs are applied to the circuits after passing through buffer and capacitor stages. The plots in Figure 3.2 show the output pulse stream for the digital input values of 800_{HEX}, $2AA_{HEX}$ and $C54_{HEX}$, from left to right. As shown in the graphs, digital synaptic data are encoded in the time domain of each pulse stream. Averaging these signals in time domain outputs a voltage value equivalent to the corresponding digital weight value.

3.1.2 Multiplier

The very first analog multipliers were designed to be used in mixers and amplitude modulators for multiplying two analog operands with a suitable multiplication constant with a desired dimension. Analog multipliers can be classified as single-quadrant (where both operands are unipolar), two-quadrant (where one of the operands is unipolar and the other one is bipolar) and four-quadrant (where both operands are bipolar).

An analog multiplier is basically a non-linear block that receives two signals, such

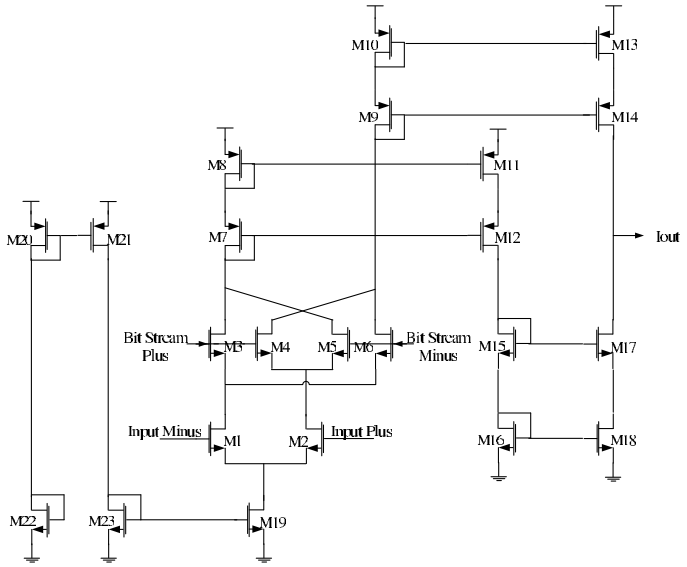

Figure 3.3: Schematic diagram of the analog multiplier employed in the proposed MDAC module

as $u_1(t)$ and $u_2(t)$, as inputs and generates polynomial functions with different orders, such as $u_1^2(t)$, $u_2^2(t)$, $u_1(t)u_1(t)$ at its output.

In the proposed architecture, multiplication is performed in mixed-signal domain implemented using a CMOS Gilbert multiplier, shown in Figure 3.3. This configuration is the CMOS version of the bipolar wide-range multiplier designed by G. Gilbert [70, 71].

Figure 3.3 shows the schematic of the analog multiplier used in this architecture. In general, the multiplication result is as following:

$$I_{out} = I_{o1} - I_{o2} = Kxy \tag{3.1}$$

where x and y are the two operands involved in the multiplication operation, the bit stream and the analog input.

For a linear multiplication, K should be constant and can be adjusted by changing the size of transistors $M1$ and $M2$ used in differential configuration. Larger width to length ratio for transistors $M1$ and $M2$ increases the gain of the differential pair which leads to a larger K.

Transistors $M19$, $M20$, $M21$, $M22$ and $M23$ are used to bias the differential pair. I_{o1} is inverted using a current mirror including $M15$, $M16$, $M17$ and $M18$ in order to be subtracted from I_{o2} and yield $I_{out} = I_{o1} - I_{o2}$. Table 3.1 shows the width to length ratio of the transistors used in the analog multiplier circuit.

The first input is bipolar and comes from the delta sigma modulator which has coded the synaptic weight into a duty cycle of a high bit rate delta sigma pulse stream. The second input is unipolar with an analog value which is the inputs to the network. A two-quadrant multiplier for the proposed network architecture is used. The multiplier receives the pulse stream coming from the modulator and changes its level depending on the analog input it is receiving as its second input.

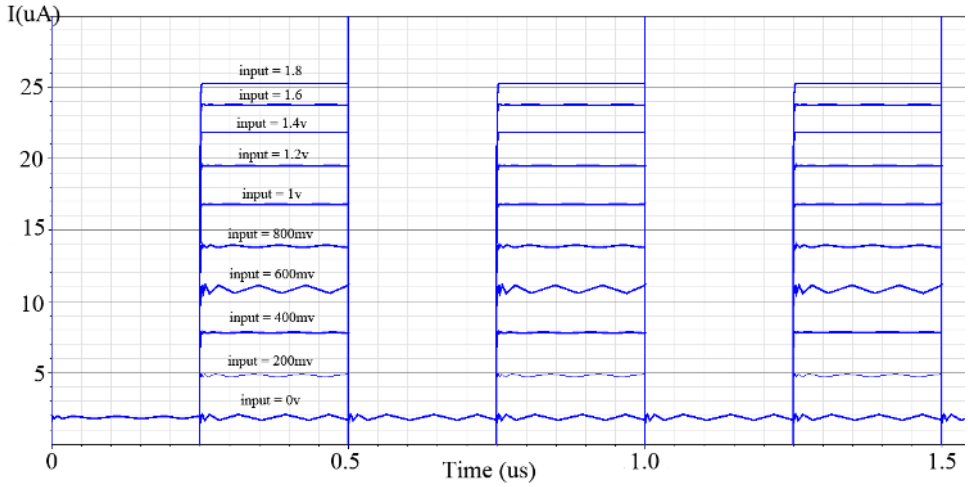

Figure 3.4: Transient analysis results of the linear analog multiplier circuit

Table 3.1: Transistor sizes for the multiplier circuit

M1	0.55	M7	5	M13	5	M19	1.25
M2	0.55	M8	5	M14	5	M20	1.25
M3	1.5	M9	5	M15	5	M21	2
M4	1.5	M10	5	M16	5	M21	1.25
M5	1.5	M11	5	M17	5	M23	1.25
M6	1.5	M12	5	M18	5		

Figure 3.4 shows the transient analysis results of the analog multiplier block after running post-layout simulations. A sample square wave with the maximum amplitude of Vdd and frequency of $2MHz$ is applied as one of the inputs. The second input is set as a parameter which changes between 0 to $1.8V$ with a $200mV$ step size between each value. The simulations are carried out using Cadence parametric analysis tool. As it can be seen in Figure 3.4, the multiplication operation is performed almost linearly for the selected range of inputs since the the difference between output currents for different input values is almost constant. This is because K in Equation 3.1 is kept constant.

Cadence DC analysis is carried out to better show how linearly the output changes in our desired range of input, 0 to $1.8V$. The DC simulation results are shown in Figure 3.5. The input voltage changes between 0 to $1.8V$ on the horizontal axis and the output current changes between $12.5\mu A$ to $26\mu A$ on the vertical axis.

Monte Carlo Analysis (MCA) is run to show the effect of process variations and transistor mismatches on the circuits. MCA gives a better understanding on circuits behavior than Corner Analysis (CA) as CA does not include the effect of transistor

mismatches.

Figure 3.6 shows the MCA results for the analog multiplier circuit after 200 runs

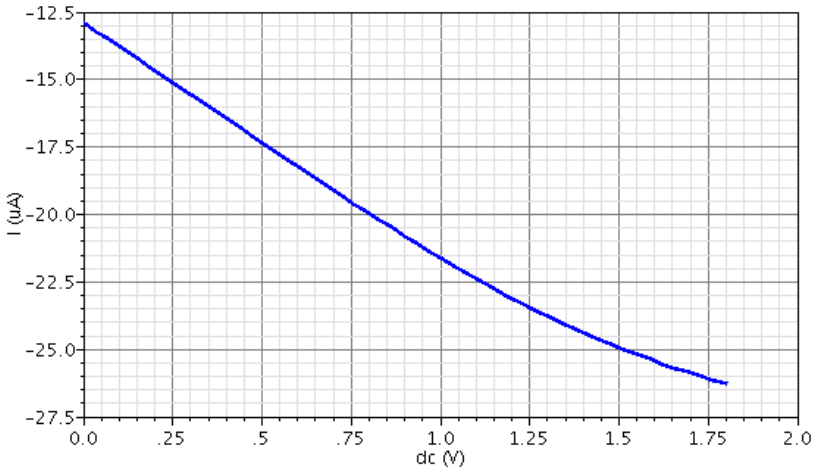

Figure 3.5: DC analysis results of the linear analog multiplier circuit

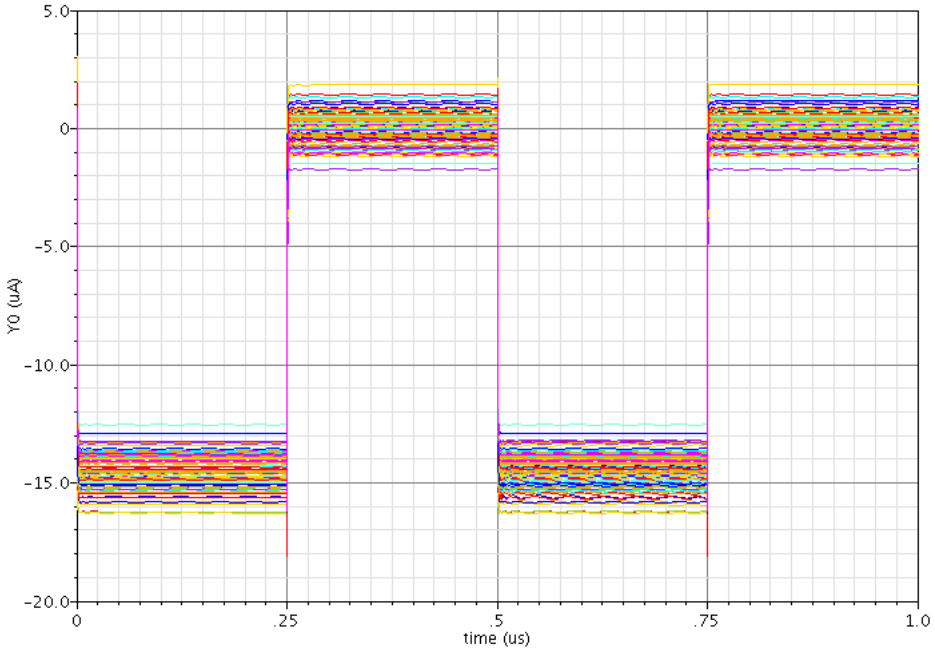

Figure 3.6: Monte Carlo analysis results for the multiplier circuit

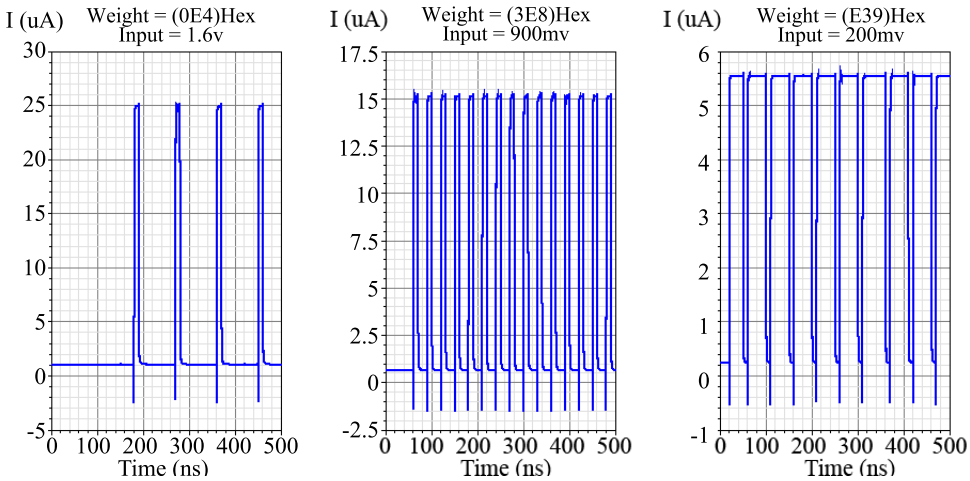

Figure 3.7: Transient analysis results of the MDAC module for three different input and weight values

of transient analysis. As an example, an arbitrary square wave with the maximum amplitude of Vdd and frequency of $2MHz$ is applied as one of the inputs and the other input is set at $1V$. Figure 3.6 shows that, the output current keeps its square shape under non-ideal conditions, such as mismatches and process variations, and is only subjected to a slight change in its amplitude. In addition to that, the operation region of the transistors does not change in spite of the changes in their threshold voltage.

3.1.3 MDAC Simulation Results

Figure 3.7 shows the transient analysis results for the MDAC after connecting the delta sigma DAC in Figure 3.1 and the analog multiplier in Figure 3.3. Three simulation results are shown that each represents the multiplication result between different input values ($1.6V$, $900mV$ and $200mV$) synaptic weight values ($(0E4)_{HEX}$, $(3E8)_{HEX}$ and $(E39)_{HEX}$). The synaptic weights are presented in hexadecimal num-

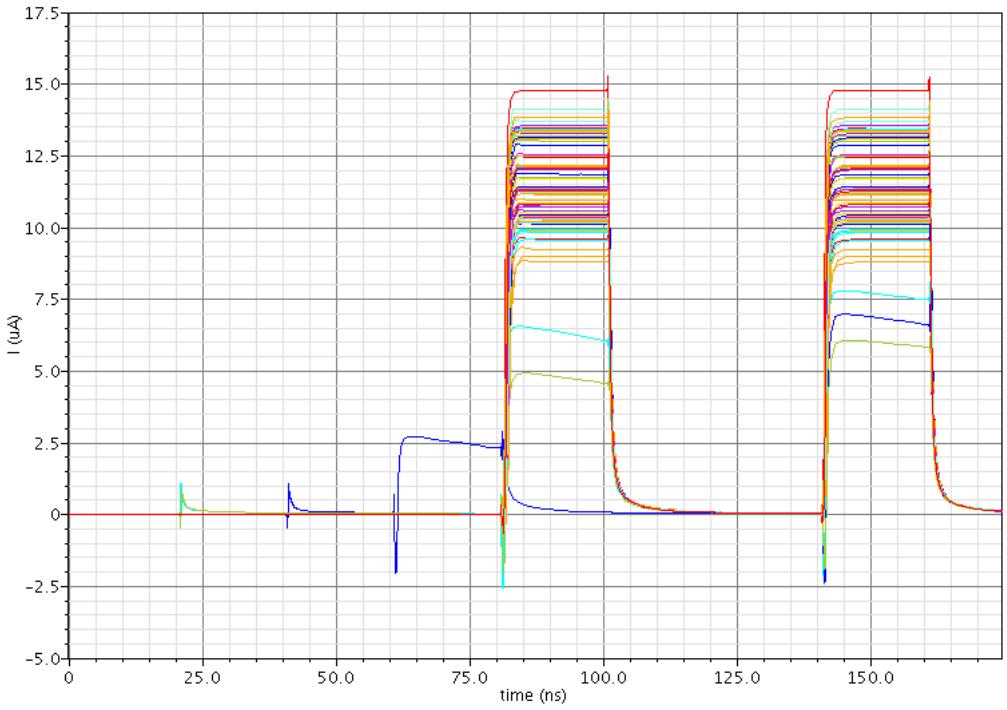

Figure 3.8: Monte Carlo analysis results for the MDAC

ber system while the inputs are analog voltage values. As it can be seen, amplitude of the output current is reduced from left to right as the analog input value decreases. However, the pulse stream in the plot on the left has the smallest duty cycle since its corresponding synaptic weight has the smallest value. Consequently, the average of the pulse streams over time increases from left to right. The frequency of the output pulse stream only depends on the synaptic weight values and is not affected by the multiplication operation.

Figure 3.8 shows the transient MCA result for the MDAC block, with the DAC and multiplier connected to each other. The simulations are run 100 times for both mismatch and process variations. The digital input at the DAC is set to $(AAB)_{HEX}$ and the analog input is $1V$. This figure shows that the delta sigma modulator out-

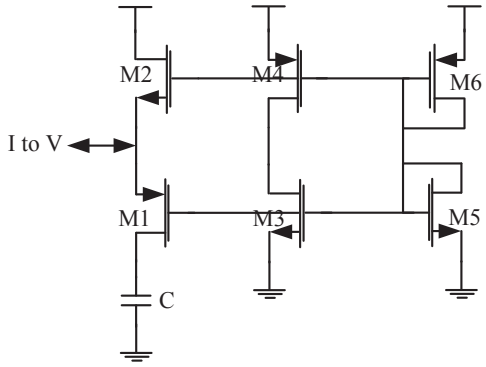

Figure 3.9: Non-linear resistive type neuron

put changes reasonably under transistor mismatches and process variations since the pulse streams have the same duty cycles. That means the high resolution weight values encoded in time domain of the delta sigma pulse streams remain unchanged for worst case conditions.

3.2 Neuron

Neurons in the proposed neural network are resistive type and distributed in the network to increase signal to noise ratio of the network [72, 73]. The neuron transfer function self-adjusts, preventing the saturation of neurons when inputs increase. The neuron uses the basic non-linearity in V-I characteristics of the MOS transistors to approximate the sigmoid-like function.

All delta sigma digital to analog converters include a low-pass filter at the final stage to average the bitstream and to relate it to a voltage level. This in turn requires large resistor and capacitor values for a typical RC implementation. To overcome this

Table 3.2: Transistor sizes for the neuron circuit

M1	7.5	M4	20
M2	1.25	M5	1
M3	4	M6	8

limitation, in this architecture the neuron circuit acts also as a low-pass filter. Therefore, the existing resources in the network are used for the filter design and there is non need for additional circuitries.

Because of the neuron's relatively high resistance, required capacitance for the low-pass filter reduces significantly. Additionally, using a capacitor in the neuron's circuit is essential since the network is multiplexed and different sets of data are transmitted through one data path. This capacitor stores the output value of a layer and keep them stored until all the outputs are synchronized and ready to be processed. The schematic of the neuron cell is shown in Figure 3.9 and the transistors width to length ratios are given in Table 3.2.

Figure 3.10 shows the simulation result for the neuron transfer function. The neuron's bias point is set by adjusting size of $M5$ and $M6$, choosing smaller transistors will bias the circuit at a higher voltage. We can also define the shape of the transfer function by changing the length of $M1$, $M2$, $M3$ and $M4$. Smaller length for these transistors makes the transfer function more linear.

Figure 3.11 shows the MCA transient results for the neuron circuit after 100 runs of simulations. This figure illustrates how neuron output voltage is affected by a change in its input current in time domain and how this response changes by mis-

match and process variations. Figure 3.12 shows the MCA results for the neuron circuit acting as a low-pass filter in frequency domain. As it can be seen, there is a variation for maximum amplitude when process and mismatch variations are taken into consideration. However, the cut-off frequency remains almost constant.

3.2.1 Low-Pass Filter Considerations

In order to see how the neuron works as a low-pass filter, we need to calculate the impedance seen from the capacitor C. For this purpose transistors are replaced with their small signal model and the capacitor C is replaced with test voltage source V_T with test current of I_T. Equation 3.2 is the KCL rule for node V_S which is the node where sources of $M1$ and $M2$ meet. Equation 3.3 is the KCL rule for the test voltage node V_T.

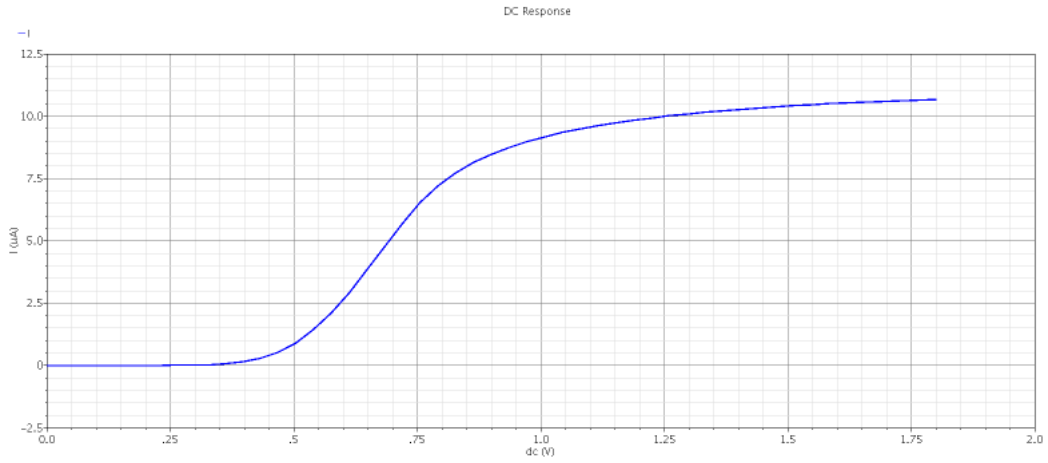

Figure 3.10: Neuron transfer function

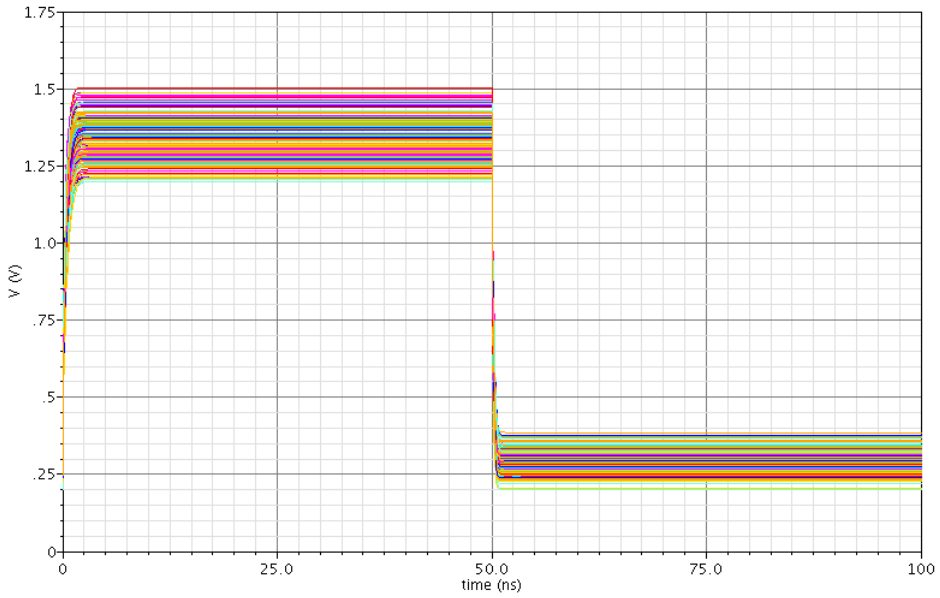

Figure 3.11: Monte Carlo transient analysis result for neuron output

$$\frac{-V_S}{r_{o1}} - gm_1 V_S + gm_2 V_S + \frac{V_T - V_S}{r_{o2}} = 0 \tag{3.2}$$

$$V_S = (\frac{1}{gm_2 + \frac{1}{ro2}})(I_T - \frac{V_T}{r_{o2}}) \tag{3.3}$$

By combining equations 3.2 and 3.3 we have:

$$-(\frac{1}{gm_2 + \frac{1}{r_{o2}}})(I_T - \frac{V_T}{r_{o2}})(\frac{1}{r_{o1}} + gm_1 - gm2 + \frac{1}{r_{o2}}) + \frac{V_T}{r_{o2}} = 0 \tag{3.4}$$

$$R = \frac{r_{o2}(\frac{1}{gm_2+\frac{1}{r_{o2}}})(\frac{1}{r_{o1}} + gm_1 - gm_2 + \frac{1}{r_{o2}})}{1 + (\frac{1}{gm_2+\frac{1}{r_{o2}}})(\frac{1}{r_{o1}} + gm_1 - gm_2 + \frac{1}{r_{o2}})} = \frac{Ar_{o2}}{1+A} \tag{3.5}$$

Equation 3.5 defines the impedance seen from capacitor C, which linearly depends on the output resistance of transistor $M2$. Using this equation proper value of C can

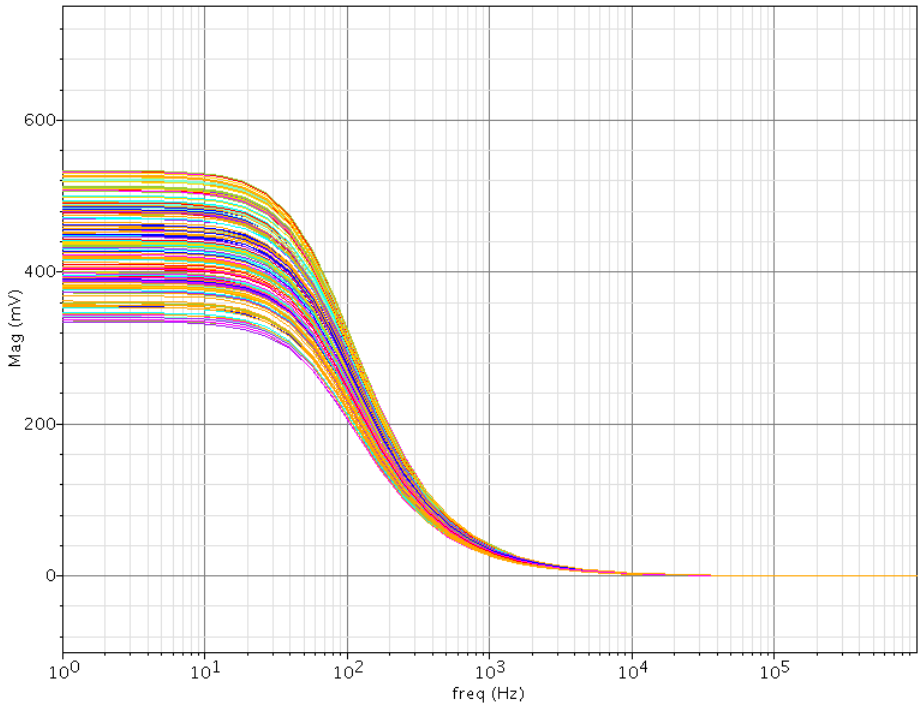

Figure 3.12: Monte Carlo ac analysis result for neuron output

be obtained for the desired time constant RC.

As discussed in details in Chapter 2, number of bits of resolution defines the maximum allowable ripple at the low-pass filter output. This ripple voltages along with the operating frequency decide the time constant of the filter, RC. It should be also noted that the maximum allowable frequency of the pulse streams are limited by the low-pass filter cut-off frequency, $\frac{1}{2\pi RC}$. Consequently, the maximum value of RC depends of the frequency in which the network is implemented.

Figure 3.13 shows the MCA result of the low-pass filter output after 500 simulation runs. As it can be seen, the average of the output is $289.3mV$ with a the standard deviation of only $34.6mV$ which is quite low compared with its output range.

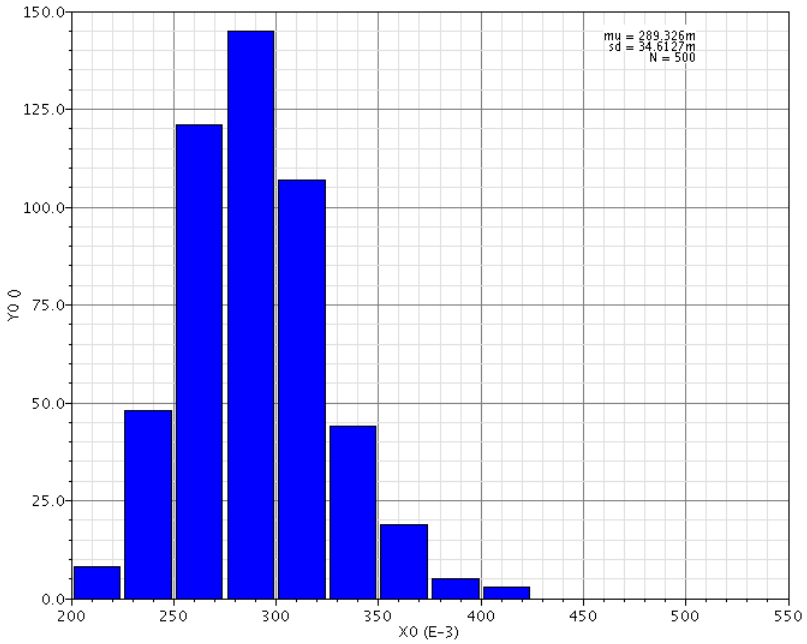

Figure 3.13: Monte Carlo analysis result for the LPF output

3.3 Summary

The fundamental building blocks in the proposed architecture were discussed in general in Chapter 2. In this chapter, each of these building blocks were discussed in more details providing their circuit designs and simulation results. Different type of simulations were carried out in order to test the circuits with any type of input variations, such as DC, AC, transient and Monte Carlo. All the circuits are designed, simulated and laid out in TSMC CMOS 180nm Technology.

Chapter 4

Solving XOR Problem

XOR logic function and parity problem is among the most difficult problems for neural networks to solve because the input patterns are not linearly separable. A network with at least one hidden layer and non-linear activation function is capable of solving such problems. In this chapter, we are going to employ the proposed architecture for a high resolution feed-forward neural network to solve XOR problem. In order to do so, the architecture presented in Chapter 2 is expanded by cascading two layers in the proposed configuration. As it was mentioned before, using high-resolution Multiplying Digital to Analog Converter (MDAC) blocks make the network design more modular. Hence, this architecture can be easily adjusted for any network size and structure.

4.1 Network Configuration

The neural network configuration employing the proposed architecture has two inputs, a hidden layer and one output. The learning process and weight calculations are performed in Matlab as follows:

$$IW = \begin{bmatrix} 010B & 010B \\ 001B & 001B \end{bmatrix}_{HEX} \quad LW = \begin{bmatrix} 0D56 & 1FFF \end{bmatrix}_{HEX}$$

where IW and LW represent Input Weights and Layer Weights, respectively.

The training function is backpropagation which updates the synaptic weights according to Levenberg-Marquardt optimization. Network's bias values are initialed at zero.

Figure 4.1 shows the block diagram representation of a $2-2-1$ neural network implementation to solve the XOR problem using the proposed architecture in Chapter 2. As it can be seen in this figure, there are three neurons in this network, denoted as f_1, f_2 and f_3. The number of MDAC modules in each layer is equal to the number of inputs in that layer. Hence, the hidden layer has two MDACs since there are two inputs to the network. The output layer has two inputs coming from the hidden layer in the network and therefor, it also has two MDACs as well.

The output of each MDAC block is a pulse stream current signal that enters a summation node. $M1$-$M4$ are the memory elements that provide the required synaptic data for each multiplication operation which takes place in MDAC modules. The enable signals marked as $EN1$ and $EN2$ are generated in the multiplexer and control the synaptic weights flow in the data buses as well as triggering current switches.

Figure 4.2 shows the simulation results of the $2-2-1$ network trained to solve the XOR problem implemented in TSMC CMOS $180nm$ technology. The transient

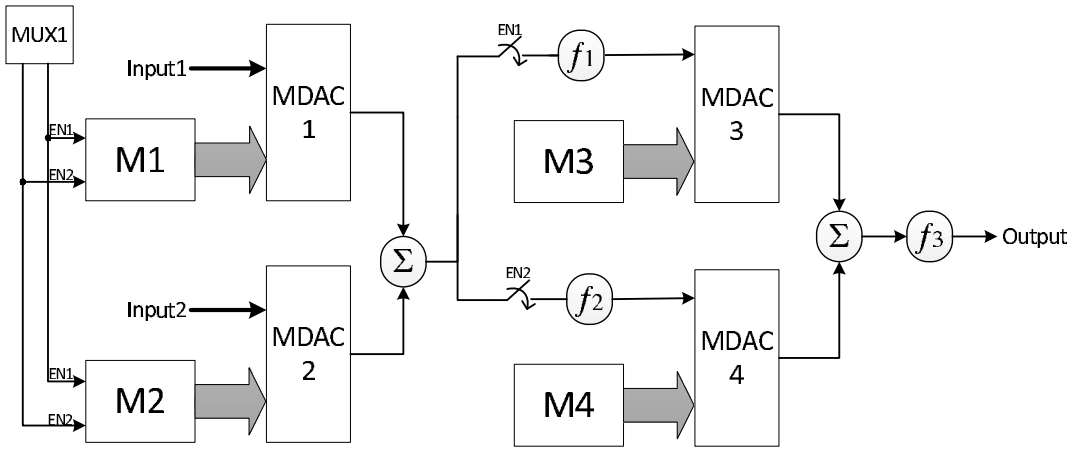

Figure 4.1: Block diagram of the $2-2-1$ network using the proposed architecture

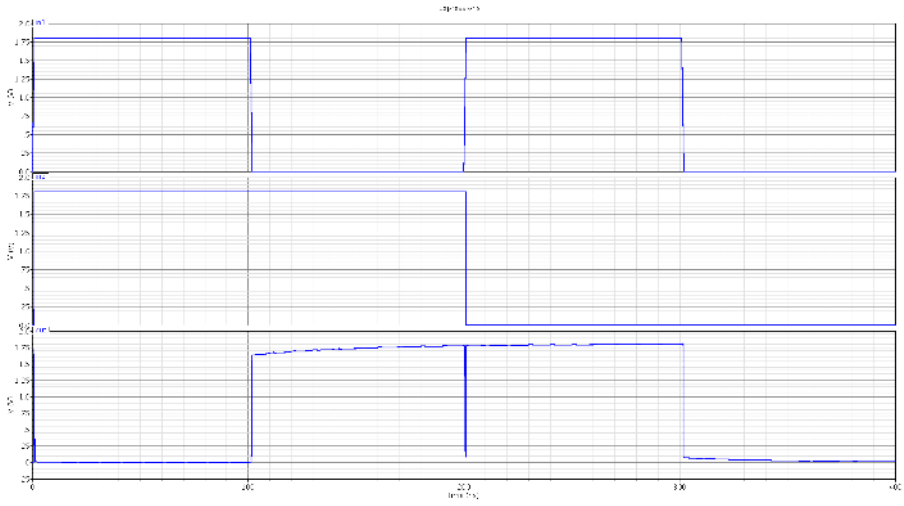

Figure 4.2: Transient analysis result of the $2-2-1$ network implemented using the proposed architecture

simulation is performed under non-ideal conditions. The input and clock signals are passed through buffer stages so that the rise and fall time are taken into consideration. Additionally, all the parasitic capacitances are included in simulation results. Table 4.1 summarizes the network features.

Technology	TSMC CMOS $180nm$
Power Supply	$1.8v$
Clock Frequency	$100MHz$
Weight Resolution	12 bits
Multiplexing Delay for Full Resolution	$81.9\mu s$
Time Constant	$3.62\mu s$
Settling Time	$18.1\mu s$
Cut-off Frequency	$176MHz$
Total Power Consumption	$0.538mW$
Active Die Area	$28503.16um^2$
Synapse Cell (MDAC) Area	$6454.22um^2$
Total Transistor Count	5662
Synapse (MDAC) Transistor Count	1408

Table 4.1: Network general features

4.2 Comparisons

The proposed neural network is implemented in mixed-signal domain using high resolution multiplying digital to analog converter blocks. The 12-bit synaptic weights are stored in registers and the inputs are analog. However, the architecture can be adjusted for higher resolutions with a cost of speed and area.

In the presented neural network architecture, all the arithmetic operations, such as addition, multiplication, and nonlinear transfer function realization, are performed

in analog domain which makes in more area-efficient compared with networks implemented in digital using large look-up-tables [32–34]. Considering the fact that all synaptic weights have 12 bits of resolution, performing parallel multiplication and addition in that resolution consumes a considerable portion of chip area.

Neural networks implemented in analog are generally susceptible to noise and process variations comparing to digital networks. In the proposed network, delta sigma modulation method has been employed in order to convert digital data to analog with high accuracy and perform high resolution synaptic calculations. Additionally, the quantization noise in the network is moved to higher frequency bands due to oversampling modulation employed in this network. Moreover, the overall signal to noise ratio of the network is improved due to distributing neuron's non-linear transfer functions in the network. In addition to these, the proposed architecture benefits from other advantages of distributed networks such as scalability and extra robustness [72, 73].

Comparing with neural networks implemented using PWM techniques [74, 75], the propsed architecture benefits from simpler and smaller filter design due to high frequency of delta sigma pulse stream. In addition, choosing delta sigma modulation specially for digital to analog data conversion makes the design process simpler, more modular and robust. This is because in a digital to analog converter, delta sigma modulator is implemented in digital; however, PWM design requires accurate OTA and sample-and-hold design. It is worth to mention again that, unlike PWM modulation, delta sigma modulation is capable of shifting quantization noise to a a higher band of frequency.

One of the advantages of the proposed architecture over analog neural networks is the reconfigurability [11, 76]. The MDAC blocks along with the mutual design of

neuron and the low-pass filter make this architecture more modular and easier to be adapted for different network sizes and configurations. Furthermore, the high resolution synaptic weights used in this architecture makes it an appropriate choice for larger network sizes. However, neural networks designed based on common CMOS MDACs can not be used for high resolution calculations required in larger networks [35, 36].

In terms of weight storage, saving synaptic data on capacitors or multiple-valued memories [9] suffer from inaccuracy due to the leakage currents in capacitors and therefore, require extra refreshing circuitries which require larger chip area. These information can be stored on registers more reliably.

The proposed architecture consumes more area due to its high resolution nature, specially when compared to fully analog implemented networks, or other low resolution mixed-signal networks [9, 35, 36]. However, due to different techniques used, such as multiplexing and employing analog components for arithmetic calculations, the proposed neural network architecture consumes less area than digital networks, considering 12 bits of resolution.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

In this thesis, a mixed-signal feed-forward neural network architecture using high resolution Multiplying Digital to Analog Converter (MDAC) blocks is presented. The network inputs are in analog domain and synaptic weights have 12 bits of resolution stored in registers. All the arithmetic operations, such as addition, multiplication and neuron nonlinear activation function realization, are performed using analog components to save area and power along with maintaining high operation speed. The proposed network is trained off-line and the synaptic weights are downloaded to the chip after being calculated off the chip.

Each MDAC block employs a digitally implemented Delta Sigma Modulator in order to convert digital synaptic values to high bit rate pulse streams. Therefore, all the synaptic values are encoded in time domain rather than a voltage level which is

more accurate and reliable. Using this method of data conversion, all the synaptic operations are performed in pulse stream domain in order to maintain high resolution calculations for the network. Additionally, number of interconnections and overhead area is considerably lower in pulse stream domain comparing with digital.

In order to finalize the digital to analog data conversion, a new method of low-pass filtering is proposed which exploits neuron blocks existing in the network. This method requires less chip area comparing to conventional RC filters. Moreover, applying a time-multiplexing scheme becomes feasible using this type of low-pass filtering method. A time-multiplexed neural networks uses the same data path for different synaptic values in order to decrease the chip size and compensate for the extra overhead area caused by the registers storing high resolution synaptic weights.

A $2-2-1$ network is designed using the proposed architecture to solve the XOR problem. All the circuits are designed and laid out in a full custom fashion in TSMC CMOS $180nm$ Technology. The network is tested under non-ideal conditions and for different mismatches and process variations.

5.2 Future Works

The proposed neural network architecture can be used in applications where high resolution synaptic weights are required for the network, particularly in larger network sizes. This thesis is the first step in a high-resolution neural network in which the synaptic weights are calculated and updated off the chip. Using a hardware-friendly learning rule, such as MRIII, required blocks for training the network on the chip can be designed and added to this work as a second step. Taking advantage of pulse stream calculations can provide the accuracy that a high resolution neural network

requires for its weight calculation and update.

Appendices

Appendix A

Layout Diagrams

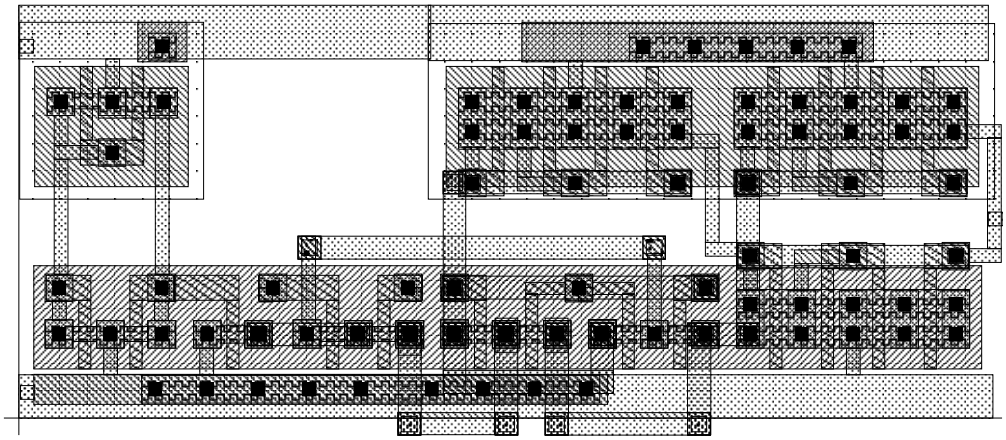

Figure A.1: Layout design of the analog multiplier

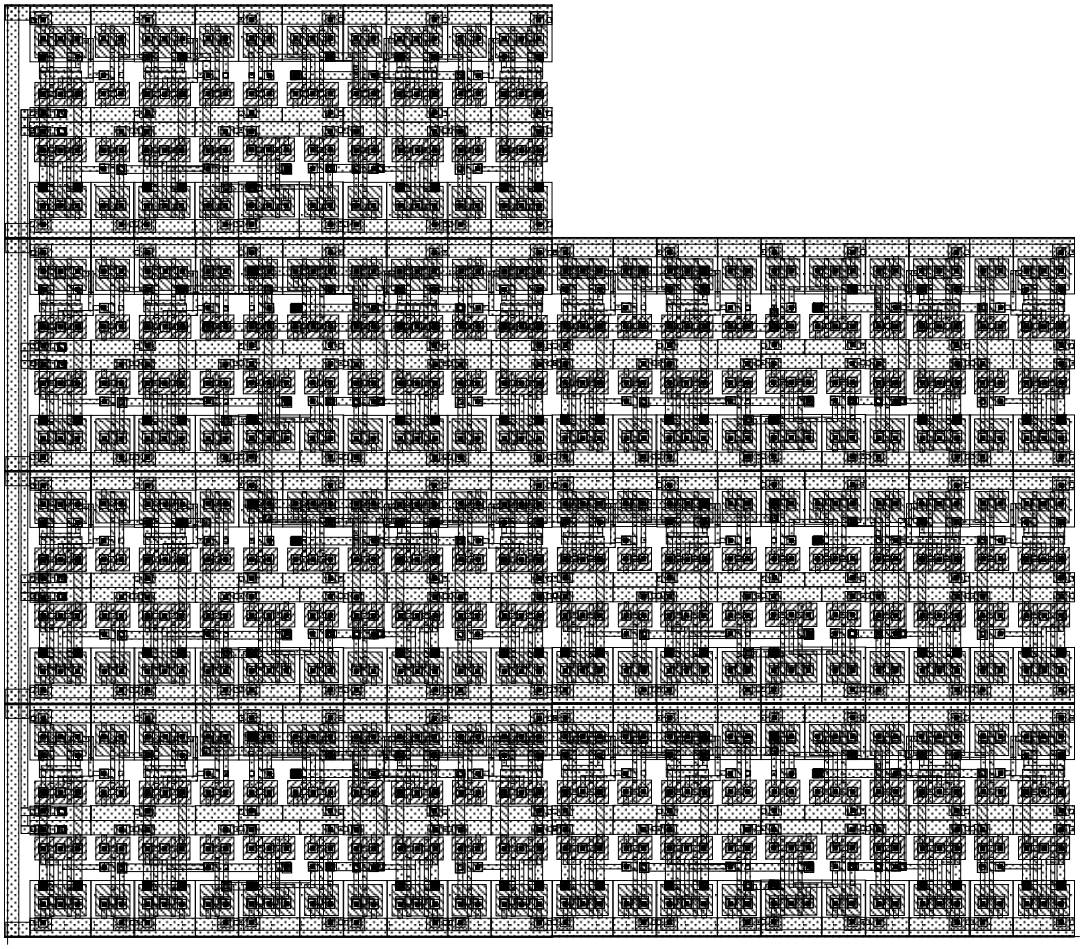

Figure A.2: Layout design of the 14-bit full adder

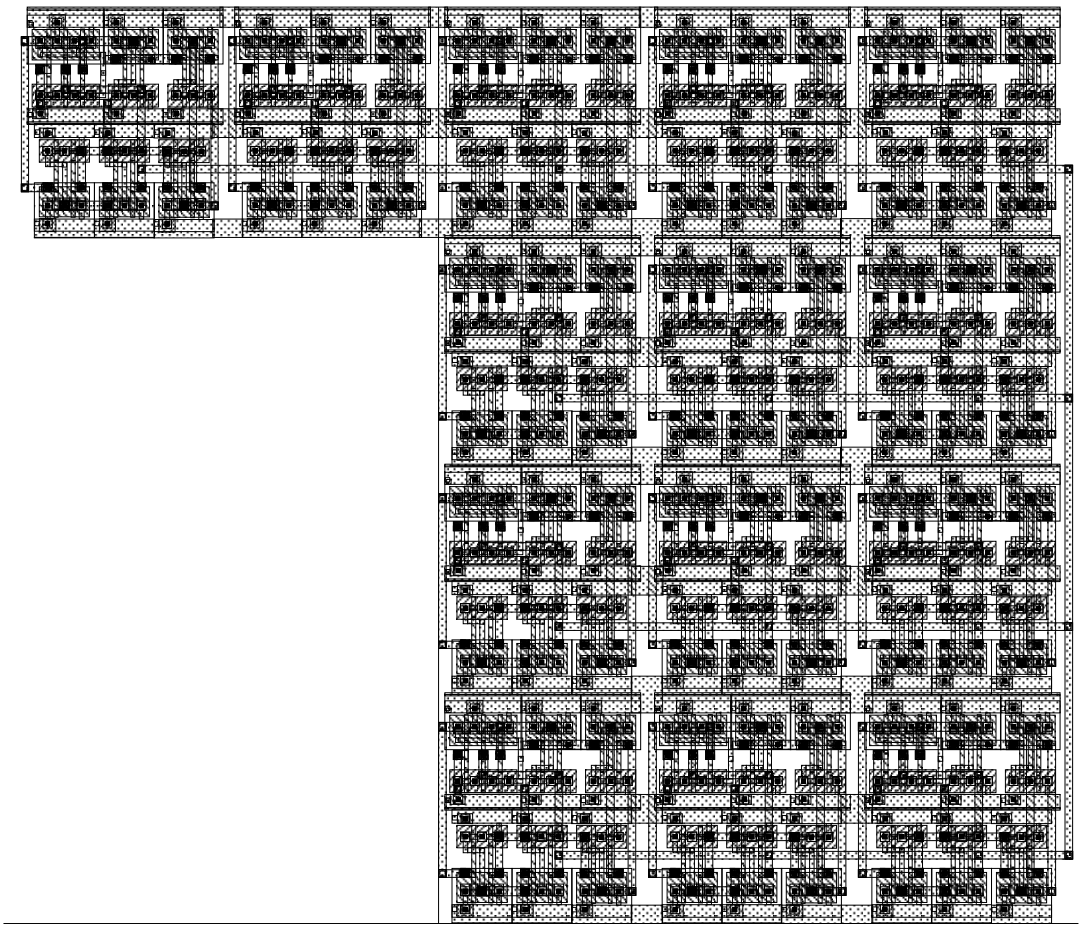

Figure A.3: Layout design of the 14-bit register

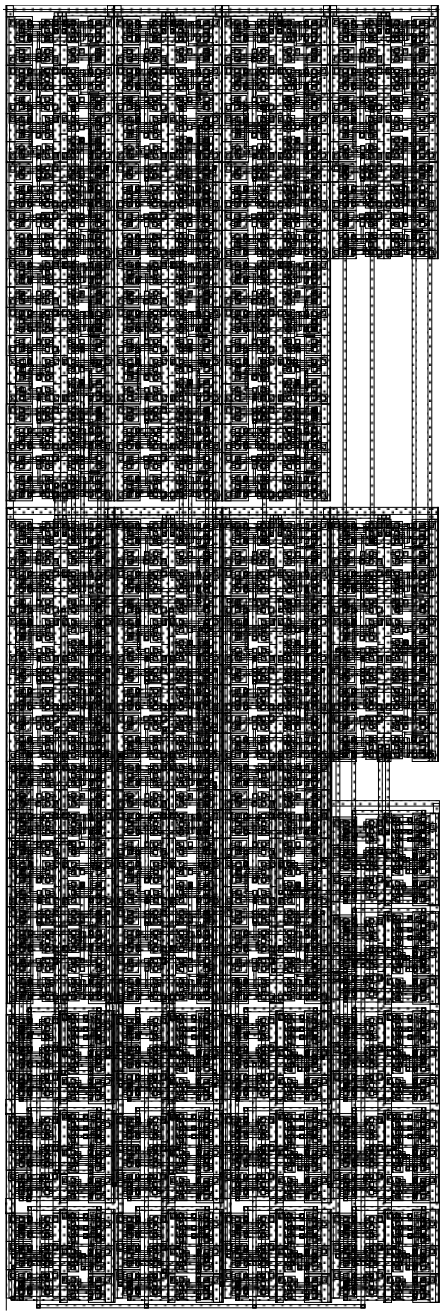

Figure A.4: Layout design of the delta sigma modulator

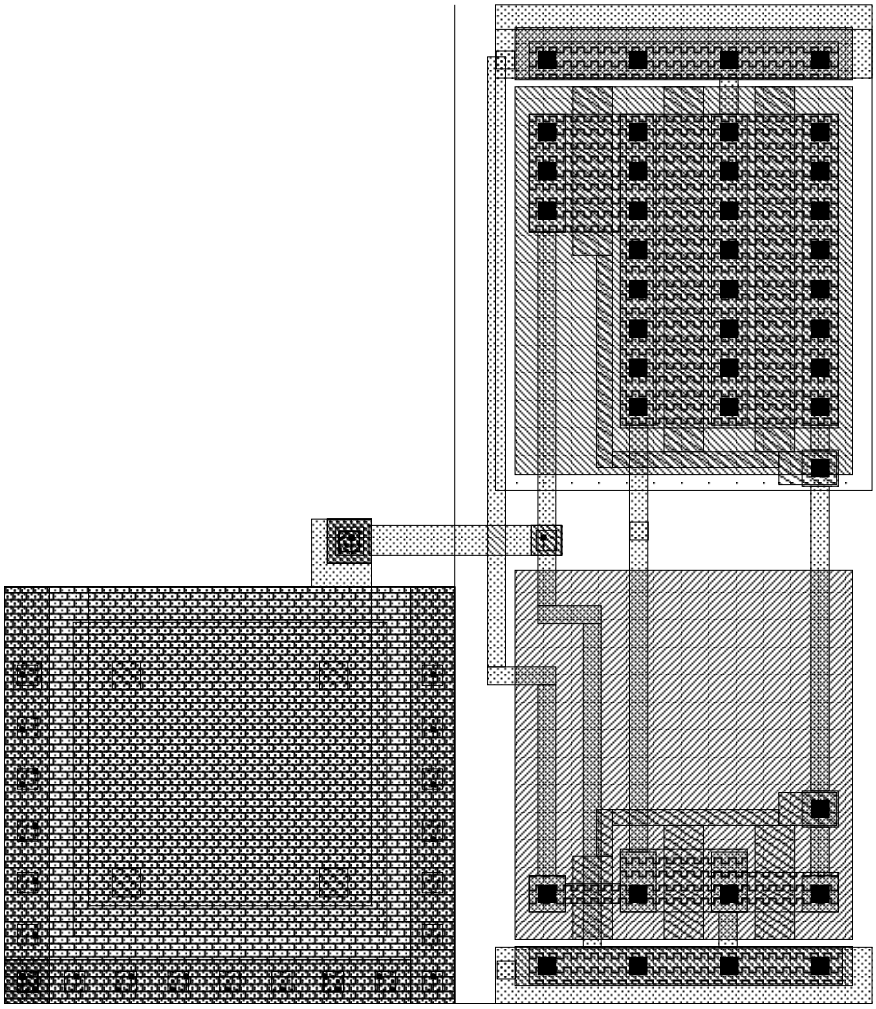

Figure A.5: Layout design of the neuron

Appendix B

Low Pass Filter

In this appendix, details on how the discrete-time transfer function are given of a typical low-pass filter is calculated. The transfer function of a low-pass filter with time constant RC iin Laplace domain is described as:

$$\frac{V_{out}(s)}{V_{in}(s)} = \frac{1}{1 + RCs} \tag{B.1}$$

By converting the Laplace domain to z-domain we have:

$$\frac{V_{out}(z)}{V_{in}(z)} = \frac{1}{1 + \left(\frac{z-1}{z+1}\right) RC} \tag{B.2}$$

$$\frac{V_{out}(z)}{V_{in}(z)} = \frac{z+1}{z+1+(z-1)RC} \tag{B.3}$$

After a multiplication through, we have:

$$V_{out}(z)\,(1 + RC)\,z + V_{out}(z)(1 - RC) = V_{in}(z)z + V_{in}(z) \tag{B.4}$$

By rearranging the above equation, we have:

$$V_{out}(z)\,(1+RC)\,z = V_{in}(z)z + V_{in}(z) - V_{out}(z)(1-RC) \tag{B.5}$$

$$V_{out}(z) = \frac{1}{1+RC}(V_{in}(z) + V_{in}(z)z^{-1}) - \frac{1-RC}{1+RC}V_{out}(z)z^{-1} \tag{B.6}$$

Hence, we have:

$$V_{out}(z) = \frac{1}{1+RC}(V_{in}(n) + V_{in}(n-1)) - \frac{1-RC}{1+RC}V_{out}(n-1) \tag{B.7}$$

The block diagram of Equation B.7 is shown in Figure B.1, where A is $\frac{1}{1+RC}$ and B is $\frac{1-RC}{1+RC}$.

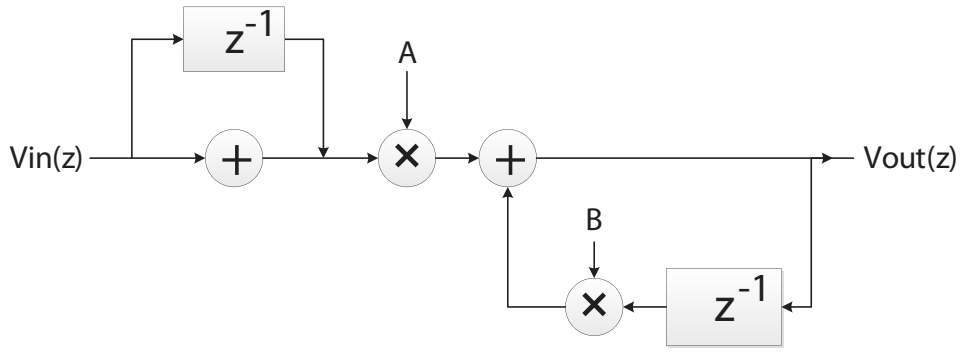

Figure B.1: Block diagram representation of discrete-time low pass filter

Appendix C

Delta Sigma Modulation in Matlab

This section includes Matlab codes used from the Delta Sigma Toolbox available at http://www.mathworks.com/matlabcentral/fileexchange/19. For further information regarding the toolbox and delta sigma refer to [77].

```
function ntf = synthesizeNTF(order,osr,opt,H_inf,f0)
%ntf = synthesizeNTF(order=3,osr=64,opt=0,H_inf=1.5,f0=0)
%Synthesize a noise transfer function for a delta-sigma
modulator.
%       order = order of the modulator
%       osr =   oversampling ratio
%       opt =   flag for optimized zeros
%               0 -> not optimized,
%               1 -> optimized,
%               2 -> optimized with at least one zero at
```

```
band-center
%               3 -> optimized zeros (Requires MATLAB6 and
Optimization Toolbox)
%       [z] -> zero locations in complex form
%       H_inf = maximum NTF gain
%       f0 =    center frequency (1->fs)
%
%ntf is a zpk object containing the zeros and poles of
the NTF. See zpk.m
%
% See also
%  clans()   "Closed-loop analysis of noise-shaper." An
alternative
%            method for selecting NTFs based on the 1-norm
of the
%            impulse response of the NTF
%
%  synthesizeChebyshevNTF()    Select a type-2 highpass
Chebyshev NTF.
%            This function does a better job than synthesizeNTF
 if osr
%            or H_inf is low.
% This is actually a wrapper function which calls either the
% appropriate version of synthesizeNTF based on the availability
% of the 'fmincon' function from the Optimization Toolbox
% Handle the input arguments
parameters = {'order' 'osr' 'opt' 'H_inf' 'f0'};
```

```
defaults = { 3 64 0 1.5 0 };
for arg_i=1:length(defaults)
    parameter = char(parameters(arg_i));
    if arg_i>nargin|( eval(['isnumeric('parameter')']) &  ...
     eval(['any(isnan('parameter'))|isempty('parameter')']))
        eval([parameter '=defaults{arg_i};'])
    end
end
if f0 > 0.5
    fprintf(1,'Error. f0 must be less than 0.5.\n');
    return;
end
if f0 ~= 0 & f0 < 0.25/osr
    warning('(%s) Creating a lowpass ntf.', mfilename);
    f0 = 0;
end
if f0 ~= 0 & rem(order,2) ~= 0
    fprintf(1,'Error. order must be even for a bandpass
    modulator.\n');
    return;
end
if length(opt)>1 & length(opt)~=order
    fprintf(1,'The opt vector must be of length %d(=order).\n',
     order);
    return;
end
if exist('fmincon','file')
```

```
    ntf = synthesizeNTF1(order,osr,opt,H_inf,f0);
else
    ntf = synthesizeNTF0(order,osr,opt,H_inf,f0);
end

function [v,xn,xmax,y] = simulateDSM(u,arg2,nlev,x0)
%[v,xn,xmax,y] = simulateDSM(u,ABCD,nlev=2,x0=0)
% or
%[v,xn,xmax,y] = simulateDSM(u,ntf,nlev=2,x0=0)
%
%Compute the output of a general delta-sigma modulator with
input u,
%a structure described by ABCD, an initial state x0 (default
zero) and
%a quantizer with a number of levels specified by nlev.
%Multiple quantizers are implied by making nlev an array,
%and multiple inputs are implied by the number of rows in u.
%
%Alternatively, the modulator may be described by an NTF.
%The NTF is zpk object. (The STF is assumed to be 1.)
%The structure that is simulated is the block-diagional
structure used by
%zp2ss.m.

fprintf(1,'Warning: You are running the non-mex version of
simulateDSM.\n');
fprintf(1,'Please compile the mex version with
"mex simulateDSM.c"\n');
```

```
if nargin<2
    fprintf(1,'Error. simulateDSM needs at least two
    arguments.\n');
    return
end

% Handle the input arguments
parameters = {'u','arg2','nlev','x0'};
defaults = [ NaN NaN 2 NaN ];
for i=1:length(defaults)
    parameter = char(parameters(i));
    if i>nargin|( eval(['isnumeric('parameter')']) & ...
     eval(['any(isnan('parameter'))|isempty('parameter')']))
        eval([parameter '=defaults(i);'])
    end
end
nu = size(u,1);
nq = length(nlev);
if isobject(arg2) & strcmp(class(arg2),'zpk')
    ntf.k = arg2.k;
    ntf.zeros = arg2.z{:};
    ntf.poles = arg2.p{:};
    form = 2;
    order = length(ntf.zeros);
elseif isstruct(arg2)
    if any(strcmp(fieldnames(arg2),'zeros'))
```

```
        ntf.zeros = arg2.zeros;
    else
        error('No zeros field in the NTF.')
    end
    if any(strcmp(fieldnames(arg2),'poles'))
        ntf.poles = arg2.poles;
    else
        error('No poles field in the NTF.')
    end
    form = 2;
    order = length(ntf.zeros);
elseif isnumeric(arg2)
    if size(arg2,2) > 2 & size(arg2,2)==nu+size(arg2,1) %
    ABCD dimesions OK
        form = 1;
        ABCD = arg2;
        order = size(ABCD,1)-nq;
    else
        fprintf(1,'The ABCD argument does not have proper dimensions.\
        if size(arg2,2) == 2            % Probably old (ver. 2) ntf fo
            fprintf(1,'You appear to be using the old-style form of N
            specification.\n Automatic converstion to the new form wi
            done for this release only.\n');
            ntf.zeros = arg2(:,1);
            ntf.poles = arg2(:,2);
            form = 2;
            order = length(ntf.zeros);
```

```
        else
            error('Exiting simulateDSM.')
        end
    end
else
    error('The second argument is neither an ABCD matrix nor
     an NTF.\n');
end
if isnan(x0)
    x0 = zeros(order,1);
end

if form==1
    A = ABCD(1:order, 1:order);
    B = ABCD(1:order, order+1:order+nu+nq);
    C = ABCD(order+1:order+nq, 1:order);
    D1= ABCD(order+1:order+nq, order+1:order+nu);
else
    [A,B2,C,D2] = zp2ss(ntf.poles,ntf.zeros,-1);
    % A realization of 1/H
    % Transform the realization so that C = [1 0 0 ...]
    Sinv = orth([C' eye(order)])/norm(C); S = inv(Sinv);
    C = C*Sinv;
    if C(1)<0
        S = -S;
        Sinv = -Sinv;
    end
```

```
    A = S*A*Sinv; B2 = S*B2; C = [1 zeros(1,order-1)];
    % C=C*Sinv;
    D2 = 0;
    % !!!! Assume stf=1
    B1 = -B2;
    D1 = 1;
    B = [B1 B2];
end

N = length(u);
v = zeros(nq,N);
y = zeros(nq,N);
if nargout > 1  % Need to store the state information
    xn = zeros(order,N);
end
if nargout > 2  % Need to keep track of the state maxima
    xmax = abs(x0);
end

for i=1:N
    y(:,i) = C*x0 + D1*u(:,i);
    v(:,i) = ds_quantize(y(:,i),nlev);
    x0 = A * x0 + B * [u(:,i);v(:,i)];
    if nargout > 1      % Save the next state
        xn(:,i) = x0;
    end
    if nargout > 2      % Keep track of the state maxima
```

```
        xmax = max(abs(x0),xmax);
    end
end
return

function v = ds_quantize(y,n)
%v = ds_quantize(y,n)
%Quantize y to
% an odd integer in [-n+1, n-1], if n is even, or
% an even integer in [-n, n], if n is odd.
%
%This definition gives the same step height for both mid-rise
%and mid-tread quantizers.

if rem(n,2)==0  % mid-rise quantizer
    v = 2*floor(0.5*y)+1;
else            % mid-tread quantizer
    v = 2*floor(0.5*(y+1));
end

% Limit the output
for qi=1:length(n)      % Loop for multiple quantizers
    L = n(qi)-1;
    i = v(qi,:)>L;
    if any(i)
        v(qi,i) = L;
    end
```

```
    i = v(qi,:)<-L;
    if any(i)
        v(qi,i) = -L;
    end
end
```

Appendix D

Proposed Delta Sigma Modulator Described in Verilog

In this section the Verilog code describing proposed delta sigma modulator implemented using CMOS 180nm library cells is presented.

```
module DelSigDAC ( DACout, DACin, clk, reset, VDD, VSS );
  input [3:0] DACin;
  input clk, reset;
  input VDD;
  input VSS;
  supply1 VDD;
  supply0 VSS;
  output DACout;
```

```
wire    n6, n7, n8, n9, n10, n11, n12, n13, n14, n15, n16;
wire    n17, n18, n19, n20;
wire    [4:0] SigmaLatch;
wire    [5:0] DeltaAdder;
wire    [5:0] SigmaAdder;
assign DeltaAdder[3] = DACin[3];
assign DeltaAdder[2] = DACin[2];
assign DeltaAdder[1] = DACin[1];
assign DeltaAdder[0] = DACin[0];

DFFSX1 \SigmaLatch_reg[3]   ( .D(SigmaAdder[3]), .CK(clk),
.SN(n20), .QN(n6), .VDD(VDD), .VSS(VSS));
DFFRHQX1 \SigmaLatch_reg[0]   ( .D(SigmaAdder[0]), .CK(clk),
.RN(n20), .Q(SigmaLatch[0]), .VDD(VDD), .VSS(VSS) );
DFFRHQX1 \SigmaLatch_reg[1]   ( .D(SigmaAdder[1]), .CK(clk),
.RN(n20), .Q(SigmaLatch[1]), .VDD(VDD), .VSS(VSS) );
DFFRHQX1 \SigmaLatch_reg[2]   ( .D(SigmaAdder[2]), .CK(clk),
.RN(n20), .Q(SigmaLatch[2]), .VDD(VDD), .VSS(VSS) );
DFFRHQX1 \SigmaLatch_reg[4]   ( .D(SigmaAdder[4]), .CK(clk),
.RN(n20), .Q(SigmaLatch[4]), .VDD(VDD), .VSS(VSS) );
DFFRHQX1 \SigmaLatch_reg[5]   ( .D(SigmaAdder[5]), .CK(clk),
.RN(n20), .Q(DeltaAdder[4]), .VDD(VDD), .VSS(VSS) );
DFFRHQX1 DACout_reg ( .D(DeltaAdder[4]), .CK(clk), .RN(n20),
.Q(DACout), .VDD(VDD), .VSS(VSS) );
INVX1 U6 ( .A(reset), .Y(n20), .VDD(VDD), .VSS(VSS) );
OAI21XL U7 ( .A0(n7), .A1(n8), .B0(n9), .Y(SigmaAdder[5]),
  .VDD(VDD), .VSS(VSS) );
```

```
OAI2BB1X1 U8 ( .A0N(n8), .A1N(n7), .B0(SigmaLatch[4]),
  .Y(n9), .VDD(VDD), .VSS(VSS) );
INVX1 U9 ( .A(DeltaAdder[4]), .Y(n8), .VDD(VDD), .VSS(VSS) );
XOR2X1 U10 ( .A(n10), .B(n7), .Y(SigmaAdder[4]), .VDD(VDD),
.VSS(VSS) );
AOI21X1 U11 ( .A0(n11), .A1(DeltaAdder[3]), .B0(n12), .Y(n7),
.VDD(VDD), .VSS(VSS) );
AOI2BB1X1 U12 ( .A0N(DeltaAdder[3]), .A1N(n11), .B0(n6),
.Y(n12), .VDD(VDD), .VSS(VSS) );
XNOR2X1 U13 ( .A(DeltaAdder[4]), .B(SigmaLatch[4]), .Y(n10),
.VDD(VDD), .VSS(VSS) );
XNOR2X1 U14 ( .A(n11), .B(n13), .Y(SigmaAdder[3]), .VDD(VDD),
.VSS(VSS) );
XOR2X1 U15 ( .A(n6), .B(DeltaAdder[3]), .Y(n13), .VDD(VDD),
.VSS(VSS) );
OAI2BB1X1 U16 ( .A0N(n14), .A1N(DeltaAdder[2]), .B0(n15),
.Y(n11), .VDD(VDD), .VSS(VSS) );
OAI21XL U17 ( .A0(DeltaAdder[2]), .A1(n14), .B0(SigmaLatch[2]),
.Y(n15), .VDD(VDD), .VSS(VSS) );
XNOR2X1 U18 ( .A(n16), .B(n14), .Y(SigmaAdder[2]), .VDD(VDD),
.VSS(VSS) );
OAI2BB1X1 U19 ( .A0N(n17), .A1N(DeltaAdder[1]), .B0(n18),
.Y(n14), .VDD(VDD), .VSS(VSS) );
OAI21XL U20 ( .A0(n17), .A1(DeltaAdder[1]), .B0(SigmaLatch[1]),
.Y(n18), .VDD(VDD), .VSS(VSS) );
XNOR2X1 U21 ( .A(DeltaAdder[2]), .B(SigmaLatch[2]), .Y(n16),
.VDD(VDD), .VSS(VSS) );
```

```
  XOR2X1 U22 ( .A(n17), .B(n19), .Y(SigmaAdder[1]), .VDD(VDD),
  .VSS(VSS) );
  XOR2X1 U23 ( .A(SigmaLatch[1]), .B(DeltaAdder[1]), .Y(n19),
  .VDD(VDD), .VSS(VSS) );
  AND2X1 U24 ( .A(SigmaLatch[0]), .B(DeltaAdder[0]), .Y(n17),
  .VDD(VDD), .VSS(VSS) );
  XOR2X1 U25 ( .A(SigmaLatch[0]), .B(DeltaAdder[0]),
  .Y(SigmaAdder[0]),
  .VDD(VDD), .VSS(VSS) );
endmodule
```

References

[1] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of mathematical biology*, 5(4):115–133, 1943.

[2] B. Widrow, M.E. Hoff, et al. Adaptive switching circuits. 1960.

[3] M. Minsky and P. Seymour. Perceptrons. 1969.

[4] P. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

[5] B. Widrow and M.A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

[6] M. Jabri and B. Flower. Weight perturbation: An optimal architecture and learning technique for analog vlsi feedforward and recurrent multilayer networks. *Neural Networks, IEEE Transactions on*, 3(1):154–157, 1992.

[7] D. Psaltis and Y. Qiao. Adaptive multilayer optical networks. *Prog. Opt*, 31:227–261, 1993.

[8] RD Brandt and F. Lin. Supervised learning in neural networks without explicit error back-propagation. In *Proceeding of the Annual Allerton Conference on Communication Control and Computing*, volume 32, pages 294–294. UNIVERSITY OF ILLINOIS, 1994.

[9] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi. A prototype cvns distributed neural network using synapse-neuron modules. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(7):1482 –1490, july 2012.

[10] M. Jabri and B. Flower. Weight perturbation: An optimal architecure and learning technique for analog feed forward and recurrent multiplayer networks. *Neural Networks, IEEE Transactions on*, 3(1), 1992.

[11] V.F. Koosh and R.M. Goodman. Analog vlsi neural network with digital perturbative learning. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 49(5):359–368, 2002.

[12] D. Maliuk, H.G. Stratigopoulos, and Y. Makris. An analog vlsi multilayer perceptron and its application towards built-in self-test in analog circuits. In *On-Line Testing Symposium (IOLTS), 2010 IEEE 16th International*, pages 71–76. IEEE, 2010.

[13] G. Zatorre-Navarro, N. Medrano-Marqués, and S. Celma-Pueyo. Analysis and simulation of a mixed-mode neuron architecture for sensor conditioning. *Neural Networks, IEEE Transactions on*, 17(5):1332–1335, 2006.

[14] D. Kim, H. Kim, H. Kim, G. Han, and D. Chung. A simd neural network processor for image processing. *Advances in Neural Networks–ISNN 2005*, pages 815–815, 2005.

[15] A. Schmid, Y. Leblebici, and D. Mlynek. Mixed analogue-digital artificial-neural-network architecture with on-chip learning. In *Circuits, Devices and Systems, IEE Proceedings-*, volume 146, pages 345–349. IET, 1999.

[16] A. Tisan, M. Cirstea, S. Oniga, and A. Buchman. Artificial olfaction system with hardware on-chip learning neural networks. In *Optimization of Electrical and Electronic Equipment (OPTIM), 2010 12th International Conference on*, pages 884–889. IEEE, 2010.

[17] S. Jung and S. su Kim. Hardware implementation of a real-time neural network controller with a dsp and an fpga for nonlinear systems. *Industrial Electronics, IEEE Transactions on*, 54(1):265–271, 2007.

[18] L. Tarassenko, J. Tombs, and G. Cairns. On-chip learning with analogue vlsi neural networks. *International journal of neural systems*, 4(04):419–426, 1993.

[19] Y. Xie and M.A. Jabri. Training algorithms for limited precision feedforward neural networks. *Technical Rep*, (1991-8):3, 1991.

[20] N. Rochester, J. Holland, L. Haibt, and W. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *Information Theory, IRE Transactions on*, 2(3):80–93, 1956.

[21] K. Boahen. Neuromorphic microchips. *Scientific American*, 292(5):56–63, 2005.

[22] K.A. Zaghloul and K. Boahen. A silicon retina that reproduces signals in the optic nerve. *Journal of neural engineering*, 3(4):257, 2006.

[23] K. Boahen. Neurogrid: emulating a million neurons in the cortex. In *IEEE international conference of the engineering in medicine and biology society*, 2006.

[24] C. Lu, B.X. Shi, and L. Chen. An on-chip bp learning neural network with ideal neuron characteristics and learning rate adaptation. *Analog Integrated Circuits and Signal Processing*, 31(1):55–62, 2002.

[25] L. Gatet, H. Tap-Béteille, and M. Lescure. Real-time surface discrimination using an analog neural network implemented in a phase-shift laser rangefinder. *Sensors Journal, IEEE*, 7(10):1381–1387, 2007.

[26] R. Perfetti and E. Ricci. Analog neural network for support vector machine learning. *Neural Networks, IEEE Transactions on*, 17(4):1085–1091, 2006.

[27] B.W. Lee and BJ Sheu. General-purpose neural chips with electrically programmable synapses and gain-adjustable neurons. *Solid-State Circuits, IEEE Journal of*, 27(9):1299–1302, 1992.

[28] S.M. Gowda, B.J. Sheu, J. Choi, C.G. Hwang, and J.S. Cable. Design and characterization of analog vlsi neural network modules. *Solid-State Circuits, IEEE Journal of*, 28(3):301–313, 1993.

[29] S. Bettola and V. Piuri. High performance fault-tolerant digital neural networks. *Computers, IEEE Transactions on*, 47(3):357–363, 1998.

[30] D. Zhang and MI Elmasry. Vlsi compressor design with applications to digital neural networks. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(2):230–233, 1997.

[31] K. Basterretxea, JM Tarela, and I. Del Campo. Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons. In *Circuits, Devices and Systems, IEE Proceedings-*, volume 151, pages 18–24. IET, 2004.

[32] K. Leboeuf, A.H. Namin, R. Muscedere, H. Wu, and M. Ahmadi. High speed vlsi implementation of the hyperbolic tangent sigmoid function. In *Convergence and Hybrid Information Technology, 2008. ICCIT'08. Third International Conference on*, volume 1, pages 1070–1073. IEEE, 2008.

[33] P.K. Meher. An optimized lookup-table for the evaluation of sigmoid function for artificial neural networks. In *VLSI System on Chip Conference (VLSI-SoC), 2010 18th IEEE/IFIP*, pages 91–95. IEEE, 2010.

[34] A.H. Namin, K. Leboeuf, R. Muscedere, H. Wu, and M. Ahmadi. Efficient hardware implementation of the hyperbolic tangent sigmoid function. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 2117–2120. IEEE, 2009.

[35] H. Djahanshahi, M. Ahmadi, G.A. Jullien, and W.C. Miller. Design and vlsi implementation of a unified synapse-neuron architecture. In *VLSI, 1996. Proceedings., Sixth Great Lakes Symposium on*, pages 228 –233, mar 1996.

[36] M. Mirhassani, M. Ahmadi, and W.C. Miller. A new mixed-signal feed-forward neural network with on-chip learning. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 3, pages 1729 – 1734 vol.3, july 2004.

[37] G. Khodabandehloo, M. Mirhassani, and M. Ahmadi. Resistive-type cvns distributed neural networks with improved noise-to-signal ratio. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(10):793–797, 2010.

[38] A.F. Murray, D. Del Corso, and L. Tarassenko. Pulse-stream vlsi neural networks mixing analog and digital techniques. *Neural Networks, IEEE Transactions on*, 2(2):193 –204, mar 1991.

[39] A.F. Murray and A.V.W. Smith. Asynchronous arithmetic for vlsi neural systems. *Electronics Letters*, 23(12):642 –643, 4 1987.

[40] Alan F. Murray and Anthony V. W. Smith. A novel computational and signalling method for vlsi neural networks. In *Solid-state Circuits Conference, 1987. ESSCIRC '87. 13th European*, pages 19 –22, sept. 1987.

[41] L.M. Reyneri. A performance analysis of pulse stream neural and fuzzy computing systems. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 42(10):642 –660, oct 1995.

[42] M. Chiaberge, E.M. Sologuren, and L.M. Reyneri. A pulse stream system for low-power neuro-fuzzy computation. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 42(11):946 –954, nov 1995.

[43] A.F. Murray and L. Tarassenko. *Analogue neural VLSI: a pulse stream approach.* Chapman & Hall, Ltd., 1994.

[44] K. Boahen. A throughput-on-demand address-event transmitter for neuromorphic chips. In *Advanced Research in VLSI, 1999. Proceedings. 20th Anniversary Conference on*, pages 72–86. IEEE, 1999.

[45] A.G. Andreou and KA Boahen. Neural information processing ii. *Analog VLSI signal and information processing*, pages 358–409, 1994.

[46] A. Mortara and E.A. Vittoz. A communication architecture tailored for analog vlsi artificial neural networks: intrinsic performance and limitations. *Neural Networks, IEEE Transactions on*, 5(3):459–466, 1994.

[47] W. Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[48] B. Ruf and M. Schmitt. Self-organization of spiking neurons using action potential timing. *Neural Networks, IEEE Transactions on*, 9(3):575–578, 1998.

[49] S. Wolpert and E. Micheli-Tzanakou. A neuromime in vlsi. *Neural Networks, IEEE Transactions on*, 7(2):300–306, 1996.

[50] A.M. Chiang and M.L. Chuang. A ccd programmable image processor and its neural network applications. *Solid-State Circuits, IEEE Journal of*, 26(12):1894–1901, 1991.

[51] R.R. Harrison. A low-power analog vlsi visual collision detector. *Advances in Neural Information Processing Systems*, 16, 2003.

[52] M. Milev and M. Hristov. Analog implementation of ann with inherent quadratic nonlinearity of the synapses. *Neural Networks, IEEE Transactions on*, 14(5):1187–1200, 2003.

[53] J. Liu, M.A. Brooke, and K. Hirotsu. A cmos feedforward neural-network chip with on-chip parallel learning for oscillation cancellation. *Neural Networks, IEEE Transactions on*, 13(5):1178–1186, 2002.

[54] T. Schoenauer, S. Atasoy, N. Mehrtash, and H. Klar. Neuropipe-chip: A digital neuro-processor for spiking neural networks. *Neural Networks, IEEE Transactions on*, 13(1):205–213, 2002.

[55] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel. Evolutionary bitsnspikes. In *Artificial Life VIII. Proceedings of the Eighth International Conference on Artificial Life. MIT Press, Cambridge, MA*, 2002.

[56] L.M. Reyneri, M. Chiaberge, and L. Zocca. Cintia: A neuro-fuzzy real time controller for low power embedded systems. In *Microelectronics for Neural Networks and Fuzzy Systems, 1994., Proceedings of the Fourth International Conference on*, pages 392–403. IEEE, 1994.

[57] J. Wang. Analogue neural network for solving the assignment problem. *Electronics Letters*, 28(11):1047–1050, 1992.

[58] A. Rodríguez-Vázquez, G. Liñán-Cembrano, L. Carranza, E. Roca-Moreno, R. Carmona-Galán, F. Jiménez-Garrido, R. Domínguez-Castro, and S.E. Meana. Ace16k: the third generation of mixed-signal simd-cnn ace chips toward vsocs. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 51(5):851–863, 2004.

[59] Á. Zarándy and C. Rekeczky. Bi-i: a standalone ultra high speed cellular vision system. *Circuits and Systems Magazine, IEEE*, 5(2):36–45, 2005.

[60] P. Arena, L. Fortuna, M. Frasca, and L. Patané. A cnn-based chip for robot locomotion control. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 52(9):1862–1871, 2005.

[61] P. Arena, L. Fortuna, M. Frasca, L. Patane, and M. Pollino. An autonomous mini-hexapod robot controlled through a cnn-based cpg vlsi chip. In *Cellular Neural Networks and Their Applications, 2006. CNNA'06. 10th International Workshop on*, pages 1–6. IEEE, 2006.

[62] JC Lopez-Garcia, M.A. Moreno-Armendáriz, J. Riera-Baburés, M. Balsi, and X. Vilasis-Cardona. Real time vision by fpga implemented cnns. In *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, volume 1, pages I–281. IEEE, 2005.

[63] D.L. Wang and D. Terman. Image segmentation based on oscillatory correlation. *Neural Computation*, 9(4):805–836, 1997.

[64] S. Bellis, KM Razeeb, C. Saha, K. Delaney, C. O'Mathuna, A. Pounds-Cornish, G. De Souza, M. Colley, H. Hagras, G. Clarke, et al. Fpga implementation of spiking neural networks-an initial step towards building tangible collaborative autonomous agents. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 449–452. IEEE, 2004.

[65] M. Krips, T. Lammert, and A. Kummert. Fpga implementation of a neural network for a real-time hand tracking system. In *Electronic Design, Test and Applications, 2002. Proceedings. the First IEEE International Workshop on*, pages 313–317. IEEE, 2002.

[66] N. Yazdi, M. Ahmadi, G.A. Jullien, and M. Shridhar. Pipelined analog multilayer feedforward neural networks. In *Circuits and Systems, 1993., ISCAS '93, 1993 IEEE International Symposium on*, pages 2768 –2771 vol.4, may 1993.

[67] A. Nosratinia, M. Ahmadi, M. Shridhar, and G.A. Jullien. A hybrid architecture for feed-forward multi-layer neural networks. In *Circuits and Systems, 1992. ISCAS '92. Proceedings., 1992 IEEE International Symposium on*, volume 3, pages 1541 –1544 vol.3, may 1992.

[68] S.W. Piche. The selection of weight accuracies for madalines. *Neural Networks, IEEE Transactions on*, 6(2):432–445, 1995.

[69] W. Maass and C.M. Bishop. *Pulsed neural networks*. MIT press, 2001.

[70] B. Gilbert. A high-performance monolithic multiplier using active feedback. *Solid-State Circuits, IEEE Journal of*, 9(6):364–373, 1974.

[71] B. Gilbert. A precise four-quadrant multiplier with subnanosecond response. *Solid-State Circuits, IEEE Journal of*, 3(4):365 –373, dec. 1968.

[72] S. Satyanarayana, Y.P. Tsividis, and H.P. Graf. A reconfigurable vlsi neural network. *Solid-State Circuits, IEEE Journal of*, 27(1):67–81, 1992.

[73] H. Djahanshahi, M. Ahmadi, G.A. Jullien, and W.C. Miller. Quantization noise improvement in a hybrid distributed-neuron ann architecture. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 48(9):842–846, 2001.

[74] Hong-Kui Yang and E.I. El-Masry. A cmos current-mode pwm technique for analog neural network implementations. In *Circuits and Systems, 1994. ISCAS '94., 1994 IEEE International Symposium on*, volume 6, pages 355 –358 vol.6, may-2 jun 1994.

[75] Lu Chen and Bingxue Shi. Cmos pwm vlsi implementation of neural network. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 485 –488 vol.3, 2000.

[76] L. Gatet, H. Tap-Béteille, and F. Bony. Comparison between analog and digital neural network implementations for range-finding applications. *Neural Networks, IEEE Transactions on*, 20(3):460–470, 2009.

[77] R. Schreier and G.C. Temes. *Understanding delta-sigma data converters*, volume 74. IEEE press Piscataway, NJ, USA, 2005.

Vita Auctoris

Farinoush Saffar, was born in 1989, in Tehran, Iran. She received her Bachelor of Science degree in Electrical Engineering in 2010 from Shahid Beheshti University, Tehran, Iran majoring in Electronics. She moved to Canada and attended University of Windsor in January 2011 where she completed her Master of Applied Science in Electrical Engineering in December 2012.