A Study on Hardware Implementations of Artificial Neural Networks

Ganesh Vhatkar
School of Electrical and Computer Engineering
Oregon State University
vhatkarg@oregonstate.edu

Abstract— **Artificial Neural Networks (ANN) are computing systems which are inspired by the biological neural networks that constitutes a human brain. These systems operate by learning or progressively improving the performance of the tasks [5]. ANN have long been used to solve complex Machine Learning problems and are also used for Deep Learning. The neuron-synapse pair is the basic element of ANN. The neuron are the processing elements while the synapses constitutes the wiring of multiple neurons. ANN can be implemented using software as well as hardware. However, hardware implementation allows large gains when scaling the network sizes. This paper talks about various types of hardware implementations of ANN. One of them is implementing the ANN using Field Programmable Gate Arrays (FPGA). Other involves CMOS implementation using digital or analog circuitry. The paper also highlights the limitations of each implementation technique in short and introduces a new techniques which uses memristor and Processing In-Memory for ANNs.**

I. INTRODUCTION

Artificial Neural Networks (ANNs) have found tremendous and widespread categorization in a broad spectrum of perception, association, and control applications. As the modern-day computing is becoming advance with the high-speed computing, there is an aspiration to build intelligent systems that can learn on itself by progressive realization and execution. Artificial Neural Networks (ANNs) consists of basic essential building blocks or elements like the neurons and the synapses. A Neuro-computing system is made up of several artificial neurons and a huge number of interconnections between them. The artificial neuron given in Figure 1. has N inputs which are denoted by x1, x2,.xN,. The lines, each of which connects these inputs to the neuron is assigned a weight, denoted as w1, w2,wN respectively[2]. As the ANNs scale in size, the number of synapses grows quadratically for fully connected networks, which becomes impractical to wire [3]. The activation function is another important element that determines whether a neuron is to be fired or not. Different methods of implementing the activation functions involve linear, ramp, and sigmoid functions which are used respectively depending on the situations.

The output, y of the neuron is given as:

y = f (a)

a = (wj xj)

Recently, there has been a large push towards the hardware implementation of the ANNs to overcome the calculation complexity of software implementations. This has led to a new class of circuits, called the neuromorphic circuits, that imitate the behavior of neurons. These circuits provide advantages like high network connectivity, simple base processing element, and distributed memory and computation.Fig 1 shows the structure of a Neuron.

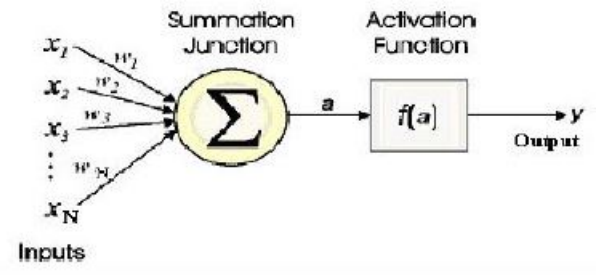

Fig. 1. Structural diagram of a Neuron[2]

II. TYPES OF ARTIFICIAL NEURAL NETWORKS

ANNs can be categorized into two main classes, Feed-Forward Networks and Recurrent Networks. In the former, the computation is performed in a layer-by-layer fashion from the input to the output of the network. The applications include supervised classification performed by a perceptron algorithm. The later have an interconnect network structure including cycles which are more diverse, and applications include self-organizing maps, associative memory, Boltzmann networks, etc. Also, artificial neurons can also be used to implement digital-like logic using spikes, and therefore reproduce a universal Turing machine [3].Fig. 2 shows a Feed-Forward Neural Network.

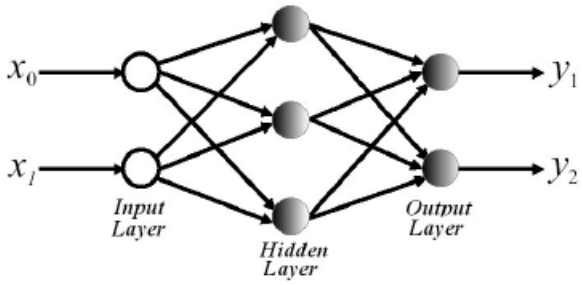

Fig. 2. Feed-Forward Neural Network [2]

III. Hardware Implementations of ANNs using FPGA

Hardware realization of a Neural Network (NN), to a large extent depends on the efficient implementation of a single neuron. In this study, a hardware implementation of a fully interconnected feed forward back-propagation artificial network using Xilinx FPGAs is presented. Each node or neuron is implemented with two XC3042 FPGAs and a 1K x 8 EPROM [1]. The ANN deployed here is a three-layer network. The network consists of an input layer with five nodes, a single hidden layer with four nodes, and an output layer with two nodes. These nodes are fully interconnected to each other between adjacent layers [1]. Fig.3. shows a ANN. The input to this network is a continuous valued vector x1, x2,...x5. The output of the node j, y_j, is computed as follows:

yj = f [(wj yi) - θj]

where,

θ = offset (bias) of j

Wij = weight of the connection between node j and i

f = sigmoid non-linearity

Training is done off-line on a conventional digital computer where the final values of the weights are obtained. It is done by off-line simulation of the network on a PC [1].The training of the network is given below:

a) Initialize the weights and offsets. Every node excluding the input node is assigned to an initial offset as the input nodes act as buffers.

b) Tracing the path from the output layer and heading back to the input layer, the weights and the offsets are adjusted recursively until they are stabilized. This is done with the help of the following equations:

W_{ijnew} = W_{ijold} + u ϵ_j x_i

j_{new} = j_{old} . 1

where,

u = gain factor (presumed 0.5)

ϵ_j = error

The weights are stabilized if the value of each new weight is 95 % of its old value. The weights in the other layers first form a weighted sum of their inputs. The input is the 8-bit which is applied to the hidden and the output nodes y_j is multiplied by the 8-bit signed weights Wij to form a 16-bit signed product. The products and a 16-bit signed bias value theta j are accumulated into a 20-bit sum stored in the accumulator. This 20-bit sum is then scaled down to a 10-bit value. This value is selected because it is the minimum number of bits that is retained without deteriorating the accuracy of the sum and it is the address to the 1K x 8 EPROM where the sigmoid activation function f is realized as a look-up table. The activation function produces an 8-bit output [1]. Fig.4 and Fig.5 shows general architecture of the network and schematic of digital neuron.

The system clock is running at 4 MHz which is the maximum speed that could be achieved due to the use of slower EPROMS and two FPGAs per node. A micro programmed controller drives the entire network. The first FPGAs carries input latches and multipliers. The second carries a 20-bit fast adder/accumulator circuit and a scaling

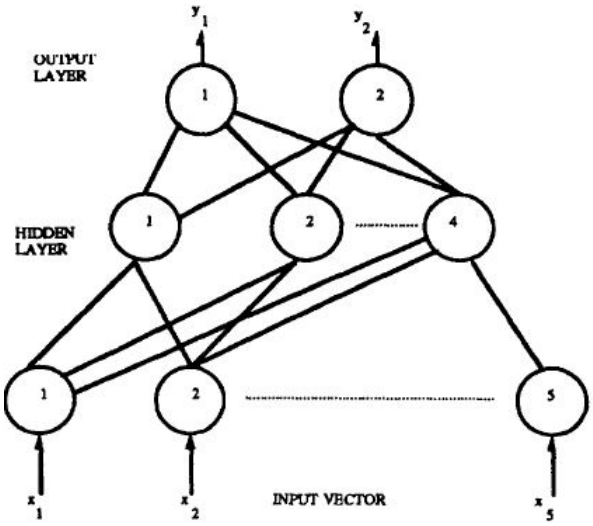

Fig. 3. The Artificial Neural Network [1]

logic. The network was simulated by using software and the same input patterns are applied to both the software and the hardware network. The output for both the cases are calculated. However, it is observed that the hardware implementation computes 4 million interconnections that accounts to approximately 70,000 decisions per second. This speed thus allows the implementation of the network in real-time applications [1]. Reconfigurability and adaptability are the main features of the hardware. Xilinx FPGAs are found feasible and efficient tools for the design of neutral nets. They provide acceptable densities without the cost and length design cycles of full custom circuits. However, the two FPGAs XC3042 and a 1K x 8 EPROM makes the network bulky and slow. The size and the speed can be increased considerably by using higher density FPGAs like XC3090. This FPGAs not only reduce the size but also increase the speed by eliminating the 55ns delay between the I/O pins of the two FPGAs.

A. Another approach of ANNs using FPGAs

A digital system architecture can be designed to realize a feed forward multilayer neural network. The designed architecture is described using Very High Speed Integrated Circuits Hardware Description Language (VHDL). The parallel structure of the neural network helps it to be fast for doing the computations. FPGA-based reconfigurable computing architectures are suitable for hardware implementation of neural networks. ASICs are application specific integrated circuits which are manufactured for serving of any one application. Whereas, FPGAs can be re-programmed based on the requirement of the application. FPGAs not only provides parallelism but also flexible designs, cost reduction and design cycle [2].Fig.6. shows Neuron architecture.

In this design, the neuron architecture is implemented to avoid a large number of multipliers used in conventional design techniques. The Multiplier/Accumulator architecture has been selected. It takes the input serially, multiplies them

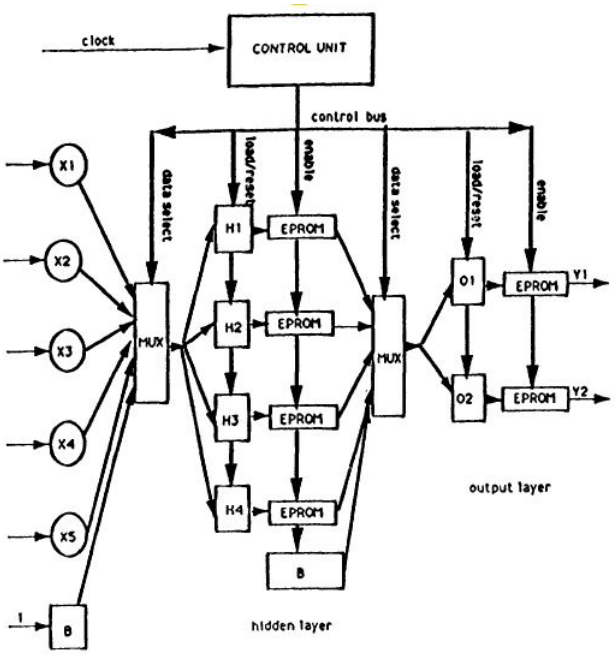

Fig. 4. General Architecture of the Network [1]

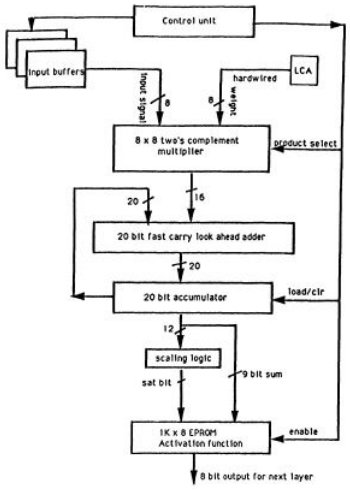

Fig. 5. Schematic of digital Neuron [1]

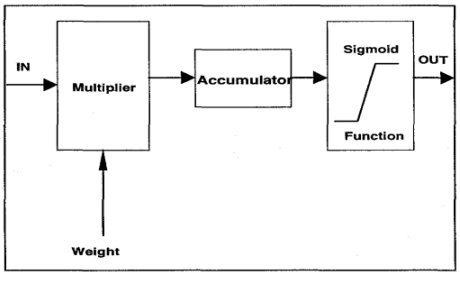

Fig. 6. Neuron Architecture [2]

with the corresponding weight and accumulates their sum in a register. The processes are synchronized to clock signal. The number of clock cycle s for a neuron to finish its work, equals to the number of connections from the previous layer.

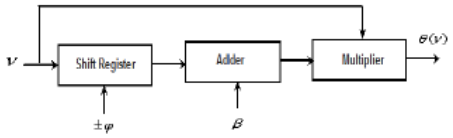

Fig. 7. Sigmoid Function Structural Diagram [2]

There are two ways to implement the sigmoid functions with simple FPGAs design. 1) Piece-wise linear approximation describes a combination of lines in the form of y = ax+b which is used to approximate the sigmoid function [2]. 2) Look-up tables, in which uniform samples taken from the center of a sigmoid function can be stored in a table for a look-up [2]. The structural diagram of the approximated sigmoid function implemented using this process Gs(z) is shown in Figure 7. and represent the slope and the gain of the nonlinear function Gs(z) between the saturation regions. A fully parallel network is fast but inflexible. Because, in a fully parallel network the number of multipliers per neuron must be equal to the number of connections to this neuron. Fig.7 shows Sigmoid function structural diagram.

IV. Hardware Implementations of ANNs using CMOS:

A. Digital Circuitry

ANNs can also be implemented using CMOS digital circuits. The advantage of using CMOS digital circuitry is because of its simple design and build. They can take full advantage of the advancement in the digital circuits. In this design, the synaptic weights can be implemented using digital memory cells, or even latches. The number of bits used to store the synaptic weights is critical to the accuracy of the algorithm. Also, the neuron state summation can be easily implemented using common multipliers and adder stages. The activation function is typically more complex to be implemented due to the fact that it has to be highly nonlinear by definition. However, the activation function such as the sigmoid function, requires a look-up tables. These slow down the computations considerably and require significant power and area if a good precision desired.

Although implementing ANNS in CMOS digital leads to simple designs, the result is not power, and area optimized. But the advantages overshadow the few limitations of the CMOS implementation. One of the major advantage of CMOS is that it can be integrated alongside the standard circuitry which can be fabricated using the same CMOS process [3]. IBM have implemented the Synapse chip using this design technique. A global synchronization clock has to be kept running throughout the chip at 1 KHz. Fig. 8 shows IBM Synapse chip.

B. Analog Circuitry

In general, analog circuits are much more complicated compared to the digital circuits, especially when scaling to large number of gates. But implementation of ANNs using analog circuitry can lead to more power and area efficient

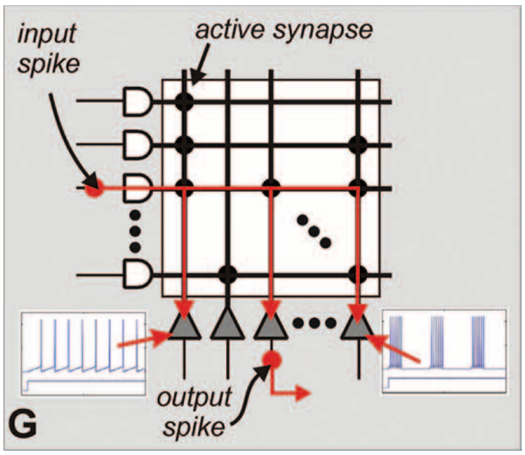

Fig. 8. IBM Synapse Chip[5]

circuits. Instead of storing a digital weight and then converting it to analog using ADC elements for computations, the weights can be stored in passive elements like resistors or capacitors. This can be done directly if the weights are to be fixed by design. Summation elements are not an issue for analog circuitry, since the neuron input and outputs will be represented as either currents or voltages. Currents branches in parallel are summed up whereas voltages in series are added together with no extra computation element. In the class of oscillatory neural networks, phase is used to represent a neuron state. Here, Phase Locked Loops (PLLs) are required to sum the neuron inputs. The activation function can be easily implemented by using a highly non-linear amplifier in the saturation regime. However, manufacturing variations inherent to the analog circuits limits the achievable accuracy [3].

Although, CMOS implementation shows some promising design techniques, they also suffer from few drawbacks as stated above. One of the major drawback of analog/digital CMOS implementation is when we attempt to scale the networks to large numbers of neurons and synapses. Increasing number of neurons and synapses, makes the circuit wiring very complex and nearly impossible. However, there are several promising novel devices that might be easier for implementation of the neural network synapse. Chief among them is the Memristor [3].

C. Memristor

Memristor is the long-anticipated fourth circuit element. A simplified view of the memristor consists of seeing it as a two-port programmable resistance. If we use resistance to encode synaptic weights, the memristor will enable an efficient programmable analog synapse without requiring heavy additional digital circuitry [3]. The memristive effect of this device occurs at nanoscale which is promising with regards to area scaling. The challenge of using memristor device is the reliability in their fabrication and integration with the standard manufacturing process.

D. ANNs using Processor In-Memory(PIM)

As ANNs have constantly changing algorithms, the computational complexity increases. Multiple computational elements are required to perform the tasks like Boolean or arithmetic calculations. These tasks can be performed using the traditional CPUs, but the performance obtained is substantially less due to the nature of ANNs. This is due to the fact that the CPU has to access the memory every time it needs to compute the operation. Also, these computations are very frequent in ANNs. A Processing in-memory architecture however can efficiently handle the operations of the algorithms in ANNs [4]. For example, in the Figure 9 there are multiple inputs to the ANNs which require fine computations like AND, OR etc,. Instead of performing these computations in the CPU, the same can be done using the Processor in-memory(PIM). In PIM, each processing element (performing tasks like AND, OR etc) is interfaced with the memory elements. The advantage of using PIM for ANN is that it handles the major load of the CPU, thus realxing the bus required for other IO operations. Also, fine grained parallelism can be obtained by employing PIMs [4].

V. CONCLUSIONS

Hardware implementations of the ANNs can be done using several design techniques. The most traditional design technique was to implement ANNs using Xilinx XC3042 FPGAs. Another design technique involved using FPGA along with digital circuitry and VHDL programming language. Recent implementations highlight the effort for ANNS implementation using CMOS circuitry. CMOS being small in size, area, cheap cost attracts the attention. CMOS implementations includes digital and analog circuitry deployment. However, due to the challenges of large number of neuron-synapse network that leads to complex wiring, researchers are now trying to implement ANNS using devices like Memristor which have advantages of less complex interconnects. Future implementation can be ANNs using PIM that will further speed up the processing and computation.

The full power of hardware ANNs has not been seen yet. But with coming release of commercial chips implementing arbitrary neural networks, more efficient algorithms will be realized in this domain, where neural networks are known to dramatically improves the performance of future applications like pattern recognition, associative memory, machine learning, etc.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] Botros, Nazeih M., and Abdul Aziz. Hardware Implementation of an Artificial Neural Network. IEEE, 1993.

[2] Mohammed, Esraa Zeki, and Haitham Kareem Ali. Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array. International Journal of Computer Theory and Engineering, Vol. 5, No. 5, October 2013, 2013.

[3] Forssell, Mats. Hardware Implementation of Artificial Neural Networks. 18-859E INFORMATION FLOW IN NETWORKS, 2015.

[4] J. Schabel, L. Baker, S. Dey, W. Li, and P. D. Franzon, Processor-in-memory support for artificial neural networks, 2016 IEEE International Conference on Rebooting Computing (ICRC), 2016
[5] Free Encyclopedia, Wikipedia the. Artificial Neural Network. https://En.wikipedia.org/Wiki/Artificial_neural_network, accessed on March 2018.