# A SIMPLIFIED DESIGN OF MULTIPLIER FOR MULTI LAYER FEED FORWARD HARDWARE NEURAL NETWORKS

**V.Parthasarathy[1], B.C.Hemapriya[2], H.M.Ravikumar[3]**

[1]*EEE Department, Nitte Meenakshi Institute of Technology, Bangalore, India*
[2]*ECE Department, Nagarjuna College of Engineering and Technology, Bangalore, India*
[3]*EEE Department, Nitte Meenakshi Institute of Technology, Bangalore, India*

## Abstract

*In the span of last twenty years, a lot of software solutions were proposed to utilize the inherent parallelism of the Artificial Neural Networks (ANNs). In order to take the full advantage of Neural Networks, dedicated hardware implementations are essentially required. But still, very few hardware models of multi layer feed forward networks with simplified activation functions are available today. Hence the effective utilization of Hardware neural Networks (HNNs) is restricted to only simple applications rather than complex power system problems. This paper analyzes the complications in the HNN design and a simplified algorithm for designing the multiplier part of a HNN is proposed. A comparison between existing and proposed model is provided.*

***Keywords:*** *Hardware Neural Networks, Multipliers, Multi layer feed forward networks.*

---------------------------------------------------------------------***---------------------------------------------------------------------

## 1. INTRODUCTION

The Artificial Neural Networks (ANNs) have found numerous applications in the electric power system including load forecasting, stability enhancement, unit commitment and economic dispatch problems. Also they are playing very crucial role in control engineering, signal processing and pattern recognition areas. But still, ANN is a research field with many open challenges in the topics of theory, applications and implementations. The hardware development of ANNs is lagging far behind compared to the conventional neural theories. It is because of the fact that lot of computations, huge investment and skilled man power is required for designing a simple Hardware neural Network (HNN). Also the low cost micro controller or DSP based controllers are found to be equally effective for the above applications. But due to the "self understanding and decision making", the unique ability of a HNN controller, it is certainly superior in power system applications. Hence it is important to address the key issues related to the design of cost effective HNNs. Due to the rapid growth of VLSI technology, and their simple, cost effective algorithms, there is a scope for the future of HNNs for power system applications.

## 2. BASICS OF PROPOSED DESIGN

With the currently available ASIC and FPGA technologies, the digital implementation of Neural Networks become very attractive. However, when it is required to work with the Multilayer Feed Forward Neural Networks (MFNN), the direct implementation is near impossible because of the involvement of large number of multiplications. Many of the power system applications normally require large scale neural networks with hundreds of neurons and synapses. It is possible to design analog VLSI circuits for such kind of applications. But there are certain difficulties like noise and un availability of high precision resistors makes this task very complicated.

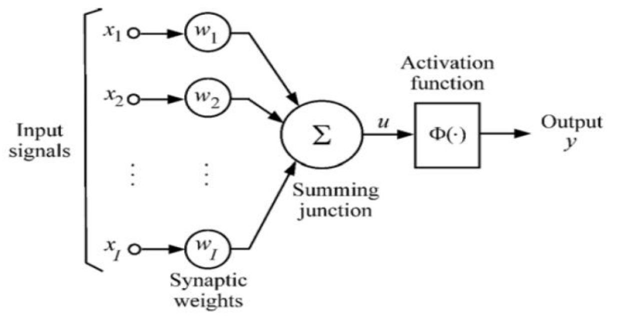


**Fig: 1** A typical Neuron in MFNNs

The fig (1) shows the basic neuron in MFNN architecture with summation, multiplication and activation blocks. A practical MFNN may be the combination of any number of such neurons. Basically, each neuron has 'n' multipliers to multiply each input value by the corresponding weight. The 'n' results of the multipliers are added with the bias and finally, the transfer function delivers the neuron's output.

Therefore two important arithmetic operations are involved in the hardware implementation of an ANN. The evaluation of the inner products is the first and the foremost operation. The computation of activation function is the second work in Hardware Neural Network (HNN) fabrication.

The design of multipliers which are used as synapse in neural network circuits creates many often conflicting constraints on the designer. Among these are small size, high speed, linearity, and four quadrant multiplications. The inputs are in the form of voltage differences and are denoted by the couples $x_1$-$x_2$, and $y_1$-$y_2$. The output of the multiplier is a current difference which is proportional to the multiplication of the voltage differences $(x_2 - x_1)$ and $(y_1 - y_2)$ $I_{diff}$ □ □$K$ $(x_2 - x_1)$ $(y_1 - y_2)$ where $K$ is the proportionality constant.

The current difference is then converted to a single ended current ($Z$) through current mirrors. This improves the linearity of the multiplier. Therefore, modeling procedure becomes easier which is one of the most important tasks during the training phase. Beside the improvement in linearity, the current mirror plays a role as buffer which will allow easy interfacing with the following circuitry.

## 2.1 Limitations Experienced

As shown in the fig 1, the solution of the inner products is obtained by simple mathematical summation. It seems to be a very easier task to handle.

$$u = \sum_{i=1}^{I} w_i x_i$$

But during the HNN implementation, large number of steps are required for obtain the same. In a fully parallel network, the number of multipliers per neuron must be equal to the number of connections to this neuron. Since all of the products must be summed, the number of full adders equals to the number of connections to the previous layer minus one.

For example, in a 4-8-1 network the output neuron must have 8 multipliers and 7 full adders while the neurons in the hidden layer must have 4 multipliers and 3 full adders. Hence, different neuron architectures have to be designed for each layer. Similarly selecting an appropriate activation function from the available choices of threshold, ramp and sigmoidal expressions are required very complex procedures. Once the hardware model is implemented with a particular activation function, it cannot be modified for the other and hence the lack of flexibility makes the system for selected applications only.

The primary function of MFNNs is to pass a weighted sum of inputs through a non linear activation function. Hence the digital implementation of MFNNs consists of several major functional blocks including multiplications, summations and calculation of non linear functions. The summations can be obtained by using adders and accumulators. The non linear calculations can be designed using look up tables.

But it is not advisable to design the multiplication between the inputs and the weights using VLSI since it requires large chip space and very slower operation.

## 2.2 Hardware Representation

To implement the neural network into hardware design, it is required to translate generated model into device structure and implemented using sophisticated digital or analog circuits. Despite the fact that FPGAs do not achieve the power, clock rate or gate density, they are preferred over the typical software solutions because of their re-configurable nature and fastness. The number of processing elements in a hardware implementation can be used to characterize the degree of parallelism achieved. With the introduction of FPGAs, it is feasible to provide custom hardware for application specific computation design.

But in the FPGA design, the balancing of achieving the required bit precision along with the increased cost is a challenging task. For the inputs, weights and activation function the degree of precision to be such that the iterations for the desired output must be lower. During the learning phase, precision has a significant impact. Only during the propagation phase, the precision can be slightly sacrificed. Hence the effective memory space will be enormously increased.

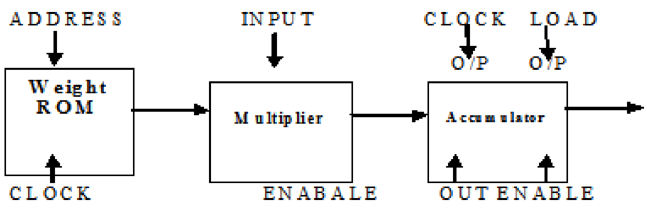


**Fig: 2** Basic representation of a single neuron

An artificial Neuron can be visualized as the combination of a multiplier and an accumulator. Fig 2 represents a simple hardware neuron. Every Single neuron will have its own weight storage ROM. The inputs from the previous stages will enter into the present neuron serially, and are multiplied with the corresponding weights. The accumulator will add all the multiplied values. All this functions will be synchronized by suitable clock signals. If there are 'N' connections are present in the previous stage, then at least 'N-1' clock pulses required for the present neuron to complete its task. The accumulator has a load signal for loading the bias values to every neuron at starting. The proposed structure is maintained fixed for various parts of the entire network. This may create some sort of error during the training process which can be neglected.

A circuit which is very closely imitates a biological neural structure is known as Neuro morphic (NM).Mostly NMs are involved with analog components despite the final outcomes may be digital. An essential aspect of NM is Address Event Representation (AER) protocol. In most of the power system applications, it is always required to have point to point pulse communication between neural assemblies.AER is used to develop such kind of communication paths by using a suitable algorithm.

## 3. PROPOSED FORMAT FOR MFNNs

For minimizing the time delay in the multiplier operation, a simplified logic termed ***"Equivalent Shifted Powers of Multiplication" (ESPM)*** is proposed in this paper. In this format, equivalent powers of two valued connection weights can be used instead of original continuously valued weights. Hence the multiplication can be replaced by shifting operation. The net silicon area needed for shifting operation for the same input data will be a fraction of the space for multiplication for the problem considered. It is possible to extend admissible weight values to be sum of two or more "terms of powers of two" and hence the complexity can be minimized.

When the 'ESPM' format is used, all weights in the MFNN only to be taken in the following set of discrete values:

$$\mathbf{W_{ESPM} = \{\pm 1, \pm 2^{-1}, \pm 2^{-2}, \pm 2^{-3}, \ldots \pm 2^{-X}, 0\}}$$

Where 'X' is the maximum number of bits that may be shifted For the given value of 'X', there are 2X+3 weight values are available to choose from. Again for some cases more than one 'ESPM' value may be suitable. In such scenario, the chip area requirement to be evaluated for identifying the optimum value of shifted weight.

## 4. HYBRID QUANTIZATION

Neural Networks, in general, work with floating-point numbers. Working with floating-point numbers in hardware is a difficult problem because the arithmetic operations are more complex than with integer numbers. Furthermore, the dedicated circuits for floating-point operations are more complex, slower, and occupy a larger chip area that integer number circuits. A solution used to make this project easier and improve its performance has been converting the floating point numbers to integer numbers.

Of course it implies in some loss of precision but in this particular case, good results have been achieved. The outputs of activation functions and connection weights in an MFNN are evaluated by back propagation algorithm. These quantities are continuously valued, so that the multiplication is un avoidable. Even though the ESPM values may reduce this burden considerably, the realization of the resultant activation function is not easier. Alternatively, continuous weights with approximated (quantized) neurons may provide the necessary freedom to be adopted for diverse problems. For combining both these futures, it is proposed to go for adopting the 'ESPM' weights and further successive approximation for the sake design the HNN. This combination of quantization and ESPM implementation is known as 'HYBRID' design of multipliers. There are three stages involved in the hybrid design .

(i) The conventional Back Propagation (BP) algorithm to be applied to find the set of continuous weights for the given problem.
(ii) Then the approximation to be applied to convert the obtained weights to the 'ESPM' values.
(iii) Adoption of the slope of the activation function to be employed for fine-tuning the post-approximation network to the pre-determined error level.

### 4.1 Step 1: Evaluation of Weights

Until the output error falls below a pre-set value 'e', the following sigmoid activation functions to be used.

$$f(x) = p\ (1-\beta^{-\alpha x})\ /\ (1+\beta^{-\alpha x})$$

Here the constant 'p' is introduced for balancing the non linear coefficients during the convergence process. Upon the convergence, a network with continuous weights and sigmoid activation function is obtained. Let it be # Net:1. This network should produce an effective error '$e_K$' which should be less than or equal to the pre set error 'E'. During the real time implementation, the values of weights may further to be approximated for the sake of accommodating newer neurons. If one half of the activation function is quantized and the remaining part retain as it is, this task can be easily achieved. But such kind of partial quantization may leads to the actual solution to divert from its desired value.

The flowchart shown in fig:3 represent this process. The flow chart does not show some more calculation steps and it is only used as the basic representation.

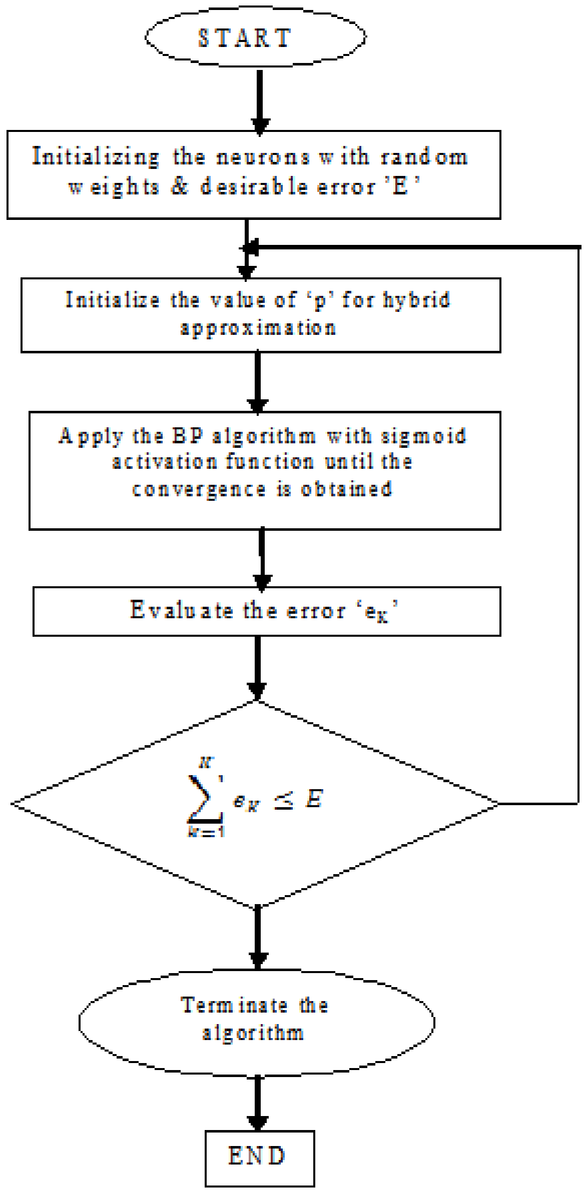


**Fig 3:** Flow Chart for Proposed Quantization

### 4.2 Step 2: Replacing the Weights

Keeping the topology of the network remains unchanged, the neurons in # Net: 1 to be replaced by their approximated values. Then the existing weights to be adjusted with the corresponding ESPM weights. The resultant neuron will have different activation function from its original one. During the replacing of original neurons with the quantized weights, it was found that the output error of the system was increased. The issue was sorted out by adjusting the system parameters.

### 4.3 Step 3: Design of Shifter

The basic function of the shift block is to shift a weight according to the activation of the corresponding neuron which is of the 'ESPM' format, instead of doing multiplication. Since the output of a neuron is different under different input patterns, the number of bits to be shifted in the corresponding weight varies from one pattern to another. Hence the shift block should be able to detect and control the actual number of bits to be shifted. This is achieved by employing two shift registers. Let them be $SR_1$ and $SR_2$.$SR_1$ is assigned for control the number of bits to be transferred and $SR_2$ is involved in the actual shifting operation.$SR_1$ is 8 bit device and $SR_2$ is 16 bit component. Now to be fitted into a neurons operation in MFNNs with 'ESPM' weights the pin 'A' is connected to a particular input to the neuron. The control vector is mapped to the corresponding ESPM weight.

**Table 1**: Shifting Operation for X=2

| Sl.No | Control vector (C) | Input vector (A) | Output vector (Y) |
|---|---|---|---|
| 1 | 00000 | $A_7 \ldots A_0$ | 000000000000 |
| 2 | 10000 | $A_7 \ldots A_0$ | $A_7 \ldots A_0 0000$ |
| 3 | 01000 | $A_7 \ldots A_0$ | $A_7 A_7 \ldots A_0 000$ |
| 4 | 00001 | $A_7 \ldots A_0$ | $A_7$ $A_7$ $A_7$ $A_7 A_7 \ldots A_0$ |

In the table shown above the control vector 'C' is deciding the number of bits to be shifted. Vector 'Y' is the shifted version of input vector 'A'. This operation can be implemented either by using multiplexers or simple combinational logic.

Taking the negative powers of ESPM values, the shifter has shift-right moments. For example, let X=2. Then, the possible ESPM choices are $2^0$, $2^{-1}$, $2^{-2}$ and 0. The related control vectors of the shifter are 10000, 01000, 00100 and 00000 respectively. A separate VHDL code is created for this shifting operation.

**Table 2**: Comparative Table

| Sl. No | CRITERIA | MFNN with direct Multiplying components | MFNN with ESPM implementation |
|---|---|---|---|
| 1 | Calculation | $Z_j(h)^* W_{ij}(h)$ | $Z_j(h) * W_{ij}^{(h)}$ |
| 2 | Implementation | Multiplier (4 X 4) | Shifter |
| 3 | Area ( No of gates) | 126 | 87 |
| 4 | Delay (s) | 6.2 | 1.8 |

## 5. RESULTS AND DISCUSSIONS

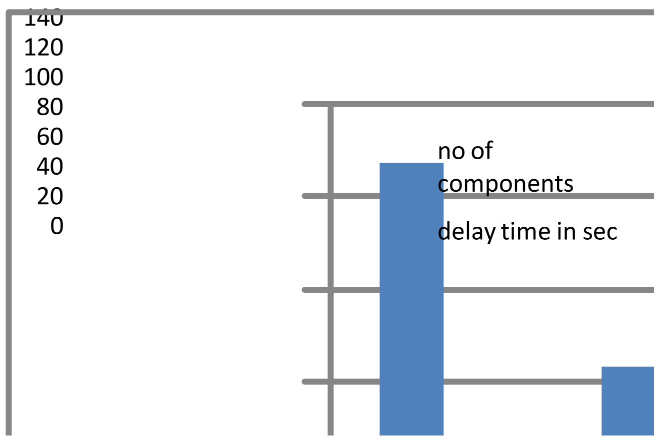


**Fig 4:** Graphical Representation of the Results

The above table gives the comparison between conventional multiplier and a shifter for the same specifications. It is to be noted from the table that the number of gates required for the digital implementation for the same operation is reduced considerably.

Also the delay time for the execution of one complete set of input vector (A) is reduced in case of the shifter implementation. But it is to be noted that there is a deviation in the convergence when the shifter is too much trained.

## 6. CONCLUSIONS

Because of the implementation of 'ESPM' weights, the multiplications needed for weighted sum operations were replaced by shift operations. Hence there is a considerable improvement in both silicon area and operation speed in digital VLSI design of MFNNs. It also ensured that the proposed network is capable of very closely achieving the same generalized performance of the network with conventional multipliers. But it is to be mentioned that the overall cost of this improved system will be comparatively higher than the multiplying system. Also the mapping ability of the derived multiplier with the activation function is below average in this work. This can be enhanced by increasing the number of control registers. Over all the designed equivalent multiplier is capable of reducing the chip area approximately by 31% which is a desired outcome of this work.

## REFERENCES

[1] A. Muthuramalingam, S. Himavathi, E. Srinivasan, “Neural Network Implementation Using FPGA:Issues and Application” International Journal of Information Technology Volume 4 Number 2,2007

[2] Prashant D.Deotale, Lalit Dole, “Implementation of FPGA based Multilayer Perceptron using VHDL” , International Journal of Advanced Research in Computer Science and Software Engineering, Volume 4, Issue 2, February 2014.

[3] Amit Garg,Sanjai Kumar Agarwal, “Dynamic Stability Enhancement of Power Transmission System using Artificial Neural Network Controlled Static Var Compensator” International Journal of Computer Applications, Volume 53– No.9, September 2012.

[4] Esraa Zeki Mohammed 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.

[5] Leonardo Maria Reyneri “Implementation Issues of Neuro-Fuzzy Hardware: Going Towards HW/SW Codesign” IEEE Transactions on Neural Networks, vol.14, no.1, pp. 176-194, 2003.

[6] Y.J.Chen, Du Plessis, “Neural Network Implementation on a FPGA ”, Proceedings of IEEE Africon, vol.1, pp. 337-342, 2002.

[7] Sund Su Kim, Seul Jung, “Hardware Implementation of Real Time Neural Network Controller with a DSP and an FPGA ”, IEEE International Conference on Robotics and Automation,vol. 5, pp. 3161-3165, April 2004.

[8] Turner.R.H, Woods.R.F, “Highly Efficient Limited Range Multipliers For LUT-based FPGA Architectures”, IEEE Transactions on Very Large Scale Integration Systems, Vol.15, no.10, pp. 1113-1117, Oct 2004.