

Analog Implementation of Artificial Neural Networks Using Forward only Computation

Subha Mada, Srinivas Mandalika

Department of Electrical Engineering
BITS Pilani, Hyderabad Campus
Hyderabad, India.
Email: subhasekhar@gmail.com

Abstract — The algorithm used to train an Artificial Neural Network (ANN) plays an important role in its implementation. Analog VLSI implementations of ANN using back propagation algorithm for multi-layer perceptron (MLP) architectures were reported earlier. In this paper, we used an algorithm which uses forward only computation to update the weights, instead of forward and backward computation resulting in reduced computation time. The chosen algorithm, can train all types of architectures in less time, even where back propagation and other second order algorithms fail. An analog VLSI implementation of this algorithm can further reduce the area and power dissipation. To validate our idea, we designed and implemented a two input-one hidden layer-one output MLP network. All the blocks were implemented in CADENCE Virtuoso tool using the 180nm technology library. The resultant network architecture was tested successfully for digital applications like AND, OR and analog applications - compression and decompression.

Keywords — Feed forward Neural networks, Analog VLSI design, Forward only computation, weight update, without back propagation

I. INTRODUCTION

Neural networks have many real time applications. In realizing the neural networks using analog VLSI blocks, the training algorithm used to train the neurons in the network is very critical.

Traditional back propagation algorithm is simple to implement. The error, the difference between the expected and actual result is back propagated by updating the weights and finally the minimum error point is reached at the convergence point. But it slows down when the number of outputs of the neural network increases, when compared to the second order algorithms [1].

The second order Levenberg- Marquardt algorithms used in the neural network tool box is fast and can train the neural networks for which EBP algorithm has convergence problem. But, this also has some disadvantages if the network has more number of outputs [1].

Analog VLSI implementations that were reported use back propagation algorithm [2-4]. In this paper, we implement an algorithm [5] which uses forward only

computation and does not use back propagation to update the weights in training the neural network.

We propose the design of various blocks to implement the neural network architecture using this forward only computation algorithm. The blocks designed were tested successfully for some of the digital and analog applications.

A. Simple neural network architecture

The neural network shown in Fig.1 is considered for implementation using the forward only algorithm. It has 4 neurons and two inputs and one output.

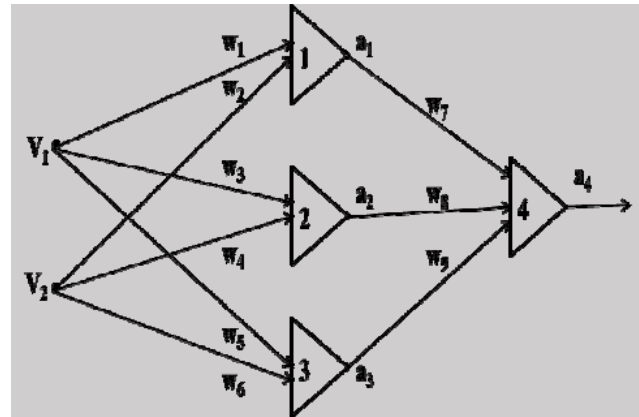


Fig.1. Neural network

The analog VLSI blocks that were used in implementing the network shown in Fig.1 are listed below.

1. Four quadrant analog multiplier: to multiply the updated weights with the inputs.
2. Tan sigmoid function generator: To act as an activation function and to derive the slopes of the individual neurons.
3. Delta module: to find the delta matrix entries
4. Gradient vector module : to compute the gradient vector
5. Capacitor block: to store the present weights.
6. Weight update module: to update the weights.

A. Multiplier [6]

The four quadrant analog multiplier shown in Fig.2 is used to multiply the inputs, with the associated weights.

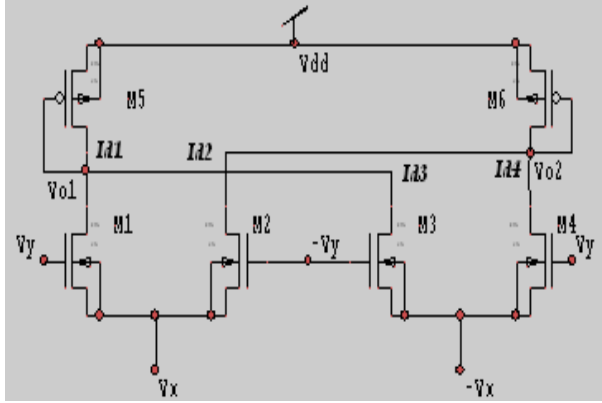


Fig.2 Multiplier

B. Subtractor

The circuit shown in Fig.3 is used to compute the error between the actual output and target output [7].

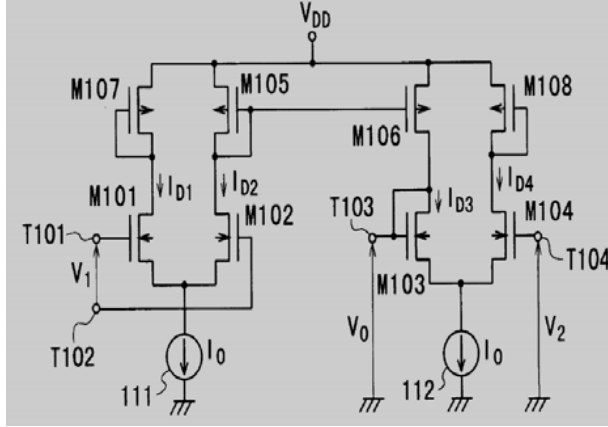


Fig.3 Voltage subtractor

C. Activation Function with Slope

The circuit in Fig.4, which generates the tan sigmoid along with its derivative is used as the activation function in the neural network architecture.

II. CALCULATIONS FOR THE PROPOSED DESIGN

For the network considered in Fig.1, the neural architecture is designed according to the modified LM algorithm which uses only forward computation to update the weights.

Delta matrix is calculated using the equation (1) and there by the Jacobean and gradient vector are computed as explained in the section B and C.

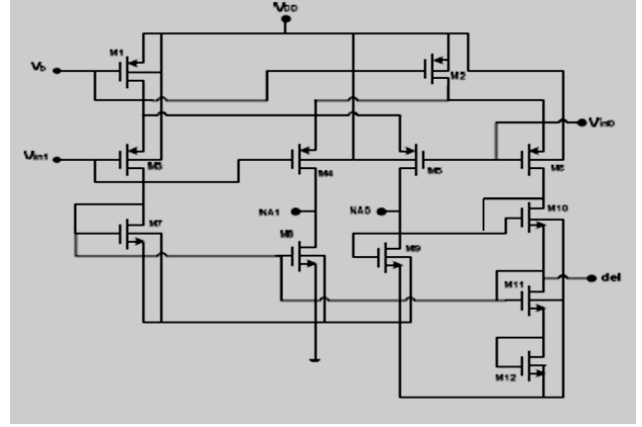


Fig.4. Activation function with slope

In the delta matrix mentioned in Table I, S_k indicates the slope of neuron i. $\delta_{k,j}$ is the signal gain between neurons j and k.

$F_{k,j}(y_j)$ is the non-linear relationship between output node of neuron k and output node of neuron j. Delta is defined as

$$\delta_{k,j} = \frac{\partial F_{k,j}(y_j)}{\partial \text{net}_j} = \frac{\partial F_{k,j}(y_j)}{\partial y_j} \cdot \frac{\partial y_j}{\partial \text{net}_j} = F'_{k,j} \cdot S_j \quad (1)$$

If $k = j$, it is the slope (S) of neuron j

$$\delta_{j,j} = S_j$$

A. Calculation of Delta matrix

The delta matrix for the network considered will be as shown in Table 1.

TABLE I. δ MATRIX

Index	1	2	3	4
1	S1	0	0	w7
2	0	S2	0	w8
3	0	0	S3	w9
4	$\delta_{4,1}$	$\delta_{4,2}$	$\delta_{4,3}$	S4

where the entries of the delta matrix are defined by equations (2),(3), and (4) as:

$$\delta_{4,1} = S_4 \cdot w_7 \cdot S_1 \quad (2)$$

$$\delta_{4,2} = S_4 \cdot w_8 \cdot S_2 \quad (3)$$

$$\delta_{4,3} = S_4 \cdot w_9 \cdot S_3 \quad (4)$$

where S_j is the slope of the neuron 'j'.

B. Calculation of Jacobean matrix

The elements of the Jacobean matrix, computed using the delta matrix entries of section A are shown using equations (5) – (13).

$$f[1] = S_4 \cdot w_7 \cdot S_1 \cdot f_1 \quad (5)$$

$$f[2] = S_4 \cdot w_8 \cdot S_2 \cdot f_2 \quad (6)$$

$$j[3] = S_4, w_8, S_2, f_1 \quad (7)$$

$$j[4] = S_4, w_8, S_2, f_2 \quad (8)$$

$$j[5] = S_4, w_9, S_2, f_1 \quad (9)$$

$$j[6] = S_4, w_9, S_2, f_2 \quad (10)$$

$$j[7] = S_4, \alpha_1 \quad (11)$$

$$j[8] = S_4, \alpha_2 \quad (12)$$

$$j[9] = S_4, \alpha_3 \quad (13)$$

C. Calculation of Gradient Vector

From the Jacobean matrix calculated in section B, Gradient vector is calculated using the equations (14)-(22).

$$g[1] = S_4, w_7, S_1, f_1, e_1 \quad (14)$$

$$g[2] = S_4, w_7, S_1, f_2, e_1 \quad (15)$$

$$g[3] = S_4, w_8, S_2, f_1, e_1 \quad (16)$$

$$g[4] = S_4, w_8, S_2, f_2, e_1 \quad (17)$$

$$g[5] = S_4, w_9, S_2, f_1, e_1 \quad (18)$$

$$g[6] = S_4, w_9, S_2, f_2, e_1 \quad (19)$$

$$g[7] = S_4, \alpha_1, e_1 \quad (20)$$

$$g[8] = S_4, \alpha_2, e_1 \quad (21)$$

$$g[9] = S_4, \alpha_3, e_1 \quad (22)$$

D. Weight Update

After the Gradient vector calculation, the weights of all the neurons are updated using the following weight update rule shown in equation (23).

$$W_{n+1} = W_n - \alpha g_n \quad (23)$$

Where g_n denotes the gradient vector, W_n is the previous weight and W_{n+1} is the updated weight.

III. PROPOSED DESIGN FOR THE IMPLEMENTATION OF NEURAL NETWORK USING FORWARD ONLY ALGORITHM

A. Neural network

In Fig.5, V_1 and V_2 are the inputs to the neural network and w_n indicate the weights, S_j indicate the slope of the j^{th} neuron and ' a_j ' indicate the output of the neuron j . The blocks 'mult' and 'Actfun' used in the figure indicate the multiplier and activation function respectively.

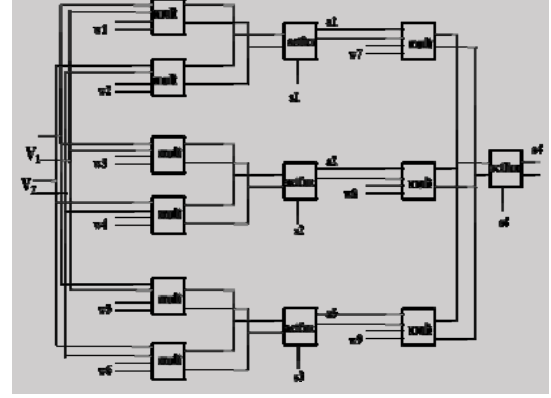


Fig.5. Implementation of neural network shown in Fig.1

B. Delta Calculation Block

The equations (2), (3) and (4) which are used to compute the delta are realized using the multipliers connected as shown in the Fig. 6.

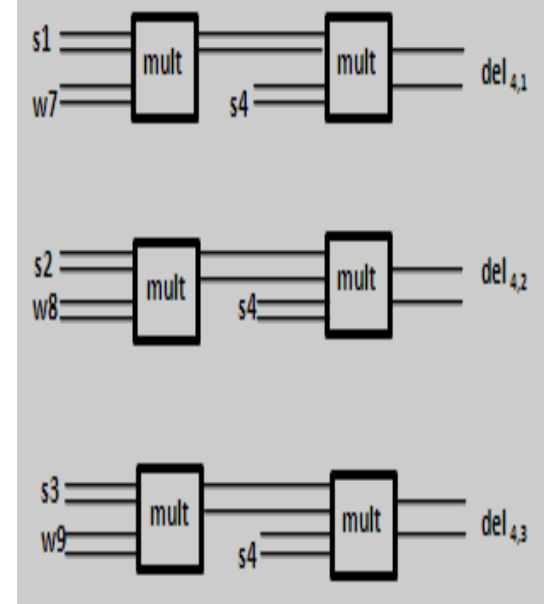


Fig.6. Delta calculation

After this calculation, these delta values $del_{4,1}$, $del_{4,2}$ and $del_{4,3}$ are used to compute the entries of the Gradient vector $g[1]$ to $g[9]$.

The number of the entries in the gradient vector depends on the number of outputs of the neural network and number of neurons used in the network.

C. Gradient vector calculation Block

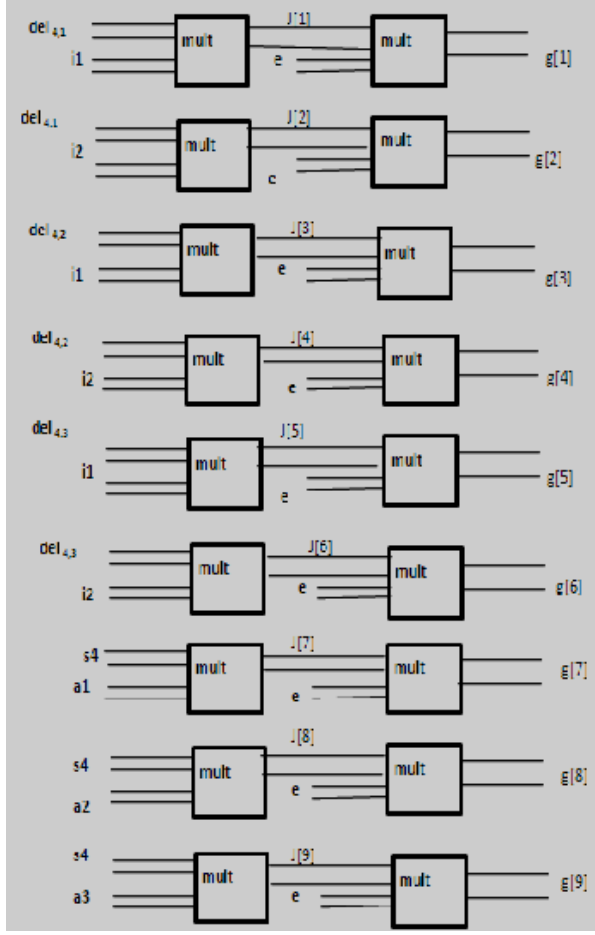


Fig. 7. Gradient calculation block.

D. Weight storage block

The weights are initialized to some predefined values and are applied in synchronous with the weight initializing clock. The updated weights are calculated using Forward only computation method as explained in previous section. The outputs of weight update block which is shown in Fig.15 are connected to this storage block of Fig.8, in synchronous with the weight update clock.

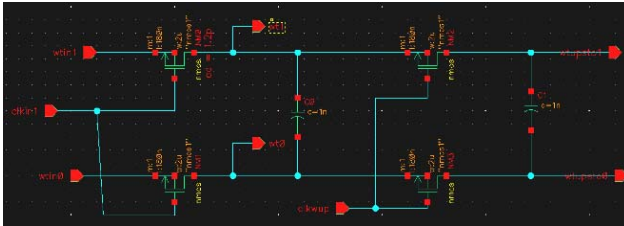


Fig. 8. Weight storage block

All the blocks that were designed and implemented are connected to build and train the neural network which is shown in Fig.1. The resultant network was successfully

tested for basic digital applications like AND, OR and analog applications like Compression and Decompression.

The block diagram of the setup used for testing compression and decompression is shown in Fig.9.

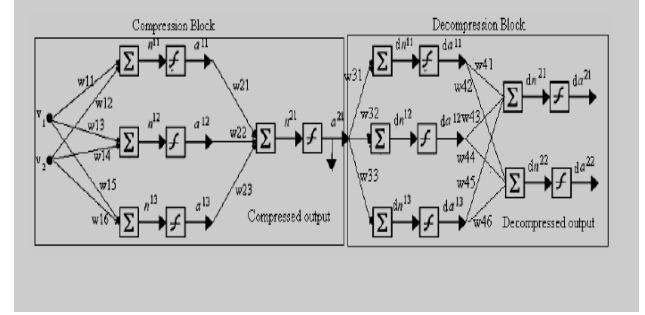


Fig.9. Block diagram for Compression and Decompression.

IV. CIRCUITS AND SIMULATIONS

The simulations of the multiplier and activation function are shown in the Fig. 10 and Fig.11 respectively.

A. Multiplier

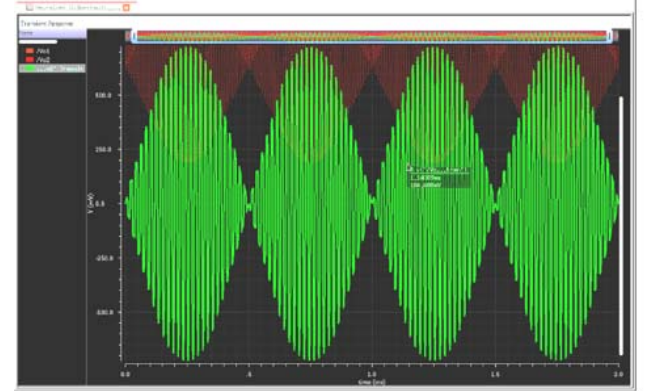


Fig.10 Transient analysis of multiplier.

B. Activation Function with Slope

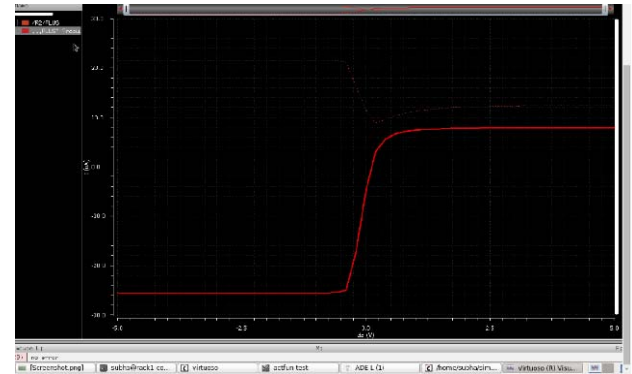


Fig. 11. DC Analysis of Activation function block

C. Neural Network

The neural network shown in Fig.1 is implemented using multiplier and activation function blocks as shown in Fig.5.

The output nodes of multiplier in Fig.2 will itself act like an adder, and the summation of weight - input products can be computed at those respective nodes. The resultant schematic is shown in Fig.12.

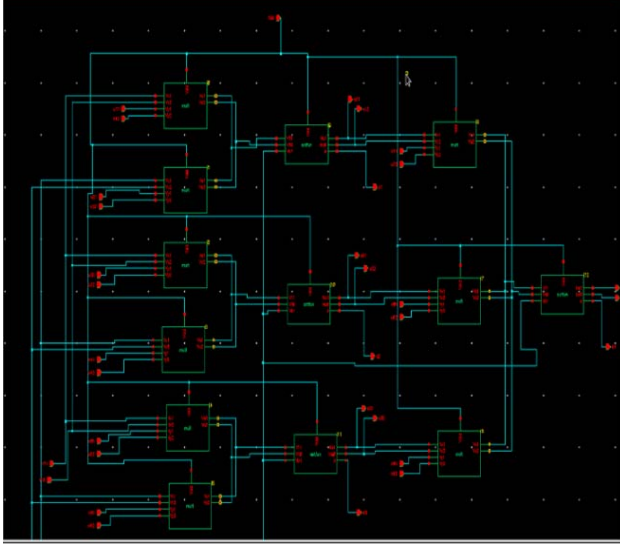


Fig. 12 Neural network

D. Delta Calculation Block

Fig. 13 shows the implementation of equations 2, 3 and 4 using which the delta parameters for the network are computed. The number of delta values to be calculated depend on the total number of outputs for the network. Here, as the network shown in Fig.1, is having single output, only the last row of the delta matrix which is shown in Table I need to be computed.

E. Gradient calculation

Once the delta values are calculated, the gradient vector is calculated as shown in Fig. 14. The equations from (14) to (22) are implemented using this block. The number of entries in the gradient vector will be equal to the number of weights that need to be updated in every iteration.

F. Weight update block

Updating the weights is done by the block shown in Fig. 15 in which the gradient vector is multiplied with α , the learning constant and is subtracted from the previous weight to get the updated weight.

The outputs of this block are connected to the associated weight storage modules and are stored by the capacitors for further iterations.

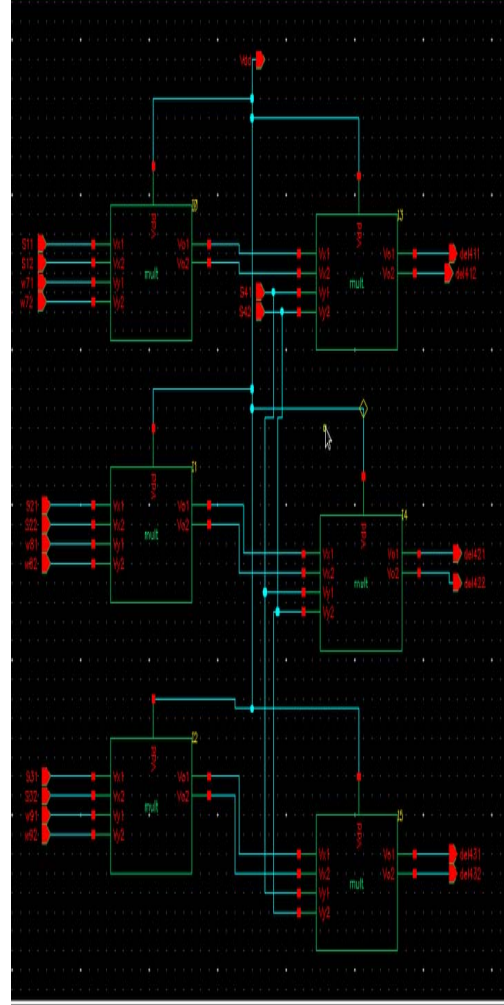


Fig. 13 Delta calculation block

G. Implementation of entire neuralnetwork shown in Fig.1

All the blocks that are necessary to do the computations were then interconnected as shown in the Fig. 16 to implement the complete neural architecture.

Delta values computed are applied as the inputs to the gradient block and outputs from the gradient block used for updating the weights.

The updated weights are fed to the weight storage block, in which the weights are stored using capacitors until the next iteration. Finally once the weights are converged, we can verify the output of the neural network at the output neuron.

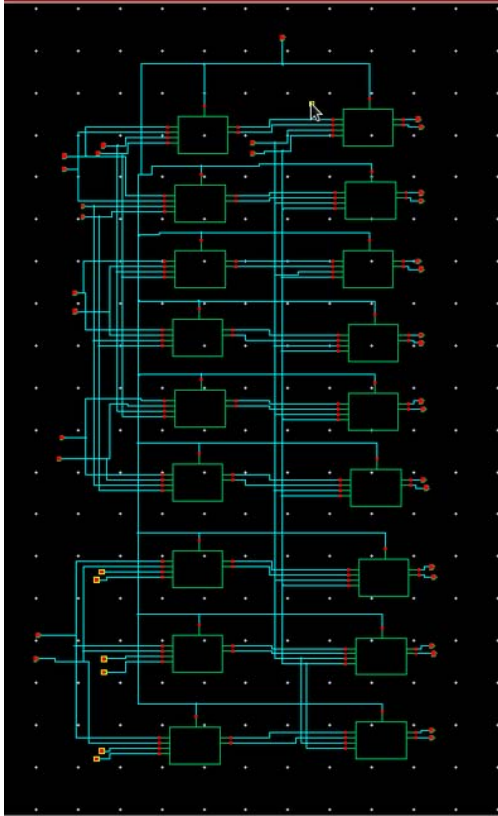


Fig 14. Gradient calculation block

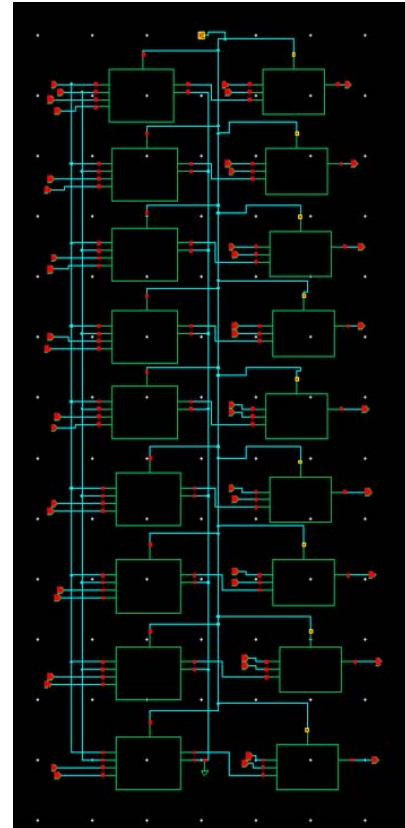


Fig 15. Weight update block

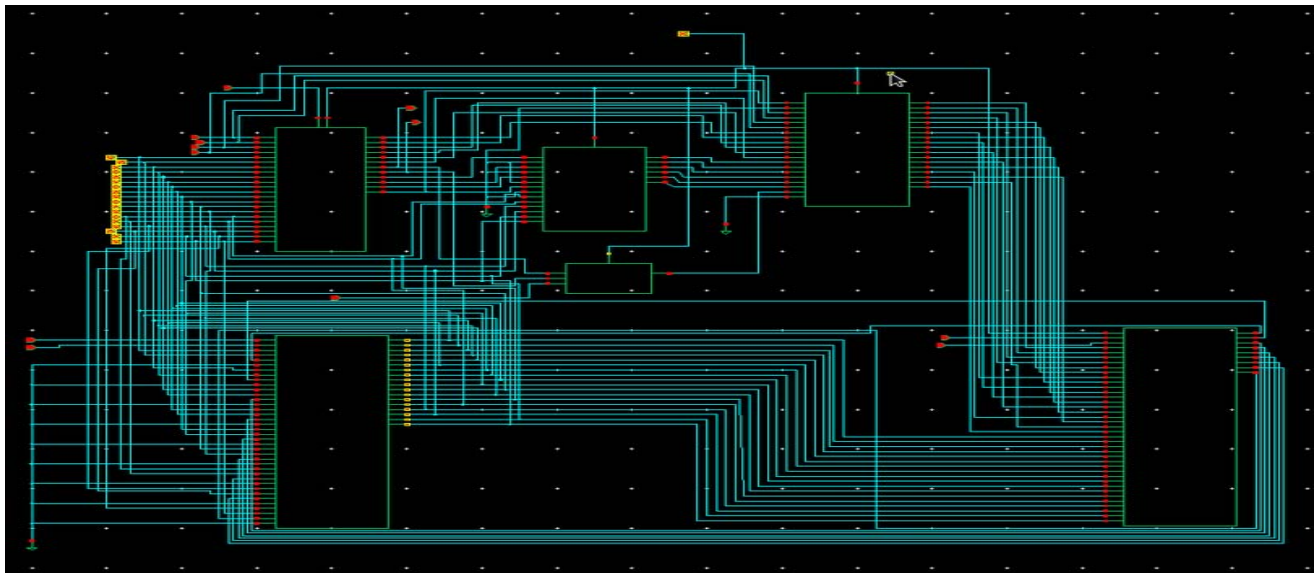


Fig 16. Complete Neural network

V. RESULTS

The neural network that is designed is tested for the digital operation AND, OR and analog applications like Compression and Decompression successfully.

Two input signals $i/p1$ and $i/p2$ were applied along with the expected target to the test circuit of the network that was designed by interconnecting all the modules. The results obtained are shown in Fig. 17, Fig. 18 and Fig. 19.

A. AND Operation

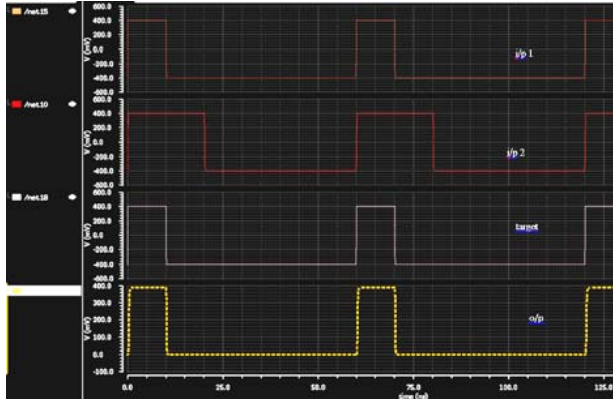


Fig 17. AND Result

B. OR Operation

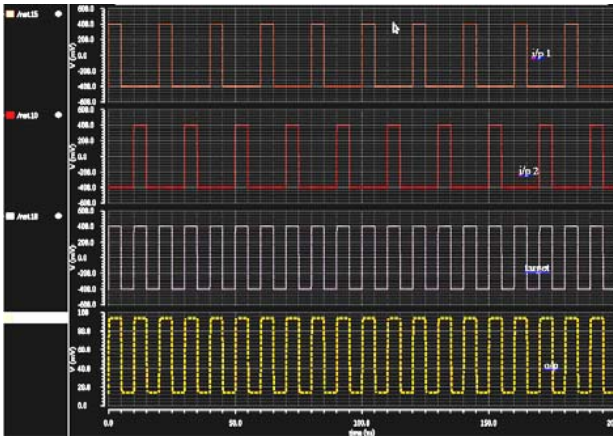


Fig 18. OR Result

C. Compression and Decompression

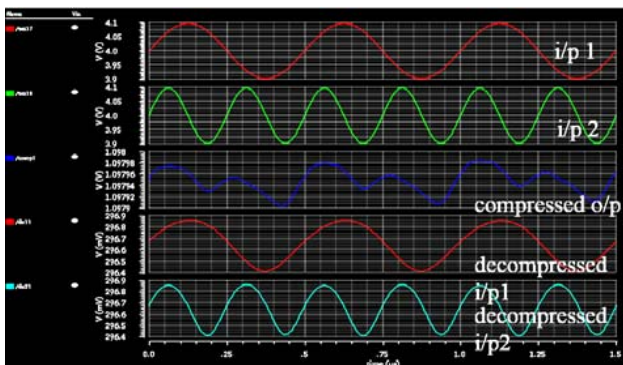


Fig 19. Compression and Decompression Result.

VI. CONCLUSIONS

The neural network shown in Fig.1 was implemented with Analog modules using forward only computation for

updating the weights (Without using back propagation) using CADENCE virtuoso tool with 180nm technology library. The Basic digital operations like AND, OR and analog applications like Compression and Decompression were tested successfully.

A more efficient weight update rule can improve the speed of the architecture but at the cost of additional and complex circuitry. Similarly, alternate multiplier and activation functions can be tried for better performance. The work can be extended for the implementation of multiple output neural networks, arbitrarily and fully connected neural networks which can be trained for complex patterns. Then, comparison studies in terms of area and power dissipation can be done against the other architectures using EBP and second order algorithms.

REFERENCES

- [1] M T Hagan and M B Menhaj, "Training Feedforward Networks with the Marquardt Algorithm", IEEE Transactions on Neural Networks, VOL. 5, NO. 6, NOVEMBER 1994.
- [2] Laurent Gatet, Hélène Tap-Bêteille, and Marc Lescure, "Analog Neural Network Implementation for a Real-Time Surface Classification Application", pp 1413-1417, IEEE SENSORS JOURNAL, VOL. 8, NO. 8, AUGUST 2008.
- [3] C P Raj P, S.L. Pinjare, "Design and Analog VLSI Implementation of Neural Network Architecture for Signal Processing", European Journal of Scientific Research, Vol.27, No.2, pp.199-216, 2009.
- [4] Bapuray. D. Yammenavar, Vadiraj. R. Gurunaik, Rakesh. N. Bevinagidad and Vinayak. U.Gandage, "Design and Analog VLSI Implementation of Artificial Neural Network", International Journal of Artificial Intelligence & Applications (IJAIA), Vol.2, No.3, July 2011
- [5] B M Wilamowski and Hao Yu, "Neural Network Learning without Back propagation", IEEE Transactions on Neural networks, VOL. 21, NO. 11, NOVEMBER 2010.
- [6] Akshatha B C, A Vijay Kumar, 'Low Voltage, Low Power, High Linearity, High Speed CMOS Voltage Mode Analog Multiplier" Second International Conference on Emerging Trends in Engineering and Technology, pp 149-154, ICETET-09
- [7] Katsuji Kimura, "Voltage Adder/Subtractor Circuit with two differential transistor pairs", United States Patent, Patent number: 5,909,137, Date of Patent: Jun 1, 1999