Logic gate verification using Neural Network

Aseel Mousa Ja'afer

Abstract

This work concentrates on showing the applicability of neural networks to
The field of design automation (CAD system), especially in the area of hardware verification.

The proposed algorithm is valid for combinational circuits verification, which is based on merging two of the well-known learning algorithms for neuralnetworks .The first one, ***Perceptron Convergence Procedure,*** which is usedfor learning the functions of the standard logic gates, in order to simulate thewhole circuit .The second, is the learning algorithm ***of Back-propagation*** neuralnetworks, which is modified to be used in the verification of the hardware design .Also, it predicts the gate that causes the malfunction in the incorrect design.

An important part of the designed systemis the graphical representation of the hardware, which is an interactive user interface for the hardware description .There is a translator that translate the graphical description into a multi-layer feedforward neural network.

After describing the hardware design, and stating it's specifications, which includes two methods of specifications :the

truth table method and the logical statements method .The verification step begins.

الخلاصة

ان هذا البحث يوضح تطبيق مفاهيم الشبكات العصبية في حقل تصميم الدوائر المنطقية باستخدام الحاسبة, وخصوصا في مجال التحقق من صحة التصاميم. ان الخوارزمية المقترحة مبنية على دمج خوازميتان معروفتان من خوارزميات التعلم الخاصة بالشبكات العصبية. حيث ان الخوارزمية الاولى (perceptron convergence procedure) تستخدم في تعلم الوظائف الخاصة بالبوابات المنطقية الاساسية. اما الخوارزمية الثانية وهي خوارزمية التعلم الخاصة بالشبكة (Back-propagation) حيث قمنا بتحوير هذه الجوارزمية للاستفادة منها في عملية التحقق من صحة التصميم للدوائر المنطقية. بالاضافة الى ذلك فان الخوارزمية المقترحة تتوقع البوابة المنطقية التي تسبب الخلل في التصميم الخاطيء.

جزء من النظام المصمم هو الوصف بالرسوم للدوائر المنطقية التي تعتبر واجهة متفاعلة مع المستخدم, حيث ان هناك مترجم يقوم بتحويل الوصف المرسوم الى شبكة عصبية متعددة المراحل مربوطة بموصلات امامية الاتجاة

(Multi-layer feedforword neural netwok)

بعد وصف التصميم واعطاء مواصفاته والتي تشمل طريقتين الاولى جدول الحقيقة (Truth table) والثانية بالعبارات المنطقية (Logical statement method), فان عملية التحقق تبدأ باستخدام الخوارزمية المقترحة.

1- Introduction

The brain is regarded by many as the last remaining frontier of exploration. Although scientists do not yet have a good understanding of how the human brain operates, some simplified models of its neural network have been proposed. Desiring to achieve human-like capabilities (e.g., vision, speech understanding, learning, etc.) in a computer, researchers have

been working for several years to implement neural network models into a new breed of computer system. These efforts have recently been successful, as commercial neurocomputers have recently become available from several firms [2].

A typical definition of neural network given in reference [2], is: "A cognitive information processing structure based upon models of brain function. In a more formal engineering context : a highly parallel dynamically system with the topology of a directed graph that can carry out information processing by means of its state response to continuous or initial input".

Neural networks are constructed of processing elements (PEs, or neurons) that are interconnected via information channels called interconnects. A neuron is a two state processing element. The state 0 or 1, of a neuron is its activation value. In our model, the neurons are connected by links. Boolean operations are realized by specifying the thresholds for the neurons and the weights associated with the links. [1].

The weight Tij characterizes the link connecting neurons i and j.li and Vi are the threshold and activation value of neuron i. A neuron determines its state by calculating the weighted sum of the states of all the connected neighboring neurons and comparing this sum with its threshold.Neural networks are typically used for two types of applications. In one application, the network performs constraint satisfaction tasks involving multiple constraints (combinatorial optimization) to find a consistent set of states. In the second application, the neural network learns from examples. The essential difference between the two applications is

as follows : in the first application the network adjusts the states of the neurons to a given fixed set of connection strengths, whereas in the second application the network adjusts the values of the connection strengths to given example sets of neuron states. In the application of neural networks to hardware verification, we require both techniques, hence the first one is used to verify the hardware design, and the second is used to teach the neuron the function of a logic gate [1]. Further discussion on the history of neural networks is given in references [3], [4], [5], and [6].

2- Logic circuits modeling by neural networks

We use the Rosenblatt's ***Perceptron***[33], which is one of the earliest models of neural networks, to simulate the functions of logic gates .According to Rosenblatt, the perceptron structure is shown in Fig (1) .In this figure, the perceptron have three major parts :the input vector, the weighted summation, and the threshold value.

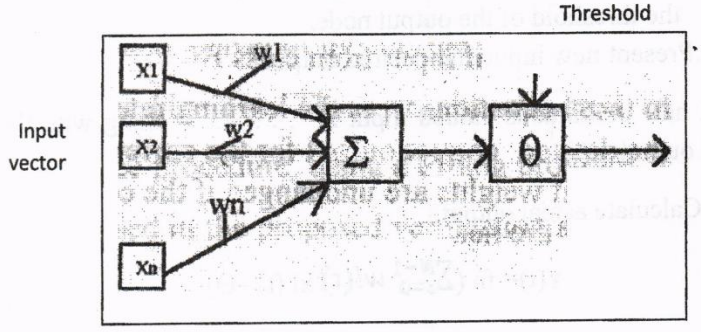


Fig .1 Perceptron Structure.

The perception can be trained on sample input/output pairs until it learns to compute the correct function of the specified logic gate . This training is done by the ***Perceptron Convergence Procedure .*** This learning algorithm is a search algorithm --it begins with a random initial state and finds a solution, state .The search space is simply all the possible assignments of real values to the weights of the perceptron, and the search strategy is the gradient descent [7].

Gradient descent as employed by neural networks, is a learning strategy, analogous to inductive techniques in symbolic .AI .The perceptron convergence procedure, according to Rosenblatt, guarantees the finding of a solution .

This Procedure as stated in reference [3], is shown below:

Step 1 .Initialize weights and threshold.

Set wi(0) (0 < = i < =N-l)and Θ to small random real values .Here wi(t)is the weight of input i at time t and 0 is the threshold of the output node.

Step 2 .Present new input and desired output.

Present new continuous valued input xl . . . .Xn-1 along with the desired output d(t)

Step 3 .Calculate actual output.

$$Y(t)= fn\left(\sum_{i=0}^{N-1} wi(t)\ xi\ (t) - \Theta\right)$$

Step 4.Adapt weights.

$$W_i(t)= W_i((t)+(\eta[d(t)- y(t)] xi(t)$$
$$0<=i<=N-1$$

{ +1 if input from class A d(t)= { -1 if input from class B

In these equations □ is the learning rate and d(t) is the desired correct output for the current input .Note that weights are unchanged if the correct decision is made by the net.

Step 5 .Repeat by going to Step 2.

The process of teaching the perceptron was applied on two computers, requires time and iterations as shown in Table (1).

Table (1) The required time and iterations for learning process.

Rate of Convergence	Number of Iterations	Learning time on 386DX/4QMHz	Learning time on 286/10MHz
75%	45-55	0.30 -0.35 sec.	1.0-1.5 sec.
90%	85-95	0.40 -0.45 sec.	1.75 -2.25 sec

2-1 The limitations of the perceptron

Minsky and Papert[38].noticed that while convergence procedure guaranteed correct classification of linearly separable data, there are problems which do not supply such data (for example XOR problem) .There is no one straight line which separates the decision surface of XOR problem as shown in Fig.2 (a) .Other logic gates can be modeled

by perceptron because their decision surfaces are linearly separable as shown in Fig (2) (b,c). To overcome the problem of XOR gate, it can be built from other gates such as NAND gates .The main advantage of the perceptron model is the self-organization of the neuron and the ability to adapt itself for various numbers of inputs.

Examples of learned perceptrons for the functions of two inputs logic gates are shown in Fig.3, These results obtained by simulating the perceptron convergence procedure, using a Prolog program, and the results are saved in order to be used in the proposed verification algorithm.

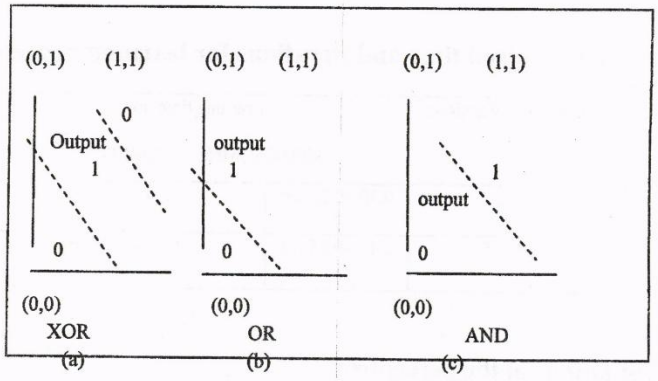


Fig 2 logig gates decision surface

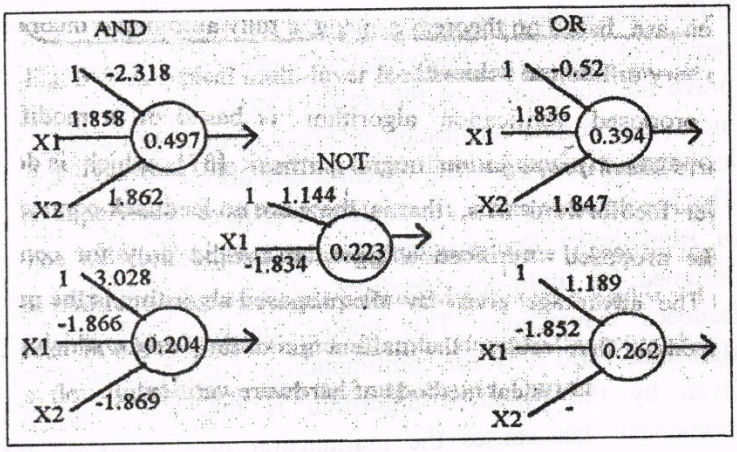


Fig 3 logic gates simulated by perceptron

3- A technique for hardware verification

The emergence of neural networks brought about the researches in the field of CAD systems to utilize the advantages of neural networks to improve the performance of such systems. The aim of the present work is to utilize the concepts of neural networks in the above mentioned field, especially in the area of hardware design verification.

The proposed verification algorithm is an attempt to transfer design automation techniques (in particular design verification) from conventional computers to neurocomputers, since there are a

number of attempts to build such computers (see references [4], [8]). In this approach, we present the utilization of neural networks concepts in a field of hardware verification, to overcome the difficulties embedded within classical verification methods, which require of a thorough knowledge of algebra or high order logic to prove the correctness of the design. Since most of the classical methods of formal verification are based on theorem proving, a fully automated theorem proving system is very difficult to achieve.

The proposed verification algorithm is based on a modification of Rumelhart's ***Back-propagation*** neural network [7], which is designed for multi-layer feedforward nets, that is, there are no feedback connections. This makes the proposed verification algorithm valid only for combination a; circuits. The advantage given by the proposed algorithm is the prediction of the component that causes the malfunction of the design, which provides an extra task over the classical methods of hardware verification.

3-1 Back-propagation neural network

The back-propagation neural network is one of the most important historical developments in neurocornputing .It is a powerful mapping network that has been successfully applied to a wide variety of problems ranging from credit application scoring to image compression [9] .Fig.4 shows a typical multi-layer feedforward neural network.

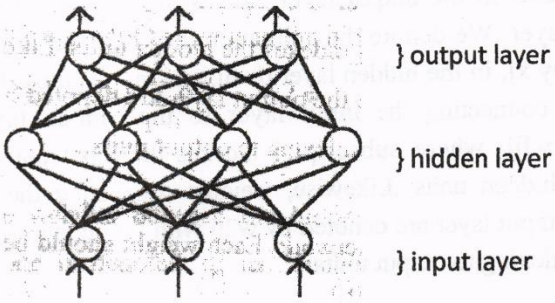


Fig.4 A typical multi-layer feedforword neural network

Like a perceptron, the learning algorithm of back-propagation neural network is supervised by a teacher .Hence, it starts out with a random set of weights .The network adjusts its weights each time it sees an input/output pair .Each pair requires two stages :a forward pass and a backward pass .The forward pass involves presenting a sample input to the network and letting activations flow until they reach the output layer .During the backward pass, the network's actual output is compared to the desired output, and error estimates are computed for the output units .The weights connected to the output units can be adjusted in order to reduce these errors .Then, we can use the error estimates of the output units to derive error estimates for the units in the hidden layers .Finally, errors are propagated back to the connections stemming from the input units [7] .The learning algorithm for back-propagation neural network as stated by reference [7], is shown below:

1 :Let A be the number of units in the input layer, let C be the number of units in the output layer, and B the number of units in the hidden layer .We denote the activation levels of the units in the input layer by xj, in the hidden layer by hj, and in the output layer by oj .Weights connecting the input layer to the hidden layer are denoted by wljj, where subscript i indexes the input units, and j indexes the hidden units .Likewise, weights connecting the hidden layer to the output layer are denoted by w2ij with i indexing to hidden units and j indexing to output units.

2 :Initialize the weights in the network .Each weight should be set randomly to a number between -0.1 and 0.1.

Wlij= random(-0.1 ,0.1) for all i =O . . . A , j =O . . . B

w2ij = random(-0.1 , 0.1) for all i =0 . . .B , j =0 . . .C

3 :Initialize the activations of the thresholding units .These should never change their values.

4 :Choose an input/output pair .Suppose the input vector is xi and the desired output vector is yi .Assign activation levels to the input units.

5 :Propagate the activations from the units in the input layer to the units in the hidden layer, using the activation function

$$h_j = \frac{1}{1 + e^{-(\sum_{i=0}^{A} w1_{ij} \cdot x_j - \theta)}}$$

6 :Propagate the activations from the units in the hidden layer to the units in the output layer, using the activation function :

7 :Compute the errors of the units in the output layer, denoted 52j . Errors are based on the network's actual output (oj) and the desired output (yj).

$$\delta 2\, j = oj(1 - oj)(yj - oj) \qquad \text{for all } j=1 \ldots C$$

8 :Compute the errors of the units in the hidden layer denoted 5lj.

$$\delta 1j = hj(1 - hj)\sum \delta 2\, j.w2ji \qquad \text{for all } j=1 \ldots B$$

$$i=o$$

9 :Adjust the weights between the hidden layer and the output layer .The learning rate is denoted T|; its function is the same as in perceptron learning. A reasonable value of T| is 0.35.

$$\Delta w2ij = \square \bullet \delta 2j \bullet hj \qquad \text{for all } i=0 \ldots B,\ j=1 \ldots C$$

10 :Adjust weights between the input layer and the hidden layer.

$$\Delta wlij = \square \bullet \delta 1j \bullet xi \qquad \text{for all } i=0 \ldots A,\ j=1 \ldots B$$

11 :Go to step 4 and repeat.

The algorithm described above, is a generalization of the Least Mean Square (LMS)algorithm .It uses a gradient technique to minimize a cost function equal to the mean square difference between the desired and actual net outputs .An essential component of the algorithm is the iterative method described in the algorithm, that propagates error terms required to adapt

weights back from nodes in the output layer to nodes in lower layers [3].

Back-propagation suffers from several drawbacks .As a biological model, it is implausible; there is no evidence that synapses can be used in both directions .Even so, it does not seem readily believable that neurons can spread signal-activity levels or error signals — using nonlinear or linear input-output functions depending on the direction of movement .Nevertheless, from the point of view of computer engineering and connectionist modeling, biological implausibility is not a fundamental deficiency of back-propagation .Back-propagation's real limitations are those of all gradient-descent techniques, namely the possibility of getting stuck in local minima and a slow convergence rate[8].

3-2 The Proposed Algorithm

The proposed algorithm is based on the idea of representing each gate in the combinational circuit under consideration by a single perceptron, and then connecting these perceptrons in orde "to construct the multi-layer network corresponding to the circuit under consideration .Then applying a modified version of back-propagation algorithm to evaluate the output of the network (according to design specification).

Notice that we use perceptrons in order to eliminate the iterated steps of back-propagation algorithm, that consumed in learning the behavior of the problem .That means if the design matches the specification, then only one forward pass will be applied for each input/output pair of the design specifications .In case of mismatch of the design with its specification, at this time, the design is known

to be incorrect .Now, the extra task (prediction part); over the classical verification methods will be performed, which is represented by a backward pass, performed in order to compute the errors in the neurons of each layer (the same happens in back-propagation algorithm) .Then, these errors will be sorted in decreasing order, and the neuron with the highest error is considered as the niost probable neuron (component)that causes the malfunction in the design .The proposed algorithm is given below:

1-Simulate the logic circuits as à network of perceptrons(using Perceptron Convergence Procedure to teach each perceptron the function of one of the standard logic gates).

2-Choose an input/output pair from the set of design specification .Suppose the input vector is xi, and the target output vector is yi .Assign activation levels to the input units.

3 -Propagate the activations from the units in the input layer to the units in the first hidden layer, using the activation function hj.

$$h_j = \frac{1}{1 + e^{-(\sum_{i=1}^{B} w_{ij} \cdot x_i + \theta}}$$

Where A is the number of units in the input layer, which are connected to the unit in the first hidden layer indexed by j .Note that 6 is the threshold value of the neuron under consideration, wij is the connection weight.

4 -Propagate the activations from units in the first hidden layer to the units in the next hidden layer up to the output layer, using the activation function Oj.

$$O_j = \frac{1}{1 + e^{-(\sum_{i=1}^{B} w_{ij} \cdot x_i + \theta}}$$

Where B is the number of neurons in the previous layers connected to the unit under consideration, which is indexed by the subscript j.

5 -If the actual output matches the desired output, go to step 10.

6- Compute the errors of the units in the output layer denoted by:

Oe = Oj(1 - Oj)(yj - Oj)

7- Compute the errors of each unit in the network from hidden layers down to the input layer denoted by :

$$H_e = h_j(1 - h_j) \sum_{i=1}^{N} O_e \cdot W_{ji}$$

where N represents the number of units in the higher layers than the layer of the unit under consideration, which is indexed by the subscript j.

8- Sort the computed errors of the network neurons in decreasing order. The neuron (gate) with the highest error value is the most probable

9- neuron that causes the malfunction of the design, and so on.

Terminate to redesign.

10- If specifications are not finished, go to step 2, else terminate with matched design.

The proposed algorithm consists of two major parts, the first one (step 1). performs the simulation of the circuit under consideration. We use Perceptron Convergence Procedure, to simulate the functions of st?"dard logic gates, and the perceptrons resulting from this simulation could be saved to be used in the first step of the algorithm. Hence, we use perceptron convergence procedure only once, and each time we need to simulate the circuit under consideration. we have the perceptrons corresponding to the gates of this circuit.

The simulation of logic circuits will be done by representing each gate in the circuit by its corresponding perceptron (neuron). Thus, a constructed network corresponds to the logic circuit under consideration.

The second part (step 2 - step 10), represents the verification part of the simulated circuit. As shown above, the verification method is based on the learning algorithm of back-propagation neural networks.

The verification part, in turn, is subdivided into two passes: forward pass and backward pass. In the forward pass (steps 2-5), an input/output pair, and the activations of the neurons will be propagated up to the output layer. The formula in step 3 propagates the activations from the input layer to the first hidden layer in the network. Then the formula in step 4, is used to propagate the activations from the first hidden layer up to the output layer. Note,

there is a difference between the equations of steps 3 and 4, and the corresponding steps in the back-propagation algorithm, which is the adding of the threshold value in the proposed algorithm; and subtracting this value in back-propagation algorithm. This difference conges to increase the convergence of the produced output of the simulated network to the desired output (logic 1 or logic 0).

Now, the actual output will be computed, which its a real value results from the neural network corresponding to the circuit under consideration. Noting that, if this real value is larger than (0.5), it will be considered as logic 1; if it is smaller than (0.5), then it shall be considered as logic 0. So, if the actual output obtained from the network matches the desired output, then another pair will be chosen, and the above steps are repeated again. If the actual output does not match the desired one, this means there is no matching between the desired specification of the circuit and its actual input/output pattern. Then, there is an incorrect design according to the specification.

This will lead to performing the backward pass (steps 6-9), to compute the errors in the neurons and therefore predict the gate that causes the malfunction in the design. The errors in the output layer will be computed using the formula in step 6, and then computing the errors in all hidden layers up to the input layer using the formula in step 7 (this corresponds to error signals propagation in back-propagation algorithm).

The prediction technique of the gate that causes the malfunction in the design, will be performed by sorting the errors of

network neurons in decreasing order (step 8). Then, the first neuron in the sorted list is the most probable neuron (gate) that causes the malfunction in the design (the most probable component that needs to be changed in the design).

However, the prediction part of this algorithm, is not so accurate, but it helps the designer to debug the design in less time than the time required for design debugging in the case of no prediction at all. The inaccuracy of the prediction comes from an important point -- the random numbers generated in (step 1) to simulate the function of each logic gate. Simulating these gates b>different sets of perceptrons (weight vectors and thresholds), will lead to different predictions in the design. Some of these sets converge to the corrcct prediction, and others diverge from it.

The time needed for this algorithm is at most $O(k \cdot 2^n)$, where n represents the number of input variables. So, (2^n) represents the input/output pairs of the design specifications. The constant (k) represents the number of calculations performed by the algorithm for each input/output pair given to the network. Thus, the proposed verification algorithm requires exponential time However, even with this disadvantage, this algorithm is favorable when compared with the classical methods of hardware verification, which deals with theorem proving techniques to prove the correctness of the design, and these methods are difficult to be computerized or even solved by hand.

Another important point to note is that, the techniques used to derive the specifications of the circuits by the user do not require much time: when the number of input variables is

small, a truth table method can be used t specify the behavior of the circuit. However, the required time in this method is exponential, but. it is not of great concern. When the number of input variables is large (which leads to a large exponential time in truth table method) the specification of such circuit will be given by the second method, which uses ***Propositional Logic*** implemented by Prolog to state the behavior of the circuit. These two methods will be discussed in the next chapter in more detail.

Attempts have been made to enhance the prediction part of the algorithm. They are concentrated on two ways. The first one uses probabilistic rules, especially the ***Bayesian Statistical Form*** [10]. The second concentrates on the prediction of the path that causes the malfunction of the design, by using the ***Critical Path Calculations*** as used in networks for Project cheduling: a method of Operations Research [11]. But these attempts did not produce any improvement to the already obtained results.

4- Verification example

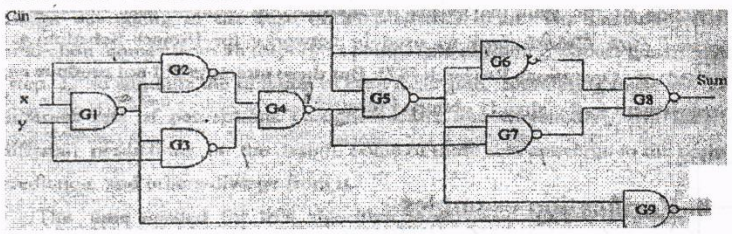
To show the applicability of the proposed algorithm in verification, we will consider ano example, which is solved by hand. It is the well-known full-adder circuit

4-1 Example

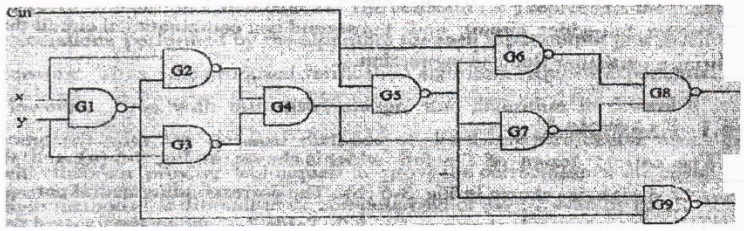
The correct design of the full adder is shown in Fig. 5 (a), and the incorrect design is shown in Fig. 5 (b). The corresponding neural network of the incorrect design is shown in Fig. 6. Table(2) shows the forward part of the verification algorithm to compute the

actual output. Table (3) shows the backward part of the algorithm to compute the errors of neurons in the network. The table also shows the sorted list of these neurons according to their error values.

Note that the prediction of the component that causes the malfunction diverges from the correct prediction (Gate 4). However, the prediction is not so accurate, but it can help the designer in case of large circuits, since this prediction reduces the debugging time of the design by at least 50% of the debugging time performed by the designer in the classical methods of hardware verification.



-a-



-b-

Fig 5 full-adder circuit. (a) correct design, (b) incorrect design

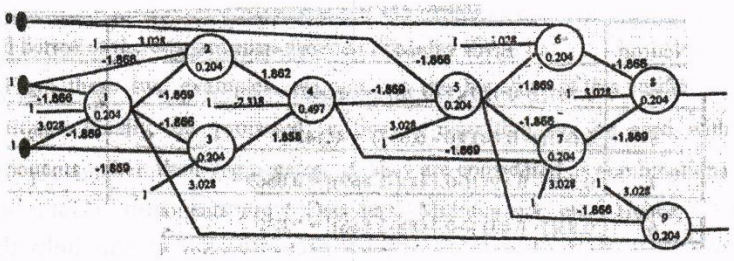


Fig. 5 Full-adder Fig. 6 The corresponding neural network for the incorrect full-adder circuit

Table 2 : The activations of network neurons.

Neuron	Weighted sum	h j & oj
1	1(3.028)+ 1 (-1.866) + 1(-1.869) + 0.204 = -0.503	0.377
2	1(3.028) +!(-!. 866)+ 0.377(-1.869) + 0.204 = 0.66	0.66
3	1(3. 028) +!(-!. 869) + 0.377(-1. 866) + '0.204 - 0.659	0.66
4	1(-2.318) + 0.66(1.862) + 0.66(1.858) + 0.497 - 0.635	0.654
5	1(3.0281+ 0(-1.866) + 0.654(-1.869) + 0.204 = 2.01	0.882
6	1(3.028) + 0 (-1.866) + 0.882(-1.869) + 0.204 = 1 . 5 8	0.83
7	1(3.028) +0.882(-1.866) +0.654(-1.869) +0.204 =0.364	0.59
8	1(3.028) +0.83(-1.866) +0.59(-1.869) + 0.204 = 0.581	0.641*
9	1(3.028) +0.882(-1. 866) +0.377(-1.869) +0.204 - 0,88	0.707

*: The activation value of neuron 8 is 0.641 , that is larger than or equal 0.5, which corresponds to logic 1. According to the correct design this value must be logic 0, that means a value less than 0.5.

Table(3) : The errors in network neurons.

Neuron	Error values in network neurons	Sorted list
9	0.707(1- 0.707)(1- 0.707) - 0.0607	2
8	0.641(1- 0.641)(0 - 0.641) = -0.148	9
7	0.59(1- 0.59) [(-0.148X-1.869)! = 0.0669	1
6	0.83(1- 0.83) [(-0.148X-1.866)] = 0.039	3
5	0.882(1- 0.882) [(0.039)(-1.869) + (0.0669)(-1.866) + (0.0607)(-1. 866)] = -0.0324	8
4	0.654(1-0.654) [(-0.0324)(-1.869) + (0.0669)(-1.866)]= -0.0145	7
3	0.66(1- 0.66) [(-0.0145)(1.858J] = -0.006045	4
2	0.66(1- 0.66) [(-0.0145X1.862)] = -0.006058	5
1	0.377(1- 0.377) [(-0.006058)(-1.869) +' (-0.006045X-1.866) + (0.0607)(-1.869)] =.-0.0213	6

Conclusions

No solution exists until now, which uses neural networks for complicated problems such as the mathematical proving. In our work we have attempted to utilize neural networks to solve the problem of hardware verification, and some success has been achieved. This was due to the use of the simulation technique in verification, since it is possible to model the verification by simulation technique using the concepts of neural networks.

As a new trend in CAD systems, this work comes with some embedded limitations. These limitations could be summarized as follows:

1- The verification algorithm is valid for combinational circuits only. This is due to the bases of the proposed verification algorithm. Hence, it's based on the learning algorithm for *back-propagation* neural networks which are feedforward networks.

2- The prediction technique used to predict the component that causes the malfunction in the design lacks the high accuracy in predictions made. The probability of the correct prediction ranges from 50%.

3- The verification time is exponential, that means a long time is needed if the number of inputs is large, hence the order of the proposed algorithm is $O(k.2^n)$.

Even with the above limitations there are some advantages in this work:

1- NeuroCAD provides two different ways of describing the hardware design, and two different methods of defining the specifications of this hardware design. This made the user feel free when choosing the methods he needs to describe his design.

2- Although the proposed algorithm has exponential complexity, it does not use the mathematical proving methods -- which might be of the same degree of complexity, or even higher, and a sophisticated task to be computerized.

3- The already mentioned point, that is introducing the concepts of neural networks to the field of verification.

4- This work gives the basics for implementing ***First-Order Predicate Logic*** as an HDL for the description and documentation of the hardware.

5- This work gives the basics for using ***Temporal Logic*** to describe timing relations among hardware modules.

6- This work may be considered as a step toward ***Distributed CAD Environments,*** depending on parallel processing architecture, in particular the ***Neurocoinputer*** architecture.

Refrences

1 -S. T. Clmkradhar, M. L. Bushnell, and V, D. Agrawal, *"TowardMassively/Parallel Automatic Test Generation"*, IEEE trans, on CAD, vol. 9, no. 9,September 1990.

2- **R. K. Miller,** ***"Neural Networks",*** Prentice-Hall Inc., U.S.A., 1990.

3- R. P. Lippmann, ***"An Introduction to Computing with Neural Nets", IE*** ASSP Magazine, vol. 4, no. 4, April 1987.

4- B. Soucek, and M. Soucek, ***"Neural and Massively Parallel Computers",*** John Wiley & Sons Inc. U.S.A., 1988.

5- B. Soucek, ***"Neural and Concurrent Real-Time Systems",*** John Wiley & Sons Inc., U.S.A, 1989.

6- T. Khanna, ***"Foundations of Neural Networks",*** Addison-Wesley Publishing Company Inc., U.S.A., 1990.

7- K.Knight, ***"Connectionist Ideas and Algorithms",*** Communication of the ACM, vol.33,no.11,November 1990.

8- J. R. Millan, and P. Bofill, *"Learning by Back-propagation: a Systolic.Algorithm and its Transputer Implementation", Neural Networks, vol. I, no.*3, Julv 1989.

9- R. Hecht-Nielson, *"Neurocomputing"*, Addison-Wesley Publishing Co. Inc.,U.S.A., 1990.^"

10-N.. Matloff, *"Probability Modeling and Computer Simulation"*, PWS-KENT Publishing Co., U.S.A.. 1988.

11-H. A. Taha, *"Operations Research"*, Macmillan Publishing Co., U.S.A.,1987.