ARTIFICIAL NEURAL NETWORKS FOR REALIZATION AND VERIFICATION OF DIGITAL LOGIC CIRCUITS

Reza Raeisi & Amanpreet Kaur, California State University—Fresno

Abstract

Most of the time the processing capability and efficiency of computers are compared to that of a human brain. This comparison lays the basic foundation for Artificial Neural Networks (ANN). In today's technological age, neural networks are finding applications in almost every field. In this paper, an application of an ANN approach is presented to implement digital circuits. This approach provides an easier structural method for verification of digital logic circuits. Additionally, information about various neural network modeling tools is given. Also, a software tool is used to design a 5-bit Arithmetic Logical Unit (ALU).

Introduction

An Artificial Neural Network (ANN) is a network of several processing units known as neurons, where the information about the inputs and outputs is stored in the network in the form of interneuron connections known as weights. Each neuron has a fixed threshold value. Knowledge is acquired by the system through a learning process and is stored in the form of synaptic weights. When an input is given to the neuron, via an interneuron connection, the summation of the product of the synaptic input weights forms the weighted input. Then, it is subtracted from the threshold value to generate the activation of the neuron. Eventually, the activation is passed through the activation function to produce the output. The important part of ANN modeling is training, during which the value for the synaptic weight is computed by correlating the input with the output. This training procedure is similar to the learning process of the human brain. According to our experiences and knowledge, the strength of interneuron connections, or synapses, is altered in our brain [1].

The most significant characteristic of ANN is its use in applications where the user does not know the exact relationship between input and output. In this study, this advantage of ANN was explored by designing combinational digital circuits. Even though the relationship between input and output is known in combinational digital circuits, it is sometimes very complex and requires a lot of computing. Therefore, in this study, only the inputs and outputs were used for training and designing ANN models equivalent to the combinational digital circuit. The benefit of doing so is that this kind of modeling can be used for digital circuit functional verification and testing.

A discrete neuron model element can assume one of two possible states, 0 or 1. Figure 1 shows a neural architecture, referred to as a perceptron, which serves as a building block for primitive digital logic gates. Each perceptron can have multiple inputs and one output.

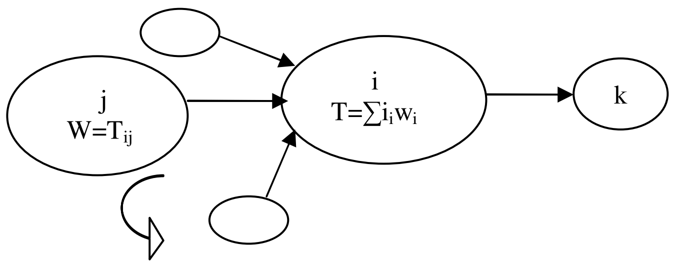


Figure 1. A Typical Neural Perceptron

A perceptron is connected to its neighbors through nets or signals similar to logic circuits and associated with a real number, T_{ij} , which connect perceptron *j* to *i.* A perceptron receives inputs from its neighbors and processes the inputs to produce its output function. A neural representation of digital logic gate primitives is shown in Figure 2 as a weighted graph, with the perceptrons represented as vertices and signals or nets as weighted edges. Each perceptron is also assigned a threshold. As an example, consider the neural network representation of a two-input AND gate shown in Figure 2. The AND gate neural element has three perceptrons labeled a, b, and c and whose thresholds are 0.5, 0.5 and 0.8, respectively.

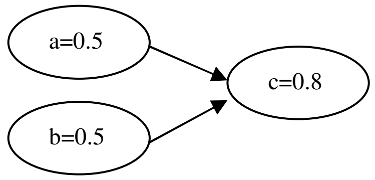


Figure 2. AND Neural Representation

A digital logic circuit with N gates can be realized with a neural network with N perceptrons, where the interconnections between the perceptron elements represent a specific Boolean logic expression. As an illustration, consider a neural network realization of an AND gate, with positive or negative weights for edges in the range of -1 and 1, and the threshold to be assumed in the range of -1 to 1. In an AND operation, both of the inputs need to be adequately high in order to produce an output. Assuming a threshold value of 0.8 for perceptron c, and setting the weights of the two edges at 0.5, then if both inputs of the AND gate are 1, the perceptron will have an activation output function of (1 * 0.5 + 1 * 0.5), which is greater than the output perceptron threshold and causes the perceptron to fire.

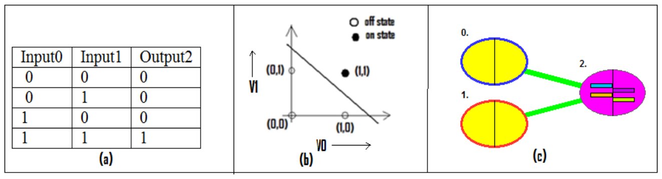
Modeling Digital Gates using ANN

The input and output signal states of a logic gate can be expressed in terms of minimum-energy function states, defined for an ANN. Similarly, a digital circuit can be expressed by a single energy function. In the development of an ANN model for digital circuits, it is important to determine if it is going to be a single layer or multiple layer ANN model. The decision variable for deciding the number of layers in ANN modeling is Decision Hyperplane. A hyperplane associated with neuron i can be expressed by Equation (1) [2]:

$$I_i + \sum_{j=1}^{n} T_{ji} V_j = 0 \qquad (1)$$

where I represents the threshold of the neuron; T represents the interconnection weight between neuron i and neuron j; and, V_i is the activation value of neuron i.

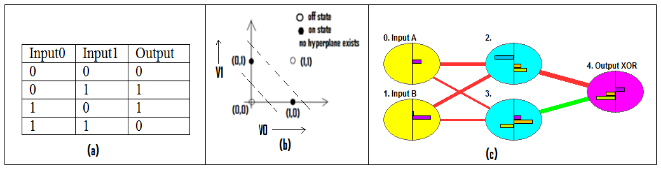
A hyperplane associated with neuron i is termed as decision hyperplane if the ON(1) and OFF(0) states of the system lie on the opposite sides of hyperplane and all other points lie on it. In such a case, ANN will form a single layer network. Otherwise, a multilayer ANN is formed. For instance, consider the case of a logical AND gate, the truth table for which is shown in Figure 3(a). In this case, the ON and OFF states of the AND gate can be separated by using a hyperplane, as shown in Figure 3(b). So, a single layer neural network is formed, as represented by Figure 3(c). This is also true for OR, NAND, NOR, and NOT gates.



Input0	Input1	Output2
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3. Neural Modeling of an AND Gate

In the case of an XOR gate, however, the ON and OFF states cannot be separated using a single hyperplane, as shown in Figure 4(b). So it forms a double layer network. An XNOR would represent a similar case.



Input0	Input1	Output
0	0	0
0	1	1
1	0	1
1	1	0

Figure 4. Neural Modeling of an XOR Gate

Designing an ANN Model

Once there is a basic understanding of the concept, there are various ways to design neural networks. Users can design and train neural networks using programming languages such as C++ , C# , Python, and Java. Beyond programming there are various software tools available which can be used to design neural networks. Some of them are listed here:

1. Matlab-Neural Network Toolbox
2. EasyNN-plus
3. Java Object Oriented Neural Engine
4. NNDef-Toolkit
5. Sharky Neural Network
6. A.I.Solver Studio
7. C# Neural network library

Amongst all these tools, EasyNN-plus was selected because it is particularly user friendly, and multilayer neural networks can be designed; the details of the neural network can subsequently be imported as text files. The only drawback with this tool is that it designs only feed-forward ANN models. In this study, 1-bit and 5-bit ALUs were designed using this tool.

Designing a 1-Bit ALU using EasyNN-plus

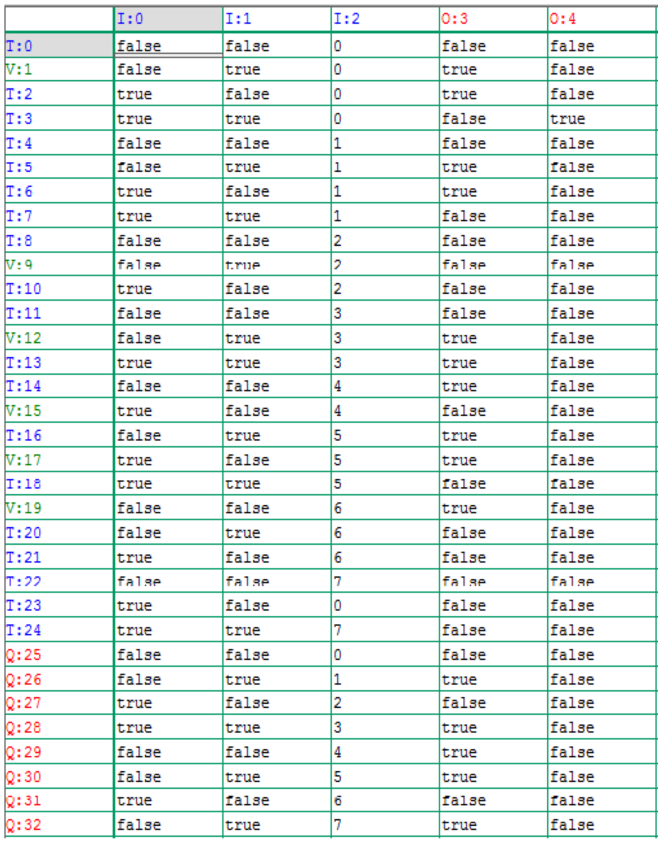
To design a 1-bit ALU using EasyNN-plus, the only thing required is ALU realization and then training the data for a functional 1-bit ALU and then just setting the required parameters. Once the parameters are set, the neural network can be designed and, eventually, verified by adding the query data to the training data. The opcodes used for designing this 1-bit ALU have been tabulated in Table 1. Using these opcodes, the data used for training, validating, and querying the neural network model are shown in Table 2.

Table 1. Opcode for a 1-Bit ALU

OPCODE	OPERATION
0	ADD
1	SUB
2	AND
3	OR
4	NOT
5	NAND
6	NOR
7	XOR

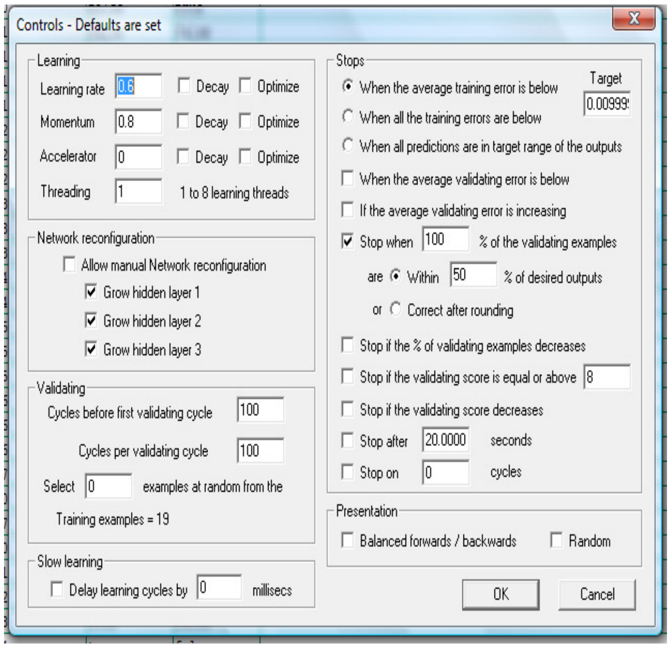
In Table 2, columns I:0 and I:1 represent the 1-bit inputs to the ALU; I:2 represents the opcodes; O:3 is the output; and O:4 represents the carry-over flag. In addition, the rows labeled T:n, where n is the row number, are the training examples. Examples used to validate the neural network are labeled V:n, where n is again the row number.

Table 2. Training for a 1-Bit ALU



	I:0	I:1	I:2	O:3	O:4
T:0	false	false	0	false	false
V:1	false	true	0	true	false
T:2	true	false	0	true	false
T:3	true	true	0	false	true
T:4	false	false	1	false	false
T:5	false	true	1	true	false
T:6	true	false	1	true	false
T:7	true	true	1	false	false
T:8	false	false	2	false	false
V:9	false	true	2	false	false
T:10	true	false	2	false	false
T:11	false	false	3	false	false
V:12	false	true	3	true	false
T:13	true	true	3	true	false
T:14	false	false	4	true	false
V:15	true	false	4	false	false
T:16	false	true	5	true	false
V:17	true	false	5	true	false
T:18	true	true	5	false	false
V:19	false	false	6	true	false
T:20	false	true	6	false	false
T:21	true	false	6	false	false
T:22	false	false	7	false	false
T:23	true	false	0	false	false
T:24	true	true	7	false	false
Q:25	false	false	0	false	false
Q:26	false	true	1	true	false
Q:27	true	false	2	false	false
Q:28	true	true	3	true	false
Q:29	false	false	4	true	false
Q:30	false	true	5	true	false
Q:31	true	false	6	false	false
Q:32	false	true	7	true	false

Finally, the query data are given by the user to verify the neural network model. In case of a query, the neural network model tries to predict the output for the given set of inputs. In this case, by giving a query and then verifying its output, the user can determine whether the neural network meets minimum error conditions or not. The other important factor is setting up the control values like average training error, number of layers, and learning rate. The dialog box for setting the controls is shown in Figure 5.



Controls - Defaults are set

Learning
- Learning rate: 0.6 — ☐ Decay ☐ Optimize
- Momentum: 0.8 — ☐ Decay ☐ Optimize
- Accelerator: 0 — ☐ Decay ☐ Optimize
- Threading: 1 — 1 to 8 learning threads

Network reconfiguration
- ☐ Allow manual Network reconfiguration
- ☑ Grow hidden layer 1
- ☑ Grow hidden layer 2
- ☑ Grow hidden layer 3

Validating
- Cycles before first validating cycle: 100
- Cycles per validating cycle: 100
- Select 0 examples at random from the Training examples = 19

Slow learning
- ☐ Delay learning cycles by 0 millisecs

Stops
- ● When the average training error is below — Target 0.00999
- ○ When all the training errors are below
- ○ When all predictions are in target range of the outputs
- ☐ When the average validating error is below
- ☐ If the average validating error is increasing
- ☑ Stop when 100 % of the validating examples are ● Within 50 % of desired outputs or ○ Correct after rounding
- ☐ Stop if the % of validating examples decreases
- ☐ Stop if the validating score is equal or above 8
- ☐ Stop if the validating score decreases
- ☐ Stop after 20.0000 seconds
- ☐ Stop on 0 cycles

Presentation
- ☐ Balanced forwards / backwards
- ☐ Random

Figure 5. ANN Controls for Designing a 1-Bit ALU

After setting up the controls, the designer would press the "Start learning" button in the main panel window and the software will start designing the neural network and stop when the minimum validating error time is below the set value. The ANN model realization of a 1-bit ALU is shown in Figure 6. The details of this model are shown in Figure 7. The average training error was 0.13.

Designing a 5-Bit ALU

The design of a 5-bit ALU is similar to that of a 1-bit. Figure 8 shows the opcode table and control values for a 5-bit ALU. For this case, 11 input columns were used—ten columns for input data and one for the opcode. Using a procedure similar to the one described earlier, a neural network was designed and is shown in Figure 9. The average training error was 0.141056 for the training process of this 5-bit ALU neural network.

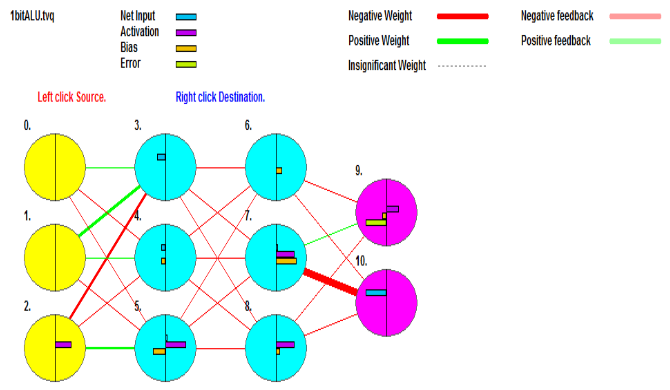
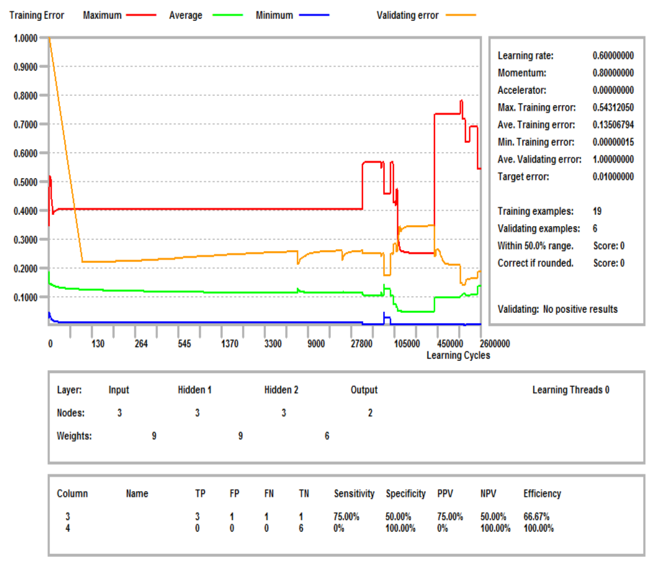


Figure 6. Neural Network Model for a 1-Bit ALU

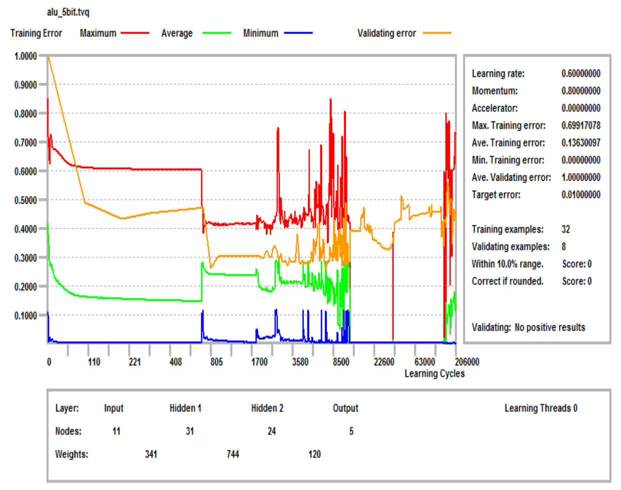


```
Learning rate:            0.60000000
Momentum:                 0.80000000
Accelerator:              0.00000000
Max. Training error:      0.54312050
Ave. Training error:      0.13506794
Min. Training error:      0.00000015
Ave. Validating error:    1.00000000
Target error:             0.01000000

Training examples:        19
Validating examples:      6
Within 50.0% range.       Score: 0
Correct if rounded.       Score: 0

Validating:  No positive results
```

```
Layer:    Input      Hidden 1      Hidden 2      Output              Learning Threads 0
Nodes:      3           3             3             2
Weights:         9             9             6
```

```
Column    Name    TP    FP    FN    TN    Sensitivity    Specificity    PPV       NPV       Efficiency
  3               3     1     1     1     75.00%         50.00%         75.00%    50.00%    66.67%
  4               0     0     0     6     0%             100.00%        0%        100.00%   100.00%
```

Figure 7. A 1-Bit ALU ANN Model



```
Learning rate:            0.60000000
Momentum:                 0.80000000
Accelerator:              0.00000000
Max. Training error:      0.69917078
Ave. Training error:      0.13630097
Min. Training error:      0.00000000
Ave. Validating error:    1.00000000
Target error:             0.01000000

Training examples:        32
Validating examples:      8
Within 10.0% range.       Score: 0
Correct if rounded.       Score: 0

Validating:  No positive results
```

```
Layer:    Input      Hidden 1      Hidden 2      Output              Learning Threads 0
Nodes:     11           31            24            5
Weights:        341           744           120
```

Figure 8. Details of a 5-Bit ALU

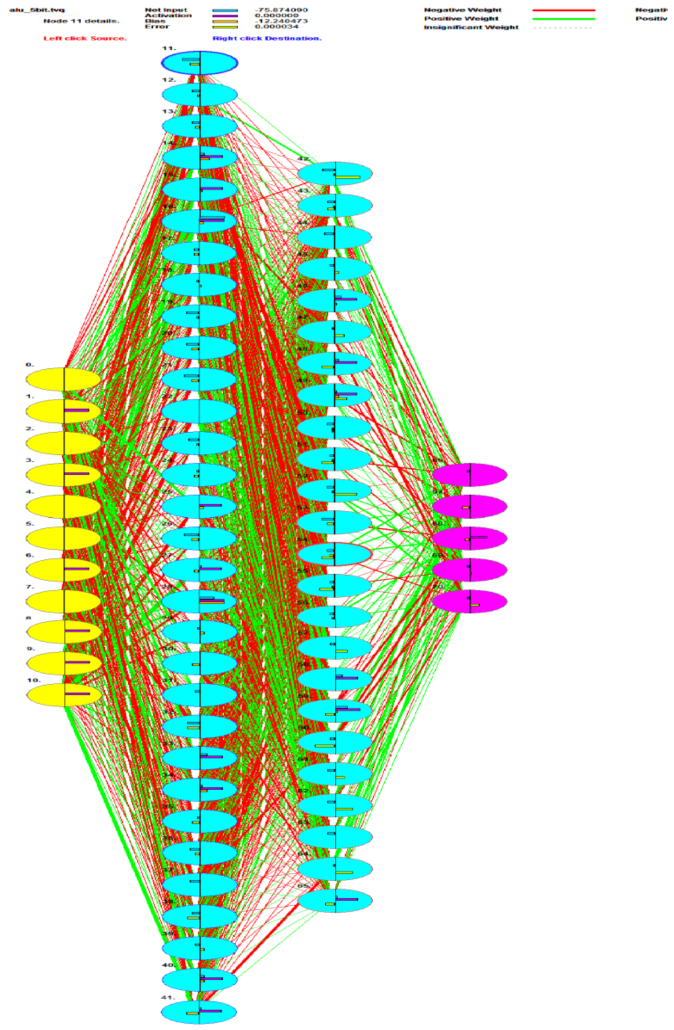


Figure 9. Neural Network Model of a 5-Bit ALU

Future Work

The Boolean Difference concept [3] was evaluated along with the proposed ANN digital logic realization technique for the testing of combinational circuits. The basic principle of the concept is to drive two Boolean expressions—one of which represents normal, fault-free behavior of the circuit, while the other represents logical behavior under an assumed single stuck-at-1 or stuck-at-0 fault condition. These two expressions are then realized using the ANN methodology for verification and Automatic Test Pattern Generation (ATPG) based on energy optimization of a Hopfield Neural Network [4], [5]. The block diagram for the proposed ATPG structure is shown in Figure 10 and consists of the following major components:

1. Circuit description containing a hard-coded form of the complete description of the circuit.

2. Energy surface generator to generate the energy surface for the circuit based on the Hopfield energy equation.
3. Test circuit is the circuit under test to be checked for faults.
4. Test circuit state table contains the functional state of the test circuit for the given inputs.
5. Fault Processing Unit takes the data from the energy surface and test circuit state table and processes them to determine if faults exist.

The implementation of the ATPG phase of this research work is currently under exploration.

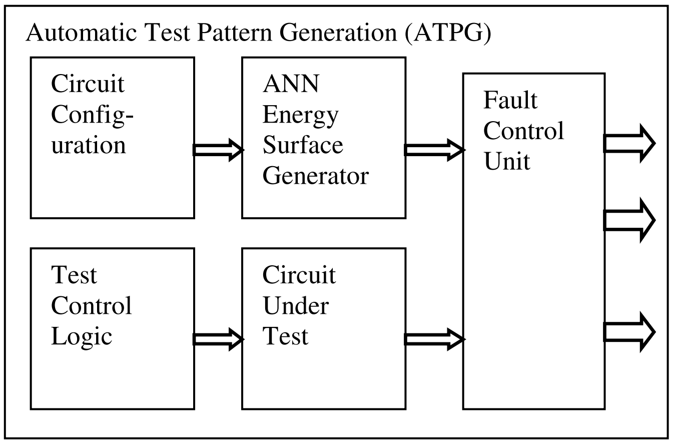


Figure 10. Block Diagram of ATPG

Conclusion

This paper presents a methodology for realization and functional verification of digital logic circuits using ANN as the building block to model digital logic circuits in a hierarchical method. Using ANN logical blocks, various combinational circuits were successfully modeled. An example of designing a 5-bit ALU with its functional verification was successfully implemented. The next phase of this research project is underway to use the ANN-modeling of digital circuits to test for faults and delays and generate automatic test patterns for a given circuit.

References

[1] Russel, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

[2] Chakradhar, S. T., Agrawal, V. D., & Bushnell, M. L. (1991). *Neural Models and Algorithms for Digital Testing*. Kluver Academic Publishers.

[3] Lala, P. K. (1985). *Fault Tolerant & Fault Testable Hardware Design*. Prentice Hall International.

[4] Agrawal, V. D., & Seth, S. C. (1988). *Test Generation for VLSI Chips*. IEEE Computer Society Press.

[5] Hopfield, J. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, 52, 141-152.

Biographies

REZA RAEISI is currently an Associate Professor of Electrical and Computer Engineering Department at California State University, Fresno. He has been also serving as the Graduate Program Coordinator for the ECE department since 2008. He received his Ph.D. in 1990 and MS degree in 1985 both in computer engineering from University of Cincinnati. His research interests include integrated circuits, embedded systems, Application of Artificial Neural Networks in VLSI testing, and VLSI-CAD technology. He serves as Pacific Southwest regional director of American Society of Engineering Education. He is a Coleman Fellow and entrepreneur with over 20 years of domestic an international experience and professional skills in both industry and academia. Dr. Raeisi may be reached at rraeisi@csufresno.ed

AMANPREET KAUR is currently a Ph.D. student at University of California, San Diego. She received her MS in electrical engineering from California State University, Fresno in 2010. Her present research interest are application of Artificial Neural Networks and Genetic Algorithms for load forecasting and methods for reliable integration of variable resources like solar energy to the present grid. Ms. Kaur may be reached at a3kaur@ucsd.ed