Design and Realization of FPGA based Off-Chip Trained MLP for Classical XOR Problem and Need of On-Chip Training

K. Packia Lakshmi
II yr M.E student
M.E Applied Electronics
Einstein College of Engineering, Tirunelveli-12

M. Subadra, PhD.
Associate Professor & Head
Department of Electronics & Communication Engg.
Einstein College of Engineering,Tirunelveli-12.

ABSTRACT

The main intension of this work is to present the importance of neural chip with learning capability. The designed sequentially trained MLP structure is used to solve the classical XOR problem and the structure is realized on FPGA device environment. By comparing the device utilization summary for the design in different families of Xilinx FPGA, the importance of platform selection for hardware implementation is presented. Finally the importance of on-chip learning is emphasized.

General Terms

Hardware based learning, Pattern classification

Keywords

ANN, FPGA, MLP, Off-chip learning, On-chip learning

1. INTRODUCTION

Artificial Neural Network (ANN) is an important soft computing tool. ANN is an adaptive statistical model which resembles the human brain activities. Artificial neuron is the main information processor of the network; it may inspired by biological neuron activities. Unlike conventional computers, ANN does the process of learning to structure an analytical model. The designed analytical model solves given tasks based on previous experience with reasonable accuracy, at reasonable cost and in a reasonable amount of time [1-3].

An important characteristic of artificial neural network (ANN) is its inherent parallelism. All the neurons available in the same layer may work instantaneously. Hardware implementation may preserve this inherent parallelism [4], so computation speed may increase compared to sequential software implementation. Neural network hardware is usually defined as those devices structured to realize neural architectures and learning algorithms, especially those devices that take advantage of the inherent parallelism characteristic of ANN [17].

Reconfigurable FPGA architectures are suitable for hardware implementation of neural networks, because it preserves the parallelism characteristics of ANN [4], [12], [19].

The main goal of the network is to learn some association between input and output patterns, or to analyze, or to find the structure of the input patterns [1]. Basic structure of a neuron-perceptron model with 'n' input is shown in Fig.1

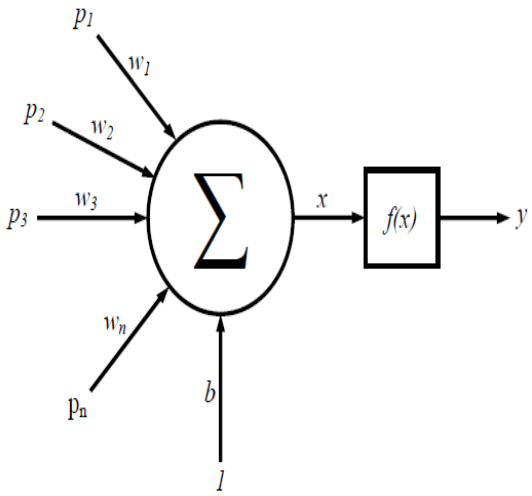

Fig 1: Structure of a neuron-perceptron model

The weighted sum value x is calculated as follows,

$$x = b + \sum_{i=1}^{n} p_i \; w_i \qquad \text{---------- [1]}$$

The output value is calculated by applying the activation function on weighted sum value as follows

$$y_k = f(x) \qquad \text{---------- [2]}$$

2. IMPLEMENTATION CONSIDERATION

The important characteristics of the network depend on

i. The activation functions of the neuron;
ii. Structure of the network;
iii. Learning mechanism of the network.

So during hardware implementation of ANN, these parameters should be considered for efficient implementation [4]. For classification task, Multilayer Perceptron architecture with back propagation learning algorithm is popularly used [5].

2.1 Network Structure

Multilayer Perceptron is the most important neural network model to solve the real world problem [3]. In Multilayer

Perceptron network, the processing elements are usually organized into several layers as input layer, hidden layer, output layer as shown in Fig.2. There is no standard available to define the number of neurons and number of hidden layer to be available in each layer to structure the network. Defining the architecture to obtain the required performance is a tedious process; usually trial and error method is used to structure the network architecture [6], [13], [16].

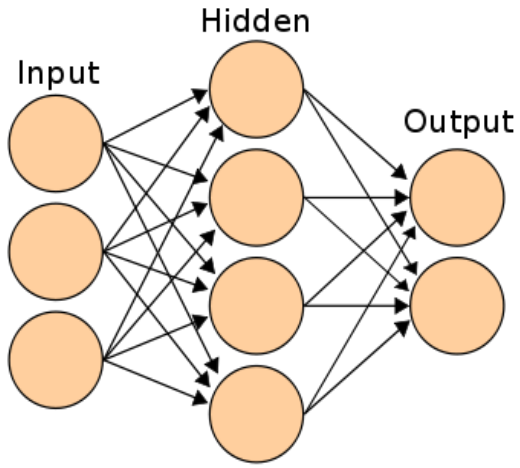

Fig 2: Multilayer perceptron architecture

Based on the input feature, count number of neurons in the input layer is defined. Test vector classes define the number of neurons in the output layer. The maximum number of neuron may be available in the hidden layer is {2(number of input feature) +1} [3]. In the network architecture if the number of neurons in the hidden layer is large in number it in turn increases the implementation complexity. So that maximum one hidden layer may enough to obtain the required performance [13], [16].

2.2 Activation Function

Generally neuron models use the same procedure to produce the total input signal, but they may differ in terms of how they produce an output response from this input. Artificial neurons use an activation function to compute their activation as a function of total input stimulus [1].

Activation functions are the mathematical formula, which is used to determine the output of a processing node. If the weighted input sum value is greater than the threshold value, means the neuron produces output. Several different functions may be used as an activation function.

Some activation functions are:

(i) Threshold activation function

(ii) Sigmoid activation function

(iii) Linear activation function

In Fig.1 f(x) denotes the activation function of the neuron. Activation functions such as sigmoid are mostly used because sigmoid activation function is continuously differentiable which are desirable for network learning [1-3].

2.3 Learning

Learning is the process of changing the weight of the network to acquire the desired behavior. During the learning process, set of training pattern is given to the network. In neural networks three types of learning is used. They are

(i) Supervised Learning

(ii) Unsupervised Learning

(iii) Reinforcement Learning

Generally, MLP network uses the supervised back propagation learning to train the network [5]. In supervised learning, the training pattern may have a set of input feature and its corresponding target output. When the network is converged, that is the difference between the target output and actual output becomes small then the training is stopped. Now the network is ready for the testing phase [3].

Due to its calculation power and simple effective concept is mostly used for pattern recognition application [5]. Popularity of BP algorithm mainly revolves around the MLP network to learn complicated multidimensional mappings.

3. BACKPROPAGATION ALGORITHM

Back Propagation (BP) Algorithm is an effective two pass multi layer supervised learning algorithm to train the multilayer perceptron network to obtain the solution for non-linearly separable problem [5-6].

There are two passes available in BP algorithm [3]. They are,

(i) Forward Pass

(ii) Feedback Pass

3.1 Forward Pass

MLP network is a fully connected network in which all the neurons in each layer are connected to all the neurons in the next layer.

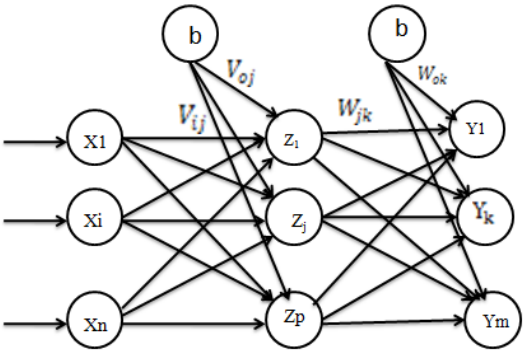

Fig 3: Generalized MLP architecture

Generalized MLP architecture is shown in Fig.3 [3].Where $\{X_1,\ldots X_i,\ldots X_n\}$ neurons available in the input layer,$\{ Z_1,\ldots Z_j,\ldots..Z_k \}$ neurons available in the hidden layer, $\{Y_1,\ldots Y_k,\ldots Y_m\}$, V_{oj}, W_{ok} - weight values of bias term. V_{ij}, W_{jk} - weight values between neurons.

In forward pass, the input data travelled from the input layer to output layer to obtain the output value at each processing element and the corresponding error value is calculated at the output layer.

During the training phase, the input vectors linearly travel through the input layer. The synaptic weight in between neurons determines the strength of the signal to be transferred. The ease of transmission of signal altered by the weighted signal is the basic for learning method

Processing steps involved in forward pass are given below,

(I) Parameter initialization. (weight, learning rate, and error value used for stopping condition)

(II) Check for the stopping condition. If the error value is equal to or lesser than the given error value means the current input vector is dropped and the next input vector is taken for forward pass. If the stopping condition is false, the consecutive steps are taken place.

(III)For each training pair do steps IV to IX.

(IV) Each input unit receives the input signal x_i and transmits the signal to all the units in the above layer, where i = 1to n.

(V) Each hidden unit Zj where j=1 to p have some weight value; Weighted sum value at each hidden node is calculated as,

$$Z_{inj} = V_{oj} + (\sum_{i=1}^{n} X_i \; V_{ij}) \quad \text{---------- [3]}$$

Applying its activation function to the weighted sum value to obtain the output of each hidden node and the result is forwarded to the above layers. The output value at hidden node is calculated as,

$$Z_j = f(Z_{inj}) \quad \text{---------- [4]}$$

(VI) Each output unit Y_k where k=1 to m have some weight value; Weighted sum value at each output node is calculated as

$$Y_{ink} = W_{ok} + \sum_{i=1}^{m} Z_j \; W_{jk} \quad \text{---------- [5]}$$

Applying its activation function to calculate the output signal as follows,

$$Y_k = f(Y_{ink}) \quad \text{---------- [6]}$$

Now the output value is compared with the target value to find the error value. After an error value calculation the values are fed backed and travel towards the input layer to do the weight updating process.

3.2 Feedback Pass

In feedback pass of BP algorithm, the error value is back propagated to do the weight update process for all the nodes available in the previous layers. Processing steps involved in feedback pass are given below,

(VII) For each output unit Y_k , where k=1 to m; 'T' is the target pattern corresponding to its input pattern. The error information calculated as,

$$\delta_k = (T_k - Y_k)\, f'(Y_{ink}) \quad \text{---------- [7]}$$

(VIII) For each hidden unit Z_j, where j=1 to p; Sums its δ_k inputs from the units in the layers above.

$$\delta_{inj} = \left(\sum_{k=1}^{m} \delta_k \, W_{jk}\right) \quad \text{---------- [8]}$$

$$\delta_j = \delta_{inj} f'(Z_{inj}) \quad \text{---------- [9]}$$

Where $f'(.)$ indicates the derivative of the continuous activation function.

(IX) Values of weights and biases are getting updated based on the back propagated gradient value.

(A). For each output unit Y_k bias and weights are updated as follows,

$$\text{Weight updation } \Delta W_{jk} = \eta \, \delta_k Z_j \quad \text{---------- [10]}$$

$$\text{Bias updation } \Delta W_{ok} = \eta \, \delta_k \quad \text{---------- [11]}$$

$$W_{jk\,(new)} = W_{jk\,(old)} + \Delta W_{jk} \quad \text{---------- [12]}$$

$$W_{ok\,(new)} = W_{ok\,(old)} + \Delta W_{ok} \quad \text{---------- [13]}$$

(B). For each hidden unit Z_j bias and weights are updated as follows,

$$\text{Weight updation } \Delta V_{ij} = \eta \, \delta_j X_i \quad \text{---------- [14]}$$

$$\text{Bias updation } \Delta V_{oj} = \eta \, \delta_j \quad \text{---------- [15]}$$

$$V_{ij\,(new)} = V_{ij\,(old)} + \Delta V_{ij} \quad \text{---------- [16]}$$

$$V_{oj\,(new)} = V_{oj\,(old)} + \Delta V_{oj} \quad \text{---------- [17]}$$

4. XOR STANDARD BENCHMARK PROBLEM

MLP model is suitable to solve the non -linearly separable problem. If a straight line is used to classify the given input problem classes means that problem is called as linearly separable. In real time applications the problems are non-linearly separable. So the standard non-linear benchmark XOR problem [10] is chosen for implementation.

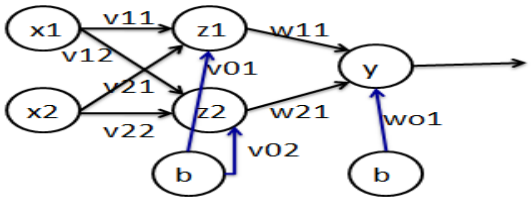

Fig 4: MLP architecture to solve XOR problem

XOR problem have 2 inputs and 1 output. Based on these requirement structured MLP architecture to solve the XOR problem is shown in Fig. 4 [6]. By using off-chip training the weight and number of neurons in each layer are fixed to construct the MLP architecture.

Table 1. MLP architecture description to solve XOR problem

Layer	Activation function	No. of neurons
Input	Linear	2
Hidden	Sigmoid	2
Output	Sigmoid	1

The detailed description of MLP architecture to solve the XOR problem is given in Table 1.

5. OFF- CHIP TRAINING OF MLP

In off-chip training, learning process relying on sequential software execution [11]. In this work training was done by using MATLAB software [8]. After the completion of training the network structure and weights were fixed. By using that

architecture and weight value the VHDL code was written to realize it on hardware [6-7].

Table 2.Initial and updated value of parameters

Parameter	Initial Value	Updated Value
v11	0.8147	5.6301
v12	0.1270	3.6205
v21	0.9058	5.6243
v22	0.9134	3.6052
w11	0.6324	6.9970
w12	0.0975	-7.5544
b1-1	0.2785	-2.2974
b1-2	0.5469	-5.5193
b2	0.9575	-3.1571

Table 3. Learning rate and error value

Parameter	Value
Learning rate	0.9
Error value	0.01

The network was trained by using back propagation algorithm, the obtained final updated weight value to solve the XOR problem is given in Table 2. The performance of the network was tested by given an unseen input vector to the network. This process output is shown in Table 4.

Table 4.Testing phase of the network

Test set	Net output	Final output
[1,1]	-2.5536	0.0722

During the feedback pass of back propagation algorithm weight updating process take place by reducing the MSE value till reach the specified error value given in Table 3. The plot for epoch VS MSE (Mean Square Error) is shown in Fig 5. The given learning rate determines the fast of convergence.

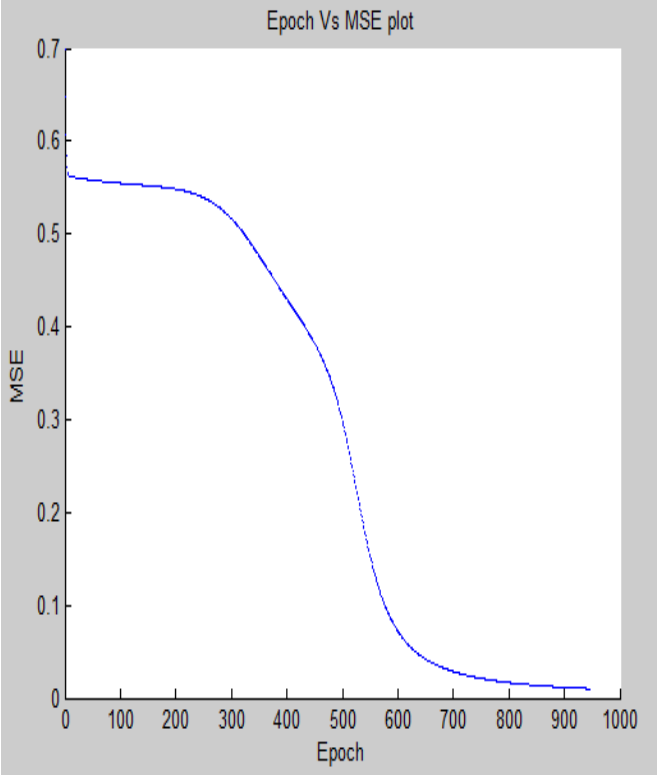

Fig 5: Epoch Vs MSE plot

From this plot it may clear that for XOR problem the specified MLP structure may take 950 epochs to solve the problem.

6. HARDWARE IMPLEMENTATION OF MLP

After the completion of training, the network structure is fixed. To do the hardware implementation of the structure the final updated weights value obtained during the training process will be used. Design entry to do the hardware implementation was done by using VHDL [6-7].

6.1. Device utilization

Device utilized by the simple classical XOR problem on FPGA device is shown in Table 5.

Table 5. Device utilized by XOR problem

Logic Utilization	Need for XOR Problem
No. of Slices	2464
No. of 4 input LUTs	4378
No. of bonded IOBs	96

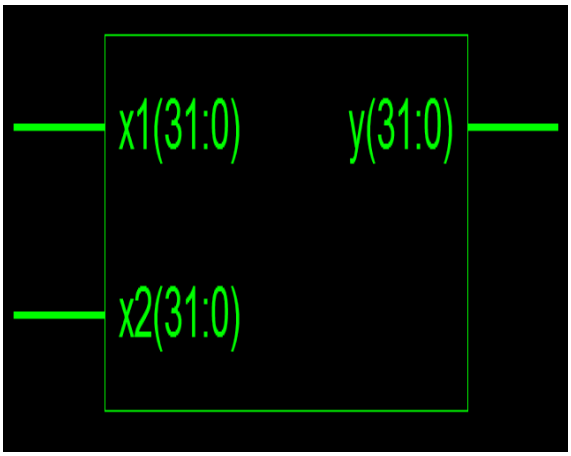

Fig 6.a: RTL Schematic View

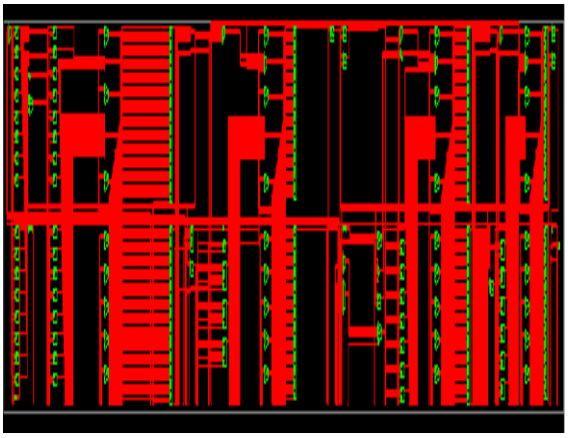

Fig 6.b: RTL Schematic View

Fig.6.a, b shows the RTL schematic view of the synthesized XOR problem. RTL schematic view shows the hardware area required to solve the XOR problem. From this it is clear that the inherent parallelism property of ANN is preserved. Fig.7. shows the simulation result for XOR problem on Modelsim simulator tool. So it is clear that off-chip trained MLP to solve the XOR problem is easily implemented in FPGA.

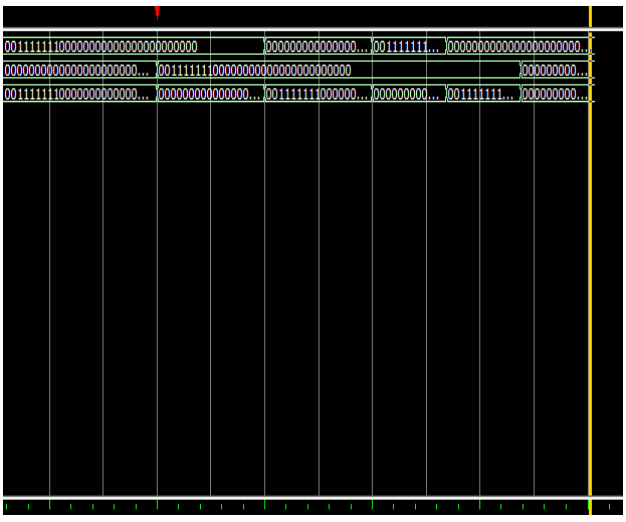

Fig 7: Simulation result

During FPGA implementation, to preserve the precision of network single precision floating point representation is used. It may give better accurate result.

7. ON-CHIP LEARNING

In section VI off-chip learned MLP network implementation for the classical XOR problem is given. In off-chip learning the process is easy to realize, because once the training and testing is completed using software simulation, the architecture and weights are fixed. Realizing this in hardware is easy. But this is not suited for real time applications and high dimensional problems such as bio-informatics and image processing [15]. Because different real time tasks need different architecture to solve the given task and the computation speed also should be high. The solution provided by on-chip learning is useful for adaptive control and system modelling for real time applications. On-chip back propagation learning is a standard bench mark learning for hardware based learning, because it preserves the inherent parallelism properties of ANN and the execution speed also high[11].

7.1 Hardware Platform for On-Chip Learning

Hardware based learning is well-suited for real world application [15-16], because the hardware may train itself to fix the architecture and weight to obtain the required performance based on the given training vector space. Also it provides device portability.

Implementation complexity is somewhat high in hardware based learning, because huge number of arithmetic operation and updating process will be take place during the back propagation algorithm execution [14]. Table 6 Shows the Comparison of Device utilization summary of off-chip based learning of XOR problem on different device environment.

From the comparison table it is clear that for the same problem xc3s4000I-4fg900 device environment is over fitted and the higher version is needed for the implementation.

Table 6. Comparison of device utilization percentage summary

Logic Utilization	xc3s4000I-4fg900	xc3s400-4pq 208	xc3s1000-fg456
No. of Slices	Over fitted	61%	8%
No. of 4 input LUTs	89%	63%	7%
No. of bonded IOBs	60%	68%	15%

In off-chip learning this may easy, because the architecture is fixed and every needed for the given task implementation is known in advance. But in on-chip learning this is not a case, because more devices may make use of more resources but some may require less compared to others.

Latest Xilinx platform suited for on-chip learning is the Xilinx Virtex-5 SX50T FPGA. This model of the Virtex-5 contains 4080 CLBs and CLBs hold 8 logic function generator, 8 storage elements, a number of multiplexers and carry logic. Now this platform is large enough to test a range of online neural network of varying size [14], [18].

8. CONCLUSION

From the proposed work finally concluded that, FPGA based hardware implementation of ANN preserves the parallelism property of ANN. Then the need for hardware based learning and the platform specification for hardware based learning were discussed. Hardware based learning is well-suited for real time application and it provide device portability.

9. FUTURE WORK

Based on the completed work it is planned to design a hardware based trained network to do the diabetic retinopathy classification.

10. ACKNOWLEDGMENTS

I would like to gratefully acknowledge the enthusiastic support of our College Management, Principal, and Einstein College of Engineering for providing me a platform where learning has known no margins.

11. REFERENCES

[1] M. Gopal. "Digital Control And Static Variable Methods", Tata McGraw Hill, New Delhi, 1997.

[2] Sathish Kumar, "Neural Networks: A Classroom Approach", Tata McGraw-Hill Publishing Company Limited, New Delhi, 2004.

[3] S.N.Sivanandam, Sumathi, Deepa.m, "Introduction to Neural Networks Using Matlab 6.0", Tata McGraw-Hill, ISBN 0-07-059112-1.

[4] Jagath C. Rajapakse, Amos R. Omondi, "FPGA implementation of Neural Networks". ISBN-10 0-387-28487-7 (e-book) @2006 Sprinker.

[5] David E. Rumelhart, Yves Chauvin, "BackPropagation: Theory, Architectures, and Applications", google preview.

[6] Martin T. Hagan, Howard B. Demuth, and Mark Beale, "Neural Network Design", Thomson Learning, New Delhi,2003,.

[7] J. Bhaskar, "VHDL Primer", P T R Prentice Hall.

[8] Volnei A.Pedroni, "Circuit Design with VHDL", ISBN 0-262-16224-5,Library of Congress Cataloging-in-Publication Data.

[9] R.Hunt,L.Lipsman, M.Rosenber, "A Guide to MATLAB for Beginners and Experienced Users", Cambridge University Press, ISBN-I3 978-0-511-07792-0 eBook.

[10] Simon Haykin, "Neural Networks: A Comprehensive Foundation", 2ed., Addison Wesley Longman (Singapore) Private Limited, Delhi, 2001.

[11] Antonio de Padua Braga, Tiago Mendonca Dasilva, Willian Soares Lacerda, "Pipelined on-line Back-propagation training of an artificial neural network on a parallel multiprocessor system", Learning and Nonlinear Module(L&NLM)-Journal of the Brazillian Society on Neural Networks, Vo1.8,I.No.2,P.No.120-123,2010.

[12] Himavathu.s, Muthuramalingam.A, Srinivasan.E, "Neural Network Implementation Using FPGA: Issues and Application" World Academy of Science, Engineering and Technology,I.No.48, P. No. 625-631,2008.

[13] Rafid Ahmed Khalil, "Hardware Implementation of Back propagation Neural Networks on Field programmable Gate Array(FPGA)", Al-Rafidain Engineering,Vol.16,No.3,Aug. 2008.

[14] Alexander Gomperts, Aghisek Ukil, " Development and Implemenation of Parameterized FPGA-Based General Purpose Neural Networks for Online Applications", IEEE Transaction on industrial informatics, Vol. 7,No.1,Feb.2011.

[15] Mark Pethick, Michael Liddle, Paul Werstein, and Zhiyi Huang , "Parallelization of a Backpropagation Neural Network on a Cluster Computer", 15th IASTED International Conference on Parallel and Distributed Computing and Systems,P.No.574-582, 2003.

[16] Zhu and Sutton. P, "FPGA implementation of Neural Networks: A Survey of a Decade of Progress", Lecture Notes in Computer Science,Vol. 2778/2003, P. No. 1062-1066,2003

[17] Yihua Liao, "Neural Network In Hardware-Survey", http://bit.csc.lsu.edu/~jianhua/shiv2.pdf,liaoy@cs.ucdavis.edu,

[18] http://alexandria.tue.nl/repository/books/644229.pdf

[19] http://business.highbeam.com/articles/436704/internatio-nal-journal-information-technology.