Implementation of RBF Neural Network
Reconfigurable Architecture – A NoC Design
Strategy

A Project Report
Submitted in partial fulfilment of the
Requirements for the Degree of
Master of Engineering
in
Microelectronics Systems

by
Renuka A

Under the guidance of
Prof. S K Nandy

Department of Electronics Systems Engineering
INDIAN INSTITUTE OF SCIENCE
BANGALORE – 560 012 (INDIA)

June 2014
Acknowledgements

I would like to take the opportunity and thank my advisor, Prof. S.K. Nandy, whose insights and suggestions helped tremendously to finish this work successfully. Further I am grateful for his patience and support he showed towards me during my stay here in CADL as a student. Secondly I express my gratitude towards Mah Naz Mohammadi and Shruthi.G my labmates, for their moral and technical support. I am sure this work would have not been possible without their help. I might have annoyed them many a times, but whenever help was expected from them, They were always there. Besides I would like to thank all my CADL lab mates, classmates and IISc-mates, IISc gave me an opportunity to be a part of Career Fair and Photography Club.

Thanks to all the people I came across in this family(CADL). This acknowledgement wouldn't be complete without a special thanks to Sanjay (for being a brother) and Nandhini Gopalan(a true friend), who made my stay in IISc and CADL a memorable one.

Most of all, I would like to thank my Parents and Brothers, the constant support and encouragement I needed to survive graduate study.
Abstract

The aim of the project is to design an Artificial neural network by using Network on Chip (NoC). Artificial Neural Networks (ANNs) are widely used in various applications such as recognition, security, Computer learning and so on. To meet requirements of higher performance, hardware implementations have been widely researched and developed. The popular implementation methods are analog, digital and hybrid methods. The digital method is also widely used due to high precision, good expansibility and a good design support by tools, but at the same time it is limited by the higher design cost, heavy communication load and less reconfigurability. In this thesis, we formulate and address problems in the key hardware implementation method, namely, digital ANN. Hardware implementation methods for Artificial Neural Network (ANN) have been researched for a long time to achieve high performance. We have proposed a Network on Chip (NoC) for ANN, and this architecture can reduce communication load and increase performance when an implemented ANN is small size, it shows that the proposed multiple NoC models can reduce communication load, increase system performance of connection-per-second (CPS), and reduce system running time compared with the existing hardware ANN. Furthermore, this architecture is reconfigurable and reparable. It can be used to implement different applications of ANN.
## Contents

**Acknowledgments**........................................................................................................i

**Abstract**..........................................................................................................................ii

1. **Introduction to ANN (Artificial Neural Network)**......................................................01
   1.1 Motivation ..................................................................................................................01
   1.2 Definition of ANN .....................................................................................................01
   1.3 History of ANN ..........................................................................................................02
   1.4 Neural Network Topologies .......................................................................................03
   1.5 A Short Introduction to Bluespec System Verilog .....................................................07
   1.6 Thesis Organization .................................................................................................08

2. **Literature Survey** .........................................................................................................09
   2.1 Survey of articles introducing RBFNN ......................................................................09
   2.2 Survey of articles motivating NoC implementation..................................................11

3. **Radial Basis Function Neural Network** .....................................................................14
   3.1 Introduction ...............................................................................................................14
   3.2 Structure of RBFNN ..................................................................................................14
   3.3 Learning Algorithms ................................................................................................15
   3.4 Determination of hidden layer ..................................................................................16
   3.5 K-means Clustering ..................................................................................................17
   3.6 Learning weights of Output layer .............................................................................18

4. **Introduction to NoC (Network on Chip)** ...................................................................19
   4.1 Router Architecture ...................................................................................................20
   4.2 Routing ......................................................................................................................24

5. **Architectural Design for ANN on NoC** ....................................................................27
   5.1 Structure of a single neuron .......................................................................................27
   5.2 Processing Element (PE) ..........................................................................................27
   5.3 Router design and packet architecture .......................................................................30
5.4 NoC Model for ANN.................................................................31

6. Simulation Results.................................................................32
   6.1 Data set used........................................................................32
   6.2 Learning Procedure............................................................33
   6.3 Design Tools......................................................................35
   6.4 Matlab Simulation.............................................................36

7. Conclusion and Future Work.....................................................39

Bibliography..............................................................................40
List of Figures

1.1 Biological Neuron ................................................................. 02
1.2 Feed Forward Network .......................................................... 03
1.3 Feed Back Network ............................................................... 03
1.4 RBF Network ................................................................. 03
1.5 BSV Example ................................................................. 06
3.1 Structure of RBF neural network ............................................... 10
3.2 Flow Chart for K-means clustering ............................................ 16
4.1 Overview of the 8x8 NoC Fabric ................................................ 19
4.2 Architectural Overview of the Router ......................................... 21
4.3 Architectural Overview of the Router - Various Modules Involve ...... 21
4.4 The muxes and demuxes of a stateless crossbar ......................... 22
4.5 A Regular honeycomb topology consisting of three combs .......... 23
4.6 A Regular honeycomb topology consisting of three combs .......... 24
4.7 The turns that are forbidden in the mesh topology are marked ........ 25
4.8 Mesh and Torus topologies .................................................. 25
5.1 Single Neuron ................................................................. 27
5.2 PE Architecture ............................................................... 28
5.3 PE Architecture ............................................................... 29
5.4 Router Packate Architecture ................................................ 29
5.5 Router Architecture ........................................................... 30
5.6 NoC Model for ANN .......................................................... 31
6.1 Block Diagram for learning procedure ...................................... 33
6.2 Gaussian Curve for Learning Procedure ..................................... 33
6.3 Output vs Desired Value Graph ............................................. 35
6.4 Learning Curve .............................................................. 36
Chapter 1

Introduction to ANN (Artificial Neural Network)

Artificial Neural Networks (ANNs) is an information processing paradigm inspired by the way a brain works. The key element of this paradigm is the processing element. The network is composed of a large number of highly interconnected processing elements. Each interconnection is called a synaptic connection and is associated with a weight that signifies the strength of the connection. An ANN is configured for a specific application such as data classification or recognition by a learning process. Learning process, similar to the counterpart in biological systems, involves adjustments to the synaptic strengths that exist between the neurons.

1.1 Motivation

The last two decades have witnessed the revival and a new surge in the field of artificial neural network research. This is an interdisciplinary area covering neuroscience, physics, mathematics, economics, electronics, communications etc. One of the reason for the popularity of the neural networks is the massive parallelism that they provide to perform computations. In order to take the benefit of this parallelism dedicated hardware implementation is definitely required. The challenge is to build complicated and larger size systems to implement efficient hardware networks to efficiently exploit new signal/state representation techniques and to apply these ideas in real-world applications where efficient hardware solutions are necessary to provide compactness, high-speed and low-power consumptions. These hardware implementations can be broadly classified as analog and digital. In the presented thesis, focus is on digital implementation.

1.2 Definition of ANN:

The neural network is a type of computer system architecture consisting of data processing by neurons arranged in layers. The corresponding results are obtained through the learning process, which involves modifying the weights of those neurons that are responsible for the error.

An Artificial Neural Network (ANN) is a computational model or mathematical model which attempts to simulate the structure and functional aspects of biological
nervous system. A neural network consists of an interconnected group of nodes which called artificial neurons, and it processes information by computing with a connectionist approach. An ANN is always an adaptive system that changes its structure according to internal or external information during the learning phase.

1.3 History of ANN:

1943 - McCulloch and Pitts introduced the first neural network computing model.
1950's - Rosenblatt's work resulted in a two-layer network, perceptron neural network.

1.3.1 Biological Neuron:


**Cell body** - soma, the axon and the dendrites.

**Axon** - a long cylindrical connection that carries impulses from the neuron

**Dendrites** - receive information from neurons through axons.

**Synapse** - axon-dendrite contact organ. Synapse is where the neuron introduces it's signal to the neighbouring neuron.

Fig 1.1: Biological Neuron
1.4 Neural Network Topologies:

The Tree Table of Neural Network Topologies given below.

**Feedforward Neural Network**

Data enters at the i/p and passes through the network, layer by layer, until it arrives at the o/p. When it acts as a classifier there is no feedback between layers, that's why it is called feedforward.

**Learning Phase:**

Supervised Learning Algorithm- besides the i/p pattern the neural net should know what category the i/p pattern

**Learning Procedure:**

When the i/p pattern reaches to o/p layer, if it is not classified correctly, weights are modified a little bit to guarantee that next time the same i/p pattern is applied to the network it is classified correctly.

**Learning Time:**

Depends on the size of the neural network, the number of patterns to be learned the speed of computer and the number of epoches.
Feedback Network/ Recurrent Networks
Can be obtained from feedforward network by connecting the neuron’s o/p to their i/p. Connection between units form a directed cycle. This creates an internal state of the network that allows it to exhibit dynamic temporal behavior. They can use their internal memory to process arbitrary sequences of inputs. Shown below Ability to do tasks such as unsegmented handwriting recognition.

![FeedForward Network](image1)

![Feedback Network](image2)

**1.4.1 Radial Basis Function:**

For the implementation of Neural Network the following one of the method of FFNN is been chosen, i.e RBF

Definition of RBF (Radial Basis Function):

\[
\phi(||x - c_i||)
\]

A function \( f: \mathbb{R}^n \rightarrow \mathbb{R} \) is termed **radial basis function net (RBF net)**

\[
f(x) = w_1 \phi(||x - c_1||) + w_2 \phi(||x - c_2||) + \ldots + w_p \phi(||x - c_q||)
\]
Consists of 3 layers:

**Input layer**: contains source nodes which are connected to the environment.

**Hidden layer**: It is only one layer and does a non-linear transformation from i/p space to hidden space.

**Output Layer**: supplies the response.

Non-linear transformation: If we can map the non-linearably separable i/p patterns to hidden layer in a linear separable way then the solution to the problem has been found.

- layered net
- 1st layer fully connected
- no weights in 1st layer
- activation functions differ

### 1.4.2 Threshold Activation Functions:

The transfer function of a neuron is chosen to have a number of properties which either enhance or simplify the network containing the neuron.

- **Activation functions** (Transfer Functions) are used for limiting the amplitude of the o/p of the neurons.

**Step Threshold Function**

The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

\[
 f(x) = \begin{cases} 
 0 & \text{if } \sum_{i=1}^{n} x_i w_i < T \\
 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq T 
\end{cases}
\]

**Sign Threshold Function**

Can be used for classification.

\[
 f(X) = \begin{cases} 
 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq 0 \\
 -1 & \text{Otherwise} 
\end{cases}
\]

**Sigmoid Function**

\[
 f(x) = \frac{1}{1 + e^{-ax}}
\]

**Gaussian**: Gaussian functions are bell-shaped curves that are continuous. The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average.
1.4.3 EXAMPLE (XOR- Problem):

Given below the example for Non-Linear

\[ f(x) = w_1 \phi(||x-c_1||) + w_2 \phi(||x-c_2||) + \ldots + w_p \phi(||x-c_q||) \]

- Two Gaussian functions are considered here as Radial Basis Functions:
- \( t_1, t_2 \) are centers of the functions.
- Dimensionality of i/p space and hidden space = 2

\[ \phi_1(\tilde{x}) = e^{-||\tilde{x}-\tilde{t}_1||^2} \]
\[ \phi_2(\tilde{x}) = e^{-||\tilde{x}-\tilde{t}_2||^2} \]
\[ \tilde{t}_1 = [1,1]^T \]
\[ \tilde{t}_2 = [0,0]^T \]
1.5 A Short Introduction To Bluespec System Verilog

Project is implemented in the high-level language Bluespec System Verilog (BSV) which allows to compile it into a clock accurate simulator that can be executed on ordinary end user PCs, but also to compile it into Verilog files for further processing by e.g. synthesis tools. By using one code for simulation and synthesis, code maintenance is minimized, since consistency does not need to be ensured among several implementations. In addition BSV allows to use certain constructs which are hard to understand for developers coming from hardware design background, but are well known to software programmers. On the other hand many times the software programmer lacks the experience and knowledge of hardware programming. As an example, if

\[ b = a \times 4 + 3 \]

needs to be calculated, a software developer will most likely write the equation directly into the program. However the hardware engineer knows that the multiplication is a simply shift of a by 2 bits to the left and the addition by 3 means that the last 2 bits of b are set after assignment of a to b. Thus, the above equation is equal to,

\[ b = (a \ll 2) \& 0x3 \]

avoiding the synthesis of power intensive multiplication and addition logic potentially requiring multiple clock cycles for calculation (refer to Figure 1). BSV tries to fill the gap between hardware and software developers by providing very abstract high level language constructs, but also allows the hardware engineers to do the operations that they are used to. The implementation of the router makes use of these high level abstractions.

![Figure 1.5:example](attachment:figure1.png)
1.6 Thesis Organization

The thesis is organized into following chapters:

Chapter 2 presents details of literature survey

Chapter 3 presents an introduction to Radial Basis Function Neural Network (RBFNN) and describes the learning process.

Chapter 4 introduces the architectural details of the RBFNN for NoC implementation.

Chapter 5 discusses the results of experiments conducted and presents its conclusions based on the results of experiments.

Chapter 6 summarizes the work done in the thesis and briefly suggests some directions for further developments.

Appendix A contains the MATLAB and Bluespec System Verilog(BSV) code for the learning the network.

Appendix B contains the details of the dataset used.
Chapter 2

Literature Survey

Scientists and engineers were always fascinated by the way the brain processes information, ever since the first steps of brain behavior research in the end of the 19th century. In early 1940s, the first mathematical model of a Neural Network (NN) was proposed, inspired by brain behavior. This achievement paved the way to further research the idea of Artificial Neural Networks (ANNs) for creating more advanced systems that abstractly mimic biological behavior. Such research as the perceptron [1] and other works on connectionist networks are more suitable for the solution of certain computational problems than typical computers, such as adaptive A.I and control applications. Contrary to the typical computer system based on the van Neumann model, a neural network does not execute explicit sequential instructions to solve its computational problems. The network is a group of processing elements interconnected to each other. There, each function on each node (or neuron) is computed in parallel and the relation between input and output of the NN is determined by the network topology and method of interconnectivity. This topology can also be adaptive, in terms of its the computational dynamics, mimicking further the biological behavior. A good amount of work has been carried out in the field of RBF neural networks and its implementation for various applications. Some of these have inspired the project work presented in this thesis. A brief review of such research work is presented in this chapter.

2.1 Survey of articles introducing RBFNN

RBF networks were introduced into the neural network literature by Broomhead/ Lowe and Pog-gio/Girosi in the late 1980s. The RBF network model is motivated by the locally tuned response observed in biologic neurons, e.g. in the visual or in the auditory system. RBFs have been studied in multivariate approximation theory, particularly in the field of function interpolation. The RBF neural network model is an alternative to multilayer perceptron which is perhaps the most often used neural network architecture. A brief introduction to artificial neural networks is required before moving on to implementing RBFNN. Simon Haykin [13] provides a comprehensive foundation of neural networks in his book. In recent years neural computing has emerged as a practical technology, with
successful applications in many fields. The majority of these applications are concerned with problems in pattern recognition, classification that make use of feed-forward network architectures such as the multi-layer perceptron and the radial basis function network. Also, it has also become widely acknowledged that successful applications of neural computing require a principled, rather than ad-hoc, approach. Bishop [12] in his book provides a focussed treatment of neural networks than previously available, which reacts these developments. By deliberately concentrating on the pattern recognition aspects of neural networks, it has become possible to treat many important topics in much greater depth. Having learnt the basics of artificial neural networks, a detailed study of radial basis function networks can be carried out. Mark J L Orr introduces RBFNN in his paper [14]. In his research document he provides an introduction to RBFNN for application to problems of supervised learning (e.g. regression, classification and time series prediction). The approach described places an emphasis on retaining, as much possible, the linear character of RBF networks, despite the fact that for good generalization there has to be some kind of nonlinear optimization. The two main advantages of this approach are keeping the mathematics simple (it is just linear algebra) and the computations relatively cheap (there is no optimization by general purpose gradient descent algorithms).

The document introduces Supervised Learning. A ubiquitous problem in statistics with applications in many areas is to guess or estimate a function from some example input-output pairs with little or no knowledge of the form of the function. So common is the problem that it has different names in different disciplines (e.g. nonparametric regression, function approximation, system identification, inductive learning). In neural network parlance, the problem is called supervised learning. The function is learned from the examples which a teacher supplies. The set of examples, or training set, contains elements which consist of paired values of the independent (input) variable and the dependent (output) variable. In classification problems the goal is to assign previously unseen patterns to their respective classes based on previous examples from each class. Thus the output of the learning algorithm is one of a discrete set of possible classes rather than, as in nonparametric regression, the value of a continuous function. However, classification problems can be made to look like nonparametric regression if the outputs of the estimated function are interpreted as being proportional to the probability that the input belongs to the corresponding class.
Data analysis plays an indispensable role for understanding various phenomena. Cluster analysis, primitive exploration with little or no prior knowledge, consists of research developed across a wide variety of communities. The diversity, on one hand, equips us with many tools. On the other hand, the profusion of options causes confusion. Thus in [16] the authors survey clustering algorithms for data sets appearing in statistics, computer science, and machine learning, and illustrate their applications in some benchmark data sets, the traveling salesman problem, and bioinformatics, a new field attracting intensive efforts. Several tightly related topics, proximity measure, and cluster validation, are also discussed.

Of the several clustering methods, k-means clustering algorithm is one of the oldest and popular clustering algorithm. Several tutorial pages are available online that discuss this popular algorithm. K-means is undoubtedly the most widely used partitional clustering algorithm. Unfortunately, due to its gradient descent nature, this algorithm is highly sensitive to the initial placement of the cluster centers. Numerous initialization methods have been proposed to address this problem. In this paper [15], the author first presents an overview of these methods with an emphasis on their computational efficiency. Then compares eight commonly used linear time complexity initialization methods on a large and diverse collection of data sets using various performance criteria. Finally, analyses the experimental results using nonparametric statistical tests and provide recommendations for practitioners. The paper demonstrates that popular initialization methods often perform poorly and that there are in fact strong alternatives to these methods.

2.2 Survey of articles motivating NoC implementation

The main problem of ANNs, that are biologically plausible, is the computational load of the networks. Furthermore, biological NNs execute these computations with massive parallelism, something that conventional CPU execution cannot cope with very well. As a result, the speed of simulations and the execution of models is quite low when running on PCs (with models implemented in MATLAB or neuron modeling languages such as NEURON and GENESIS). A good alternative to that would be the execution of neuron models in GPUs. Since DSP applications have repetitive and quite parallel functions to compute, GPUs are more capable to efficiently run neuron models. Yet, in the cases of complex models or very large-scale networks, they may not be able to provide real-time performance for the neural networks while also being power hungry and less mobile for real time applications. Another alternative would be the use of supercomputer
implementations. Although these systems can emulate behavior and parallelism with good speed, the sheer size and complexity of these solutions makes them useful only for behavioral simulations. Supercomputer systems require significant space, implementation, maintenance and energy consumption costs while lacking any kind of mobility. Mixed-VLSI is another option for such implementations. Such designs have good simulation speeds while emulating the biological systems more accurately since they also work with analog signals, just like the natural world. Mixed-VLSI designs are, on the other hand, much more difficult to implement, while often encumbered by accuracy issues, such as transistor matching. Additionally they lack edibility since each system must be tailor-made for a certain type of neuron models. Implementing the neural network in parallel hardware can exploit the parallelism of biological models fully and provide real-time or hyper real-time performance useful for simulations, prosthetics and robotics applications. ASIC design, though, is expensive, time consuming and not flexible. An ASIC chip when implemented cannot be altered if the need arises and changes on the system would required a new development cycle, just like Mixed-VLSI.

The second is a problem in digital ANN which is limited by the higher design cost, heavy communication load and less reconfigurability. Recently, Network on Chip (NoC) has attracted much attention. The packet-based network with high level parallelism architecture of NoC was used to solve complex on-chip interconnection problems for large system-on-chip (SoC). We presented a digital ANN with NoC architecture to solve the existing problems of digital ANN, such as heavy communication load and less reconfigurable. This digital ANN with NoC architecture is reconfigurable, because the weight values and activation functions can be changed as desired. We can also change the topology and routing algorithms of the NoC by sending new data to meet different kinds of ANN, so this system is easily extended. We can design this system in the style of cell-by-cell and can easily add or remove any cell to comply with different applications. The proposed NoC system can reduce the communication load of total packet size and improve the system performance of connection-per-second (CPS). This proposed NoC mapping method can make the digital ANN more efficient. The third is a discussion on general digital ANN with NoC architecture which is limited by the design cost when implementing large size ANN. A multiple NoC model is developed for a digital ANN, which can implement both a small size ANN and a large size one. Model-1 uses the general NoC ANN, all the layers of ANN can be implemented with it in one time, thus it can be suitable for ANN with small network size. Model-2 uses the same NoC architecture, whereas the implementation method is different. In this model, different
layers of ANN will be implemented with NoC architecture one by one, so that it is appropriate for ANN with large network size. The proposed multiple NoC models can reduce communication load, increase system performance of CPS, and reduce system running time compared with the existing hardware ANN. Furthermore, this architecture is reconfigurable and reparable. It can be used to implement different applications of ANN.

As the fourth issue, routing is important in order to maximize effectiveness of NoC. The traditional routing strategy limits the communication load and performance of the NoC ANN. One of popular routing strategies is Destination-Tag (DT) method which is used in the proposed NoC ANN. The advantage is that each hop could be easily controlled and different routing algorithms could be easily realized, whereas the disadvantage is that the total destination address stored in header becomes larger and larger proportional to the network size. This drawback causes that the NoC ANN could not achieve high performance and low communication load for large size ANN. Thus, a new NoC architecture is needed to implement the ANN. The main improvement is a router model with absolute address based routing strategy instead of the former router with DT method based routing strategy. This absolute address based routing strategy could reduce the header size of the packet compared with the DT method, and it can implement different routing algorithms with a little hardware change. So that the absolute address based NoC architecture is effective in reducing communication load and increasing performance.
Chapter 3

Radial Basis Function Neural Network

3.1 Introduction

Radial basis function neural network (RBFNN) is a fully connected feed-forward network with three functionally different layers: an input layer, a hidden layer and an output layer. When designing a neural network is seen as a curve fitting problem in a higher dimensional space, learning is equivalent to finding a surface in a multidimensional space that provides a best fit to the training data. Then, generalization is equivalent to the use of this multidimensional surface to interpolate the test data. In the context of neural network, the hidden units provide a set of functions that constitute an arbitrary basis for the input patterns when they are expanded into the hidden space, these functions are called radial basis functions.

3.2 Structure of RBFNN

The structure of the RBF neural network model is shown in Figure 3.1. The input data in the input layer of the network is \( x = [x_1, x_2, x_3, \ldots, x_n] \), where \( n \) is the number of dimensions. The hidden layer consists of \( m \) computing units (_1 to _m), which are connected to the output by \( m \) connection weights \((w_1, \ldots, w_m)\). The output of the network used by this algorithm has the following form:

\[
y(x) = f(x) = \sum_{j=0}^{m} \phi_j w_j
\]  

(3.1)

where \( \phi_0 = 1; \ W0 = \text{bias,} \ \phi_j \) is the response of the \( j \)th hidden neuron to the input \( x \), is the weight connecting the \( j \)th hidden unit to the output unit. Here, \( m \) represents the number of hidden neurons in the network, and \( \phi_j \) is a Gaussian function given by

\[
\phi_j = \exp \left( \frac{\|x - c_j\|^2}{\sigma j^2} \right)
\]  

(3.2)

Where \( c_j \) is the center and \( \sigma j \) is the width of the Gaussian. \( \| \cdot \| \) denotes the Euclidean norm.
3.3 Learning Algorithms

The learning algorithm consists of unsupervised learning and supervised learning. The unsupervised learning mainly adjusts the weight among input layer and hidden layer. The supervised learning adjusts the weight among output layer and hidden layer. The unsupervised k-means clustering procedure is often employed as a part of the general learning algorithm to adjust RBF centers. This involves computing the squared distance between the centers and the network input vector, selecting minimum squared distance and moving the corresponding center closer to the input vector. The computational procedure of this unsupervised clustering is as follows:

**Unsupervised Learning**

The unsupervised k-means clustering procedure is often employed as a part of the general learning algorithm to adjust RBF centers. This involves computing the squared distance between the centers and the network input vector, selecting minimum squared distance and moving the corresponding center closer to the input vector. The computational procedure of this unsupervised clustering is as follows:

1. \[ d_j(s) = ||x(s) - C_j(s-1)||^2, \quad 1 \leq j \leq n \quad (3.3) \]
2. \[ l = \arg \min_{1 \leq j \leq n} d_j(s) \quad (3.4) \]
3. \[ C_l(s) = C_l(s-1) + \alpha C(x(s) C_l(s-1)) \quad (3.5) \]
4. \[ C_j(s) = C_j(s-1), \quad 1 \leq j \leq n \text{ and } j \neq l \quad (3.6) \]
Supervised Learning

The supervised algorithm is very simple and robust. It is advisable to adjust the weights of the network so that the network can learn the general equalizer solution. The adaptation of the weights is achieved using the following supervised algorithm:

\[ \phi_j(s) = \exp\left(-\frac{||x(s) - c_j(s)||^2}{2\sigma^2}\right); \quad 1 \leq j \leq n \quad (3.7) \]
\[ e(s) = t(s - T) - n \sum_j = 1 T \phi_j(s) \quad (3.8) \]
\[ w_j(s) = w_j(s - 1) + \alpha w e(s) \phi_j(s); \quad 1 \leq j \leq n \quad (3.9) \]

In the presented work unsupervised learning technique using k-means clustering is used.

3.4 Determination of hidden layer

In order to specify the middle layer of an RBFNN, the number of neurons of the hidden layer have to be decided a priori. Hidden layer kernel functions are the radial basis functions. Here, a Gaussian function is used as a kernel function. A Gaussian function is specified by its center and width. The simplest and most general method to decide the middle layer neurons is to create a neuron for each training pattern. However the method is usually not practical since in most applications there are a large number of training patterns and the dimension of the input space is fairly large. Therefore it is usual and practical to first cluster the training patterns to a reasonable number of groups by using a clustering algorithm such as K-means or SOFM (Self Organizing Feature Map) and then to assign a neuron to each cluster. A simple way, though not always effective, is to choose a relatively small number of patterns randomly among the training patterns and create only that many neurons. A clustering algorithm is a kind of an unsupervised learning algorithm and is used when the class of each training pattern is not known. But an RBFNN is a supervised learning network. And we would know at least the class of each training pattern. So we would better take advantage of the information of these class memberships when we cluster the training patterns. The training patterns are clustered class by class instead of the entire patterns at the same time (Moody and Darken, 1989; Musavi et al., 1992). In this way the total computation time required to cluster the entire training patterns is reduced, since the number of patterns of each class is usually far less than that of the entire patterns.
3.5 K-means Clustering

K-means clustering is a method of classifying/grouping items into 'k' groups (where k is the number of pre-chosen groups). The grouping is done by minimizing the sum of squared distances (Euclidean distances) between input attributes and the corresponding centroid. A centroid is this context is the mean vector. The initial partitioning can be done in a variety of ways.

- **Dynamically Chosen:** This method is good when the amount of data is expected to grow. The initial cluster means can simply be the first few items of data from the set. For instance, if the data will be grouped into 3 clusters, then the initial cluster means will be the first 3 items of data.
- **Randomly Chosen:** Almost self-explanatory, the initial cluster means are randomly chosen values within the same range as the highest and lowest of the data values.
- **Choosing from Upper and Lower Bounds:** Depending on the types of data in the set, the highest and lowest (or at least the extremities) of the data range are chosen as the initial cluster means. The example below uses this method. In the presented work the centers are chosen randomly from the given training data set.

3.5.1 K-means Algorithm

1. Select K points as initial centroids/centres.
2. repeat: Form K clusters by assigning all points to the closest center
   (a) Compute distances of input vector to each cluster
   (b) Find the minimum distance
   (c) Assign the input to the closest cluster
3. Check if the last input vector is reached. If yes proceed with next next otherwise goto repeat
4. Recompute the centroid of each cluster
5. until: The centroids do not change.

3.6 Learning weights of Output layer

Unlike with other neural network paradigms, for RBFNN the hidden and output layer weights have different meaning and properties. It is therefore quite often different
learning techniques are used. The input to the hidden weights (centroid and spread) are learnt using unsupervised algorithms while output layer weights are learnt using supervised learning algorithms.

A supervised learning algorithm, Pseudoinverse Learning Algorithm (PIL), is used in the presented work for finding weight matrices, as the theorem from linear algebra states that pseudoinverse solution is the best approximation solution for Eq.3.10. It achieves a global minimum in the weight parameter space if the exact solution is reached. The algorithm is based on generalized linear algebraic methods, and it adopts matrix inner products and pseudo-inverse operations. PIL is a feedforward only, fully automated algorithm, including no critical user-dependent parameters such as learning rate or momentum constant. Let the responses from hidden layer be

\[ \varphi = [\varphi_0; \varphi_1; \ldots; \varphi_M] \quad (3.10) \]

Figure 3.2: Flow Chart for K-means clustering

where M is the number of neurons in the hidden layer.

The final output of the RBFNN \( y(k) \) is produced by a linear combination of the hidden layer responses as

\[ y(k) = \varphi W = \sum_{j=0}^{M} \varphi_j k \quad (3.11) \]

where \( WT = [w_0; w_1; \ldots; w_M] \) is a vector of synaptic weights. Thus the synaptic weights are calculated as follows

\[ W^T = Y^T : \varphi : (\varphi T : \varphi)^{-1} \quad (3.12) \]
Chapter 4
Introduction to NoC

1.2 Evolution of NoCs

Within a chip, complex systems and multi-core architectures consist of many units called PEs that are either highly specific to perform a single task very efficiently, or comprise ALUs for generic operations. The high level of integration of multiple (in an order of tens or even higher) PEs promise to satisfy the demand for computation power needed as of today. On the other hand these systems and architectures require a high speed communication system to exchange data among the PEs within the chip. One method is to analyze the traffic patterns of the executed application and building a communication system exactly matching these patterns by directly connecting the data exchanging PEs as shown in figure 1.1. While this point-to-point communication system is considered to be the fastest, it restricts the system designer to a few applications only. In addition depending on the richness of the system, the wiring requirements explode for n-to-n Some basic and important definitions to know before proceeding to the understanding of Network on Chip: The accumulation of the access control including the direction decision logic is called a router.

- The pattern in which the routers and their connections among each other are arranged is henceforth called topology.
- Topology and routers form an Network on Chip (NoC).
- The NoC including the PEs among which the connectivity is established, is referred to as Fabric.
- The number of connections that are established from one routers to all its neighbours, is referred to as degree. The degree depends on the topology that a router is integrated into.
- The term radix refers to the total number of connections a router provides including the links to all connected PEs (Processing elements).

All the above mentioned terminologies could be related to in Figure 2.

Few more definitions:
- An encapsulated piece of information outside the Fabric is called a packet.
- If it is inserted into the NoC, it is converted into a protocol that the router understands. This might include the necessity to divide the packet into several flits. Flits represent the largest amount of bits that can be transmitted at one instance of time between two routers.

In here, the chosen topology for the arrangements of the routers in the Fabric is the mesh topology, where-in every router is connected only to its adjacent router (which gives the rectangular mesh-like look).

Every router comprises of a control logic, for routing the data packets.

The router is also connected to the Processing Elements (also known as Computational Elements, CEs), which acts as a source or a sink in different scenarios.

Figure 4.1: Overview of the 8x8 NoC Fabric

4.1 Router Architecture

A look into the architectural overview of the Router.

Considering the Figure 3, the following can be said about the RECONNECT Router:

- The incoming data is stored in one of the Virtual Channel (VC). The VC has been calculated by the routing algorithm in the previous router.
• In the next step the neighbouring routers report their states of the VCs. This step ensures that the IP only considers flits which have a chance to be forwarded. This step creates a bit array of VCs that contain data and have a chance to be forwarded.
• An arbiter residing inside the IP, chooses one of the requesting VCs and the IP reports the desired OP to the router.
• Multiple IP might request for the same route, hence another arbitration step (IP/OP arbitration step) is required to resolve this conflict.
• The IP that won the IP/OP arbitration, transmits its flit and deletes it from the chosen VC. In case of multilfit environment the OP is bound to the IP for the entire duration of the transmission of the packet to prevent interleaving of flits from other IPs to the same OP.
• The flit traverses through the crossbar and is received at the OP in which its relative address is updated in case it is a header flit. In the relative addressing scheme, the address tuple represents the distance from the current node to the destination. Since the distance changes when the flit traverses the Network on Chip (NoC), it needs to be updated at every node the flit passes. If all elements of the address tuple equal 0, the flit reached its destination and is ejected from the network.

After the address has been updated, the flit is passed on towards the next router.

More details about the internal structure of the router, comprising the various modules:

4.1.1 Input Port (IP)

There are two different kind of IPs that provide connectivity to the router:

1. One is designed to be connected to the sink/generator which can be a PE or other modules of the target architecture. It includes the segmentation and reassembly of packets into flits and vice versa in multilfit environments.

2. The other kind of IPs is directly connected to the OPs of the neighbouring routers. It provides the functionality of VCs including the arbitration by Matrix Arbiters. If VCs are not required, each IP is reduced to provide a simple buffer of a predefined depth and a routing algorithm. Since there are no VCs anymore, obviously a VC arbitration step also becomes superfluous.
4.1.2 Assembly Unit (AU)

The AU is always instantiated regardless of the configuration depending on the defines. However, the provided functionality differs, if the router is not used in multiflit environments. In these, the AU basically only provides an interface that is compatible to the target architecture. The NoC works merely on the provision that a rule does not fire, if any of the implicit and explicit conditions that lead to the firing of the rule, is false. For example, if a rule could fire and it contains a component that writes into a FIFO, but at the same time the FIFO is full, the rule will not fire. The readiness of the FIFO buffer becomes an implicit condition for firing this particular rule. This mechanism is used to e.g. transfer data from the OP to the IP. The rule representing the OP will not transmit data, if the IP cannot accept it. Thus sending Acknowledgements (ACKs) back and forth can be omitted.

IP has an additional mechanism to choose only those flits in the VCs that can be routed (i.e. space is available in the receiving IP of the next router). Hence the implicit condition in which a rule transmits a flit to a full IP, does not occur in the first place.
If the router is embedded in a multiflit environment the task of AU is extended into segmenting a packet into flits. Depending on the available bus width and the size of the packet, multiple flits are generated. As long as the flits are generated, the AU marks itself as busy and does not send back ACKs, if new data is intended to be stored in the FIFO at its input port.

### 4.1.3 Arbiter

For selecting among the filled VCs a Matrix Arbiter has been implemented. It provides a similar scheduling of VCs such as round-robin.

1. In single cycle routers it is obvious to implement an arbiter that only considers flits that can be routed, since the buffers in the next IP are not full and can still accept data. Flits that cannot be routed, are not considered, hence increasing the throughput. In a second step multiple IPs might compete for an OP. If the flit looses this arbitration, it remains in the IP. Hence considering the state of the next IPs does not guarantee any proceeding to the next hop.

2. In multiflit environments after the arbiter chose a head flit, the following flits are taken from the same VC and the arbiter is deactivated till a tail flit passed. Even if the chosen VC runs out of data, the arbiter does not choose another VC.

   By binding the chosen VC to the output port of the IP, the interleaving of packets is avoided at the cost of a potentially lower hardware utilization which is compensated by the lack of hardware for distinguishing and reordering of incoming flits.

### 4.1.4 Crossbar

There are different ways to implement a crossbar. In single cycle routers the crossbar is stateless and comprises of muxes which are ideally arranged in the order shown in Figure 5.

![Crossbar Diagram](image)

**Figure 4.4:** The muxes and demuxes of a stateless crossbar
There are two points of view, how a flit traverses the router: An IP can send the flit towards an OP or an OP can request one specific IP to transmit its data. The latter one is implemented. For each connection from any IP to any OP exists a rule that fires upon a pulse. The pulse is generated by the arbiter that is instantiated once for each OP. The requests are coming from the IPs which wish to send their flits to this particular OP. After the arbitration step the OP is aware what IP won and request it to send its data.

Another implementation of a crossbar is a butterfly crossbar:

4.1.5 Output

Since the Fabric uses a relative addressing scheme, at each hop the address is updated based on the location of the OP.

4.2 Routing

4.2.1 Topologies

The router used, is designed to support honeycomb and mesh topologies each with a different routing algorithm. Both the topologies can be merged into a single one, and is discussed in this section.

For the honeycomb topology a rectangular brick structure is used as shown in Figure 7. It is obvious to see that the brick structure is very similar to the mesh topology with each alternating vertical link missing. Hence merging honeycomb and mesh topologies can be done easily.

Figure 4.5: A Regular honeycomb topology consisting of three combs (on the left). Same topology converted into a brick structure (on the right).
4.2.2 Virtual Channels (VCs)

It is compulsory to ensure that there cannot be any cyclic dependency in any situation in the network to avoid deadlocks. For instance in honeycomb toroidal networks are 2 different kind of dependencies possible: A cycle that comprises a comb and a cycle that circles once around the network as depicted in Figure 8. The arrows indicate the direction a flit want to be forwarded to. It is important to note that the data stored in the buffer can belong to entirely different messages and do not have to have a common source and destination. For instance the flit marked with (*) could have come from the generator that is located in the far east of the same row. To break cyclic dependencies the routing algorithm might depend on additional physical network layers which a flit is forwarded to. Each layer is equipped with a different routing algorithm and the paths from a lower layer to an upper layer are unidirectional. A message that traverses in a higher layer, cannot reenter a lower layer. Though physical separation is a commonly used implementation of complex bus systems such as in , it results in a tremendous overhead on a chip. If performance or wiring complexity is a constraint, VCs provide a solution to reduce multiple layers to a single physical layer thus flattening the topology. The input ports are equipped with a buffer for each layer. A change in a layer translates in a change of the buffer. E.g. if a message traverses from layer 0 to layer 1, it changes from buffer 0 to buffer 1. Hence each physical connection (called channel) among the routers consists of multiple Virtual Channels.
(a) The flit in the buffer (gray box) is waiting to be routed to the next router as indicated by the arrow. However it cannot proceed, because the buffer in the next router is full as well. Its flit also waits to be routed. This continues till the cycle is complete.

(b) An additional cyclic dependency, if toroidal topologies are considered.

Figure 4.6: Two examples for cyclic dependencies that can occur in honeycomb topologies.

### 4.2.3 Mesh Topology

To avoid deadlocks in mesh topologies the same technique of prohibiting turns is applied. The turns that a flit cannot take are depicted in Figure 11, which results in an algorithm that can be described as west first. Once a flit has been routed to the north or south, it cannot be forwarded towards the west anymore which means, if a packet has a destination in the west, it must first go to west before it can head anywhere else.
**Figure 4.7:** The turns that are forbidden in the mesh topology are marked. This results in routing rules in which the west direction has to be considered first. The dotted lines show the location of the date lines.

**Figure 4.8:** Mesh and Torus topologies
Chapter 5

Architectural Design for ANN on NoC

5.1. Structure of a single neuron

Neuron computing need to contain four operations: addition, multiplication, multiplier-accumulator, and function. Thus one single neuron consists of MUX (Multiplexer), MAC (Multiply Accumulate Circuit), RAM (Random Access Memory) and LUT (Look-up Table) as shown in Fig.1. In Fig. 1, inputs were chosen by MUX; multiplication and accumulation were realized by MAC; weight values were stored in RAM; activation function was expressed by LUT (Look Up Table). At least, 16 bits fixed point representation was required by ANN therefore the 16 bits data for neuron computing in this work consists of one sign bit, three integer bits and twelve fraction bits. Structure of one Neuron Design

![Single Neuron Diagram](image)

Fig5.1: Single Neuron

5.2. Processing Element (PE)

Processing Element (PE) and connected to one router for reducing total transmission packet, communication load and cost. A decoder and a control logic are also required to be consisted of the PE. When the data is transmitted from a router, a decoder decodes the 4-bit neuron address for choosing the neurons which will be used. For example, “1100” means the fourth neuron and the third neuron will be used. This design can make system
flexible, and it is easy for users to choose the neuron which they want. Control logic consists of counters and controlling RAM using virtual address. Weights are stored in RAM. When each neuron in the same PE completes its calculation task, the outputs of them hold output data as one single packet, and then the packet is sent to a router.

5.2.1 Four neurons in one PE
To attain higher performance and to reduce both communication load and cost, four neurons in one PE work in parallel. The neuron number in one PE is decided by the area and speed. We knew that the area of 4 neurons is equal to 1 LUT and for the pipeline working, one neuron needs 4 cycles, and two neurons need 5 cycle, and so on. We assume the size of one neuron is 1, and there are $x$ neurons in one PE. Implement $y$ neurons need area of ($y/x * 4 + y$) and time is $(4 + x - 1)$. So that the area * time = $(x + 7 + 12/x)y$. Thus, when the $x$ equal to 3 or 4, the value of area * time is best.

![PE Architecture](image)

$o_p$: packet from router;  $i_p$: packet to router

A PE architecture is shown above. A PE also requires a decoder, an encoder, a control logic, and LUT. PE decodes the neuron address to decide how many neurons (max is four, min is one) in this PE are used by a decoder. It requires two bits of the header packet as an indicator to distinguish the number of neurons used in PE. When one neuron is used, "00 " is assigned to the two bits. Similarly, "01 " is assigned when two neurons, "10 " when three neurons, and "11 " when four neurons are used. If no neuron is used in this PE, the packet does not transmit to this PE. The states of neurons are controlled by a control logic.
The four-stage pipeline design is shown in Fig. 4.4. The states are the decode state (D), calculation state (C), LUT state (L), and code state (CE). In D, the whole system must be configured to satisfy a real application, that is, the number of PEs and neurons, weight values, activation function, and routing paths are decided for the application, which includes the following work: load the weight value into the RAM, load the Look-Up-Table of activation function to the ROM, and load the head packet for selected PEs. This state does not appear anymore until the application will be changed. Then the chosen neurons begin calculations in C. When each neuron completes its look-up-table in L, the outputs of these neurons hold their output data as one single packet in CE. In CE, the packet that is loaded in C is used as part of the routed packet. The header phit is not changed, the payloads of this packet is used as the output of LUT, and each output corresponds to one payload, as shown in Fig. 4.5. Finally, the packet is sent to a router. The router is used to manage the transmission mechanism. Then the packet will be transmitted to all the neurons in the next layer via the router. This structure with 4 neurons in one PE can reduce the total number of transmit packets, communication load and cost.

The architecture of multiple NoC model is shown below. It consists of routers and PEs. One router is attached to one PE, and routers are connected with each other. PEs are
communicated via routers. In the NoC design, topology is very important. Different
topology can be suitable for different applications which make the system lower latency
and higher bandwidth requirement. At the same time, the power consumption may also be
reduced. The torus topology has one more direct communication channel between the
first router and the last router in each line. It is proper for the complex communication
applications, such as FF-ANN. (The comparison will be discussed later.) According to, at
most 64 neurons in one layer are used by common hardware ANN. Thus a 4x4 2D torus
topology is proposed for our system design.

5.3 Router design and packet architecture.

The architecture of the proposed router for managing the transmission mechanism is
shown in Fig.4.6. This router has 5 input ports and 5 output ports that are connected to
one PE and four routers. The proposed router consists of a buffer, MUX, an allocator, a
shifter and a register. i0, i1, i2 and i3 in Fig. 4.6 mean input phits from four directions to
the router. ii means input from PE. When phits arrive at this router, the virtual channel
that has 5 First-In First-Out (FIFO) buffers is chosen, and then it is transmitted to four 5-1
MUXs to choose the output port decided by the allocator. The selected phits are then
transmitted to a shifter. The shifter shifts 3 bits of header phits which are controlled by
the allocator, while payload phits are not shifted. Channels of our system transport are 18-
bit-width phits of data per cycle. A 2-bit field is added to each channel to decode the type
of phits (00 for dummy instruction, 10 for reader and 11 for payload). The packet format
is shown in Fig. 4.7. One packet contains one header and some payloads, and the payload
number is decided by the neurons used in the former-layer PE. The header contains 2 bits
for VCC (Virtual Channel Choice), 2 bits for PT (Phit Type), and 3 bits for each DA
(Destination Address). The payload contains 2 bits for PT and 16 bits for data.

![Figure 5.5. Router Architecture](image-url)
5.4 NoC Model for ANN

5.4.1 Design for multiple NoC models

The architecture of multiple NoC model is shown in Fig. 5. It consists of routers and PEs. One router is attached to one PE, and routers are connected with each other. PEs are communicated via routers. In the NoC design, topology is very important. Different topology can be suitable for different applications which make the system lower latency and higher bandwidth requirement. At the same time, the power consumption may also be reduced. The torus topology has one more direct communication channel between the first router and the last router in each line. It is proper for the complex communication applications, such as FF-ANN. (The comparison will be discussed later.) According to, at most 64 neurons in one layer are used by common hardware ANN. Thus a 4x4 2D torus topology is proposed for our system design.

![Figure 5.6: NoC Model for ANN](image-url)
Chapter 6
Simulation Results

6.1 Data set used

The chosen database for all verification purposes has been taken from https://archive.ics.uci.edu/ml/machine-learning-databases/iris/ Relevant Information: This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the _eld and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; theatter are NOT linearly separable from each other.

| Predicted attribute: class of iris plant. |
| This is an exceedingly simple domain. |
| This data differs from the data presented in Fishers article(identified by Steve Chadwick, spchadwick@espeedaz.net) The 35th sample should be: 4.9,3.1,1.5,0.2, "Iris-setosa" where the error is in the fourth feature. The 38th sample: 4.9,3.6,1.4,0.1, "Iris-setosa" where the errors are in the second and third features. |

6.1.1 Details of data set used

1. Number of Instances: 150 (50 in each of three classes)
2. Number of Attributes: 4 numeric, predictive attributes and the class
3. Attribute Information:
   (a) sepal length in cm
   (b) sepal width in cm
   (c) petal length in cm
   (d) petal width in cm
   (e) class:
      _ Iris Setosa
      _ Iris Versicolour
      _ Iris Virginica
4. Missing Attribute Values: None
5. Class Distribution: 33.3% for each of 3 classes.
6.2 Learning Procedure

Training of the RBF neural network, here, is an offline procedure. The data is stored and accessed repeatedly. A matlab code is used to perform clustering (k-means) to learn the cluster centers, cluster spread and also to learn synaptic weights of the output layer. The network is designed for twelve neurons in the hidden layer and three output neurons (one for each class of data). Neurons transform their net input by using a scalar-to-scalar function called an "activation function", yielding a value called the unit's "activation". Hidden layer neurons use Gaussian function as activation function. The output neurons have linear activation function. The network is trained to achieve the best possible classification rate. This also ensures a better recognition rate.

Block Diagram for Learning Procedure:

![Block Diagram](image)

Figure 6.1: Block Diagram for learning procedure

![Gaussian Curve](image)

Figure 6.2: Gaussian Curve for Learning Procedure
The design of the RBFNN for FPGA is developed in Bluespec System Verilog and Verilog hardware description language (HDL). The main components are the address generator, clock reference and controller, Floating Point Unit (FPU) multiplier, floating point adder and RAM's.

1. Lookup Tables
Gaussian function is a non-linear function. One of the available methods for its implementation is to use lookup table. Here, a table of pre-calculated values is stored in the RAM units, based on the controller inputs and address provided particular values are fetched from the memory. In this project, two lookup tables are used. One for storing the weights for the output layer. Other stores the exponential values in the range -3 to 3 computed at 0.001 step intervals. The range is chosen because the Gaussian function has significant values in this region and trails down to zero for regions outside this window.

2. Address Generator
In order to access the values stored in the block RAM, location address of the value has to be provided as an input to the RAM unit with other control signals set to proper values. This task is achieved by the address generator.

3. FPU multiplier and FPU adder
The data format used in the presented work is single precision floating point format. Thus, to perform any floating point operations, specific core are required. One can proceed to designing their own FPU's. Here, available logicore ip's for floating point operations are used. Two instances are made, one configured as the multiplier and the other configured as adder. Together they form a multiply and accumulate unit.

4. Controller
In order to synchronize various activities a clocking reference is used. This reference is derived from the clocking primitives available on the board. Other controls required to access data from BRAM's and control signals for multiplier and adder are set and their state is controlled here.
6.3 Design Tools:

- Data set used:
  - Data set: Fisher's Iris database
  - Dataset: 150 x 4
  - Training data size: 120(i/p’s) x 4(features)
  - Testing data size: 30 x4
  - Number of Class: 3 (50 instances of each class)
  - Tool used for training: Matlab

- Uniqueness:
  - One class is linearly separable from the other 2;
  - the latter are not linearly separable from each other

Best suited for demonstrating the classification capability of the RBF neural networks.

6.4 MATLAB – Simulation

1. Matlab Simulation:

   Training and Testing of the Following data sets are represented in a graph.
   1. Iris Setosa 2. Iris Versicolour 3. Iris Virginica

Figure 6.3 Graph of 3 data sets for Training and Testing
Matlab Simulated Result given in a Table Below

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Results/Values obtained in Matlab Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth of test data</td>
<td>30</td>
</tr>
<tr>
<td>Classification Rate</td>
<td>98.33%</td>
</tr>
<tr>
<td>Recognition Rate</td>
<td>97%</td>
</tr>
<tr>
<td>Time Consumed</td>
<td>0.021593 seconds</td>
</tr>
</tbody>
</table>

Figure 6.4: Output vs Desired Value Graph.

Figure 6.4: Learning Curve:
Chapter 7

7. Conclusion and Future Work

Other proposed NoC architecture has a similar implement method as this architecture, so that, it is fit for the ANN with large size. Furthermore, the NoC architecture has a smart packet based data transmission method. These advantages make NoC architecture much more suitable for hardware ANN.

A more dedicated work would be towards the Artificial Neural Network or towards the inclusion of new topologies into the existing Router architecture.

- Initial choice of centres affects the clustering
- Parameter “k” forms a trade off between performance and size of network
- With RBFNN 100% recognition can be achieved
- Hardware realisation improves speed to a greater extent
Bibliography: