DESIGN AND IMPLEMENTATION OF A DIGITAL NEURAL PROCESSOR FOR DETECTION APPLICATIONS

BALAMURUGAN BALASUBRAMANIAN

Design and Implementation of A Digital Neural Processor for Detection Applications

by

© Balamurugan Balasubramanian, B. Eng.

A thesis submitted to the School of Graduate

Studies in partial fulfillment of the

requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

January 1999

St. John's Newfoundland Canada

Abstract

The main focus of this research is to develop a digital neural network (processor) and hardware (VLSI) implementation of the same for detection applications, for example in the distance protection of power transmission lines. Using a hardware neural processor will improve the protection system performance over software implementations in terms of speed of operation, response time for faults etc. The main aspects of this research are software design, performance analysis, hardware design and hardware implementation of the digital neural processor. The software design is carried out by developing an object oriented neural network simulator with backpropagation training using C++ language. A preliminary analysis shows that the inputs to the neural network need to be preprocessed. Two filters have been developed for this purpose, based on the analysis of the training data available. The performance analysis involves studying quantization effects (determination of precision requirements) in the network.

The hardware design involves design of the neural network and the preprocessors. The neural processor consists of three types of processing elements (neurons): input, hidden and output neurons. The input neurons form the input layer of the processor which receive input from the preprocessors. The input layer can be configured to directly receive external input by changing the mode of operation. The output layer gives the signal to the relay for tripping the line under fault. Each neuron consists of datapath and local control unit. Datapath consists of the components for forward and backward passes of the processor and the register file. The local control unit controls the flow of data within a neuron and co-ordinates with the global control unit which controls the flow of data between layers. The neurons and the layers are pipelined for improving the throughput of the processor. The neural processor and the filters are implemented in VLSI using hardware description language (VHDL) and Synopsys / Cadence CAD tools. All the components are individually verified and tested for their functionality and implemented using 0.5 μ CMOS technology.

Acknowledgements

I am deeply indebted to my supervisor Dr. R. Venkatesan for his invaluable guidance, discussions and useful criticisms during the course of my research and help in preparing this manuscript. I express my sincere thanks to Dr. R. Venkatesan, the Faculty of Engineering and Applied Science and Memorial University of Newfoundland for the financial support provided to me during my M. Eng. program.

I thank Dr. R. Seshadri, Dean of Faculty of Engineering and Applied Science, Dr. M. R. Haddara, Associate Dean for Graduate Studies, the faculty members and the CCAE staff for their support during my study.

I express my gratitude to Dr. B. Jeyasurya for useful discussions and help in obtaining simulation data. I sincerely thank Dr. Paul Gillard, Head of the Department of Computer Science and Mr. Michael Rendell, Department of Computer Science for help regarding VLSI CAD tools. I thank my fellow graduate students for their moral support and encouragement throughout the course of my study in Canada.

Finally, I thank my parents Mr. & Mrs. R. Balasubramanian, my brother Mr. B. Balagurunathan and my sister Ms. M. Aruna Rajeshwari for their constant encouragement and support during my study.

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

α	: Learning rate parameter of the ANN.
Z^{-1}	: Unit delay element.
Δw	: Delta weight (to be added with the weight to be modified).
E	: Objective function of the learning algorithm.
δ	: Local gradient parameter of the ANN.
ω	: Momentum parameter of the ANN.
w	: Weight value of a synaptic connection in an ANN.
ANN	: Artificial Neural Network.
ASIC	: Application Specific Integrated Circuit.
CMOS	: Complementary Metal Oxide Semiconductor.
CMOSIS5	: CMOS Insulated Silicon 0.5μ process technology.
CUPS	: Connection Updates Per Second.
DC	: Design Compiler.
DOF	: Degree of Freedom.
FFT	: Fast Fourier transform.

GCPS : Giga Connections Per Second.

GCU : Global Control Unit.

GCUPS : Giga Connection Updates Per Second.

KCUPS : Kilo Connection Updates Per Second.

LCU : Local Control Unit.

LR : Learning Rate.

MCPS : Mega Connections Per Second.

MCUPS : Mega Connection Updates Per Second.

MFLOPS : Mega FLoating point Operations Per Second.

MLP : Multi Layered Perceptron.

RISC : Reduced Instruction Set Computing.

ROM : Read Only Memory.

SIMD : Single Instruction Multiple Data Stream.

SONAR : SOund Navigation And Ranging.

VHDL : Very high speed integrated circuit Hardware Description Language.

VLSI : Very Large Scale Integrated circuits.

VSS : VHDL System Simulator.

Chapter 1

Introduction and Literature Survey

1.1 Introduction to Neural Networks

Artificial neural networks (ANNs) form a class of systems that are inspired by biological neural networks. Artificial neural networks have proved to be a vital tool for solving problems that cannot be approached by traditional methods. McCulloch and Pitts introduced the concept of neurons in 1942 [1, 2, 3]. Since then several contributions have been made to the field of artificial neural networks. Due to their capabilities for modeling and solving complex problems, the applications of ANNs are many. The applications include classification problems, vision, speech, signal processing, time series prediction, modeling and control, robotics, optimization, expert systems and financial applications [1, 4, 5].

It can be stated that the evolution of the field of neural networks is characterized by a number of ups and downs. There was a period of hibernation, for about 25 years, from 1969 to 1982, after some initial developments in the area. This is due to the fact that neural networks without hidden layers were considered at that time and they were

not able to learn the well known XOR problem. Then a major breakthrough came with the introduction of multilayered perceptrons. These neural network structures will be discussed in detail in the following sections. From then until now, research in artificial neural networks has been blooming, as witnessed by the existence of several international neural network societies and international conferences. Progress is continuously being made to the theoretical and practical aspects of the field. As a result, the area of applications has also extended into many fields, like ATM scheduling [6], one of the new concepts in telecommunication.

1.2 Classification of Neural Networks

The artificial neural network architectures were formed resembling the biological neural architecture. The brain consists of about 10 billion neurons and 16 trillion synaptic junctions or synapses. The biological neuron (nerve cell) is shown in Figure 1.1. The figure shows the major components of a typical nerve cell in the central nervous system [2]. The synapses connect the axon of one neuron to various parts of other neurons. Depending on the stimuli at the synapses, which when exceed the activation potential (threshold potential), the neuron produces an output potential. The output potential acts as stimulus for other neurons to which it is connected. The axon carries the output of the neuron to other neurons. The artificial neurons have similar structure and functionality. The artificial neural network consists of small processing elements, called neurons, interconnected with each other. The synapses

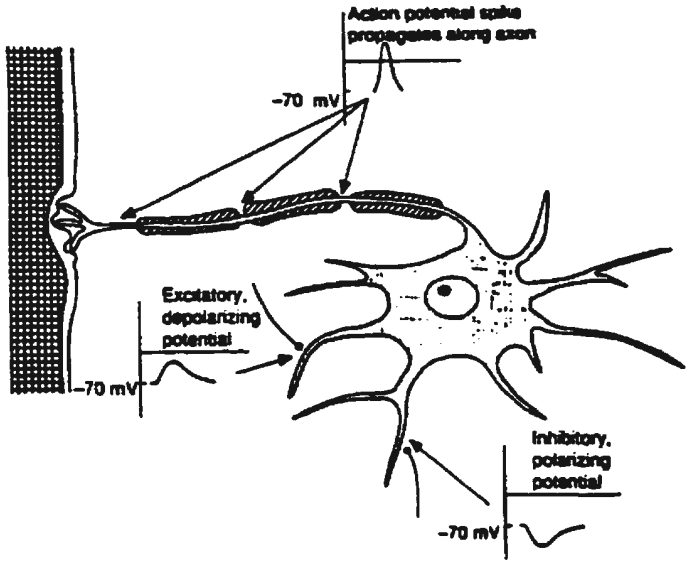


Figure 1.1: The Biological Neuron [3]

are represented by interconnection weights and the activation potential of the biological neuron is represented by the activation function. The first model of an artificial neuron was introduced by McCulloch and Pitts in 1942, which was a static nonlinear model. Later Rosenblatt [7] introduced the perceptron model, the most commonly used basic artificial neuron, in 1962. The perceptron model is shown in Figure 1.2. In the figure, φ are the inputs to the neuron and α are the synaptic weights.

1.2.1 Basic Model of a Neuron

In mathematical terms, the basic model of a neuron is given by the equation 1.1 as,

$$y = \varphi\left(\left[\sum_{i=1}^{n} w_i x_i\right] - \theta\right) \tag{1.1}$$

where φ is the activation function of the neuron; $x_1, x_2, \ldots x_n$ are the inputs; w_1, w_2, $\ldots w_n$ are the interconnection weights and θ is the threshold. The schematic represen-

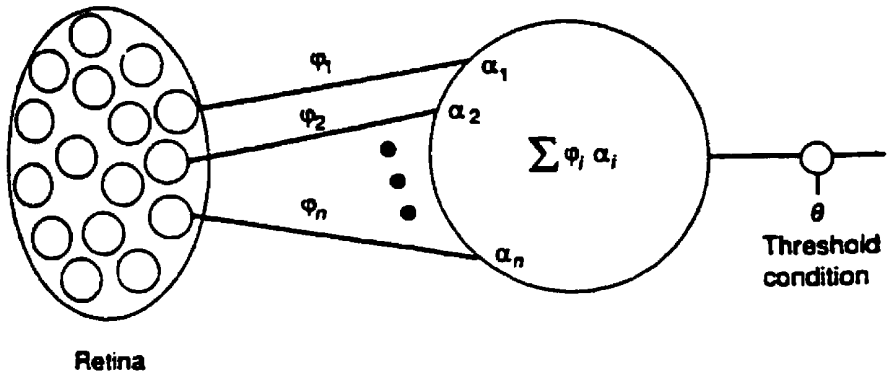


Figure 1.2: Model of a Perceptron [3]

tation of the equation is given in Figure 1.3. The activation function of the artificial neurons can be one of many, ranging from simple threshold functions to sigmoidal functions. The mathematical representation of some activation functions are as follows.

1. *Threshold Function*

$$\varphi(x) = \begin{cases} 1 & if x \geq 0 \\ 0 & if x < 0 \end{cases} \tag{1.2}$$

2. *Piecewise-Linear Function*

$$\varphi(x) = \begin{cases} 1 & x \geq 5 \\ (0.1x + 0.5) & 5 > x > -5 \\ 0 & x \leq -5 \end{cases} \tag{1.3}$$

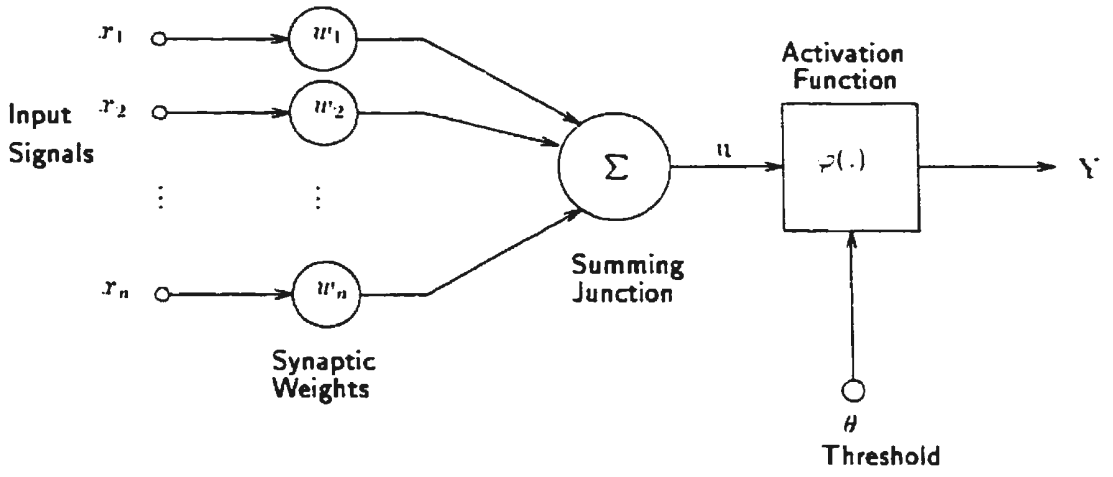


Figure 1.3: Model of a Neuron

3. *Sigmoidal Function*

$$\varphi(x) = \frac{1}{1 + exp(-ax)}, \tag{1.4}$$

where a is the gain parameter.

Some of the activation function plots are shown in Figure 1.4. The choice of the activation function depends on the application for which the neural network is used.

All existing artificial neural networks are formed using the basic artificial neuron. They are classified based on the way they are interconnected i.e. the architecture of the neural network [1]. Although these categories are based on different philosophies, all neural networks are capable of *learning*, a process by which a neural system acquires the ability to map a set of inputs to a set of outputs by modifying its internal parameters according to a scheme. The set of input/output patterns are called the

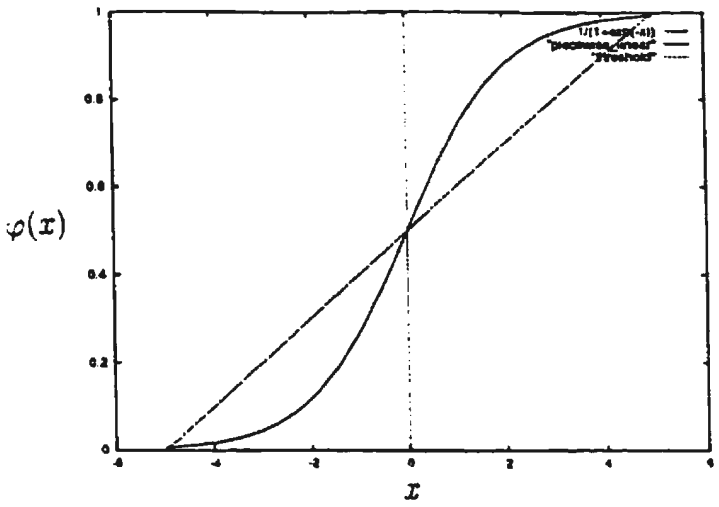


Figure 1.4: Some Activation Functions

training sample. The learning schemes are classified as *Supervised* and *Unsupervised.* In the following sections, neural network architectures and the learning schemes are discussed.

1.2.2 Neural Network Architectures

Existing artificial neural network architectures are classified into three major categories, *Feedforward*, *Feedback* and *Self Organizing* neural networks. Figure 1.5 [1] shows the classification of neural architectures. Feedforward networks are most widely used architectures. The implementation of these architectures can be in software or hardware. The work explained in this thesis uses a Multilayered perceptron with backpropagation training. The work includes implementation of the neural network in hardware as well. The following sections discuss the categories of neural architectures in detail.

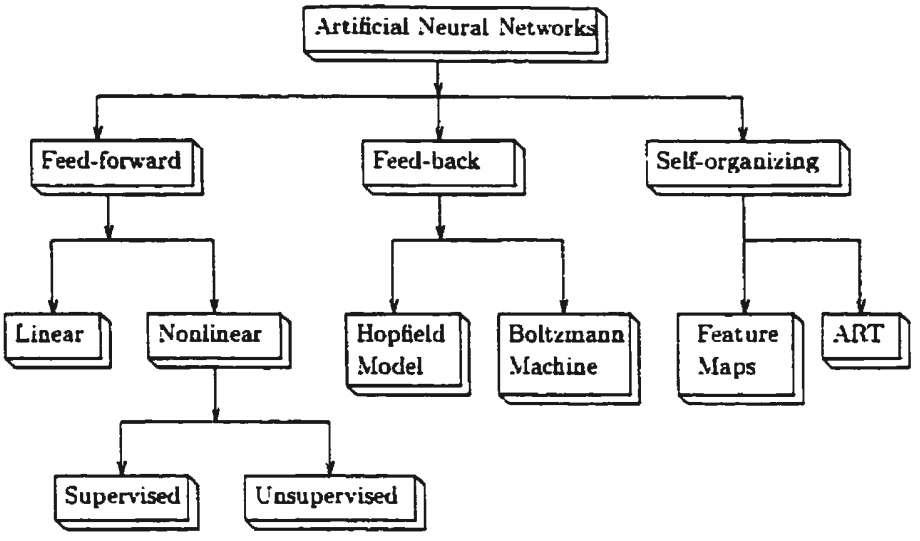


Figure 1.5: Classification of Neural Architecture [1]

1.2.2.1 Feedforward Neural Architectures

Feedforward neural networks consist of one or more layers of the basic artificial neuron, the processing elements. The neurons of the neighboring layers are interconnected by synaptic weights. The output of each neuron feeds the next layer of the network. This can be seen as a system transforming a set of input patterns into a set of output patterns. Multilayered feedforward networks consists of one or more hidden layers. The hidden layers increase the ability of the neural network to acquire higher order statistics. Multilayered networks can be fully connected or partially connected. Schematic representations of single layer, fully connected multi layer and partially connected multi layer neural architectures are shown in Figures 1.6, 1.7 and 1.8 respectively.

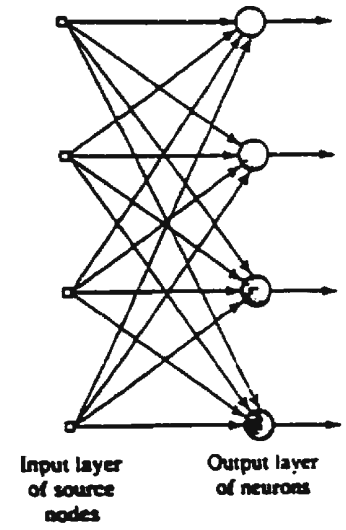


Figure 1.6: Single Layered Feedforward Neural Network [2]

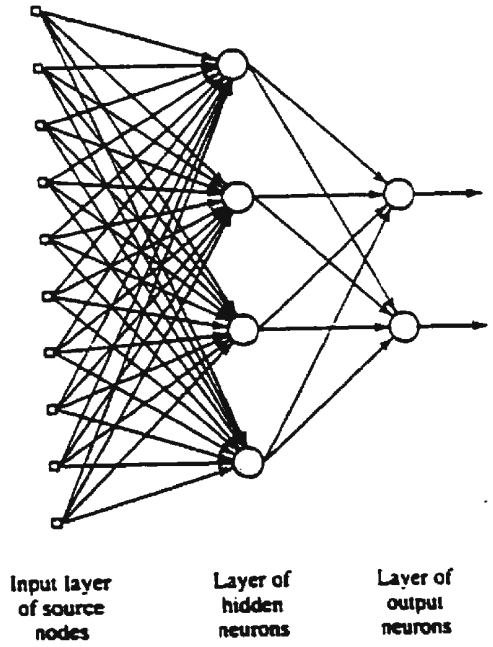


Figure 1.7: Multilayered Fully Connected Neural Network [2]

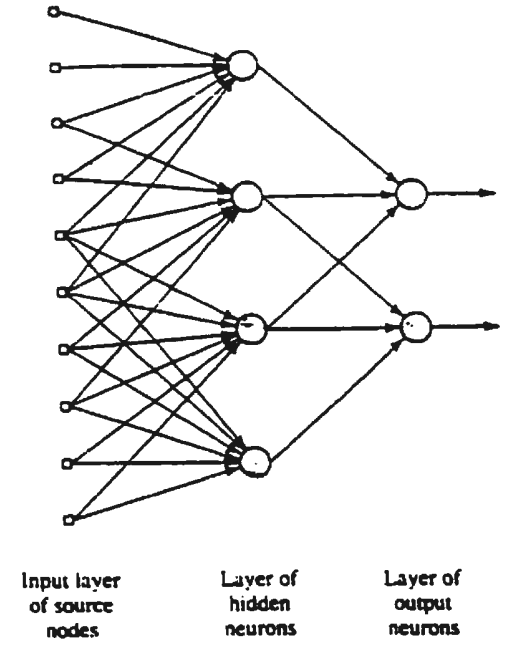


Figure 1.8: Multilayered partially Connected Neural Network [2]

1.2.2.2 Feedback Neural Architectures

Feedback neural architectures differ from the feedforward architectures by the *feedback* loop. They are also called *recurrent networks*. A feedback neural network may consist of a single layer of neuron feeding its output to all other neurons, as illustrated in Figure 1.9. The figure illustrates only a layer and not the complete network. The presence of a feedback loop has an impact on the learning capability of a neural network and on its performance. Moreover, the feedback loops involve the use of *unit-delay* elements (denoted by Z^{-1} in the figure), which result in nonlinear dynamical behavior of the neuron. Some of the feedback neural network models are

- Brain-State-in-a-Box Model
- Hopfield Model
- Boltzmann Machine

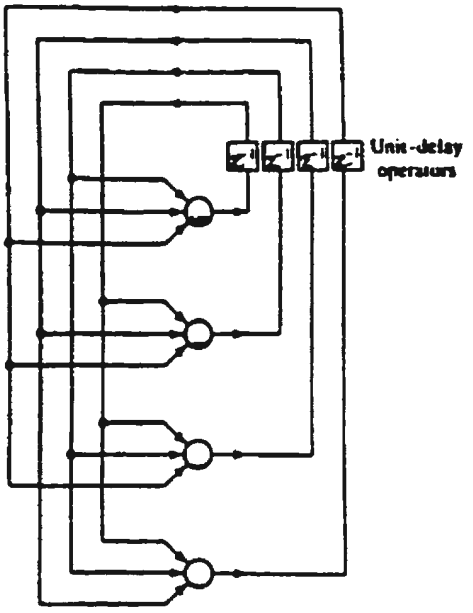


Figure 1.9: A Feedback Network without self-feedback [2]

- Recurrent Backpropagation Networks.

These models are discussed in detail in [2].

1.2.2.3 Self Organizing Neural Architectures

Human brain has the unique ability to use past experience to adapt to unpredictable changes in the environment. Such adaptation with no involvement of an external teacher is called *Self Organization*. Two of the self organizing neural networks are

- Kohonen's Feature Map
- Adaptive Resonance Theory (ART).

These networks follow the *counter propagation* or *competitive learning* scheme in which neighboring cells compete in their activation by means of mutual lateral interaction and develop into specific detectors of different signal patterns. Self Organizing feature maps are used for application like pattern recognition, robotics and process control.

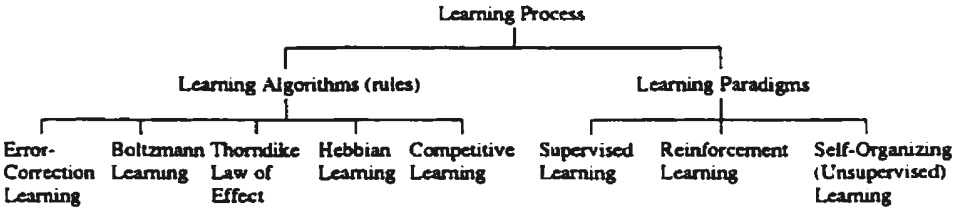


Figure 1.10: Taxonomy of Learning process [2]

1.2.3 Learning Schemes

Learning is the process of acquiring the ability to map a set of inputs to a set of outputs by adjusting the internal parameters of the system, such as *synaptic weights*, *learning rate* etc. The method followed for this process is called the *learning scheme*. When an external teacher is used to determine the training and learning process it is called *Supervised Learning*. When learning does not involve an external teacher it is called *Unsupervised learning*. Haykin [2] provides a taxonomy of the learning process which is shown in Figure 1.10. Generally supervised learning is used in the case of applications requiring specific outputs, like detection or control, and unsupervised learning is used in the case of some classification applications where the neural network determines the classification based on the input patterns.

1.2.3.1 Least Squares Method

This learning scheme, also called *outer-product rule* or *correlation training*, is among the earliest training schemes. It is not an optimal training scheme in any sense. The major advantage of this scheme is its simplicity. Section 1.3.1 discusses the performance of this learning scheme in comparison with some other learning schemes developed later. This scheme is based on the well known least squares method. Considering an output of a single layered neuron $\bar{y}_{i,k} = x_k^* w_i = w_i^* x_k$; $\mathbf{w}^*$ and $\mathbf{x}^*$ are transpose of weight and input matrices respectively, i is number of neuron, k is number of input, the optimal estimate of the synaptic weights is given by,

$$\frac{\partial E}{\partial w_i} = -\sum_{k=1}^{m} x_k \left(y_{i,k} - x_k^* w_i\right) = 0 \qquad \forall i = 1, 2, \ldots n_0, \tag{1.5}$$

where E is the objective function [1]. It can be easily verified that w_i is the solution of the set of linear equations

$$X_m X_m^* w_i = X_m y_{i,m}^* \qquad \forall i = 1, 2, \ldots n_0, \tag{1.6}$$

where X_m is a matrix of x_k and $y_{i,m}^* = [y_{i,1}, y_{i,2}, \ldots y_{i,m}]$. On simplifying the linear equation for an optimal estimate, the synaptic weight matrix solution is

$$W = \frac{1}{n_i} Y_m X_m^*, \tag{1.7}$$

where n_i is the number of inputs in a set of input patterns.

1.2.3.2 Delta Rule

Although this rule is widely used in adaptive filtering, its simplicity and flexibility made it attractive for training neural networks. However, this learning rule is characterized by slow convergence, and in some situations, can lead to local minima. This rule is based on the observation that the minimization of the objective function $E = \sum_{k=1}^{m} E_k$ (k is the number of iterations) can be performed by sequentially minimizing $E_k = \frac{1}{2}\sum_{i=1}^{no} (y_{i,k} - \hat{y}_{i,k})^2$ for $k = 1, 2, \ldots m$ using the Delta rule. Based on this the synaptic weight is updated as

$$w_{p,k} = w_{p,k-1} + \alpha \epsilon^0_{p,k} x_k, \tag{1.8}$$

where p is the synapse number, α is a positive real number, called the *learning rate* and

$$\epsilon^0_{p,k} = e_{i,k} = y_{i,k} - \hat{y}_{i,k}. \tag{1.9}$$

The network is trained until a predetermined minimum for E is obtained with the synaptic weights updated using the rule specified in equation 1.8.

1.2.3.3 Backpropagation with Gradient Descent

This is the most commonly used learning scheme for Multilayered Perceptrons (MLP). The objective of this method is to start at some arbitrary point in the error plane, by having a random synaptic weight matrix, and moving in the direction of steepest descent. The scheme consists of two distinct passes of computation called the *forward pass* and the *backward pass.*

In the forward pass, the synaptic weights remain unaltered throughout the network and the function signals are computed on a neuron-by-neuron basis. The output of a neuron j is computed as

$$y_j(n) = \varphi\left(\underbrace{\sum_{i=0}^{p} w_{ij}(n)y_i(n)}_{v_j(n)}\right), \tag{1.10}$$

where p is the total number of inputs, n is the number of iteration, $y_i(n)$ is the output of previous layer and w is the weight matrix.

In the backward pass, the error at the output neuron is propagated from the output to the hidden layers and from the hidden layers to the input layers. The weights and the parameters are modified based on the input received from the next layer. The weight update is performed as

$$\begin{pmatrix} \text{weight} \\ \text{correction} \\ \Delta w_{ij}(n) \end{pmatrix} = \begin{pmatrix} \text{learning} \\ \text{rate} \\ \alpha \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_i(n) \end{pmatrix} \cdot \begin{pmatrix} \text{input} \\ \text{signal} \\ y_i(n) \end{pmatrix}. \qquad (1.11)$$

The local gradient $\delta_i(n)$ depends on whether the neuron is an output node or hidden node.

1. If the neuron is an output node. $\delta_i(n)$ equals the product of the derivative $\varphi'(v_i(n))$ and the error signal $e_i(n) = d_i(n) - y_i(n)$ associated with that neuron.

2. If the neuron is a hidden node, $\delta_i(n)$ equals the product of the associated derivative $\varphi'(v_i(n))$ and the weighted sum of the δ's computed for neurons in the next (hidden or output) layer that are connected to that neuron.

The rate of learning is increased by introducing a parameter called *momentum*. The weight update is modified as

$$\Delta w_{ij}(n) = \Omega \Delta w_{ij}(n-1) + \alpha \delta_i(n) y_i(n), \qquad (1.12)$$

where Ω is the momentum.

1.2.3.4 Competitive Learning

In this learning scheme, as the name implies, the output neurons of a neural network compete among themselves for being the one to be active. This type of learning is useful in classification applications where a particular feature of a set of input patterns

may be used to activate a particular neuron. The basic elements of competitive learning are

- A set of neurons that are all same except for some randomly distributed synaptic weights should *respond differently* to a given set of inputs.
- A limit imposed on the strength of each neuron.
- A mechanism allows the neurons in a group to compete with each other, so that *only one neuron* is active at a time. That neuron is called the *winner-takes-all neuron.*

The synaptic weights are distributed among the inputs of a neuron i as

$$\sum_{j=1}^{n} w_{ij} = 1. \tag{1.13}$$

The synaptic weight update is given by

$$\begin{aligned} w_{ij}^{k+1} &= w_{ij}^{k} + \Delta w_{ij} \\ \Delta w_{ij}^{k} &= \begin{cases} \alpha\left(x_i - w_{ij}^{k}\right) & \text{if neuron i wins} \\ 0 & \text{if neuron i loses,} \end{cases} \end{aligned} \tag{1.14}$$

where x_i is input the neuron i.

1.3 Performance Evaluation of Neural Networks

Artificial neural networks, being emulators of human brain, have proved to be excellent performers in many application over traditional approaches. As mentioned in the earlier section, artificial neural networks are implemented in software or hardware and the performance evaluation metrics differ between these two methods of implementations. The performance of software neural networks are limited by the efficiency of the neural algorithm and the computational capability of the conventional computers that run them. Hardware neural networks enhance the performance of the neural algorithms with special hardware for implementing those algorithms. In this case the speed of execution improves many fold and the limiting factor is the cost. The efficiency of solving a problem improves dramatically as we move from traditional methods to special hardware for neural networks.

1.3.1 Evaluation of Neural Algorithms

The performance of neural networks is determined by their capacity and generalization ability or robustness. *Generalization* is the property of a trained neural network to classify an input correctly even if it is not a member of the training set. The *capacity* of the neural network is determined by the amount of information that the neural network can hold. The performance of the neural networks depends on the architecture and learning schemes employed. According to studies in the past, neural networks trained using the *outer-product rule* are characterized by low generalization

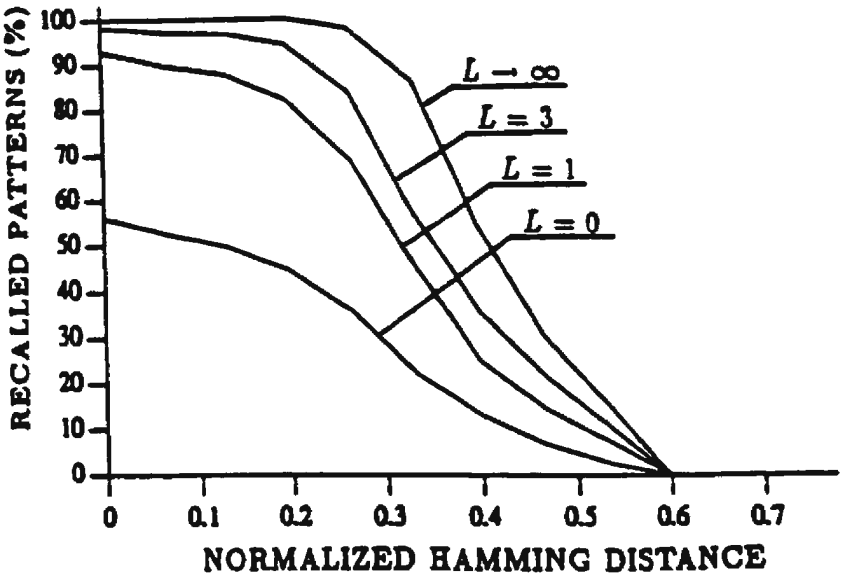


Figure 1.11: Comparison of Learning Schemes [1]

ability and low capacity [1]. The efficiency of these networks in real applications is even lower than predicted.

In [1], experimental results justifying the effect of learning schemes on the generalization ability of the neural network have been presented. The results are for the comparison of an optimally trained neural network to other neural networks of similar size and structure. The results are shown in Figure 1.11. The graphs corresponding to $L = 0$ represent a neural network training using output-product rule and $L = \infty$ represent an optimally trained network. The intermediate graphs represent neural networks with approximated synaptic weights of the optimally trained network. The Hamming distance is the difference between the input set of the test pattern and the input set of the training pattern or the stored pattern. The value of Hamming distance reflects the amount of difference between the testing and training patterns.

The results illustrate that the generalization ability of the neural network degrades as the Hamming distance increases. It also shows that, as the learning scheme changes from optimal to output-product rule, robustness of the network to difference in input patterns degrades. Similar results are presented in the literature for the capacity of the neural network as well.

1.3.2 Evaluation of Neural Hardware

Hardware neural networks are evaluated based on their performance over conventional computers and among the neural architectures. Hardware implementation of neural networks faces constraints due to cost considerations.

In [8], the authors evaluate the performance of digital neuro computers over the conventional computers. They also discuss the constraints on the hardware for integer arithmetics, pipelining, discretization of evolution of learning parameters etc. They discuss the cost associated in changing the learning parameters of a neural network in hardware realizations. They suggest methods for approximating these parameters to fewer values thus reducing the cost. These analyses have a profound impact on the design of the neural network discussed in this thesis. Discussions related to this aspect are done in Chapter 3. The comparison of the convergence speed of training between the conventional computers the neural network hardware is shown in Figure 1.12. In this figure, E is the adequate metric of convergence for a neural network model M and E_0 is some predetermined metric of convergence; t_{cc} is the time required by

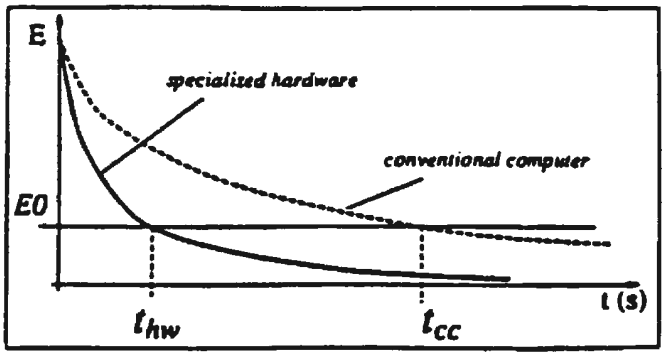


Figure 1.12: Convergence of Conventional(cc) and Neuro(hw) computers [8]

the conventional computer to reach the convergence metrics E_0, and t_{hw} is the time required by the neuro computer to reach that value. This figure clearly illustrates the improvement in performance when special hardware is used. The authors also propose a formula for calculating *speedup* in this case as

$$S_M(E_0) = \frac{t_{cc}(E_0)}{t_{hw}(E_0)}. \tag{1.15}$$

In [9], comparison of digital neural architectures is discussed. Different classes of digital neural implementations are compared quantitatively proposing some performance indices as *reconfiguration ability, virtualization ability* [9] etc. Hardware constraints with respect to implementation of backpropagation algorithm is discussed in [10]. The effect of limited weight resolution, range limitations and steepness of activation function are described. The impact of these parameters on the design of the hardware is discussed in detail in Chapter 3.

1.4 Applications of Neural Networks

Neural networks are being used in a wide variety of applications. The applications range from biological to process control applications. In a broader sense, neural network applications can be classified as detection applications, classification applications, estimation applications and control applications. In case of control applications neural networks are used along with fuzzy logic evolving into the field of neuro-fuzzy control [5]. In the following sections, some applications of neural networks are discussed.

1.4.1 Classification Applications

The application of neural networks to classification problems is conceptually most consistent with their structure and functionality. The objective of a classification application is to assign a random sample from a set of samples to one of finite output states or classes with minimum probability of error. Each sample is described by a set of parameters which form a vector, usually referred to as the feature vector. The development of such a classification system can be achieved by training a neural network to provide an output corresponding to one of the classes, when the input sample belongs to that class. The justification for use of neural networks in classification applications depends on the existence of evidence that neural network classifiers are more efficient than the alternate tools. An example classification application is described in the following subsection.

1.4.1.1 Classification of SONAR Signals

A neural network developed for classification of SONAR targets is described in [11]. The authors of this paper have analyzed the effect of hidden layers in the classification of SONAR targets. A similar application is described in [12], where the authors test the effect of finite precision calculation on the performance of the neural network.

1.4.2 Detection Applications

Detection applications are a degenerate of classification where a set of input belongs to one of two classes. Applications include pattern recognition, fault detection, medical imaging, quality control etc. One example of detection applications is presented in [13]. The authors describe the use of artificial neural networks in detecting known signals in non-Gaussian noise [13]. Another example is presented in [14]. This paper describes an application of artificial neural networks in medical signal processing. The authors describe the training and performance of a multilayered perceptron using backpropagation training for detection random signals in medical signal processing. The authors also compare the performance with other classical techniques for the same application. One more detection application in medicine is presented in [15] which is EEG spike detection using neural networks.

Artificial neural networks for fault detection is explained in [16]. This paper describes a neural network approach for the problem of sensor failure detection and identification for a flight control system without any sensor redundancy. Detection of

soft contaminants using neural networks is discussed in [17]. This paper describes a neural network based image analysis system that detects foreign objects that might be present in bags of frozen corn kernels which are not visible to a conventional camera. The following subsection discusses some of the applications in power systems, the application area of the work presented in this thesis.

1.4.2.1 Applications in Power Systems

Neural networks play a vital role in applications related to power systems due to the non-linearity of the system. A survey of the literature shows the use of neural networks in many areas of power system like distance protection, load forecasting, stability analysis, economic dispatch, security assessment etc. Contributions to the field vary from neural algorithms to dedicated hardware implementations.

Coury and Jorge [18] suggest an artificial neural network approach to distance protection of transmission lines. They describe ANN as a pattern classifier, being able to recognize the changing power system conditions and consequently improving the performance of ordinary relays. They use a Multilayered perceptron (MLP) for this purpose, with magnitude of phase voltages and currents as inputs to the ANN and a trip / no-trip as the output of the ANN. They claim improved performance of ANNs over the conventional approaches. Similar approach has been described in [19] using frequency components as inputs to the ANN instead of magnitudes of voltages and currents. Improvement in learning and convergence rate has been reported. The

work described in these papers is closely related to this thesis. Further explanation about these papers and analysis are presented in Chapter 3.

Cornu *et al.* [20] present a Kohonen feature map algorithm for security monitoring in power transmission systems. They describe the implementation of the algorithm in parallel hardware. They describe the development of a SIMD (*Single Instruction Multiple Data Stream*) array dedicated to the implementation of the algorithm. This is one of the examples of dedicated neural hardware for applications in power systems. More explanation on the development of hardware neural networks is given in the following Chapter.

1.4.3 Estimation and Prediction Applications

A large portion of scientific research is devoted to the development of systems for prediction such as weather forecasting, medical diagnosis, financial predictions, lightning strike prediction etc. Neural networks are suitable candidates for the development of systems predicting such events, due to their nonlinear structure and generalization ability. The application of neural networks to prediction application requires the determination of the parameters of the system under consideration, that most likely affect the the events or developments of interest. Provided that such a set of parameters is chosen, the network can be trained using the history of the system under consideration. After the training, the neural network must be able to use the most recent parameters in order to predict the future events or developments. The use of

neural networks for such application is based on the hypothesis that the future events or developments depend exclusively on the history of the system. Although this is the case in many systems, this hypothesis is not always true. A classical application of neural networks to weather forecasting by Widrow *et al.* is presented in [1]. They used artificial neural networks to predict the occurrence of rainfall on the following day on the basis of fluctuations in the barometric pressure in the two preceding days. The percentage of successful predictions was comparable to those predicted by the official weather prediction agency, which used a large set of parameters for forecasting.

1.4.4 Control Applications

Artificial neural networks, mimicking the human brain have demonstrated to be an attractive solution for control applications requiring some intelligent control. The application of neural network control ranges from control of electric drives to control of communication systems. Use of neural networks for the identification and control of nonlinear dynamical systems is described in [21] by Narendra and Parthasarathy. This is one of the pioneer works in the field of control using neural networks. The authors of this paper explain the practical feasibility of neural networks in identification and adaptive control schemes. The authors introduce models in which multilayer and recurrent neural networks are interconnected in novel configurations.

Neuro-fuzzy control [5] is another popular approach for intelligent control applications. It refers to the design methods for fuzzy controllers that employ neural network

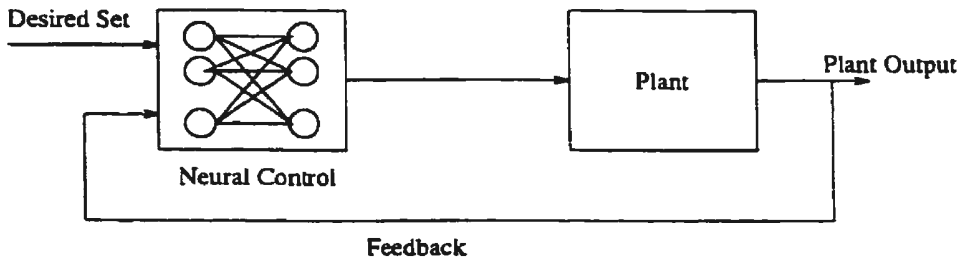


Figure 1.13: General Model of a Neural Controller

techniques. Some advantages of neural control over traditional controllers are:

1. Learning ability
2. Parallel Operation
3. Structured Knowledge representation and
4. Better integration with other control design methods.

A general model of a neural network controller is shown in Figure 1.13. An example of application of neural network control for robotic manipulator control is presented in [22]. The experimental development of a trajectory tracking neural network controller based on the theory of sliding motor control is shown. The authors have implemented the controller on a 3 DOF PUMA robot. They have also compared the performance of the neural control with that of computer torque method control and continuous sliding motor control with PI-estimator.

Another example of ANN based control is presented in [23]. This is for the control of communication system. The authors suggest that the neural networks appear well suited to applications in the control of communication systems for two reasons, adap-

tivity and high speed. They describe the application of neural network control to two problems: *admission control*, selective admission of a set of calls from a number of inhomogeneous call classes which may have different characteristics and *switch control*, the service policy used by a switch controller in transmitting packets. They address sub-microsecond optimization of these problems based on the scheme suggested.

1.5 Motivation for the Work

The earlier sections described the application of neural networks in a wide variety of applications. Different methods of implementations of artificial neural networks, software and hardware, were discussed. The performance of hardware neural networks in comparison to software implementations on conventional computers was also discussed. The advantage of using hardware neural networks is very clear from these discussions. Moreover, conventional computers do not exploit the inherent parallelism in the neural algortihms except for optimizations at the compiler level. A dedicated neural network hardware for a particular application would definitely increase the speed of a system. This would also increase the reliability of the system. So, a dedicated neural network hardware with novel design features would be a major contribution to the field of artificial neural networks. This would also be a contribution to the field of large scale integration and system on a chip research.

As discussed in section 1.4.2.1, neural networks are a vital tool in distance protection of transmission lines. As discussed in the literature, use of artificial neural

networks improve the efficiency of the protection system. If the distance protection using artificial neural networks could be implemented in a single application specific integrated circuit (ASIC), it would improve the protection system performance many fold. The paper by Coury *et al.* [18], explained in section 1.4.2.1, uses a software implementation of an artificial neural network. The authors present a learning time of 2 CPU hours and convergence at 80,000 cycles. The results of the implementation, though better than conventional approaches, are not attractive with the low convergence rates. The paper by Zahra *et al.* [19], mentioned in section 1.4.2.1, also uses software implementation of the ANN based approach to protective relaying. The speed of operation in these cases will be less when compared to a hardware implementation of the same.

The distance protection problem needs to be analyzed in detail to identify proper preprocessing methods and a suitable neural network structure, which would be feasible for implementation in hardware. A software neural network simulator which resembles the hardware implementation would acomplish this purpose. A hardware complexity optimization analysis also has to be done using the software simulation. In summary, a neural processor that is optimized (at the same time possessing adequate generality for application to similar problems) for this application has to be designed with proper preprocessors.

1.6 Organization of Thesis

In this chapter, the basics of neural networks, the classification of neural architectures and the different learning schemes were discussed. A brief description of the work done in this thesis and the motivation and background for this work were described. This chapter also discussed some applications of neural networks.

Chapter 2 discusses hardware neural networks in detail, with emphasis on VLSI neural network architectures. A survey of recent development in VLSI neural networks is given and they are classified into different categories. The chapter discusses in brief the neural network designed for this work with respect to the categories described.

Chapter 3 describes the the distance protection problem in detail. The method of solving the problem is discussed and the simulator developed for this purpose is also explained. The results of the simulation are discussed in detail and their relation to the hardware design is explained.

Chapter 4 describes the hardware design process and explains the implementation in detail. The overview of the architecture is discussed and the design is discussed in detail. Salient features of the design are described and justified.

Chapter 5 summarizes and concludes the work. The main contributions of this thesis are described. Some improvements to the current software and hardware designs are discussed. Critical assessment of the work is done and the method of approach is justified.

Chapter 2

Hardware Neural Network Architectures

2.1 Introduction

Neural networks are a promising computational technology due to their capabilities in modeling and solving problems hardly approachable by traditional methods. As the field of neural networks matures, a strong need for fast, efficient and application specific hardware for neural networks arises. In the previous chapter, basics of ANNs were discussed. Classifications of ANNs and application of ANNs were also discussed. Some literature on the performance of ANNs and methods of evaluation were described. This chapter discusses the hardware neural networks and their categories in detail with emphasis on VLSI architectures. Recent trends and developments in hardware ANNs are discussed. A brief explanation on the digital neural processor designed for this thesis is given and some features of the design are presented.

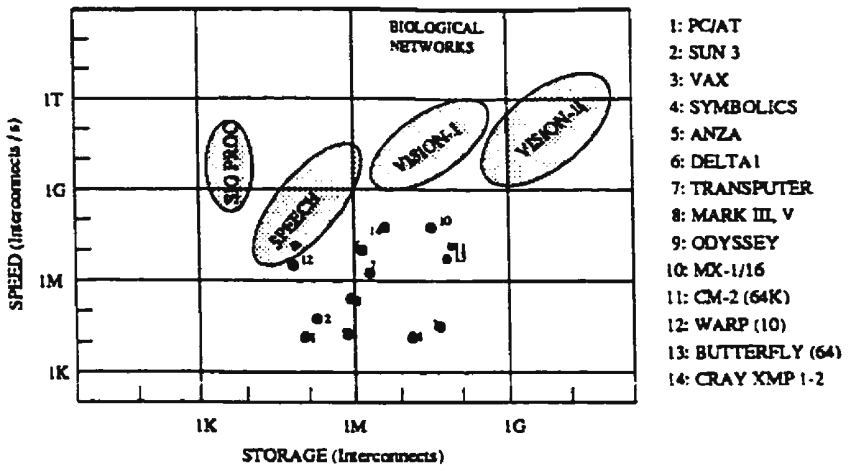


Figure 2.1: Computational Capabilities Vs. Requirements [24]

2.2 Hardware Neural Networks

Neural computing requires a tremendous number of computations and communications. The response and characteristics of the present models of ANN are primarily investigated by simulations run on workstations, special co-processors or transputer arrays. The fundamental drawback of such simulators is that the spatio-temporal parallelism that is inherent to ANNs is lost completely or partly. The computational capabilities of ANN simulators and the computational requirements of some ANN applications is illustrated in Figure 2.1 [24]. This figure clearly shows that general purpose computing machines do not meet the computational requirements for most of the applications. An appreciable reduction in computing time becomes possible with special neural hardware enabling execution of large tasks in real-time. Apart from

the improvement in execution time, special neural hardware reduces the size of equipment compared with simulators for the same task. The special neural hardware can be *general purpose neuro computers*, computers specially designed for executing neural algorithms, or dedicated *custom processors*, which are special hardware optimized for particular applications. The implementation methods for the neural hardware is classified as Direct Design and Indirect Design [25]. Direct Design is mapping the structure of a ANN model directly into hardware and indirect design is mapping ANN models into existing array processors, thus reducing the hardware complexity over single chip direct designs. The following section discusses these categories in detail.

2.3 Classification of Hardware Neural Networks

The widespread interest in hardware neural networks resulted in a number of implementations that are hard to overlook. Several books and survey papers on hardware neural networks have been published in the recent years [26, 27, 28, 29, 30, 31]. Each reference describes a different method of classifying the existing hardware implementations of neural networks. In general, the classification of hardware neural networks are *analog*, *digital* and *hybrid*, based on implementation . In [30], the authors classify the digital neural networks based on five criteria which are

- Type of system
- Numerical representation

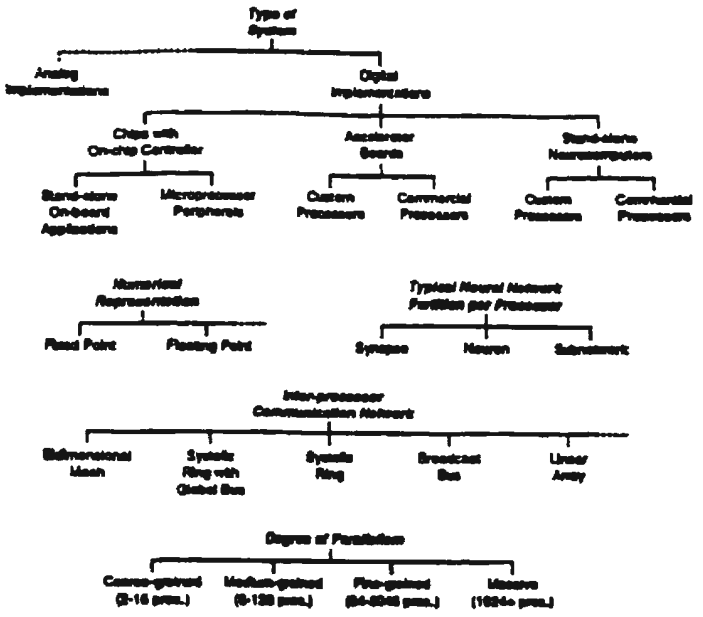


Figure 2.2: Classification of Digital Neural Hardware [30]

- Typical neural network partition per processor
- Inter-processor communication network
- Degree of parallelism.

The classification is illustrated in Figure 2.2.

In [31], the authors use a different classification based on the dedication of the hardware. They classify neural network hardware as VLSI chips, accelerator boards and multi-board neural computers, Most of the commercially available neural hardware are general purpose, programmable, reconfigurable implementations with limited number of processing elements [26]. Based on the classifications presented in the literature, the neural hardware can be classified as illustrated in Figure 2.3. As the classification shows, the indirect design methods use the existing parallel processors to implement neural algorithms. These implementations are mostly general

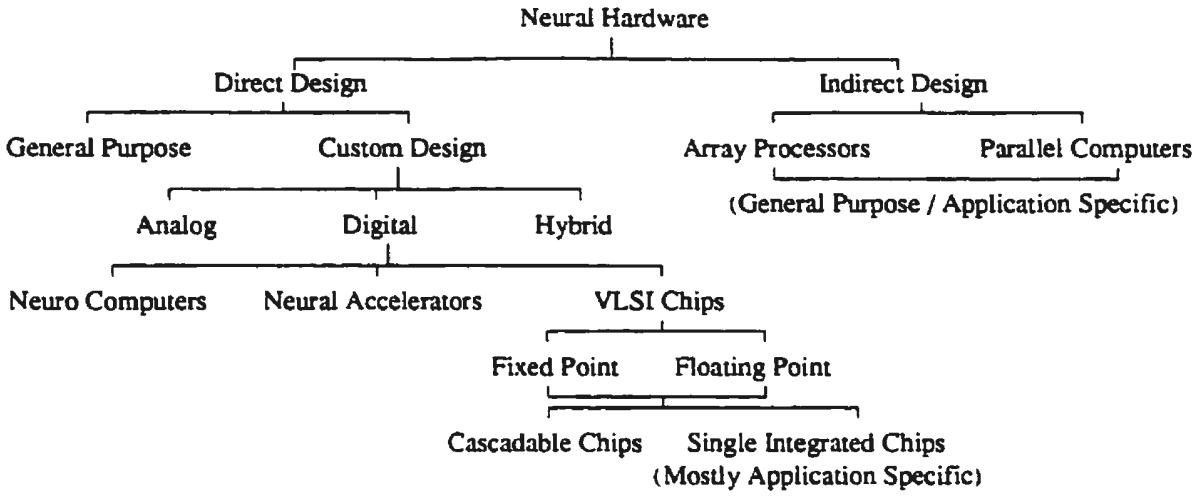


Figure 2.3: Classification of Neural Hardware

purpose neuro computers, though they exhibit less reduction in hardware complexity than application specific designs, provide good improvement in execution times when compared to the simulators. Custom design techniques involve more design issues like precision requirements, speed of operation etc. In the following sections, more explanation on the custom design of neural hardware with emphasis on digital implementation is given. Some example architectures, including commercially available architectures, are discussed. A compilation of some commercially available architectures and their features, with respect to the classifications discussed above, is given in [29, 28, 31].

Dedicated neural hardwares are naturally affected by the implementation technologies, discussed earlier in the section. Both analog and digital design techniques have demonstrated some degree of success in their areas of application. To select be-

tween digital and analog implementation techniques for neural hardware, many issues like storage and transfer of analog signals [25], the speed and precision achievable, as well as adaptivity and programmability, need to be better understood. A survey of trends in implementation techniques reveals transition from analog techniques to digital techniques. In [29], published in 1992, the authors review developments in electronic neural nets in North America during that period. In their review, they mention that analog implementations are more prevalent than digital implementations. Out of over 40 chips they have referred, only 8 are exclusively digital. In [28], published in 1993, the author mentions that the analog approach is dominant in the United States and digital techniques are preferred in Europe and Japan. This trend of analog implementations seems to have moved towards digital implementations during the recent years. The review in [27, 31], confirms this transition, where the authors mention that digital implementations are widely used and a significant fraction of neural hardware uses digital implementation. This view is supported by the architectural survey of digital neuro computers in [30].

2.3.1 Analog Implementations

Features of analog design are speed, low precision and small scale systems (single programmable interconnectable neurons or small ASICs). For dedicated applications, a neuron can be easily implemented by a differential amplifier [32, 25], with the synaptic weights implemented via resistors. This way, many neurons can be fit into one single

chip. The asynchronous updating properties of analog devices can provide extremely high speed computations that are qualitatively different from those of any digital computer [25]. Analog circuits also offer inherent advantage on the computation of sum of weighted inputs by currents or charge packets and the nonlinear effects of the devices facilitating realization of a sigmoid type function. Although analog circuits are more attractive for the biological-type neural networks, they are more susceptible to noise, cross talk, temperature effects, power supply variations etc. In general, analog circuits are limited to low precision implementations.

2.3.2 Digital Implementations

Digital implementation is suitable for dedicated connectionist type neural networks [33]. Digital techniques offer some desirable features such as design flexibility, learning, expandable size and accuracy. Digital designs have overall advantages in system level performance. Moreover, digital implementations provide more flexibility in precision than the analog techniques. Development of CAD technology also helps convenient building of modular designs with digital techniques. The disadvantages of digital implementations are: larger chip area, relatively low speed of operation, especially in the sum of weighted inputs, and conversion of analog inputs to digital form. As illustrated in Figure 2.3, the digital implementations are classified into VLSI chips, neural accelerator boards and neuro computers.

2.3.2.1 VLSI Chips

Digital implementations of this category can be a single processing element which is cascadable or multiprocessor chips, which contain many processing elements on one chip. Based on the number of processing elements in a chip, the chips can be *coarse grained*, *medium grained*, *fine grained* or *massively parallel*. The advantage of this implementation is that, generally they are optimized for a particular application and hence have a high speed and a good accuracy. The disadvantage is their custom design as they cannot adapt to changes in neural algorithms or they give poor performance for newer, improved algorithms (provided it can be programmed for accommodating different algorithms). Some example architectures are discussed in Section 2.4.

2.3.2.2 Neural Accelerators and Neuro Computers

Very large networks can be achieved by specialized neural hardware. While large general purpose parallel machines provide sufficient performance, alternatives are available with accelerators for conventional computers. Neuro computers also provide better performance with extensive software environments. Some of the available neural accelerators and neuro computers, as provided in [31], are listed in Table 2.1. Several of these accelerator cards use fast RISC chips or DSP based co-processors to speed up the network processing. These cards usually come with software that include several neural network algorithms. A disadvantage of these co-processor cards, as explained in [31], is that they do not allow signals directly to the card but over the

Type	Name	Chip	Performance
PC Accelerators	AND HNet Transputer	Transputer T400	Not Available
	BrainMaker	T1 TMS320C25 DSP	40MC, 500MF
	Current Tech. MM32k	2048 PE / Chip	4.9MC, 2.5MCU
	HNC Balbo 860	Intel i860	80MF
	IBM ZISC ISA	IBM ZISC036	800k pat/sec
	NeuralTech NT6000	T1 TMS320C20 DSP	2MC
	NeurodynamX XR50	Intel i860	45MC
	Nestor Ni1000	Nestor Ni1000	40k pat/sec
	Rapid Imaging 0491E1	Intel ETANN	2GC
	Telebyte 1000 NeuroEng.	properietary	140MC
	Vision Harvest NeuroSim.	Intel i860	30MC, 100MF
	Ward Sys. NeuralBoard	50MHz RISC	25MF
Neurocomputer	Adaptive Sol. CNAPS	Inova N64000	5.7GC, 1.5GCU
	HNC SNAP	HNC 100 NAP	500MC, 128MCU
	Siemens SYNAPSE-1	Siemens MA-16	800MC

MC - MCPS, MCU - MCUPS, MF - MFLOPS, GC - GCPS and GCU - GCUPS

Table 2.1: Neural Accelerator Cards and Neurocomputers [31]

slow PC bus. This reduces the advantage of using such cards for real-time processing. More discussion on some of the neuro computers is provided in section 2.4.

2.3.3 Hybrid Implementations

Hybrid designs combine the best of analog and digital techniques. Typically the external inputs and outputs are digital to facilitate integration with other digital systems, while internally some or all of the processing is analog. The AT & T ANNA (*Artificial Neural Network ALU*) [34] is an example of a hybrid implementation. This chip is externally digital but uses capacitor charge, periodically refreshed by DACs, to store the weights. Some other hybrid designs use digital weights but the processing is done in analog.

2.4 Example Architectures

Some of the commercially available hardware neural networks and some architectures developed by research groups and academic institutions are presented in this section. The commercial architectures are general purpose, programmable and cascadable implementations while the designs from research groups are mostly application specific implementations. The discussions on these examples give only an overview of the architecture of the hardware. Intrinsic details of the designs are given in the respective references.

2.4.1 ETANN from Intel Corporation

ETANN, the Electrically Trainable Analog Neural Network (80170NW) is the first commercial chip implementation for the general purpose application of neural networks [26]. The architecture of the ETANN chip is shown in Figure 2.4. It consists of 64 neurons and 10,240 synapses. A total of 160 synapses is connected to each output neuron. There are 128 configurable inputs available in the chip. The neuron also performs the sigmoid function for dot product between the input signal and the weight value from the synapse array. High performance is achieved through full-fledged parallel processing. The chip has feedforward processing rate of 2 GCUPS and it can support 100KCUPS learning rate for the individually addressable weight update. Learning is implemented by an off-chip approach for maximizing flexibility in order to support various learning algorithms such as the backpropagation and competitive learning. The off-chip learning is also a disadvantage. The chip has to be used in conjuction with a host station for learning and downloading the weights. This chip also has the disadvantage of analog implementation which restricts the resolution of signals. Typical resolution of the output signal is around 6 bits which is much less when compared to many other chips reported. The chip is used mostly in pattern recognition and image processing applications.

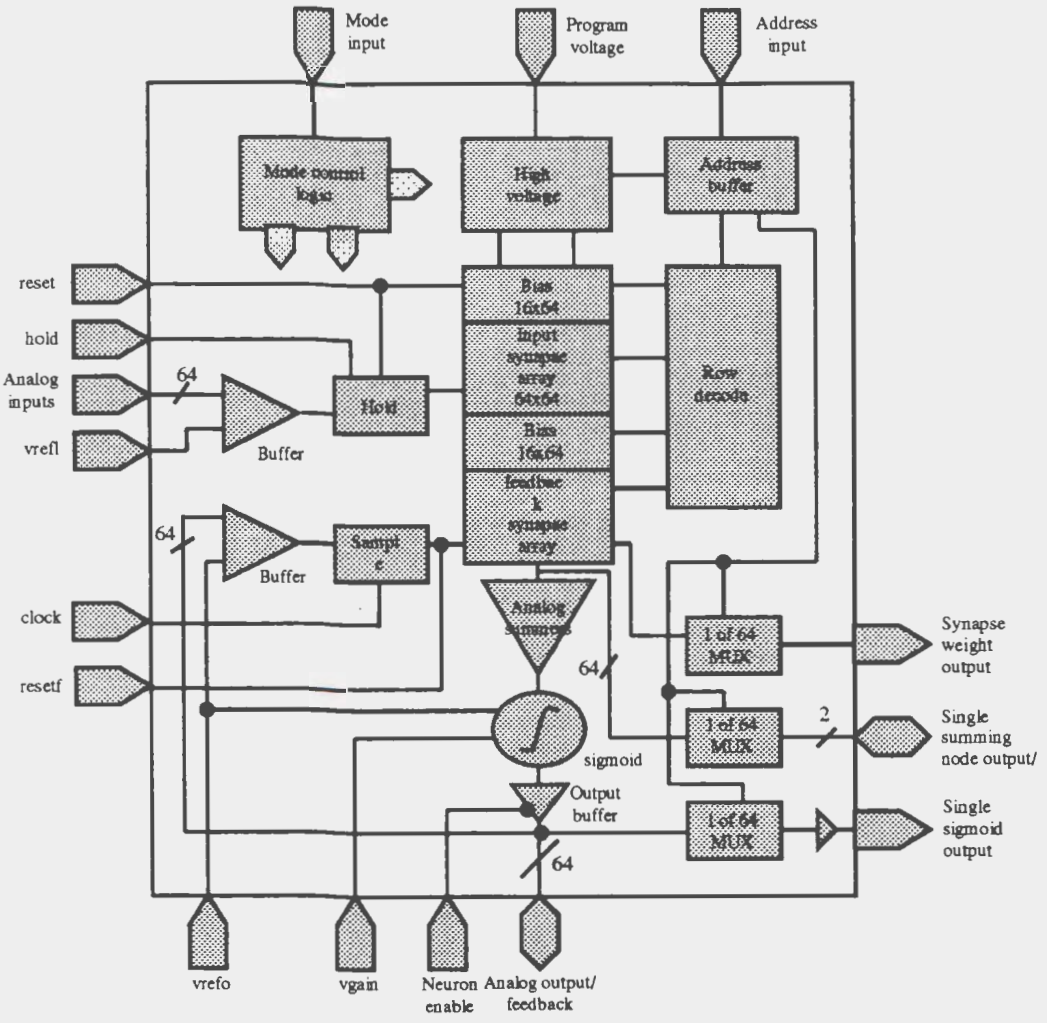


Figure 2.4: Architecture of ETANN chip [26]

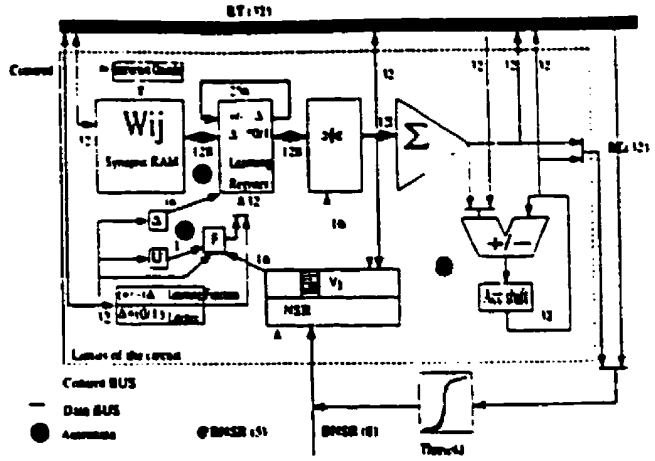


Figure 2.5: L-Neuro 1.0 Processing Element [35]

2.4.2 L-Neuro 1.0 from Philips

L-Neuro 1.0 [35] is an example of chips for sigmoid networks. The structure of the processing element of L-Neuro 1.0 is shown in Figure 2.5. In this architecture, the weights of each neuron are stored on-chip. On kilobyte of memory is arranged as 8 bit weights for 64 neurons with 16 inputs each. The design of this architecture allows reconfiguration of weights to be 4 bit weights for 256 neurons. Double precision is used for the learning process and the maximum number of neurons in this case reduces to 32. In the forward phase, a single serial-parallel multiplier performs the product and sums for a matrix vector product. Each neuron is processed sequentially, producing a single output at a time so that the external nonlinear function (a look up table) can be used by each neuron. Operation for Hebbian learning (Delta Rule) is implemented, but not the complete backpropagation algorithm. This has to be done

in the host processor, thus considerably reducing the backpropagation performance. The chip is cascadable but networks whose weights exceed the size of the on-chip memory cannot be implemented due to the low bandwidth from external memory to the internal storage. This chip is suited for small embedded applications along with traditional microcontrollers. Due to the absence of direct memory interface and limited parallelization, conventional microprocessors of future generations can easily outperform this design. An improved version of L-Neuro 1.0, called the L-Neuro 2.3 is presented in [36], overcomes the major limitations of its predecessor. It consists of an array of twelve DSPs. The new chip is able to perform 2 Giga arithmetic operations per second and has a throughput of 1.5 Gigabytes per second.

2.4.3 HNC100 Chip from HNC

HNC's processing element [30, 25] has some features of traditional processors like floating point computations and its structure is simple and orthogonal. The HNC100 processing element is shown in Figure 2.6. The core of the processing element is a 32 bit floating point multiplier and a 32 bit ALU, handling both floating point and integer operands. There are data registers, instruction registers and status registers around these functional units. The number of processing elements per chip is limited to four due to the floating point implementation. The communication between memory and processing elements is performed through bidirectional datapaths between local memory and processing elements, global memory and processing elements and

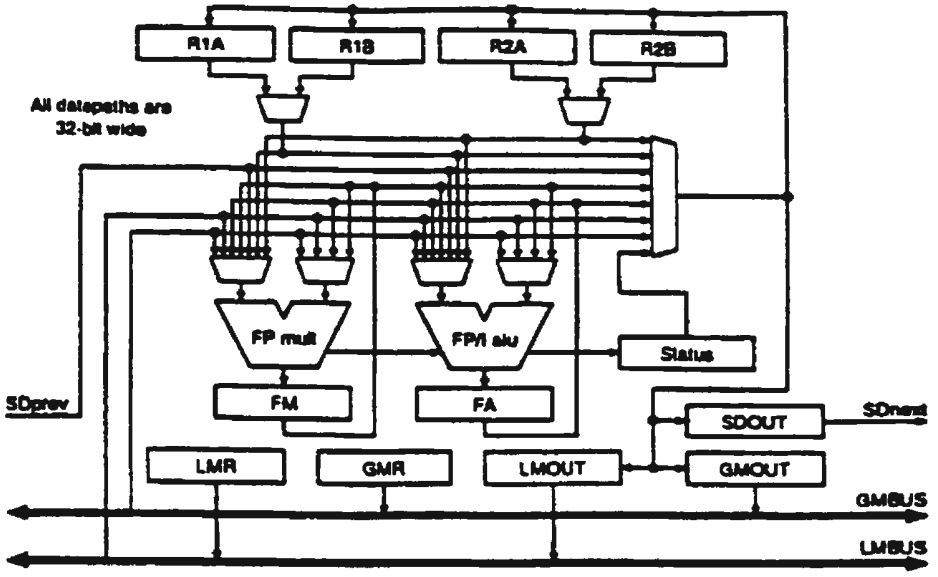


Figure 2.6: HNC100 Processing Element [30]

between neighboring processing elements. Many HNC100 chips are connected in a systolic ring structure to form the SNAP (SIMD Neurocomputer Array Processor) system [25]. The architecture of the SNAP system is shown in Figure 2.7. A complete SNAP system consists of 16 to 64 processing elements on several boards.

2.4.4 N64000 Chip from Adaptive Solutions

This chip is one of the examples of parallel neuro computers using programmable custom processing elements. Adaptive Solutions CNAPS [37] is one of the first commercial large neuro computers. This uses the regularity of the broadcast bus architecture [9] to reconfigure faulty elements (by bypassing) and improve yield. The architecture of the N64000 processing node is shown in Figure 2.8 and the CNAPS Inter-chip communication is illustrated in Figure 2.9. As the figure illustrates, the connectivity between processing elements is reduced. This gives the advantage of expansion by

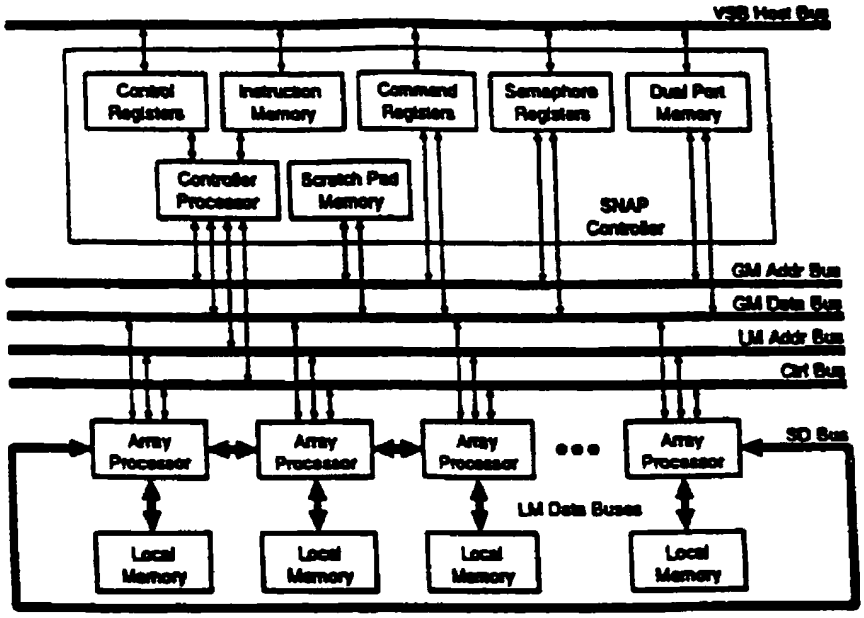


Figure 2.7: SNAP system architecture [30]

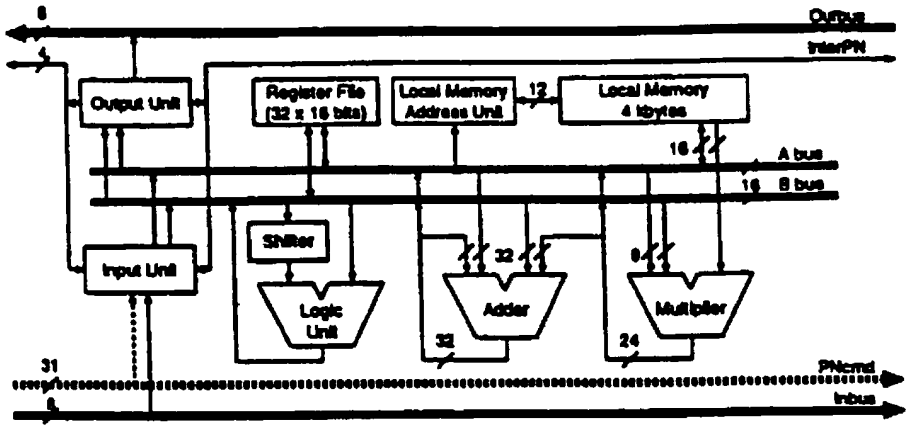


Figure 2.8: Architecture of N64000 processing element [30]

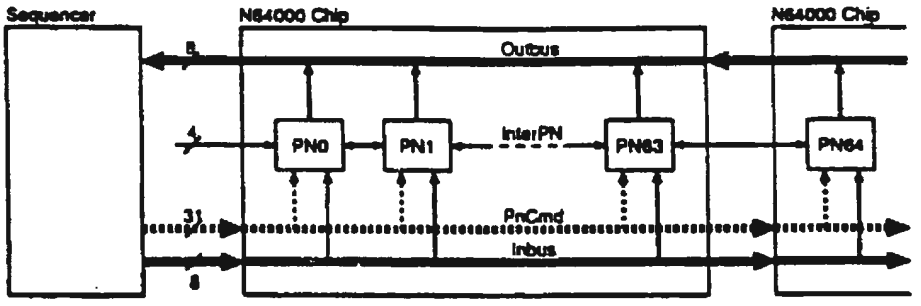


Figure 2.9: CNAPS Inter-chip Communication [30]

simple addition of N64000 chips on the bus and reduction of packaging and mounting costs. The processing element is similar to a very simple DSP and each PE (denoted by PN0 to PN64 in the figure) holds a row of the weight matrix and accumulates the products of the inputs and internal elements of the matrix. The weight update in error backpropagation is achieved by duplicating the weight matrix in the processors and both matrices are updated one after the another. The performances as reported by a study in [30], is 9.671 GCPS and 2.379 GCUPS. The great advantage of the CNAPS architecture is the versatility of the processing elements and good programmability.

2.4.5 MANTRA 1 from EPFL

This is an architecture from the research institute EPFL (École Polytechnique Fédérale De Lausanne) in Lausanne, Switzerland. MANTRA 1 [30] is a systolic mesh processor for implementing neural algorithms. This design attains one more degree of parallelism by assigning up to one processing element per synapse. The advantage

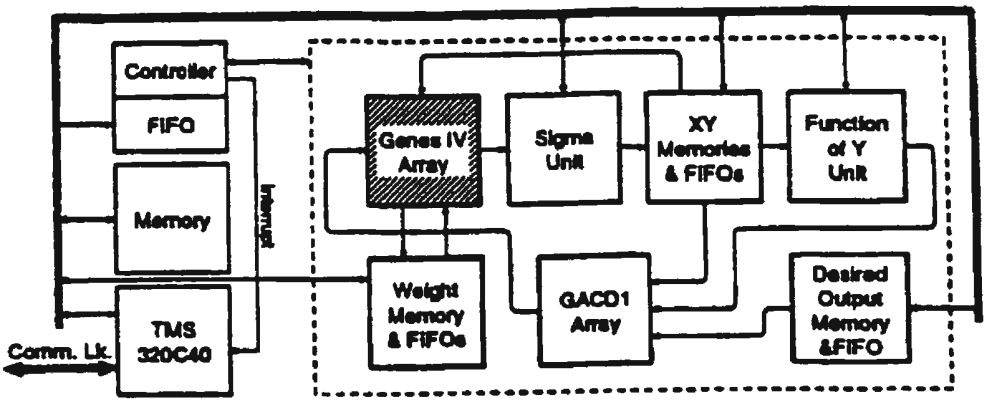


Figure 2.10: The MANTRA 1 Architecture [30]

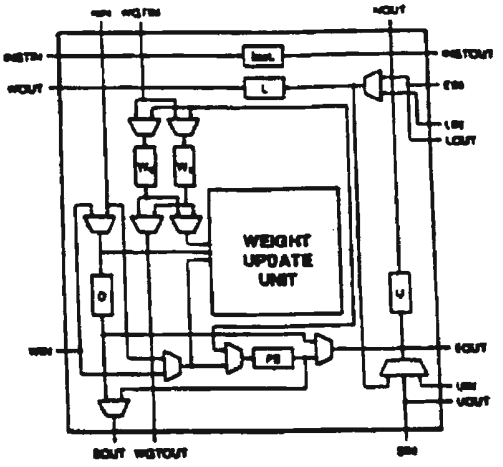


Figure 2.11: The Genes IV Architecture [30]

of this method is, higher degree of parallelization and hence higher throughput and a better PE utilization. The computational heart of this system is a bidimensional mesh of custom processing elements called GENES IV [30]. The structure of the processor is shown in Figure 2.10. The structure of the Genes IV processing element is shown in Figure 2.11. All the input and output operations are performed by the processing elements located in the North-West to South-East diagonal. The authors explain that the processing element implements a few general primitives sufficient for

backpropagation, Hopfield nets, Kohonen feature maps etc. They claim that 100% utilization rate is achieved in normal conditions. The array implemented in MANTRA 1 can contain up to 40 x 40 PEs running at 8 MHz. The system is controlled by a Texas TMS320C40 processor, which takes care of the SIMD part, instruction dispatching and input/output management. The processor also controls communication with the host computer.

2.4.6 HiPNeT-1 from ICSI

The International Computer Science Institute (ICSI) at University of California, Berkeley, presents a highly pipelined neural network architecture called the HiPNeT-1 in [38]. The authors claim that the system sustains a learning rate of one pattern per clock cycle. At a clock rate of 20MHz each neuron performs 200 MCUPS. Multiple such neurons are integrated onto a single VLSI chip. The architecture of the HiPNeT-1 neuron is shown in Figure 2.12. The pipeline operates in two basic modes, *forward* and *update* modes. In the forward mode, weight values are read from memory in one cycle and added to the accumulator in the next. In the update mode, value of delta weight Δw_{ij} is read from the error input latch and stored in the accumulator. Each weight is read from the memory, added to the update and written back to the memory. But a read after write pipeline hazard is ignored assuming backpropagation learning does not cause this hazard. The authors justify this assumption with simulations showing that the performance is not affected.

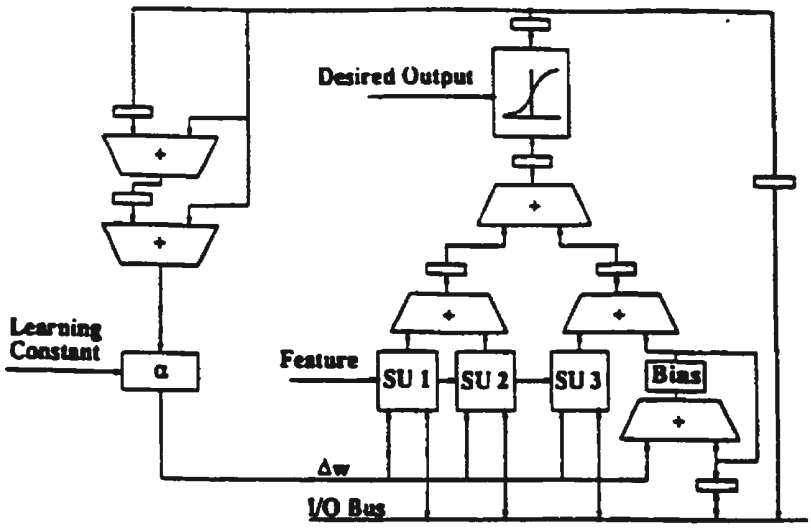


Figure 2.12: Architecture of HiPNeT-1 Neuron [38]

2.4.7 Neural ASICs

The architectures discussed in the earlier sections are general purpose, massively parallel architectures for neural algorithms. In this section, two custom designed architectures for specific applications are discussed.

2.4.7.1 Neural ASIC for real-time classification

A neural ASIC architecture for real-time classification is presented in [39]. The authors have designed a digital ASIC module which is run-time reconfigurable. The ASIC module is a multilayered perceptron (MLP) and a tree of MLPs are formed by connecting two of these modules. The authors state that the design combines high speed and precision. The architecture is presented for variable precisions and VLSI implementation is done using 8 bit integer arithmetic. The design is based on the MLP algorithm and is optimized for parallel execution. This is achieved by

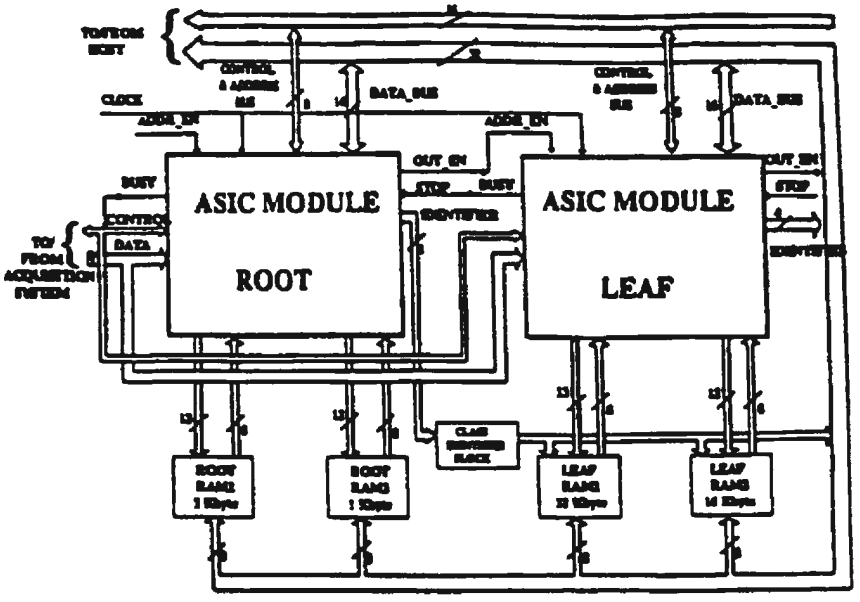


Figure 2.13: Neural ASIC architecture for classification [39]

interchanging instructions of the algorithm to attain maximum parallelism and implementing it in hardware. The disadvantage of this design is that it implements only the forward phase of the MLP algorithm and does not constitute learning. The learning has to be performed in software. The architecture is shown in Figure 2.13.

2.4.7.2 Neural ASIC for supervision of water pollution

The design of a neural ASIC that implements a system for low cost supervision of water pollution is presented in [40]. A trainable multilayer perceptron is designed which estimates the parameter to estimate the water quality. The architecture includes weight multipliers, product sum, sigmoid function and backpropagation. The architecture of the neuron is shown in Figure 2.14. The design has 8 neurons in the first layer and one neuron for the output layer. The design is implemented using 0.7 μ CMOS technology and 8 bit integer arithmetic. More general purpose and application

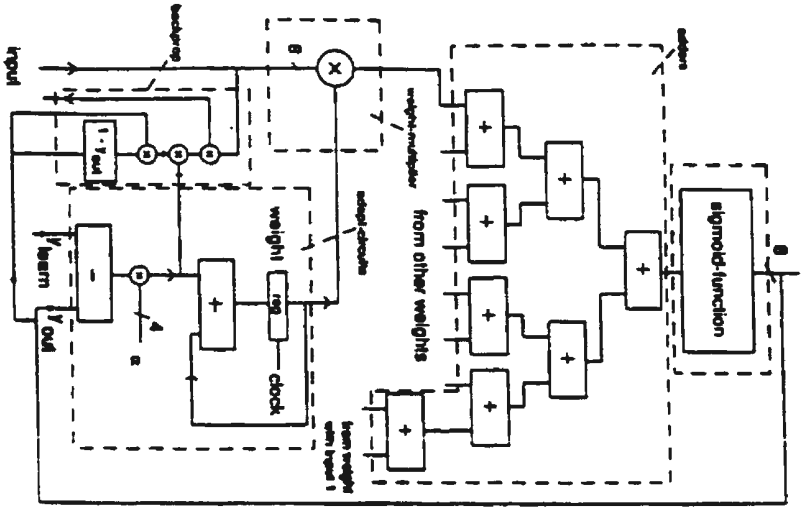


Figure 2.14: ASIC architecture for supervision of water pollution

specific designs are presented in [30, 41, 42, 25, 43, 44, 45], for further reference.

2.4.7.3 A Single Chip ASIC for Image Processing

A digital implementation of the recall phase (after training) of a backpropagation neural network for real-time image classification is presented in [46]. This implementation is application adjustable and has been implemented using similar procedures followed in this thesis. The authors claim that a network with up to 65536 inputs, 8 hidden neurons and 32 output neurons is possible. The input data range is $\approx$ 0.0...,1.0 with 8 bit resolution. The architecture of the chip, NeNEB is shown in Figure 2.15. This design is used for a real-time image classification application and uses fixed point representation for the inputs and weights. The design uses external weight storage scheme, i.e. the training is done offline and the final set of weights are loaded for use with external inputs. The design has been verified for its functionality

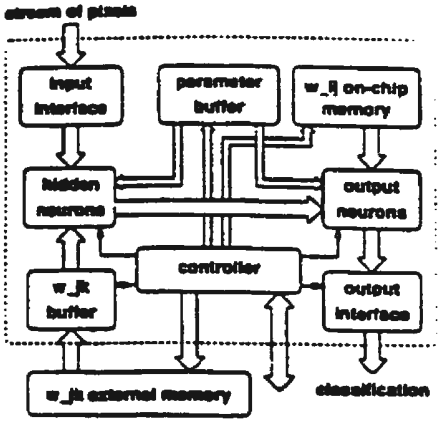


Figure 2.15: Architecture of NeNEB [46]

in comparison with the results of software simulation using a program in C language. The design is mostly suited for applications that would require low resolution. This restricts the area of application of this design.

Different commercially available and academic research level architectures of hardware neural networks were discussed in the earlier sections. Most of the commercially available, chip level architectures consisted of complex neurons that can be programmed for different applications. The hardware complexity of these designs were very large and they had very few neurons on one chip. On the other hand, some other designs had many neurons, as many as 1024, in a single chip but they were simple and can be used for only limited applications. The neuro computers that were discussed are mainly for huge applications that would require massive parallelism in their execution. But for the problem addressed in this thesis, a single chip that is

optimized for the chosen application would be most suitable. This is possible only with a custom designed neural processor chip that meets all the requirements of the application. Moreover, new ideas can be incorporated in the design which would improve the overall system efficiency. Besides, this would be a good contribution to the research in hardware neural networks.

2.5 Classification of DIANNE-D1.0

DIANNE-D1.0 (Digital Artificial Neural Network - Detector, Version 1.0), the digital neural processor developed for this thesis is a custom designed architecture with on-chip learning. Although the design is focused towards detection applications, it can be used for other applications which require similar structure and size. The partition per processing element of this design is a neuron, *i.e* a neuron forms a processing element. Eleven such neurons form the processor with four neurons in the input layer, six neurons in the hidden layer and one neuron in the output layer. The design includes an on-chip preprocessor for the example application chosen, distance protection of power transmission lines. The device can be configured to bypass the preprocessor and receive external inputs directly. The processor can be configured to learning mode implementing backpropagation algorithm or test (run) mode with the stored weights. The architecture is an interleaved pipeline structure so that all the neurons function simultaneously in real-time. The layers are pipelined so that the throughput is increased. The design is implemented using 0.5 μ CMOS technology.

More explanation on the architectural design of DIANNE is provided in Chapter 4.

2.6 Summary

In this chapter, hardware neural networks were discussed in detail. The need for hardware neural networks and the advantages and disadvantages of different methods of implementation of hardware neural networks were explained. A classification of hardware neural networks compiled from the literature survey was presented. A brief discussion on the digital neural processor designed for this thesis was presented and the features of the design were specified. Some of the commercially available neural network hardware and other interesting application specific designs were explained. A compilation of alternatives for massively parallel neural hardware was also presented.

Chapter 3

Problem Description, Software Design and Performance Analysis

3.1 Introduction

In the preceding chapters, basics of ANNs and evolution of hardware ANNs were introduced. Different categories of architectures, methods of implementations and training schemes were discussed. Some of the commercially available hardware neural chips, neural accelerators and neural computer boards were presented. Performance evaluation of hardware ANNs and methods of analysis were presented. A brief description of the neural processor developed for this thesis was given. In this chapter, the problem chosen for implementation is described. Discussion on the software design of the ANN and the preprocessing methods used on the inputs to the ANN are described. Detailed explanation on the simulator developed for simulation of the ANN used for this work is given. The performance analysis of the ANN using the software simulator is presented and quantization analysis which would affect the hardware implementation is addressed.

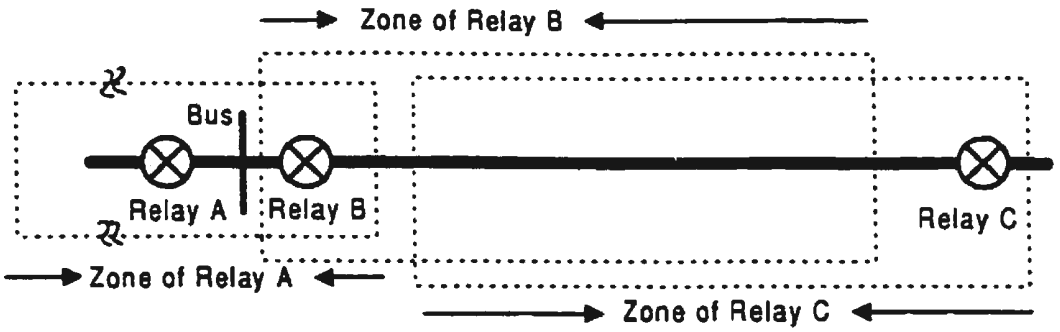


Figure 3.1: Transmission line system

3.2 Distance Protection of Transmission Lines

As explained in the earlier chapters, the objective of this work is to design and implement a digital neural processor for detection applications. As an example application, the distance protection of transmission lines [47, 48] is chosen. Detailed explanation on the problem is described in the following section. Distance protection of transmission lines is to protect the power system from transmission line faults by isolating (tripping) the line(s) under fault. The line diagram of a transmission line system is shown in Figure 3.1.

The faults in a transmission line are categorized as

- Line to Line faults
- Line to Ground faults.

Under each category there are single line, two line and three line faults. For each fault condition the fault signal is different and the protection system should be able to isolate the fault under all conditions. Apart from these, there could be conditions

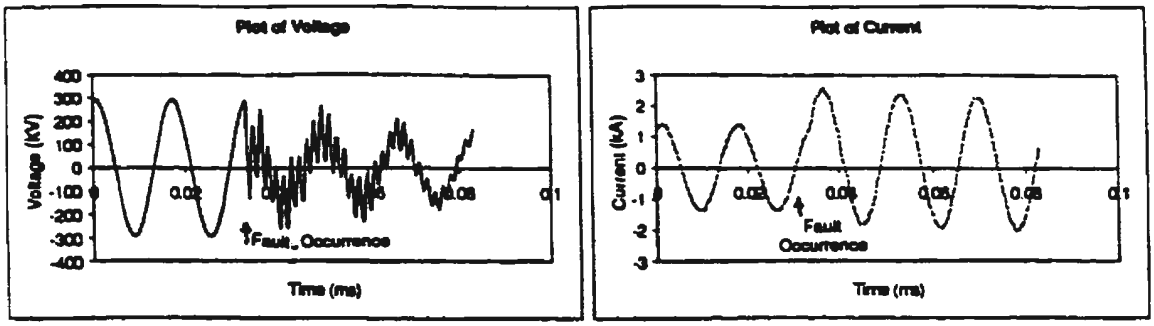


Figure 3.2: Voltage and Currents at fault condition

in which the faults are momentary, for which the system should not isolate the system even though it identifies the fault in the system. The protection system should be able to differentiate between momentary and sustained faults. The protection system should also be capable of isolating only the part of the system that is faulty. This allows other parts of the transmission system to operate without any interruption. The *zones* of operation, for a protection system of a transmission line is illustrated in the Figure 3.1. The conventional method is to use relays like impedance relays, over-current relays or over-voltage relays.

The relays operate based on the behavior of the system under fault. A typical behavior of the voltages and the currents in a transmission system under fault is shown in Figure 3.2. The voltages decrease and the currents increase, resulting in the *fault impedance* to decrease. Over-current relays identify the increase the current and the impedance relays identify the change in the fault impedance. The fault impedance for different fault conditions are significantly different. The relays are set to identify the

fault impedance that signifies a fault in the system, thereby tripping the line under fault. The momentary faults in the system are taken care by incorporating a delay in the operation of the relays, which would avoid the tripping in case of momentary faults. The disadvantage of using conventional relays is that they operate on fixed settings and have to be reset for changes in the network configuration. Changes in network condition can also affect the operation of the relays. This affects the performance of the relays to a large extent. ANNs, as explained in the previous chapters, have evolved to be an excellent tool for adapting to the changing network conditions and configurations, and provide excellent performance.

Coury *et al.* [18] has described an ANN solution for the protection system described above. A brief explanation on this work was presented in Chapter 1. The authors have presented a two layered MLP architecture with magnitudes of currents and voltages as inputs and a trip / no trip signal as the output. They have used backpropagation algorithm for training the ANN and have used 2000 sets of training data for different fault conditions. They claim the ANN improves the protection system efficiency very much. They have mentioned a training time of 2 CPU hours. The solution, though attractive in terms of improvement in efficiency, has a long training time. Moreover, the implementation has been done in software which makes the protection system less reliable. A hardware realization with proper modification in the learning methodology and the proper analysis and preprocessing of training data

would improve the learning rate, performance efficiency and the reliability by many fold.

3.3 Problem Description and Method of Solution

As described in the previous section, ANN is a better tool for the distance protection application. The objective of the thesis is to identify an ANN structure which is optimized for this application and implement it in hardware. The details available about the fault conditions are the simulation data obtained from power system fault simulation [47]. The data available for analysis are the instantaneous magnitudes of voltages and currents of the three phases for different fault conditions. A neural network simulator has been designed using C++ language to identify the ANN structure required for the training using the data available. A preliminary simulation analysis of the data shows that the data requires preprocessing instead of direct feeding to the ANN. The approach to the design of the ANN hardware for this application consists of four distinct phases. They are

I. Data Analysis and Feature Extraction

II. Software Design and Simulation

III. Quantization and Performance Analysis

IV. Hardware Design and Implementation.

The first three phases of the work are explained in the following sections in detail. These involve detailed analysis of the data to identify the inputs to the ANN and to

identify the structure and size of the ANN required for this application. The main focus of the analysis is to arrive at a set of preprocessing methods that would make the training data friendlier to the ANN, reducing the learning time and the size of the ANN, and identifying the optimum learning method for the application. This also involves quantization analysis, analyzing the optimum number of bits required to represent and store the parameters of the ANN such as the learning rate, momentum, inputs, weights, outputs etc. A detailed discussion on the software design of the C++ simulator and the results of the simulation are presented in the second and the third parts. The fourth phase, hardware design and implementation, is explained in the next Chapter. This includes the design of the hardware neural network, the main objective of the work and the VLSI implementation. The part also discusses the functional verification and testing of the hardware ANN in detail.

3.4 Data Analysis and Feature Extraction

As mentioned in the previous section, the data available for analysis are the instantaneous magnitudes of phase voltages and currents for different conditions. Each set of data consists of two cycles of pre-fault condition and three cycles of post-fault condition. The data set is obtained from simulation of a single line to ground fault on a transmission line. The data is sampled at 66 samples per cycle, i.e. 330 sample data points for one condition of fault. The fault simulation has been conducted for different fault impedences and different fault inception angles. The simulation also

includes faults at different locations of transmission line as seen by a relay at one end of the transmission line. The locations include 40%, 60%, 80%, 85%, 87%, 89%, 90%, 91%, 93% and 95% of distance from one end of the line. For each of these locations three sets of values (voltages and currents) for different fault impedance and fault inception were obtained. Of these fault locations, values within 80% are considered to be within the fault zone of the relay and values beyond 80% are considered to be outside the fault zone. These values were divided into two sets, one for training and one for testing. This amounts to 2500 sets of data for training and 600 sets of data for testing. The analysis is focused on single line to ground faults, under the assumption that the preprocessing required for all kinds of faults would be similar, based on the preliminary analysis of data. The preliminary analysis shows that the general behavior of voltage signal under single and three line fault are similar, though intrinsic details are different. This holds the preprocessor assumption good for the analysis. A plot of the data for a single line to ground fault with zero fault impedance is given in Figure 3.3. The figure illustrates that the voltages decrease and the currents increase after the fault. It can also be seen that the voltage signals have more harmonics than the current signals. From the figure it can be seen that the voltage varies significantly more than the current, which has a smooth variation. A simulation of the neural network justified the requirement of a preprocessor for the data. The results of the simulation are discussed in the next section. This section addresses

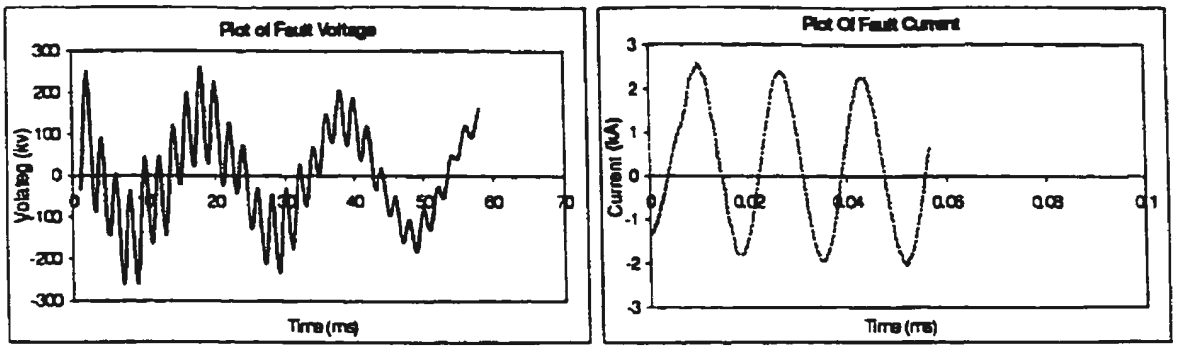


Figure 3.3: Fault voltage and current plot

the process followed to arrive at the preprocessing methods used on the data.

A closer analysis of the data, shows that two separate preprocessors are required, one for significantly separating the fault from the normal signal and other for separating the fault within the relay zone from the fault outside the relay zone. As it can be seen from the fault data plot, there are points of data which have similar magnitude but require conflicting outputs, as illustrated in the Figure 3.2, which further strengthens the necessity for a preprocessor that would eliminate the conflicts thus making the input friendly to the ANN for learning. As hardware implementation is the main focus of the thesis, hardware complexity of the preprocessing methods are given importance. Standard transforms and filters like the Fast Fourier Transform (FFT), due to their high hardware complexity are avoided, though they might solve the problem. The approach is to arrive at a preprocessor which uses the minimum hardware and provides an output which could be learned by the ANN with the least difficulty. This rules out use of many multiplications and divisions as they involve

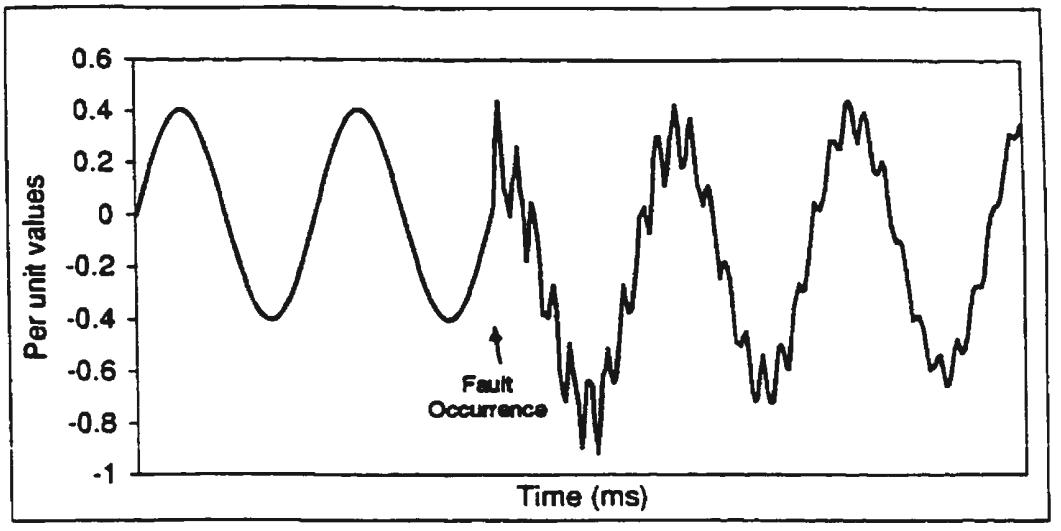


Figure 3.4: Plot of V-I Difference

high hardware complexity. Apart from the hardware complexity, the number of data points to be used before the fault could be identified should be minimum, for example, less than half a cycle (33 sample points). This also reduces the possibility of using FFTs for preprocessing as they require at least a cycle of information for a proper analysis. Following subsections discuss the methods of analysis for the preprocessors mentioned earlier in the section.

3.4.1 Fault Identification

A closer look at the Figures 3.3 and 3.2 indicates that the difference between current and voltage remains constant before the fault and increases significantly after the fault. A plot of the V-I difference is shown in Figure 3.4. The plot illustrates that the V-I difference increases significantly after the fault with many oscillations in the

signal. The signal is similar to the voltage signal but the oscillation is magnified due to the difference signal which adds to the significance in the variation. This justifies using the V-I difference instead of voltage signal alone. But just the V-I difference does not significantly differentiate the fault signal from the normal, it just magnifies the variation. The V-I signal before the fault occurrence is a proper sinusoid which means the rate of change of magnitude varies steadily. Post-fault V-I difference signal exhibits strong oscillations with the oscillations degrading towards zero. An averaged difference on the V-I difference would result in a waveform that would differentiate the part with oscillations, post-fault signal, from the normal signal. The function that was used for this is shown in equation 3.1, where Y_i is the i^{th} value of the resultant signal and X_i is the i^{th} value of the input signal, the V-I difference.

$$Y_i = (X_{i+1} - X_i) + (X_{i+2} - X_{i+3}) \tag{3.1}$$

A plot of the transformed result is shown in Figure 3.5. The figure illustrates that in the transformed signal, the post-fault region is clearly different from the normal region. The ANN would be able to learn this differentiation very quickly when compared to the original raw signal. The simulation results are discussed later in the chapter. The transformed signal still has a region of conflict as illustrated in Figure 3.6. This could be solved by accumulating points together which would eliminate the spurious points. From the equation 3.1, the accumulated version of the function,

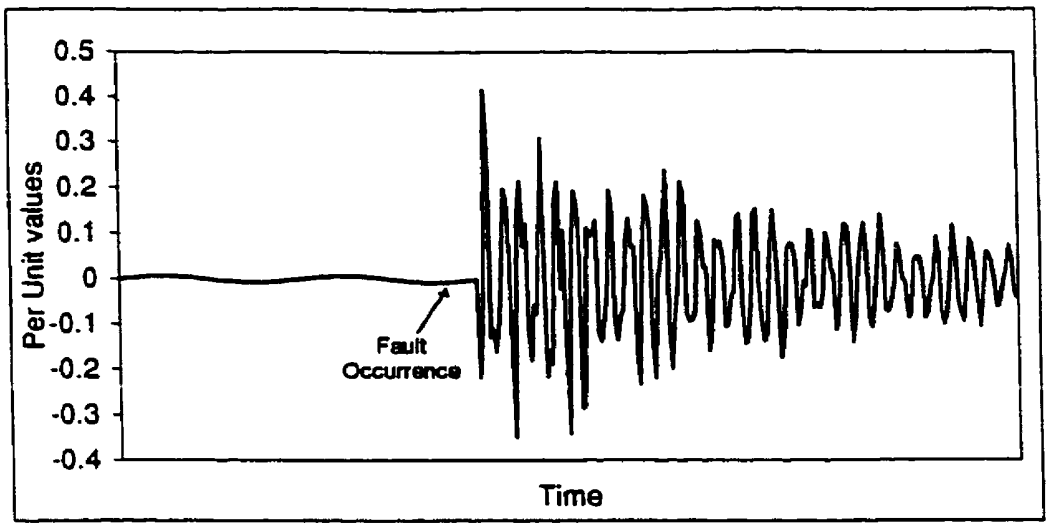


Figure 3.5: Plot of the transformed V-I difference signal

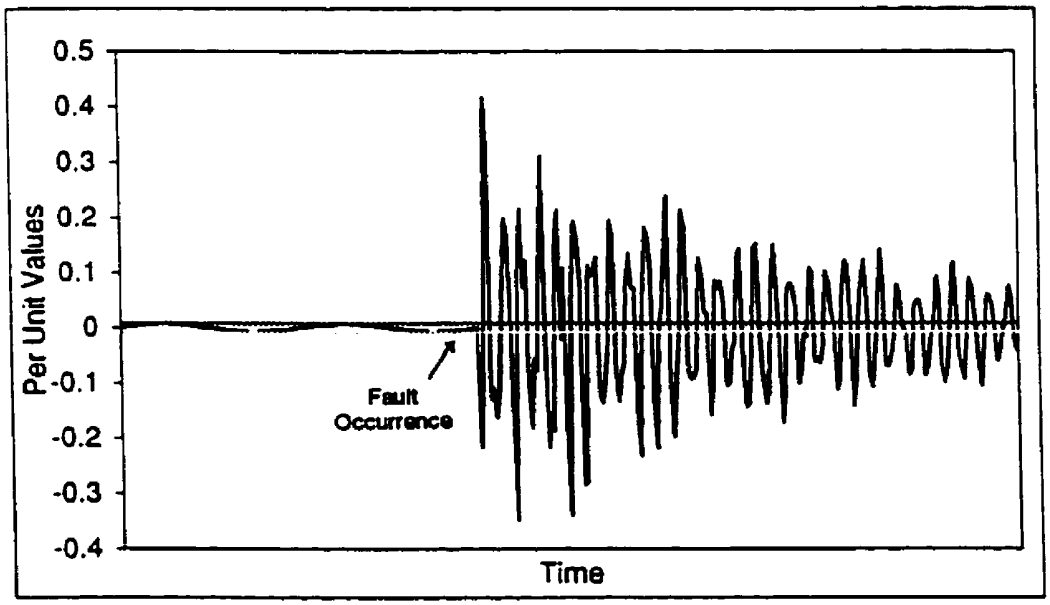


Figure 3.6: Conflict region in the transformed signal

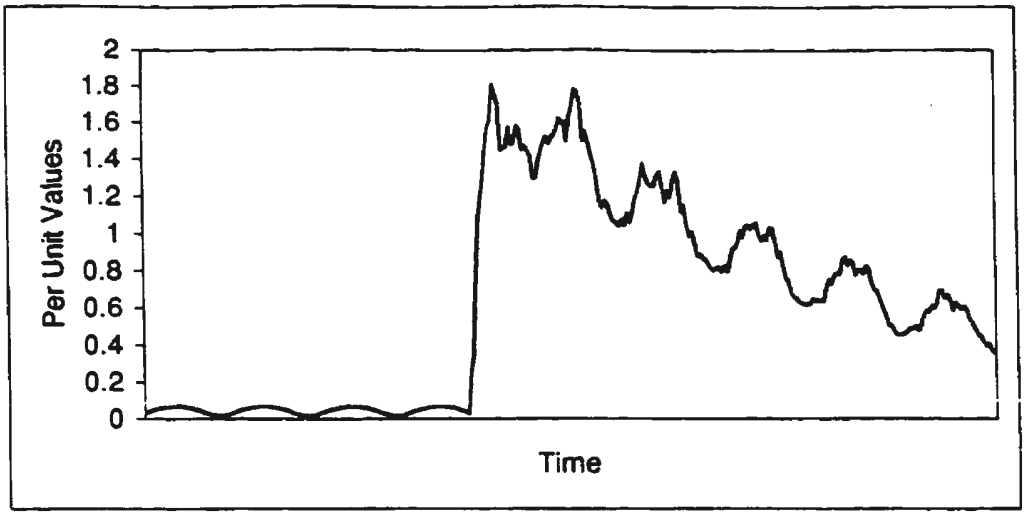


Figure 3.7: SADI Filtered Signal

named as SADI (Sum of Averaged DIfferences), is given in equation 3.2, where F_i is the i^{th} value of the SADI filtered signal.

$$F_i = \sum_{j=i-5}^{i+5} abs\,(Y_j) \tag{3.2}$$

The SADI filtered signal is illustrated in Figure 3.7. The plot shows that the fault signal is clearly differentiated from the normal signal. The neural network would be able to learn this very quickly. The following subsection discusses the preprocessor for the separation of the fault within the relay zone and the fault outside the relay zone.

3.4.2 Fault Zone Identification

This part of analysis involves detailed statistical analysis of the fault data. The difference in the fault zone identification is that the voltage and current signals for different fault locations are very similar. Analysis of the data indicates that the oscillations with respect to different fault locations are distinct to some extent. This suggests domination of different harmonics in the signals corresponding to a fault location. Using a FFT [49, 50] and analyzing the frequency components [51] would solve the problem but that would increase hardware complexity very much. This would also be a slow process, as data has to be collected for atleast one full cycle. An approach similar to the SADI approach is required to solve this. Detailed statistical analysis on half a cycle of post fault and half a cycle of pre fault data shows that *absolute differences*, difference of absolute values of successive signal variations (as illustrated in equation 3.3), differentiates the signals corresponding to different harmonics. To reduce the hardware complexity further, the sign of the absolute differences signal is alone considered. The resultant signal shows clear differences among different frequency components and exhibits different duty cycles for different fault locations. This binary signal can be easily transformed into a signal differentiating faults within the relay zone and the faults outside the relay zone, as they have distinct difference in the mix of harmonics. This preprocessor is named SIGADI (SIGn of Absolute DIfferences). The absolute differences and the SIGADI function are given in equations

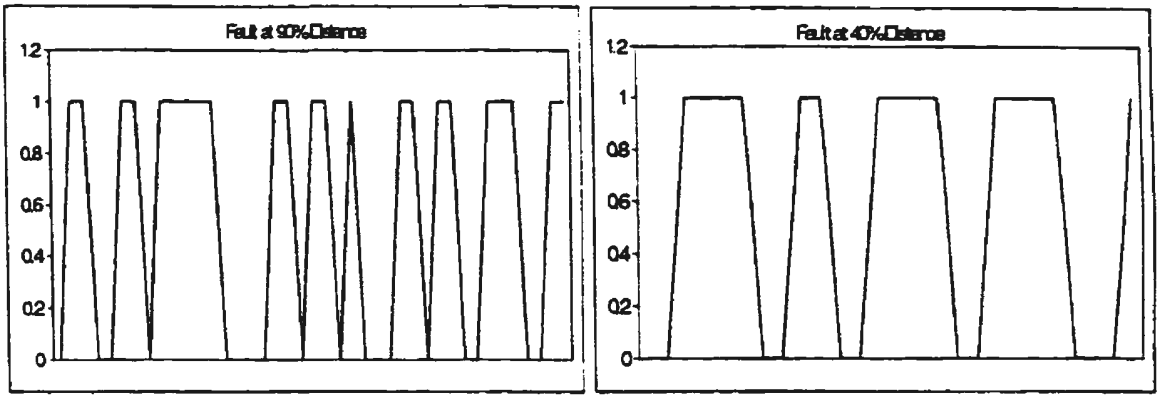


Figure 3.8: Plot of the SIGADI function results

3.3 and 3.4.

$$Y_i = abs\,(X_{i+3} - X_{i+2}) - abs\,(X_{i+1} - X_i) \tag{3.3}$$

$$F_i = \begin{cases} 0 & Y_i < 0 \\ 1 & Y_i \geq 0 \end{cases} \tag{3.4}$$

The results of the SIGADI function are illustrated in Figure 3.8.

To verify the operation of SIGADI function, a pure sinusoidal signal was mixed with known harmonics and applied with SIGADI. The results encourage the use of this approach. The plots of that analysis are shown in Figure 3.9. Modification of the resultant signal to a signal differentiating faults of different zones is achieved by simple binary polynomial transforms [52]. With these two preprocessed signals the inputs to the ANN are the three SADI filtered signals corresponding to each phase

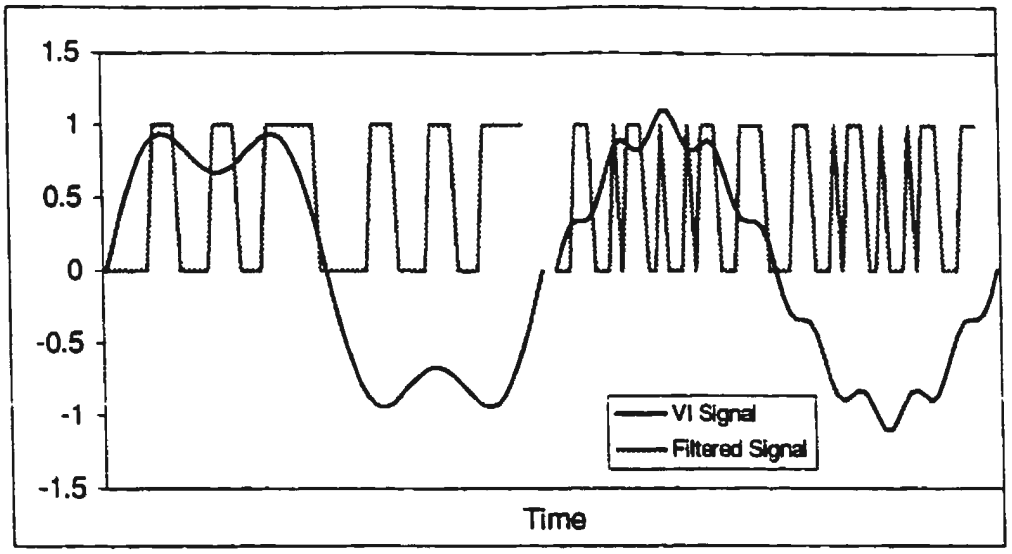


Figure 3.9: Verification of SIGADI

and the SIGADI signal for determining the fault region. The preprocessing makes the input to be ANN-friendly and hence improves the learning time dramatically. The results of the simulation and the software design are discussed in the following section.

3.5 Software Design and Simulation of the ANN

The results of the data analysis explained in the previous section were further studied for the performance with the ANN to identify the learning rate and the structure of the ANN. The objective of this analysis is to identify the optimum structure and size of the ANN corresponding to a set of filtered data and to identify methods of improving the preprocessing to minimize the size of the ANN, hence reducing the

hardware complexity of the final implementation. In the following subsections, the software design of the ANN and the results of the simulation at each stage of data analysis are discussed in detail.

3.5.1 Software Design

The analyzed data has to be used to determine the ANN size and structure. The hardware complexity analysis has also to be done. Though the available commercial versions of ANN simulators, like MATLAB and Brainmaker, allow different structures and sizes of ANNs, they have many restrictions over the number of layers and the training procedures. Moreover, they do not allow simulation using fixed point arithmetic for different bits. This makes it necessary to develop a simulator that would be flexible and can be used for floating point as well as fixed point analysis. The ANN Simulator was developed using C++ programming language [53, 54], in an object oriented manner. The simulator consists of two modes of simulations, one using the floating point arithmetic and the other using the fixed point arithmetic. The floating point simulation is used for identifying the optimum ANN structure for the application and the fixed point simulation is used for quantization analysis, which is explained in the next section. The current simulator design consists of four classes, input neuron, hidden neuron, output neuron and the multiple precision. Multiple precision class is used only in the case of fixed point analysis. The class hierarchy is shown in Figure 3.10. The current design is not fully object oriented, as the main fo-

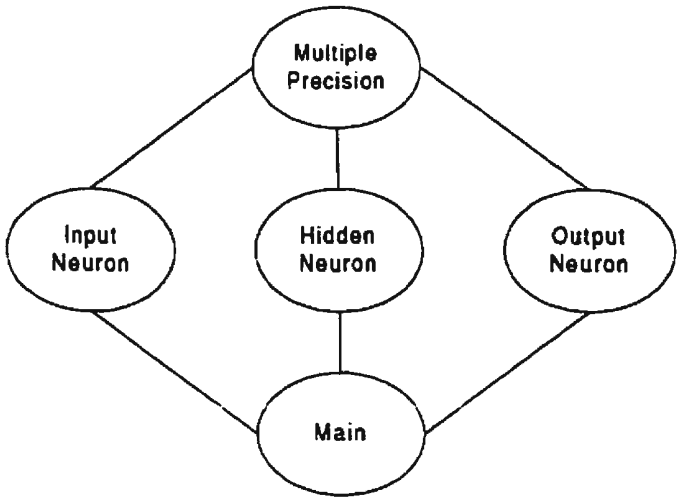


Figure 3.10: Class Hierarchy of the ANN Simulator

cus was to determine the hardware requirements and the performance metrics, which the simulator satisfies. Improvements to the current design are discussed in Chapter 5.

All the neuron classes are modularized, same as the hardware modules present in the respective neurons. The input neuron class receives input from the external sources, in this case a input file. The hidden and output neurons receive inputs from the input and hidden neurons respectively. All the three classes of neurons have similar structure except some functional differences like the backpropagation and computing of backpass sums. The ANN is integrated in the main module which uses user information to determine the network structure and the network parameters and the learning measures. The main module also acts as an interface between different layers of neurons and for file handling and error handling. The simulator uses the

backpropagation algorithm for the learning and learning is done in cycles of train and test, i.e. after each training pass a set of data is tested and the percentage of test set correctness is calculated. The test set correctness is used as the measure of learning.

3.5.2 Simulation Results

As explained in the previous section, this section addresses the simulation results at different stages of data analysis, but only for the floating point simulations. The fixed point simulation results are discussed in the next section. A preliminary simulation of the available raw data had a poor performance. This is due to the fact that similar data points required conflicting outputs. This gives oscillations in the sum of square of errors as the number of passes increases. The simulation showed that the ANN never settles and takes more than 3000 passes of the input set of data, which is expected based on the data analysis. The simulation at each stage also involves identifying the internal parameters for the data set. It should be identified by simulation with a different set of parameters. The parameters include the learning rate, momentum, delta weight, initial weights etc. The learning rate could be different at every neuron and it could be varied for each pass. In this application the learning rate was decided to be constant owing to the fact that incorporating variation of learning rate for each pass would increase the hardware complexity. Moreover, with proper preprocessing, as explained in the earlier sections, it would require very few iterations for the ANN to learn. The ANN parameters would not vary very much in these few iterations. So,

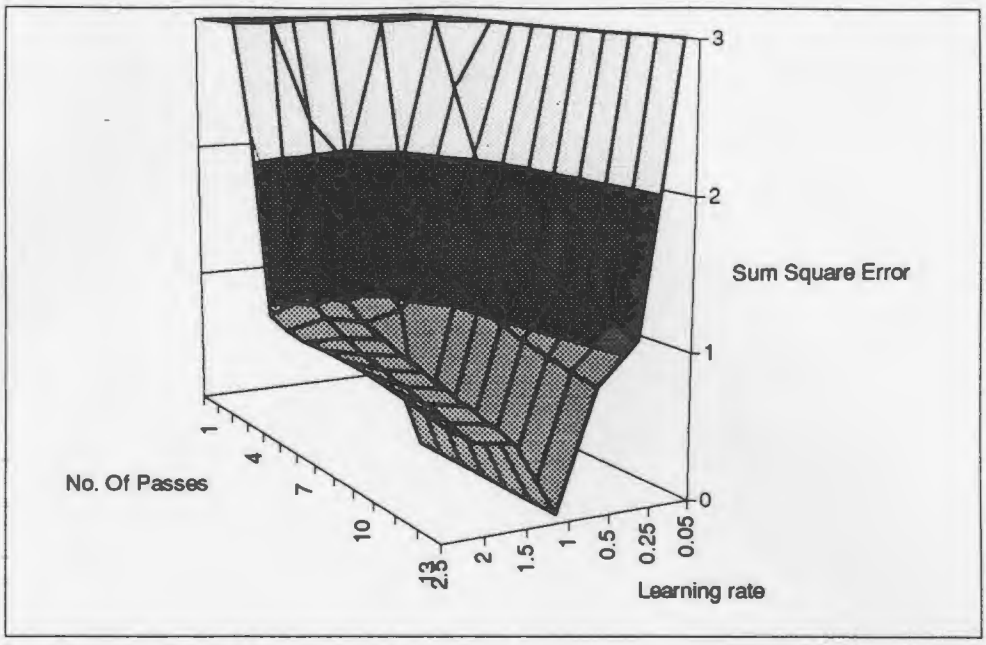


Figure 3.11: Plot of ANN performance with LR variation

it would not be required to implement the hardware for these parameter variations. A plot of the performance of the ANN with different learning rates is given in Figure 3.11. It can be seen from the plot that the variation in performance for learning rates between 0.75 and 1.5 is very low. Learning rate of 1.0 was taken to be optimum for convenience in representation as well ease of arithmetics. The ANN was trained and tested using data sets from every level of preprocessing (including the data sets preprocessed using the intermediate equations like, just the V-I differences) to analyze the performance. The final data analysis yielded an ANN-friendly data set. The data set prior to the final set (prior to summing) was also learnt by the ANN within 20 passes, which is a great improvement in performance over the initial simulation results.

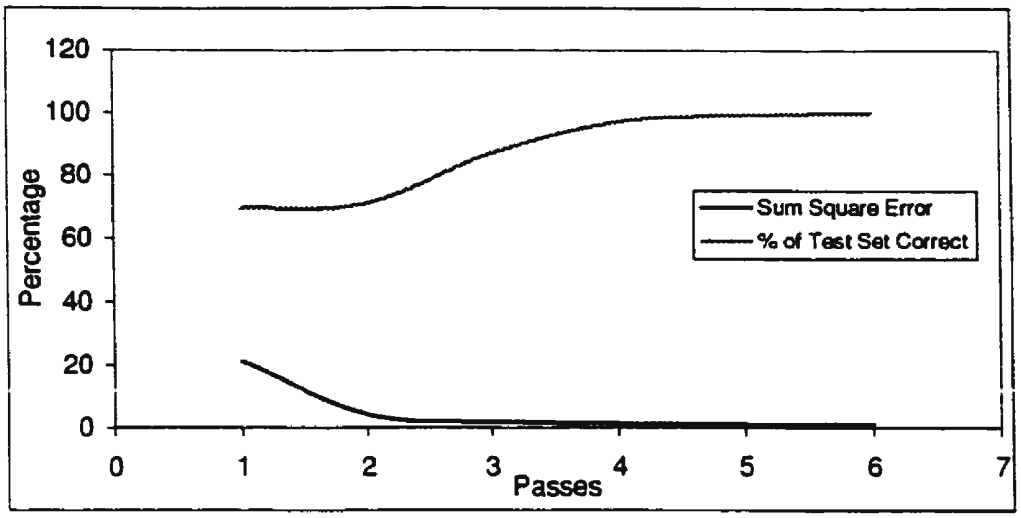


Figure 3.12: Simulation results with final data set

But those data points also had spurious points that required conflicting output values for similar inputs which resulted in oscillations in the ANN learning. The final data set preprocessed using the complete SADI filter eliminated the conflicting points, and the ANN was able to learn within 6 passes of the set of inputs, with a percentage of test set correctness at 99.8% (of the 600 data sets for test). The plot of the simulation results are shown in Figure 3.12. It can be noted that the convergence is fast and smooth without any oscillations. The ANN structure was decided to be a 4-6-1 multilayered perceptron after simulations with the preprocessed and the raw data on a trial and error basis. The ANN structure is shown in Figure 3.13.

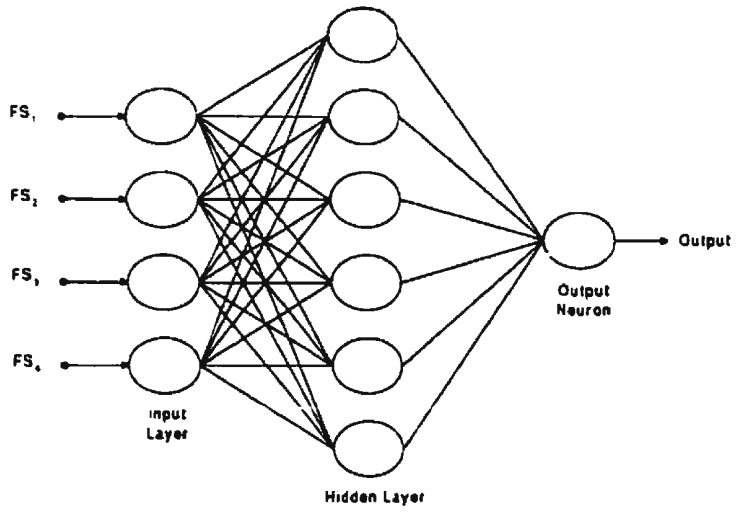


Figure 3.13: Structure of ANN used

3.6 Quantization and Performance Analysis

This section discusses the simulation results of the quantization analysis, which is fixed point simulation analysis to determine the optimum bits required to represent the inputs, outputs and the parameters of the ANN. The multiple precision class mentioned in the software design is used to achieve this. The inputs, weights, outputs and the parameters were initially represented in 32 bits, containing 16 bits of integer and 16 bits of fraction. The ANN was simulated with data of this representation to verify the results of simulation in correspondence with the floating point simulation. The results are shown in Figure 3.14. The number of bits was reduced for all the parameters and the performance was noticed to degrade below 14 bits (4 bits for the integer, 9 bits for the fraction and 1 sign bit). The results are shown in Figure 3.15.

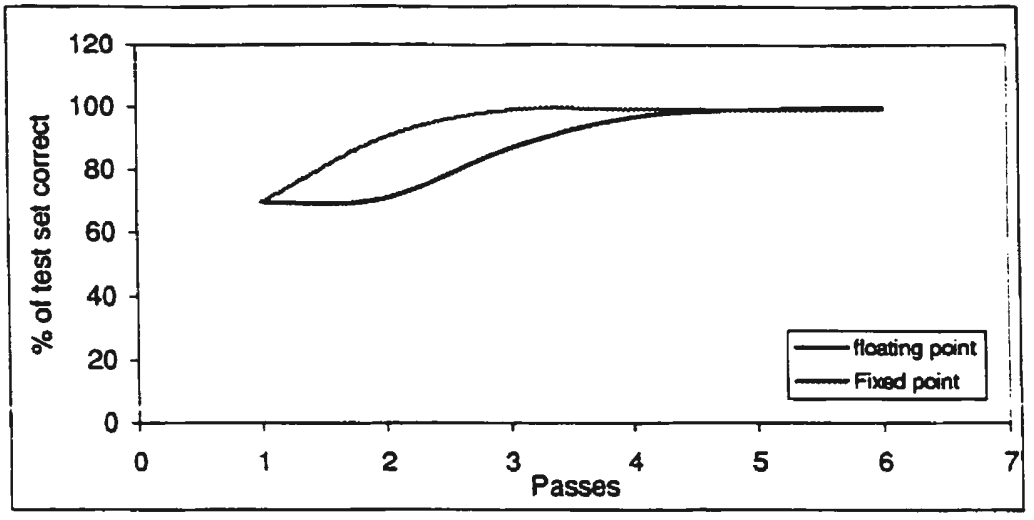


Figure 3.14: Verification of Fixed point with Floating Point Simulations

The data set appears to be representable in fewer number of bits than 14 bits. The reason for the seemingly higher number of bits required for representation is that the backpropagation needs more resolution than other learning algorithms. This is because, the error value gets very small as it propagates from the output to input layer and correspondingly the number of bits required to represent the small changes are higher. It can be noted from the figure that when the number of fraction bits is reduced below 9, the performance degrades considerably. Further reduction can be done in the bits required for inputs as the variation of weights and parameters at the output are coarser and hence can be accommodated in fewer number of bits. The results corresponding to the simulation with variable weight bits is given in Figure 3.16. The simulation showed that the reduction in parameters (momentum etc.) can

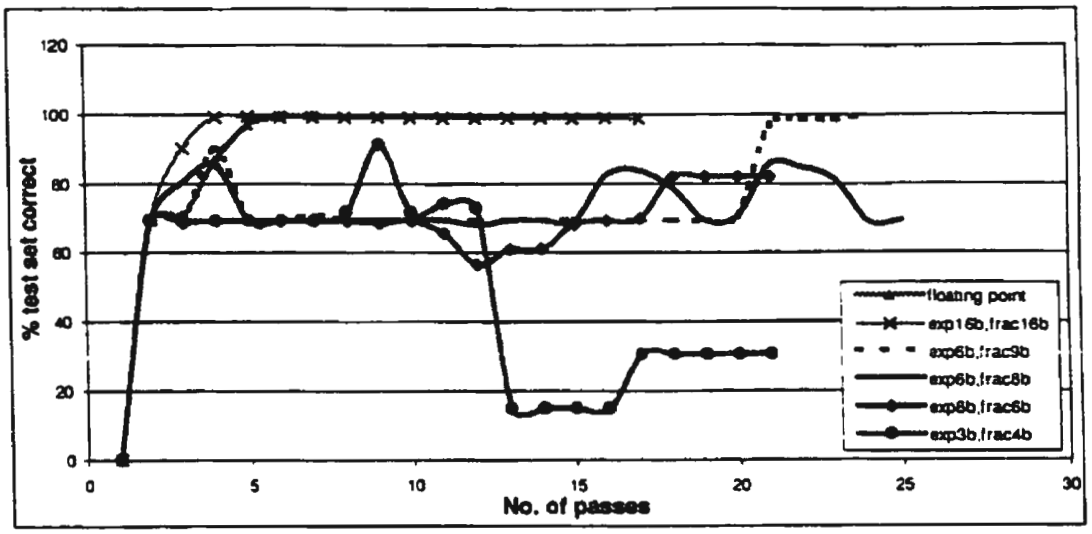


Figure 3.15: Comparison of performance with various bits

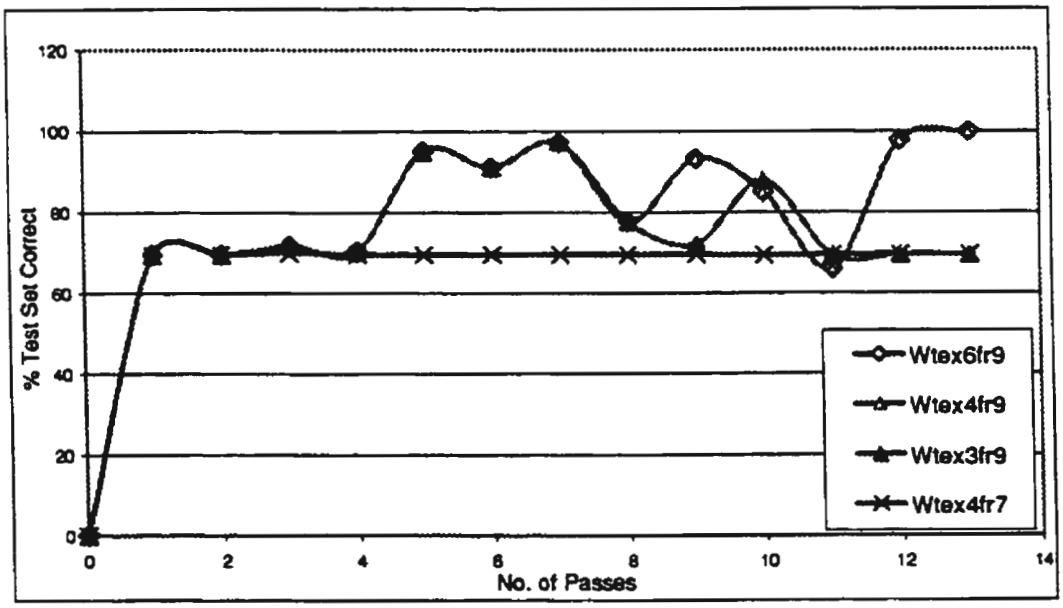


Figure 3.16: Performance with different weight bits

be two bits further making it 14 bits for the weights and 12 bits for other values like inputs, parameters etc. But to maintain generality of the hardware implementation, the number of bits was decided to be 16 bits for all, 9 bits for the fraction, 6 bits for the integer and 1 bit for sign. The hardware design aspects are discussed in the next chapter.

3.7 Summary

This chapter discussed in detail, the problem, the solution and the method of approach. The software design of the simulator was discussed in detail explaining the phases of simulation. The results of the simulation were explained in detail and the relation of the results with the hardware implementation was emphasized. The quantization analysis was discussed and the performance of the ANN for different bits was presented and the results for the same were provided. Detailed explanation on the data analysis and the method of preprocessing were presented. In the following chapter, the hardware design aspects of the ANN are discussed.

Chapter 4

Hardware Design, VLSI Implementation and Testing

4.1 Introduction

In the earlier chapters, the basics of ANN and some reported methods of ANN implementation and known hardware VLSI neural networks were discussed. In the previous chapter, the selected detection application, namely the protection of transmission line system, was discussed in detail. The software design of the neural network simulator developed for the analysis was described and the results of the simulation were presented and discussed. The methods of preprocessing and the results of the analysis were presented. The quantization analysis, the results and verification of the results were also explained. Based on the results obtained in the simulation, the hardware design of DIANNE, the DIgital Artificial Neural NEtwork, will be addressed in this chapter. The overview of the architecture and the details of the design are explained in detail. The chapter discusses the datapath and control units of the design and the issues related to the design. The testing of the design and the features of the

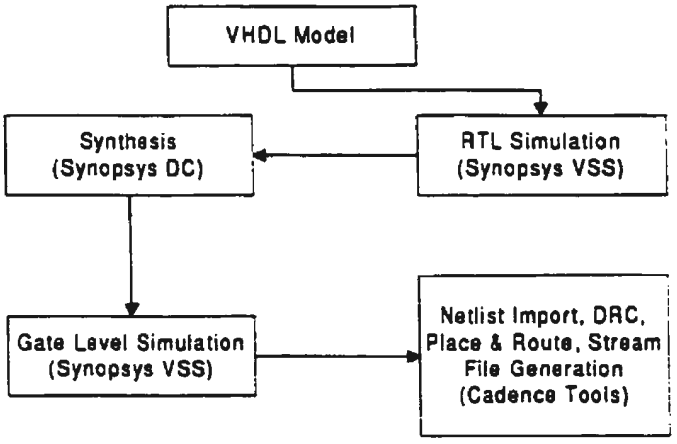


Figure 4.1: Flow chart of Design Flow

design are also described. Some of the implementation constraints and the methods of solution are also explained.

4.2 Design Cycle and Environment

The design flow and the development environment are described in this section. As explained in the previous sections, VHDL was used to simulate and synthesize the components of the neuro processor. The design flow was provided by the Canadian Microelectronics Corporation (CMC) [55]. A flowchart illustrating the design flow is given in Figure 4.1. As the figure shows, the design is coded using VHDL and analyzed for functionality using the Synopsys VHDL System Simulator (VSS). The waveform viewer helps in visualizing the functionality of the circuit designed. The waveforms corresponding to specific components are presented when the components are described in later sections. The next phase of the design flow is synthesis in which

Synopsys design analyzer is used for synthesizing the individual components. This involves optimization and mapping of components to specific cell libraries, CMOSIS5 in this case, and creating netlists. These netlists are tested and verified for functionality again (shown as gate level simulation in the figure). The verified netlists are then imported to the Cadence tools for VLSI design. This involves Verilog XL integrated simulation, placement and routing, Design Rule Checking (DRC) and stream file creation. The steps are illustrated in the figure.

4.3 Overview of the Architecture

As described briefly in Chapter 2, DIANNE is a custom VLSI neural processor with the typical partition per processing element being the neuron. The neural processor is a 16 bit architecture with integer arithmetic owing to the results of the integer arithmetic simulation explained in the previous chapter. The block diagram of DIANNE is shown in Figure 4.2. As the figure shows, there are two distinct parts of the design, the *preprocessors* and the *neural processor*. The preprocessors are the SADI and SIGADI filters explained in the previous chapter. The preprocessors are optimized for the protection application and can be used for applications requiring similar preprocessing. The preprocessors could be bypassed by configuring the initial control settings if the application does not require these preprocessors. The details of the control settings will be described in later sections. The design was carried out using VHDL (Very High Speed Integrated Circuit Hardware Description Language) [56],

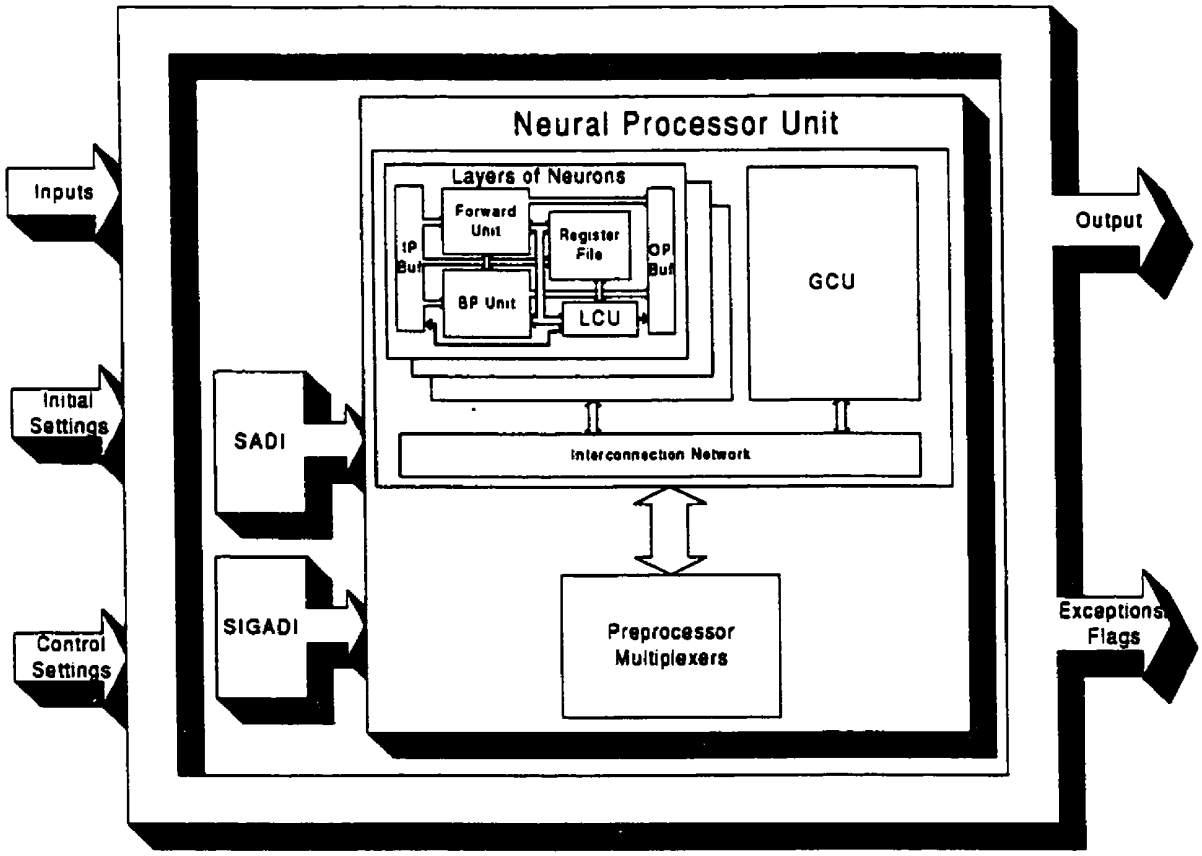


Figure 4.2: Block Diagram of DIANNE-D1.0

and functionally verified and synthesized using the CAD tools Synopsys and Cadence [57, 58]. The architecture was partitioned into neurons as mentioned before: input, hidden and output neurons. The layers of DIANNE operate in a pipelined fashion. The neurons in the layer are a mix of pipelined and multicycle implementation [59]. These implementation methods reduce the hardware complexity and increase the speed of operation. The following section describes the pipeline of DIANNE layers.

4.3.1 Pipelining of Layers in DIANNE

As mentioned in the previous sections, DIANNE can be configured to be operated in two modes, the training mode and the test mode. The training mode uses the backpropagation unit for modifying the weights and the test mode uses only the forward pass unit. The test mode is the real-time operation mode prior to which the training has to be done and the weights stored on the on-chip registers. The number of stages of pipeline differs depending on the mode of operation. Figures 4.3 and 4.4 show the different stages of pipeline in test mode and training mode of operation respectively. The stages FPL1 to 3 are the forward pass stages and the BPL1 to 3 are the backpropagation stages. As it can be seen from the figures the test mode has only the forward pass operation as no backpropagation needs to carried out. In the training mode, it can seen that the backpropagation and the forward pass operations overlap. Moreover, for the error to be calculated for a set of inputs, they have to pass over all the forward pass stages before the first backpropagation could

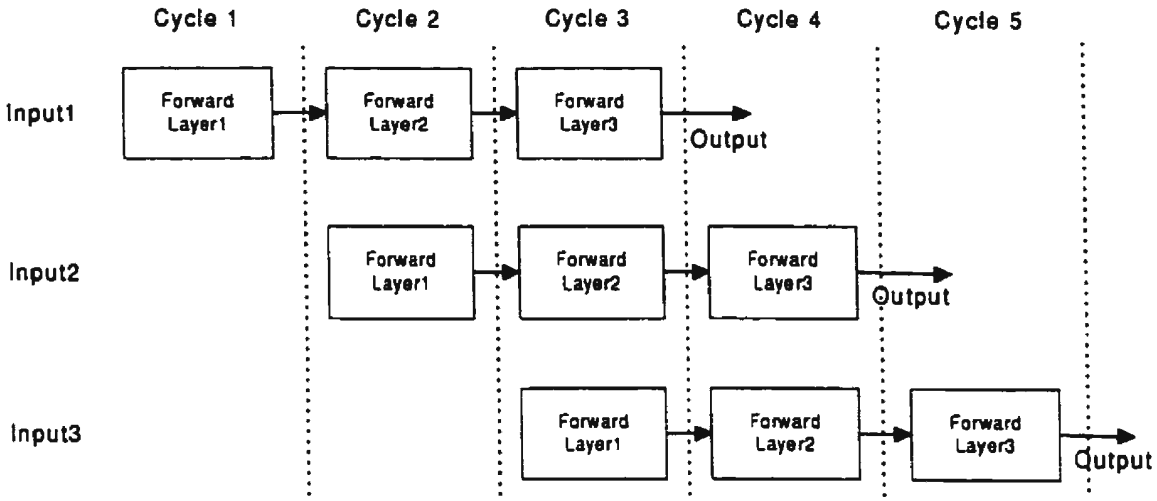


Figure 4.3: Pipeline stages of test mode

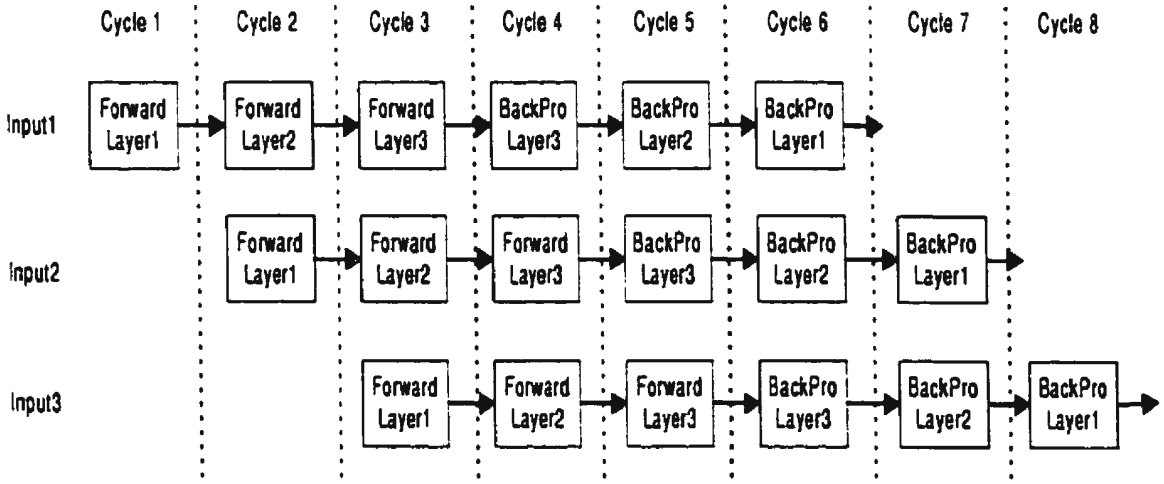


Figure 4.4: Pipeline stages of training mode

begin. This introduces a delay in the adjustment of weights which interferes with the following forward pass operations. But the design has been done neglecting the delay due to the fact that backpropagation is an iterative process and the delay in modification of weights would be adjusted with more iteration steps. This design step has been verified in the software simulation for the correctness and the delay does not affect the convergence rate of the neural network for this application. This might affect the performance in case of applications that require more accurate updates of weights. But this trade-off is better than restricting to the sequential nature of backpropagation algorithm.

The design method within one neuron is a multicycle implementation which can be called an interleaved pipeline. More explanation on the architecture of each of the class of neurons is explained in the following sections. The following subsection describes the implementation of the preprocessors.

4.3.2 Design of the Preprocessors

The previous chapter explained the method by which the preprocessors were arrived at. It was stated that the preprocessor consists of SADI filter and SIGADI filter. The block diagrams for the filters are given in Figure 4.5 and Figure 4.6. The outputs of the preprocessors are fed to the neural network block for training. The preprocessors consist of delay block and adder blocks as shown in the Figures 4.5 and 4.6. The delay blocks allow points of the cycle to be stored and processed to indicate any faults in

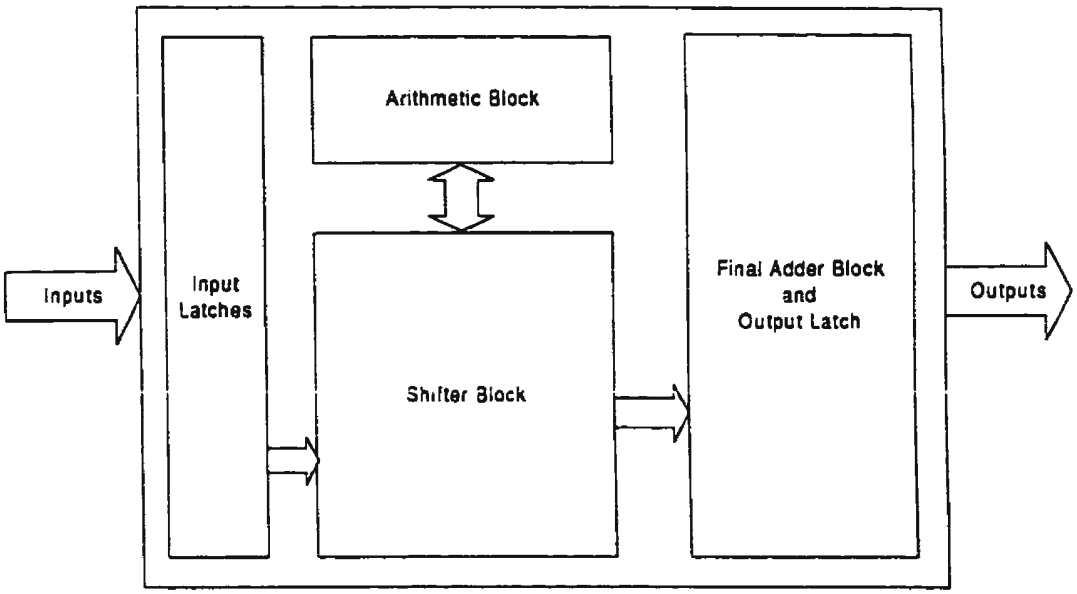


Figure 4.5: Block Diagram of SADI

real-time. This avoids the use of any external fault identifier. The filter functions explained in the previous chapter were implemented using VHDL. The preprocessors were simulated and tested for the functionality using test benches in VHDL. The results of the simulation are discussed in the section 4.6.

4.4 DIANNE-D1.0 - Datapath Design

The datapath of the processor consists of three classes of neurons. They are the input, hidden and output neurons. Each neuron consists of a datapath and a local control unit. The design of the local control unit will be discussed in the section 4.5. The datapath consists of three units called the forward pass unit, the backward pass unit and the register file unit. The functions of the units and the design are explained in

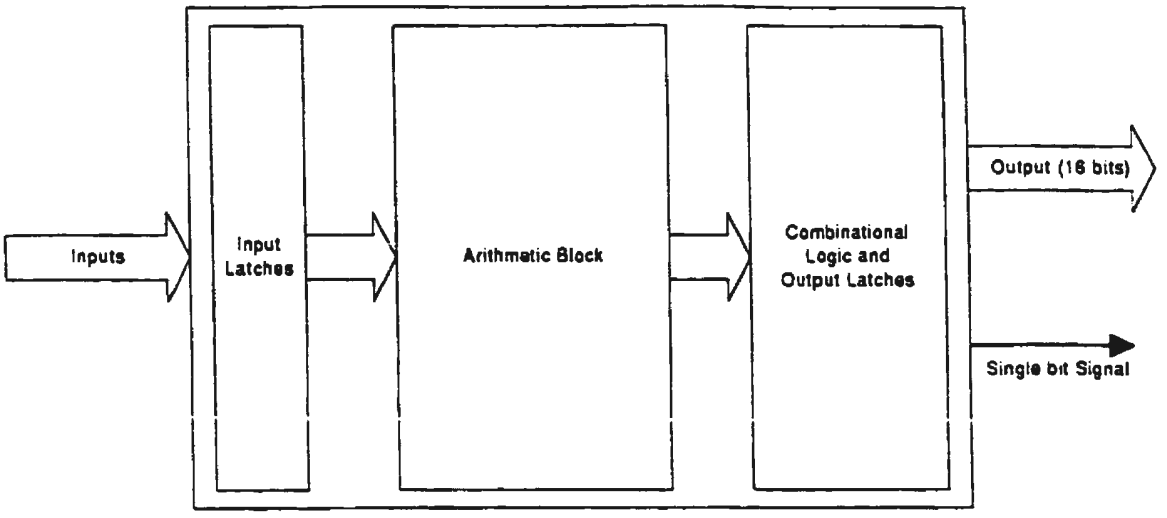


Figure 4.6: Block Diagram of SIGADI

the following subsections. The block diagram of the datapath of a general neuron is shown in Figure 4.7.

4.4.1 Forward Pass Unit

The function of the forward pass unit is the same in all classes of neurons. The forward unit computes the weighted sum of inputs and outputs the sigmoidal function equivalent of the weighted sum. The block diagram of the computational part of forward unit is given in Figure 4.8. As the figure illustrates the forward unit receives input from the external source or the preprocessors based on the initial control settings. This unit has an input buffer to store the inputs until the sum is computed. The computation is done in a pipelined fashion which reduces the hardware complexity of the design. The forward unit also holds the inputs to the unit for the

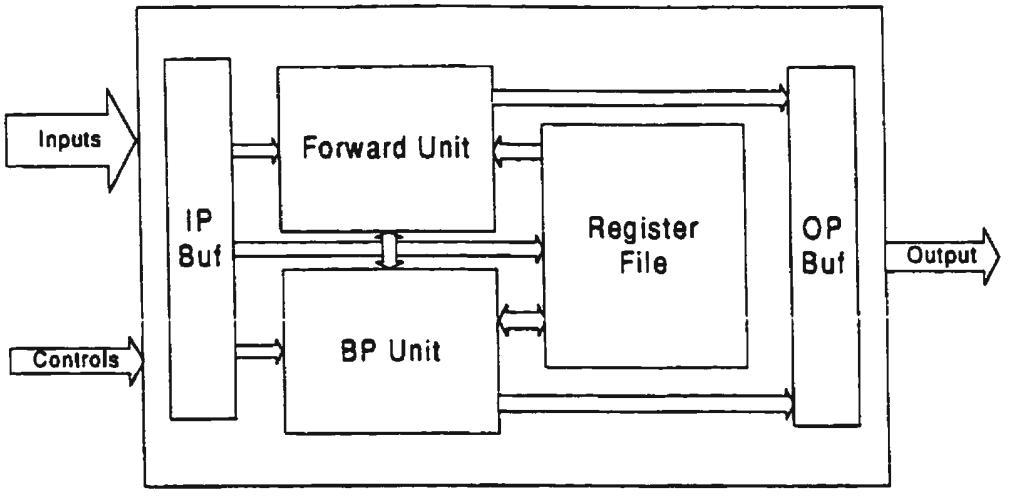


Figure 4.7: Datapath of a general neuron

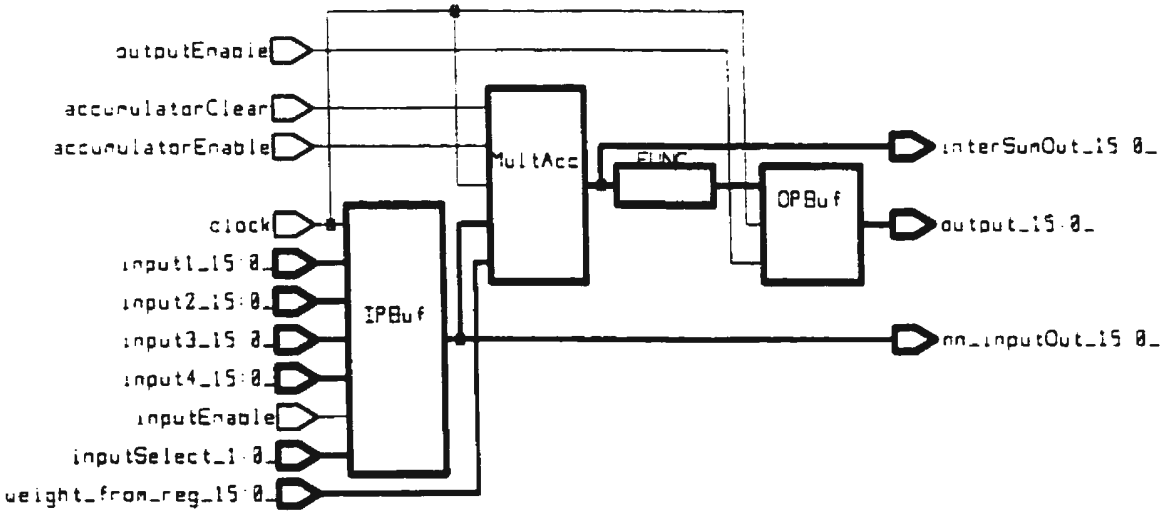


Figure 4.8: Forward Unit- Input neuron

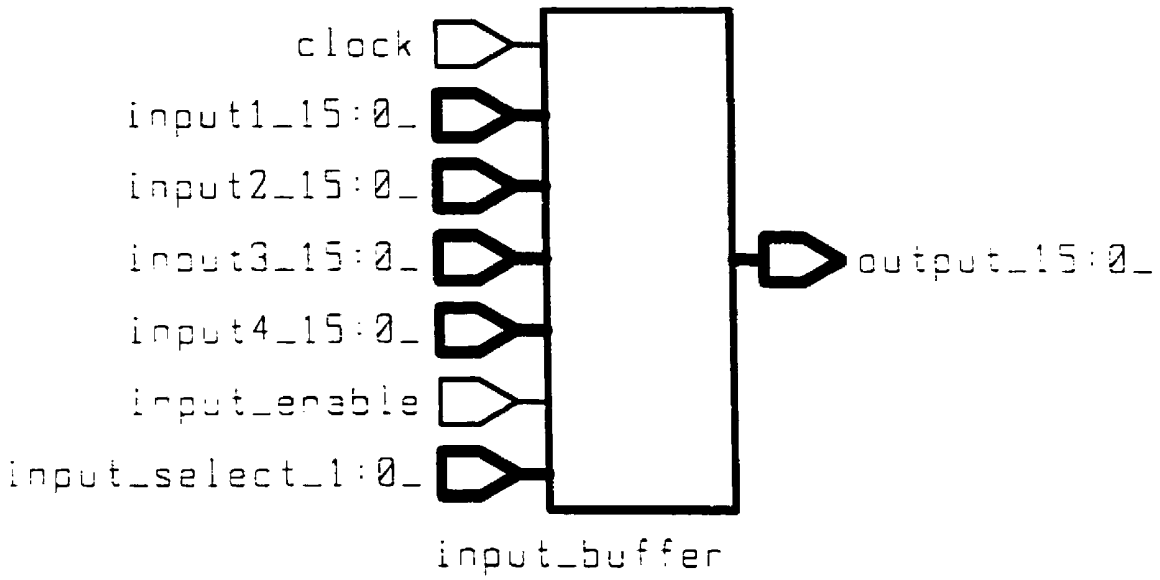


Figure 4.9: Symbolic diagram of Input Buffer

backward pass unit functions. The blocks of the forward pass unit are input buffer, multiply-accumulate, function lookup, output buffer and the hold registers.

4.4.1.1 Input Buffers

The function of the input buffers is to receive the inputs from external or previous layers and hold them for the multiply-accumulate unit to process. These are simple registers with clear and enable inputs. The output of the registers are given to a multiplexer, whose size is determined by the number of inputs. The multiplexer receives a select input from the local control unit. The select input determines the input to be processed and the sequence of selection varies among the neurons in different layers to facilitate the concurrent processing of all neurons. A symbolic diagram of the input buffer is given in Figure 4.9. The numbers of inputs handled by the input buffers at different layers differ. With the current design the input layer

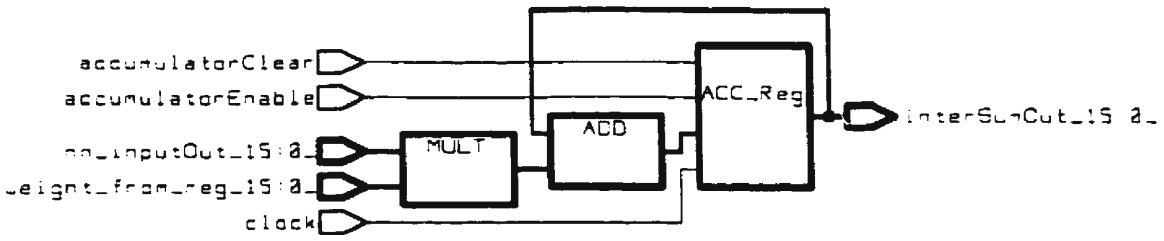


Figure 4.10: Schematic of Multiply Accumulate Unit

has four inputs, the middle layer has four inputs and the output layer has six inputs.

4.4.1.2 Multiply Accumulate Unit

The function of the multiply-accumulate unit is to calculate the weighted sum of inputs. The schematic of the multiply-accumulate unit is given in Figure 4.10. As the figure shows the multiply-accumulate unit has a multiplier and an accumulator with enable. The adder is a 16bit adder synthesized from the CMOSIS5 libraries. The multiplier is also a library synthesized component, which is a $16bit \times 16bit$ integer multiplier, modified to do fixed point multiplication. The timing specifications and other features will be discussed in the section 4.7. The block receives weights from the register file and the inputs from the input buffer. The control unit provides the signals for selecting the proper weights and inputs. This block remains the same in all neurons unlike the input buffer.

4.4.1.3 Activation Function

This block represents the activation function of a neuron designed as a table lookup. The function lookup is for the sigmoidal activation function. The block receives the

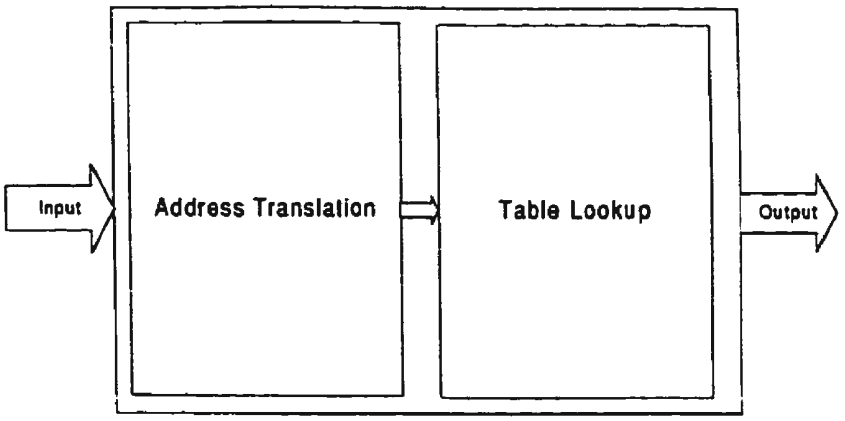


Figure 4.11: Block diagram of the Activation Function Block

16 bit input from the multiply accumulate unit and gives the sigmoidal function equivalent of the input. The function lookup is implemented using the ROM blocks of the CMOSIS5 libraries. The schematic is given in Figure 4.11. Each neuron has one function lookup in their datapath. This reduces the number of interconnections when compared to a central function lookup as in [35].

4.4.1.4 Hold Registers

The HOLD registers are used to hold the input values for the use of backpropagation block. The backpropagation algorithm requires that the error at the output for a set of inputs/weights has to propagated back to the input layer for modification of the weights. This restricts the operation of pipeline as the neuron has to wait for the error to be calculated at the output for a set of inputs and propagated back to the input layer. The hold registers are used to eliminate this restriction by holding

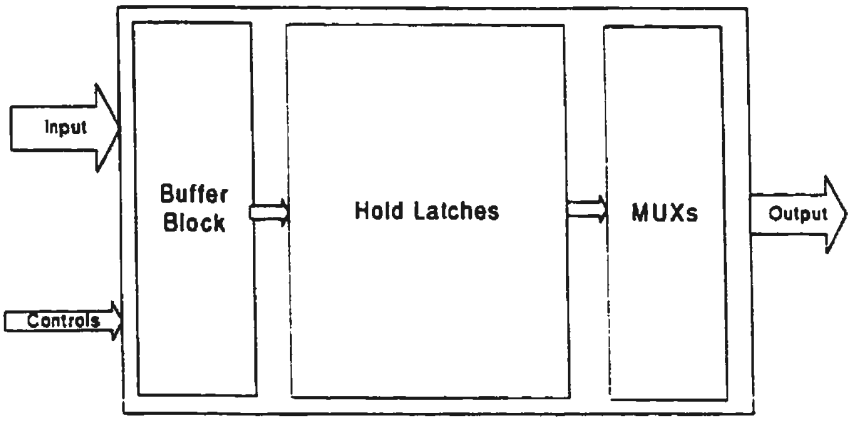


Figure 4.12: Schematic of HOLD registers

the inputs while allowing the pipeline to operate without delay. This increases the hardware complexity by a few registers but increases the speed of operation and hence the performance many folds. Each input neuron has a 5 × 4 register (to hold five sets of four inputs) and each hidden neuron has a 3 × 4 register (to hold three sets of four inputs). The schematic is given in Figure 4.12.

4.4.1.5 Output Buffer

Output buffer stores the output of the function lookup to be passed on to the next layer for processing. It receives the output enable from the local control unit. The schematic diagram is shown in Figure 4.13. The output buffer design is the same in all the neurons as only one output is passed from each neuron.

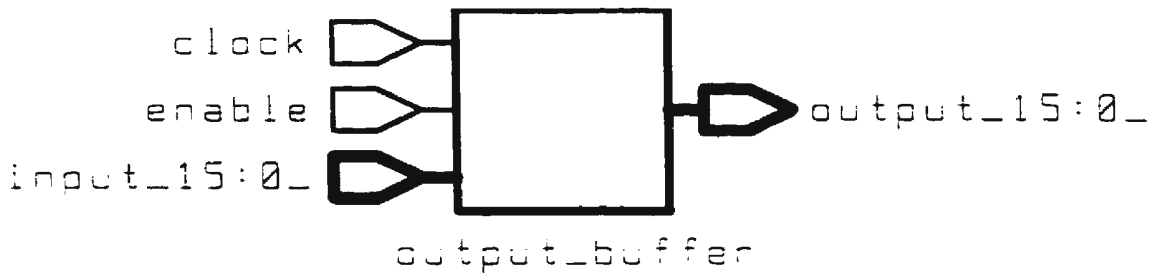


Figure 4.13: Schematic of the Output Buffer

4.4.2 Register File

Register files are used to store the neural network parameters (learning rate, momentum etc.) and the weights required for the computation. The schematic representation of the register file is shown in Figure 4.14. The register file includes a set of register to store the weights, which are readable and writable and can be initialized to a particular value before processing. Each register has read and write enable signals which are issued by the local control unit and the sequence of read and write is different among the neurons depending on their position in the layer to ensure proper computation and concurrent processing. The other register in the register file are the delta weight register, momentum register and learning rate register. These are not modified after the first write, when the processor is initialized. The register file also allows concurrent read and write operations on a weight through dual latches. This allows the backpropagation modification and the forward pass computation to

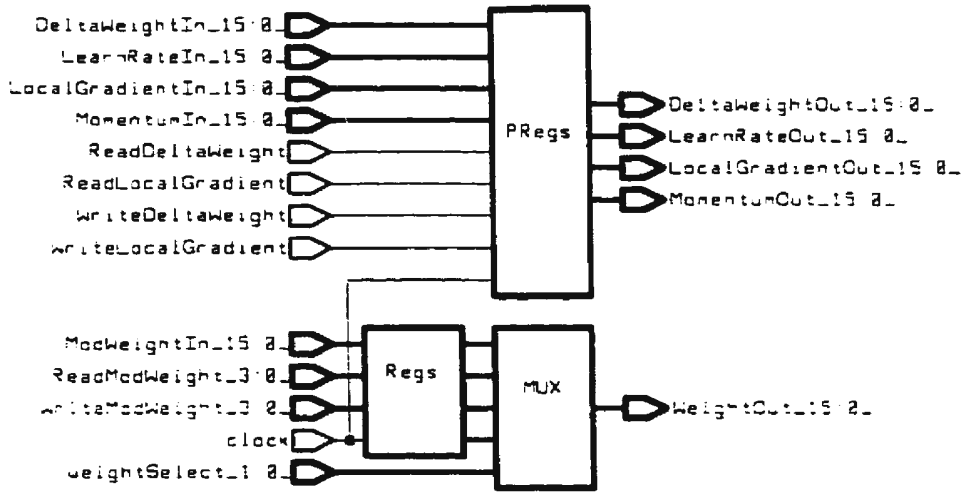


Figure 4.14: Schematic of the Register file

proceed concurrently. The operation verification is explained in the testing section.

4.4.3 Backward Pass Unit

The function of this block of the neural processor is to implement the backpropagation algorithm. This is a parallel pipeline which consists of three functional units: Compute Local Gradient Unit, Weight Adjust Unit and Compute Back Pass sum Unit. This operates in parallel with the forward pass unit. As explained in the previous subsection, the modification of weights and the forward pass computation proceed concurrently. The functional units of the block are described in the following sections.

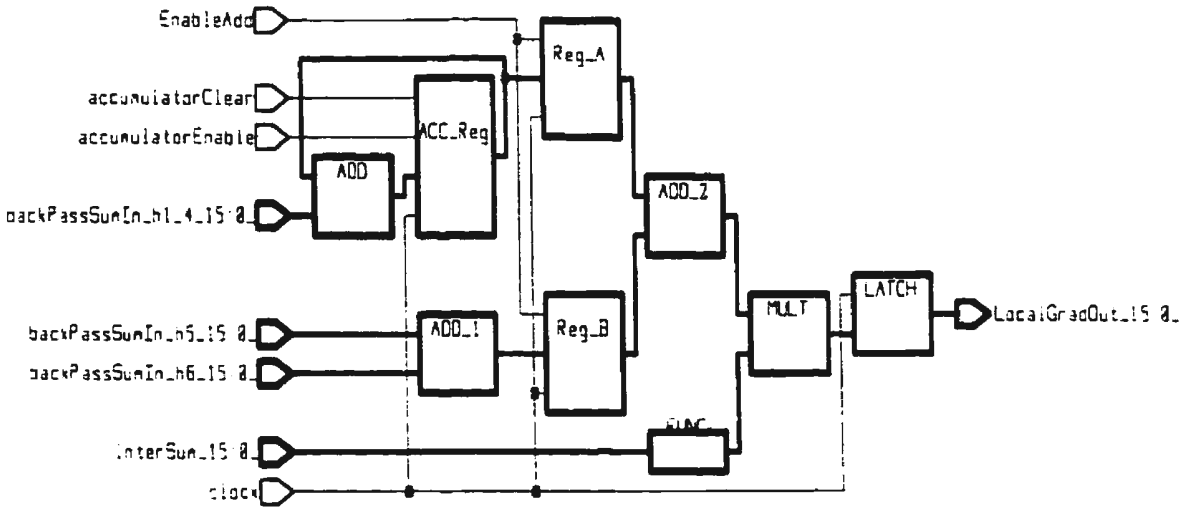


Figure 4.15: Schematic of the Compute Local Gradient Unit

4.4.3.1 Compute Local Gradient Unit

This unit computes the local gradient for each neuron. The design of the unit differs with the number of weights used in the neuron. The schematic diagram of the unit is shown in Figure 4.15. This unit consists of a derivative of activation function lookup, accumulate register, multiplier, adder and latches. The multipliers and adders are the same as the ones in the forward pass unit. The derivative lookup is the same as the function lookup in the forward pass unit but the function here is the derivative of the sigmoidal function. The unit implements the function given in the equation 4.1.

$$LocalGradient = \frac{e^{-x}}{(1+e^{-x})^2} \sum_{i=1}^{n} BPSUM_i \tag{4.1}$$

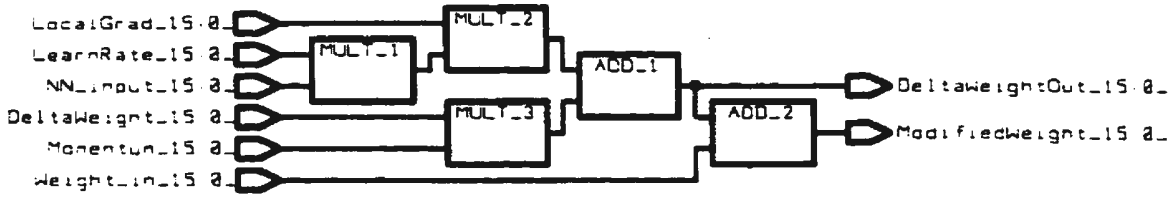


Figure 4.16: Schematic of the Weight Adjust Unit

In this equation, x is the intermediate sum calculated in forward pass unit and the $BPSUM$ is the back pass sum computed for each neuron in the previous layer. n is the number of neurons in the previous layer. The back pass sum computation is explained in the section 4.4.3.3.

4.4.3.2 Weight Adjust Unit

The weight adjust unit modifies the weights in a neuron and writes them to the register file for computation of neuron output in the next pass. The schematic of the weight adjust unit is shown in Figure 4.16. The weight adjust unit reads the weights from the register file and modifies them based on the parameter register values and the error propagated from the output layer. This unit implements the function given in equation 1.12. The design of the unit differs among different layers with respect to the number of weights associated with the neuron.

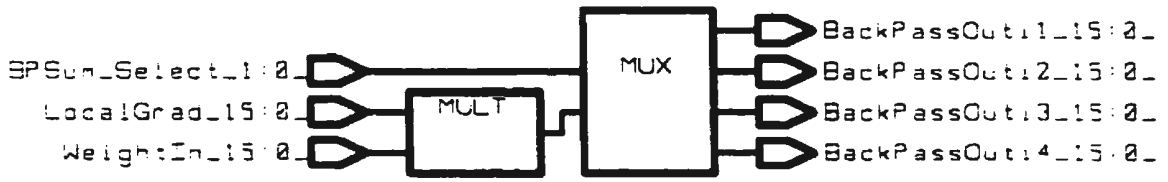


Figure 4.17: Schematic of the Compute Back Pass sum Unit

4.4.3.3 Compute Backpass Sum Unit

This unit computes the back pass sums for passing to the previous layer for computation of delta weight, the value to be added to the weights. The schematic of the compute back pass sum unit is given in Figure 4.17. The sequence of computation differs among neurons in different layers which is controlled by the local control unit. This is done to make sure all the neurons in the previous layers get the back pass sums in the same number of cycles which prevents any neuron from waiting for the back pass sum values. This ensures concurrent processing in all the neurons. The process will be explained in detail in the design of the control unit.

4.5 DIANNE-D1.0 - Control Unit Design

The control unit of DIANNE is split into two parts. One is the global control unit which controls the data flow between layers and the external inputs and outputs. The local control unit controls the flow of data between components in a neuron and

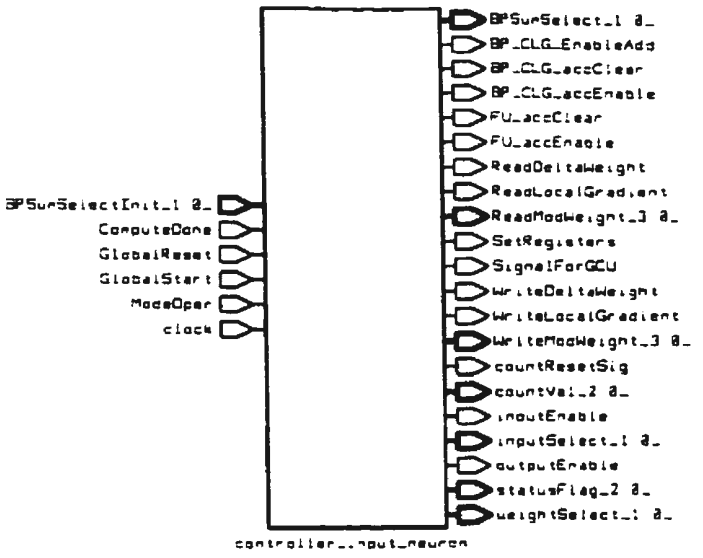


Figure 4.18: Symbolic diagram of the Local control Unit

communicates with the global control unit to ensure synchronized processing. The control units are finite state machines. A Mealy machine was used to implement the design. The following sections describe the design in detail.

4.5.1 The Local Control Unit

The local control unit is specific to a neuron. This unit communicates and gets initialization information from the global control unit. The local control unit also ensures synchronization between neurons through the global control unit. The symbolic diagram of the local control unit is given in Figure 4.18. The state diagram explaining the function of the local control unit is given in Figure 4.19. The states of the local control unit and the corresponding group of signals associated with the state are given in table 4.1. The state diagram shows some signals which are not mentioned here.

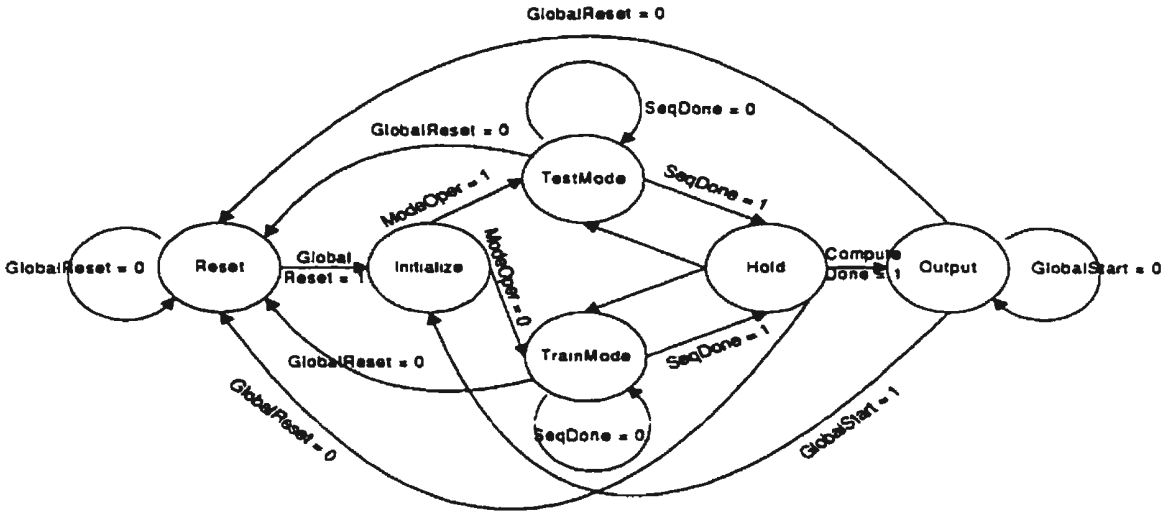


Figure 4.19: State diagram of the local control unit

These are the intermediate signals generated from the internal counters or signals derived from the main inputs to the control unit. As ilustrated by the state diagram and state description, the local control unit controls the forward and backward pass operations of a neuron based on the initial settings received from the global control unit. The following section describes the function of each group of control signals.

4.5.1.1 Description of Control Signals

GlobalStart, GlobalReset and LocalReset are the resetting signals. GlobalStart is used only in the first operation cycle after all the neurons are initialized. These are active low signals.

BPSumInit, LearnRateSet and MomentumSet are the initializing signals. BP-SumInit is a two bit signal and the other two are 16 bit words. These signals are set to certain values based on the initialization controls obtained from the global control

State	Associated Signals	Description
START	LocalReset GlobalReset	Used when GlobalReset is asserted. Not used in normal operation.
RESET	LocalReset GlobalReset FU_accClear BP_CLG_accClear outputEnable	 All the accumulators are cleared All the registers are cleared Outputs are disabled
INITIALIZE	GlobalStart BPSumInit LearnRateSet MomentumSet	 The parameter registers and the weight registers are initialized
TESTMODE	FU_accClear inputEnable inputSelect weightSelect	Weighted sum of inputs is calculated and the output is passed to next layer This is used in real time operation and in test mode
TRAINMODE	all FP signals ReadModWeight ReadDeltaWt ReadLocalGrad WriteModWeight WriteLocalGrad BP_CLG_EnableAdd BP_CLG_accEnable BPSumSelect	 This is the training mode state Along with forward pass, weight modification is done in backward pass
HOLD	SignalForGCU	Checks for exceptions and holds values

Table 4.1: State Descriptions - Local Control Unit

unit.

BPSumSelect, InputSelect, and WeightSelect are select control signals to the buffers with multiplexers corresponding to Backpass sum, inputs and weights. These are two bit or three bit (depending on the number of weights and inputs to the neuron) signals that are binary coded to represent the selection.

inputEnable and outputEnable are the input output control signals. These are active high single bit signals.

ReadModWeight, ReadDeltaWt, and ReadLocalGrad are the signals used in the training mode when weight modification are to be done using the network parameters such as the local gradient. All of these are four bit signals that are binary coded.

WriteModWeight, WriteLocalGrad and WriteDeltaWt are the signals to write to the parameter registers. These are similar to the Read signals.

FU_accClear and FU_accEnable are the clear and enable signals for the forward unit accumulator. These signals are active high signals.

BP_CLG_accClear, BP_CLG_accEnable and BP_CLG_EnableAdd are the signals for clearing the corresponding intermediate registers and enabling computation of local gradient operation. These are also active high signals as the previous signals.

One important aspect of the control unit is the sequencer which allows for con-

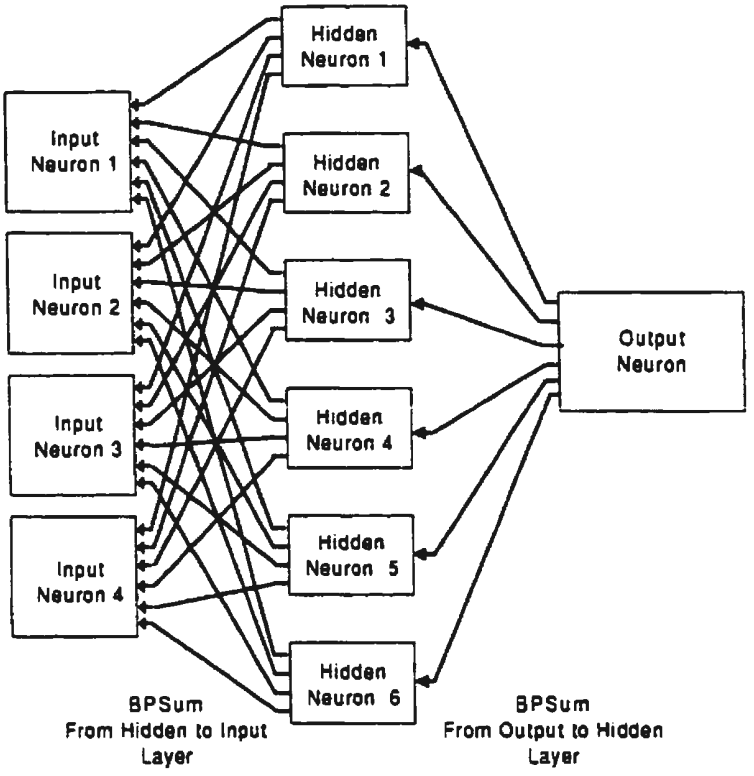


Figure 4.20: Illustration of Computation of Backpass sum

current processing in all neurons. The sequencer is described in the next subsection.

4.5.1.2 Description of the Sequencer

The backpropagation algorithm requires computation of back pass sums in each neuron which is a product of local gradient and the synaptic weight. This back pass sum is passed to the neuron in the previous layer which is connected to the computing neuron. Each neuron in a layer receives such back pass sum from all the neurons in the previous layer to which it is connected. This is illustrated in the Figure 4.20. As the figure shows, a neuron in the first layer will receive six back pass sums that will

Cycle	Neuron 1	Neuron 2	Neuron 3	Neuron 4
I	W_1H_1 & $W_1H_5 + W_1H_6$	W_2H_2	W_3H_3	W_4H_4
II	$acc + W_1H_2$	$acc + W_2H_3$ & $W_2H_5 + W_2H_6$	$acc + W_3H_4$	$acc + W_4H_1$
III	$acc + W_1H_3$	$acc + W_2H_4$	$acc + W_3H_1$ & $W_3H_5 + W_3H_6$	$acc + W_4H_2$
IV	$acc + W_1H_4 +$ $W_1H_5 + W_1H_6$	$acc + W_2H_1 +$ & $W_2H_5 + W_2H_6$	$acc + W_3H_2 +$ $W_3H_5 + W_3H_6$	$acc + W_4H_3 +$ $W_4H_5 + W_4H_6$

Table 4.2: Order of sequence - Input neurons

be used in the neuron for delta weight computation. If all the neurons compute with the same sequence, say starting from the first weight, at the end of first clock cycle, only the first neuron will have the back pass sums and other neurons need to wait for their values to arrive. Moreover, the number of back pass sums passed between layers differs among layers as the number of neurons in each layer is different. This also causes delay in processing. To eliminate these delays and to ensure synchronized concurrent processing a sequencer is required in each neuron.

The sequencer is a part of each local control unit. The sequencer receives an initial value from the control unit which is different for different neurons in a layer. The sequencer steps through the computation of back pass sum, output of neuron and the modification of weights based on the initial value. This allows for concurrent processing of all the neurons and eliminates any delay due to unavailable data. Tables 4.2 and 4.3 shows the order of computation in hidden and input neurons of DIANNE.

It can be seen from the tables that the order of computation of back pass sum in

Cycle	Neuron 1	Neuron 2	Neuron 3	Neuron 4	Neuron 5	Neuron 6
I	W_1H_1	W_2H_2	W_3H_3	W_4H_4	W_1H_5	W_1H_6
II	W_4H_1	W_1H_2	W_2H_3	W_3H_4	W_2H_5	W_2H_6
III	W_3H_1	W_4H_2	W_1H_3	W_2H_4	W_3H_5	W_3H_6
IV	W_2H_1	W_3H_2	W_4H_3	W_1H_4	W_4H_5	W_4H_6

Table 4.3: Order of sequence - Hidden neurons

hidden neurons corresponds to the order of computation in the input neurons. W corresponds to the weights and H corresponds to the hidden neuron's local gradient value.

4.5.2 Global Control Unit

The function of the global control unit is to control the flow of data between the neuron layers and to synchronize the operation of different neurons in a layer. The global control takes care of the initialization, mode of operation and exception handling as well. The symbolic diagram of the global control unit is shown in Figure 4.21. The global control unit is also a state machine similar to the local control unit. The state diagram is shown in Figure 4.22. The descriptions of the different states and the associated signals are given in Table 4.4.

4.5.2.1 Description of Control Signals

The signals described here are a group of signals that are identified under one common name. These signals are actually connected to all the neurons in the neural processor.

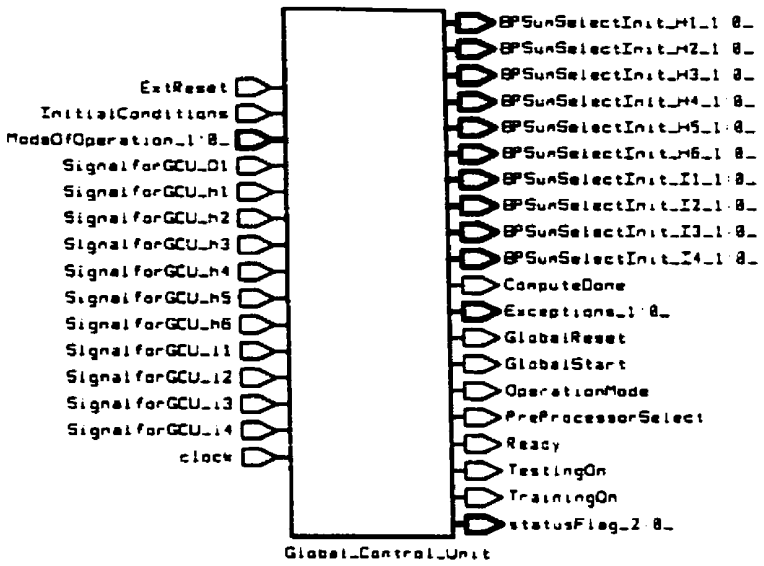


Figure 4.21: Global Control unit

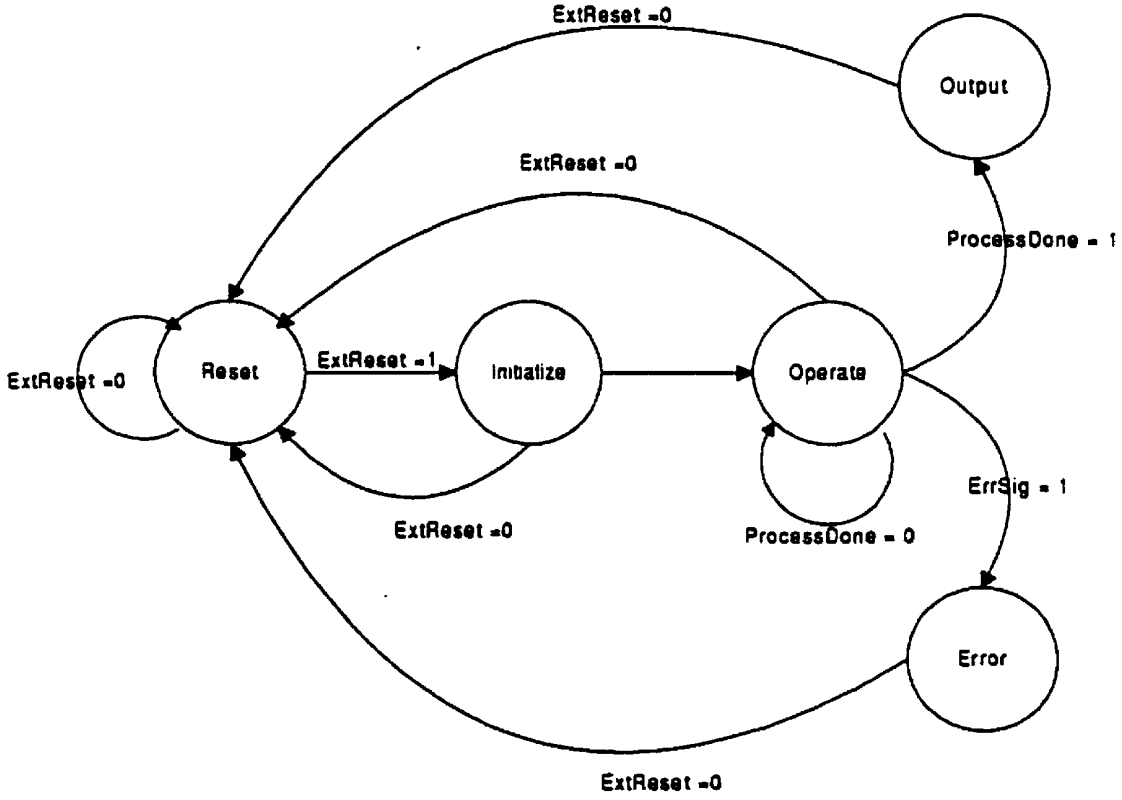


Figure 4.22: State diagram of global control unit

STATE	Associated Signals	Description
RESET	GlobalReset Exceptions	This is a starting state and exception handling state
INITIALIZE	BPSumSelectInit Ready InitialConditions ModeOfOperation	This state is used for initializing mode of operation and the parameter registers
OPERATE	TestingOn TrainingOn Exceptions	This state is main operation state and is mainly controlled by the respective LCUs
ERROR	Exceptions	Exception handling state

Table 4.4: State descriptions - Global Control unit

GlobalReset is a reset signal that is an external input which can be used to reset the whole processor. This would reset all the neurons and bring it to a fresh start state. This is an active low signal.

BPSumSelectInit, ModeOfOperation, InitialConditions are the set of signals for initializing the different neurons of their BP Sum sequencer, mode of operation and the parameter registers. These are internal signals generated by the global control unit and are passed to the local control units.

Ready, TestingOn, TrainingOn are the flag signals that indicate the operation status. These are external outputs of the processor.

Exceptions is a signal that indicates an exceptional condition in the processor. This would also initiate a global reset of all the neurons.

4.6 Testing the Design

DIANNE was tested for functionality and features at all levels of design. This section describes all the testing methods and provides the results and discussions on the results. Although the section provides most of the test results, some of the more evident test results, for example those of smaller components like the adder, multiplier and flipflops are not provided. The testing consists of three parts which are feature or functional verification, integrated random testing or global testing and exceptions testing. The test results reported for the Integrated random testing are for the "Test Mode" of DIANNE. The "Train Mode" of DIANNE in the Integrated random test has not been thoroughly verified, but the individual components of the Train Mode have been verified for their functionality. In the simulation results, the waveform viewer provides decimal equivalents of the hexadecimal valaues of the signals. But these signals are to be interpreted in the fixed point representation described in the previous chapter.

4.6.1 Functional Verification

Functional verification includes verifying the functionality of individual components and the features of the component. All the tests were carried out using the vhdlan CAD tool with a test clock period of 20 ns. The preprocessors that were explained in a earlier section were tested for the functionality and the test results are provided in Figures 4.23 and 4.24. From these figures it can be seen that the preprocessors

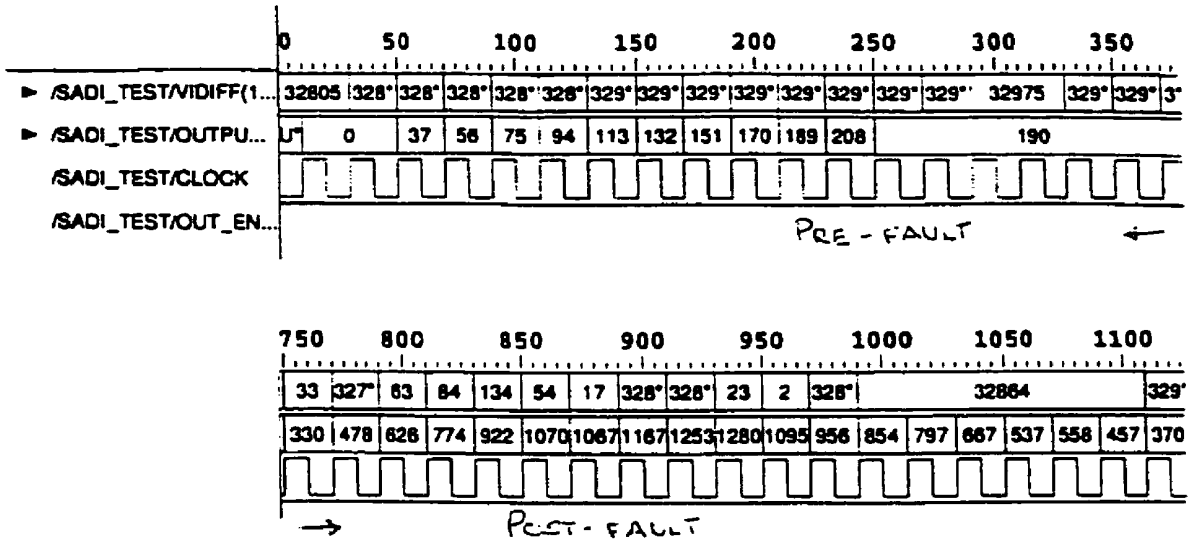


Figure 4.23: Simulation results of SADI

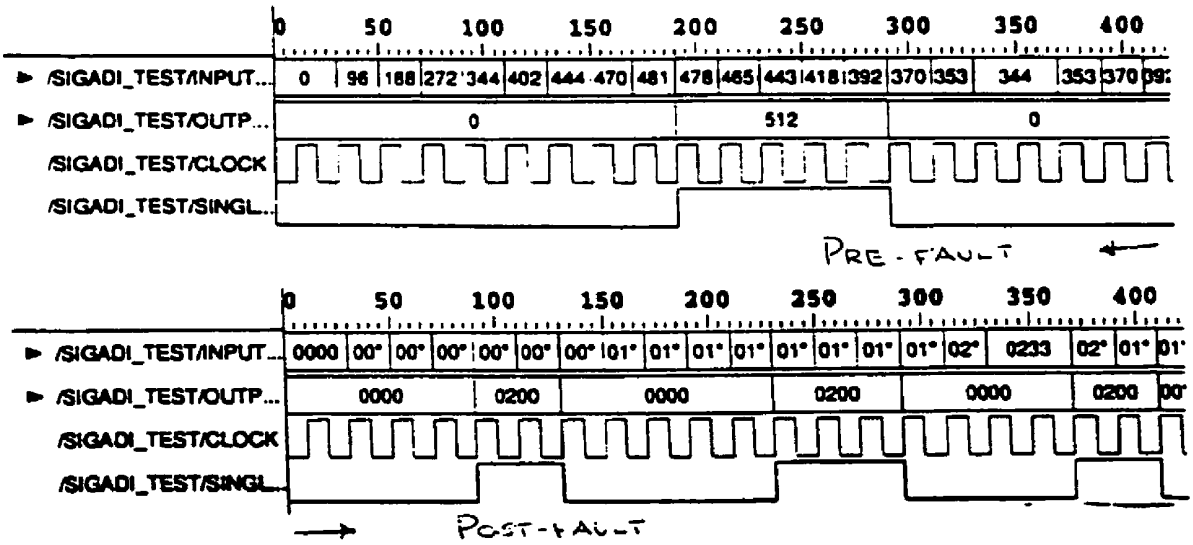


Figure 4.24: Simulation results of SIGADI

provide filtered values of the external input signals. SADI gives a value close to zero (in the figure, values less than 210) for all the inputs before the fault condition and a value close to 2 (in the figure, values more than 512) when a fault occurs at the inputs. SIGADI also gives output as expected.

The most important part of the neural processor is the control unit. The local and the global control units were tested for the functionality. The results are shown in Figures 4.25 and 4.26. From the figures it can be seen that the control units are working as expected. It can be verified from the state signals that change corresponding to the respective state diagrams. The test verifies the functionality and hence it is assumed that there are no exceptions at this point of operation. The exception case is discussed at a later section. The global control unit goes through all the states except the ERROR state that is mentioned in the description of global control unit. The local control unit test is for the training mode that includes the forward and backward pass operations. This makes sure that both the operations are verified for functionality. The switch between modes of operation is illustrated in Figure 4.27. The figure shows only those signals that are necessary to verify the operation. Other signals are asserted as illustrated in the regular operational simulations.

Another part of the control unit is the sequencer that sequences the computation in the neurons based on the settings from the global control unit. The test shows the functionality of the sequencer for different initial settings. The results are shown in

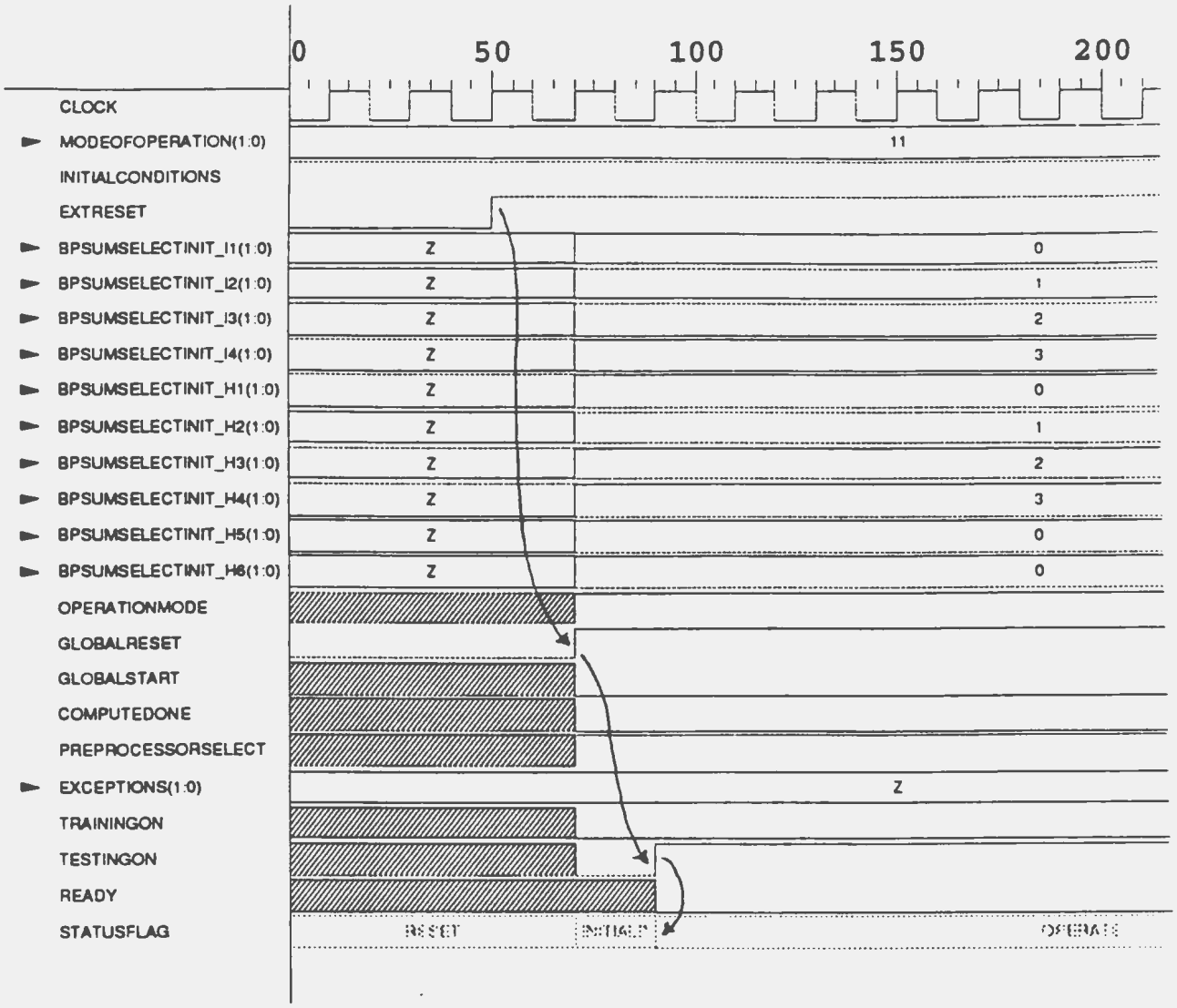


Figure 4.25: Simulation results of Global Control Unit

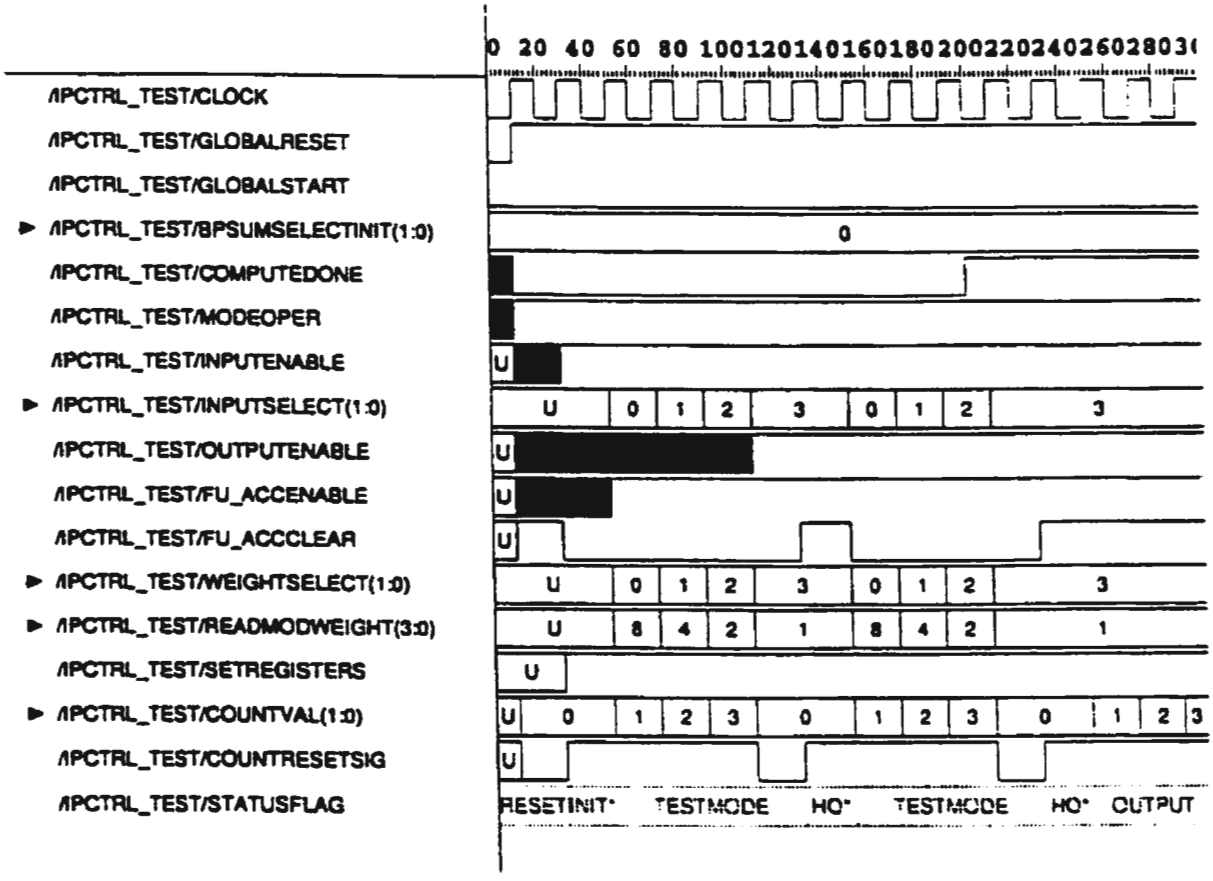


Figure 4.26: Simulation results of Local Control Unit

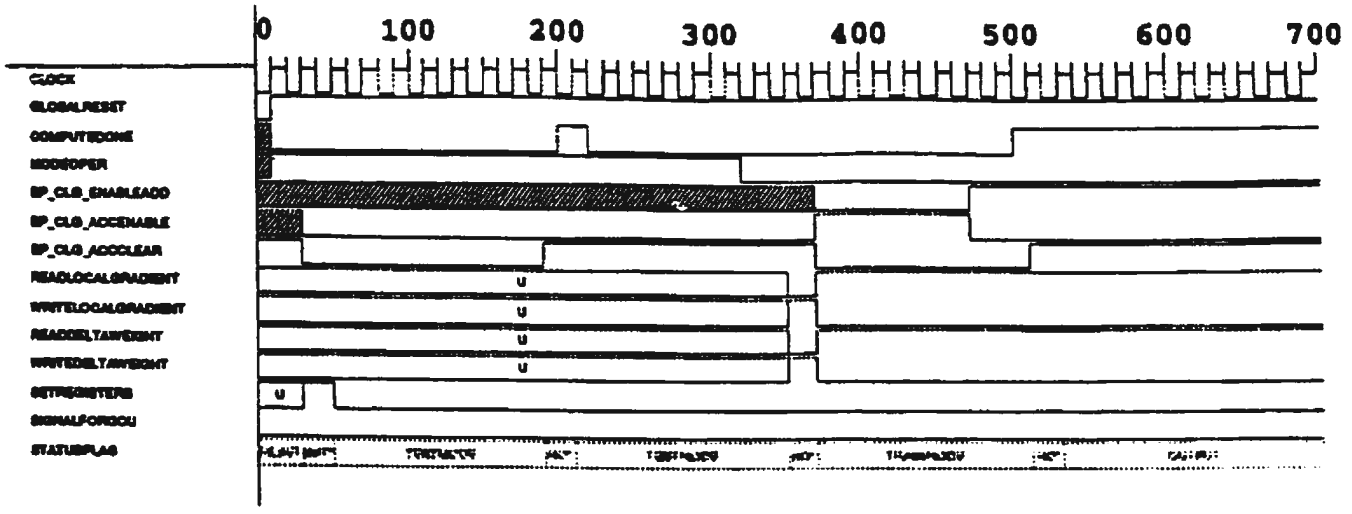


Figure 4.27: Illustration of modes of operation

Figure 4.28. It can be seen that the sequencer accepts input from the initialization signal and rotates it through the sequence that is specific to that neuron. It is generally anti clockwise rotation from the initial setting. The HOLD registers mentioned in the earlier sections are also essential part of the design which helps in concurrent operation of all neurons and in maintaining correlation between modification weights and the inputs. Figure 4.29 illustrates the functionality of the hold registers. The test shown is for the hold registers of the input neurons which is 5×3 register. This also verifies the other hold registers as they are smaller versions of the same. Functional verification of the function lookup is given in Figure 4.30. Similar verification was done for the derivative lookup as well.

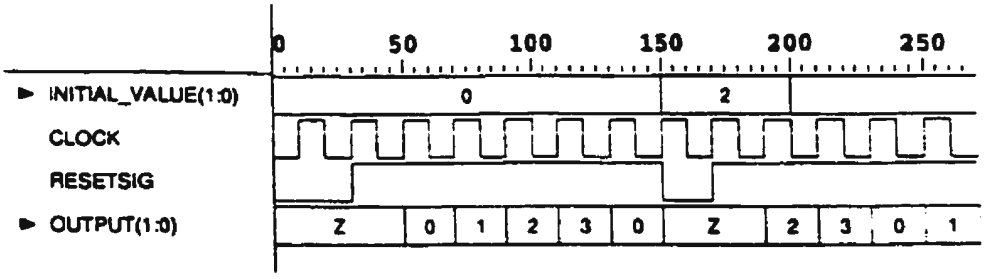


Figure 4.28: Simulation results of the sequencer

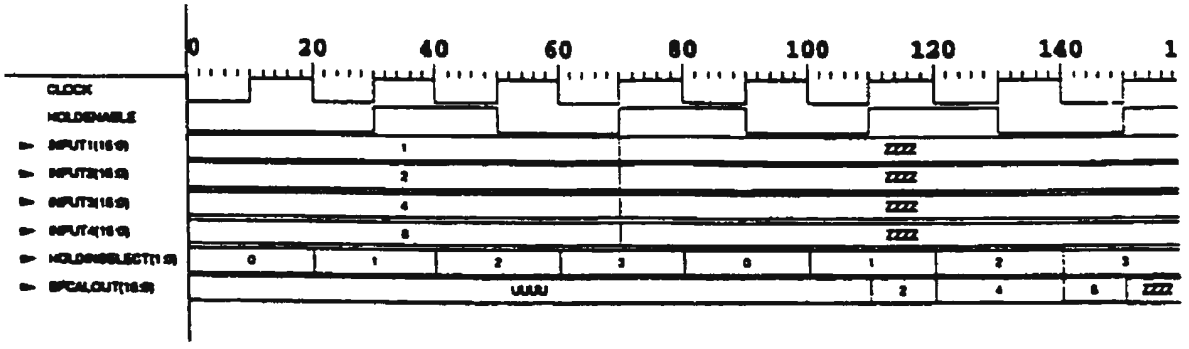


Figure 4.29: Simulation results of HOLD register

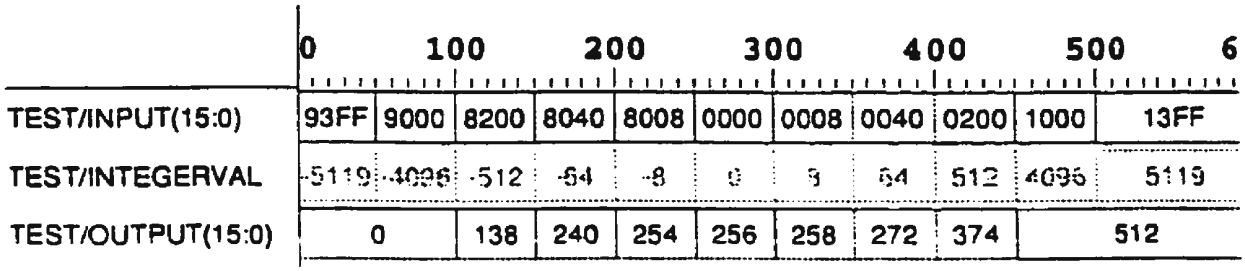


Figure 4.30: Simulation results of Function lookup

4.6.2 Integrated Random Testing

This test involves integration of smaller components to form the sub blocks of the design and to test them for their correctness as a block. The integrated random test was conducted for the forward pass unit, backward pass unit, single neuron and the neural processor. While testing the individual subblocks, the other blocks are assumed to be working without any fault. The results of the testing of forward pass unit is given Figure 4.31. The control signals were generated as designed and the unit generates output as expected. The simulation results of the backward pass unit is given in Figure 4.32. The backward pass unit also works as expected. The register file is just a set of registers which receives modified weights from the backpropagation unit and stores them for the next pass. The register file was also tested for its functionality and it works as expected. A full integrated test of a single neuron was conducted assuming

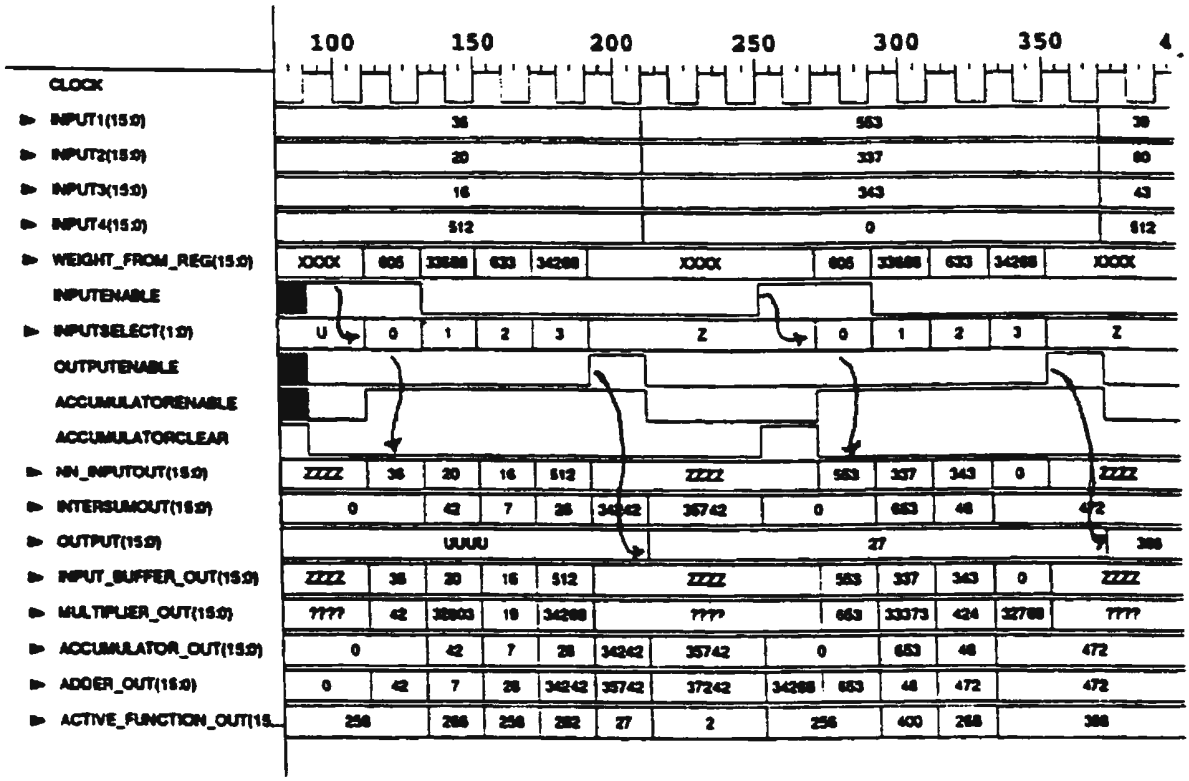


Figure 4.31: Simulation results of forward pass unit

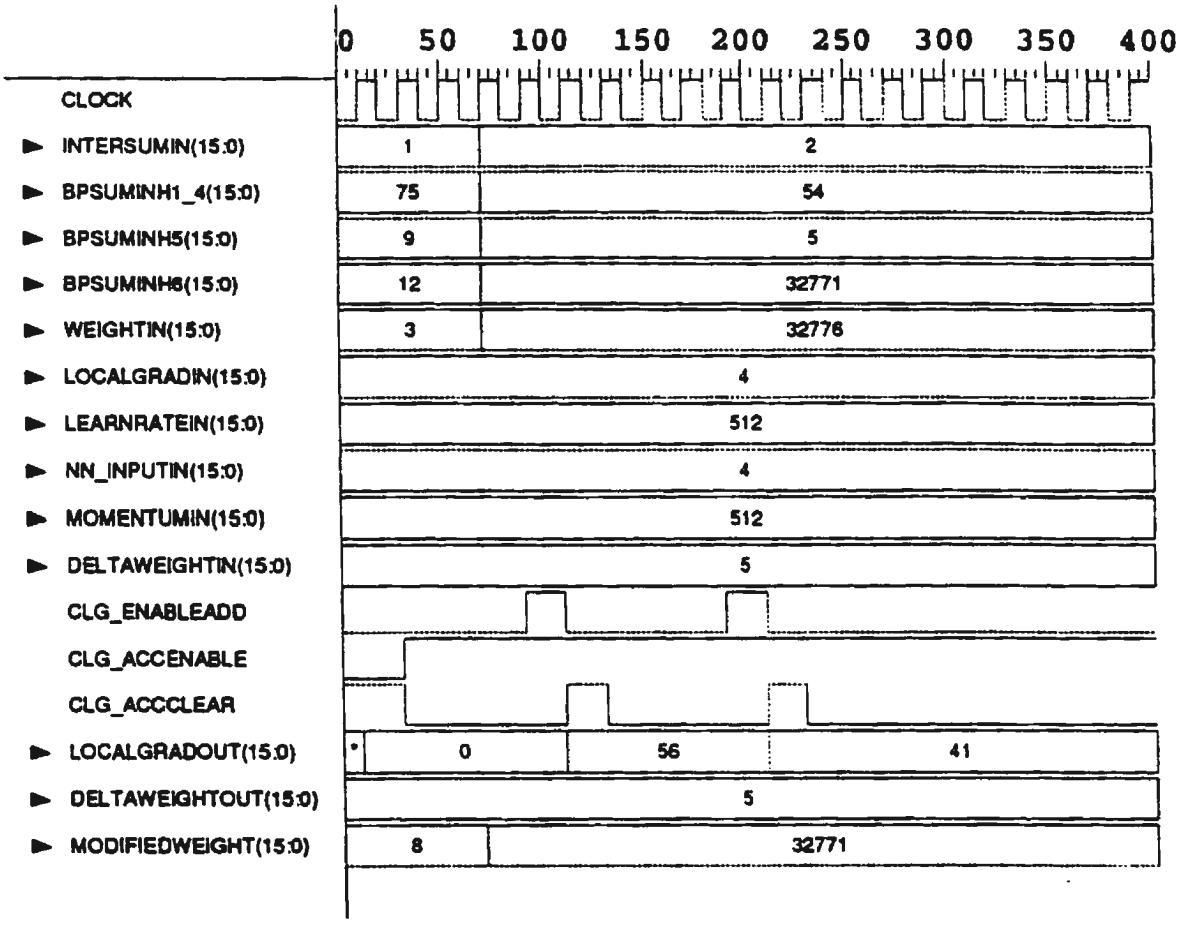


Figure 4.32: Simulation results of Backward pass unit

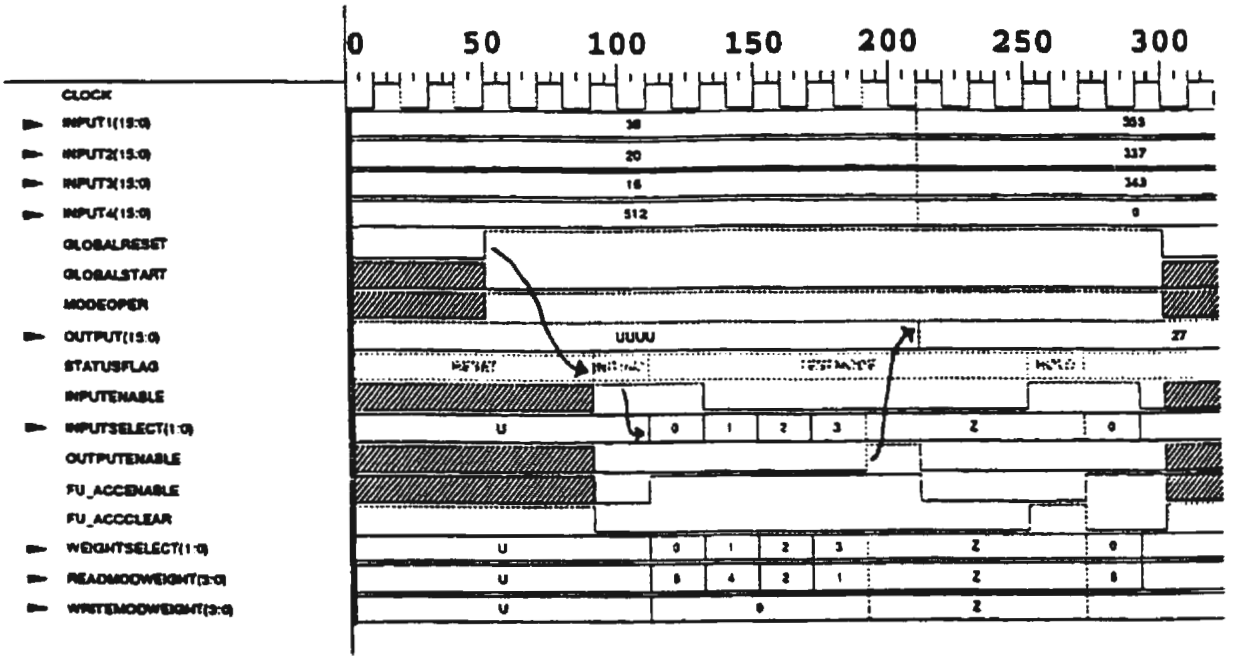


Figure 4.33: Simulation results of a single neuron

that other neurons pass values as expected. The results are shown in Figure 4.33. The neural processor as a whole was also tested for its functionality. The results are shown in Figure 4.34. As it can be seen from the figure, four sets of inputs are passed to the neural processor, which are the same as those used in the software simulator. The intermediate results as well as the final outcome of the processor are exactly the same as that obtained in software simulation. The figure illustrates the different states of the processor (shown by signal STATUS) and individual neurons (shown by signals statusflags) at each cycle of operation. The output is shown by signal NP_OUTPUT. The initial "UUUU" results are due to the fact that in the first few cycles the processor is initialized and then the values are passed between neurons. The actual outputs are available only after 590 ns. The values obtained before 590 ns

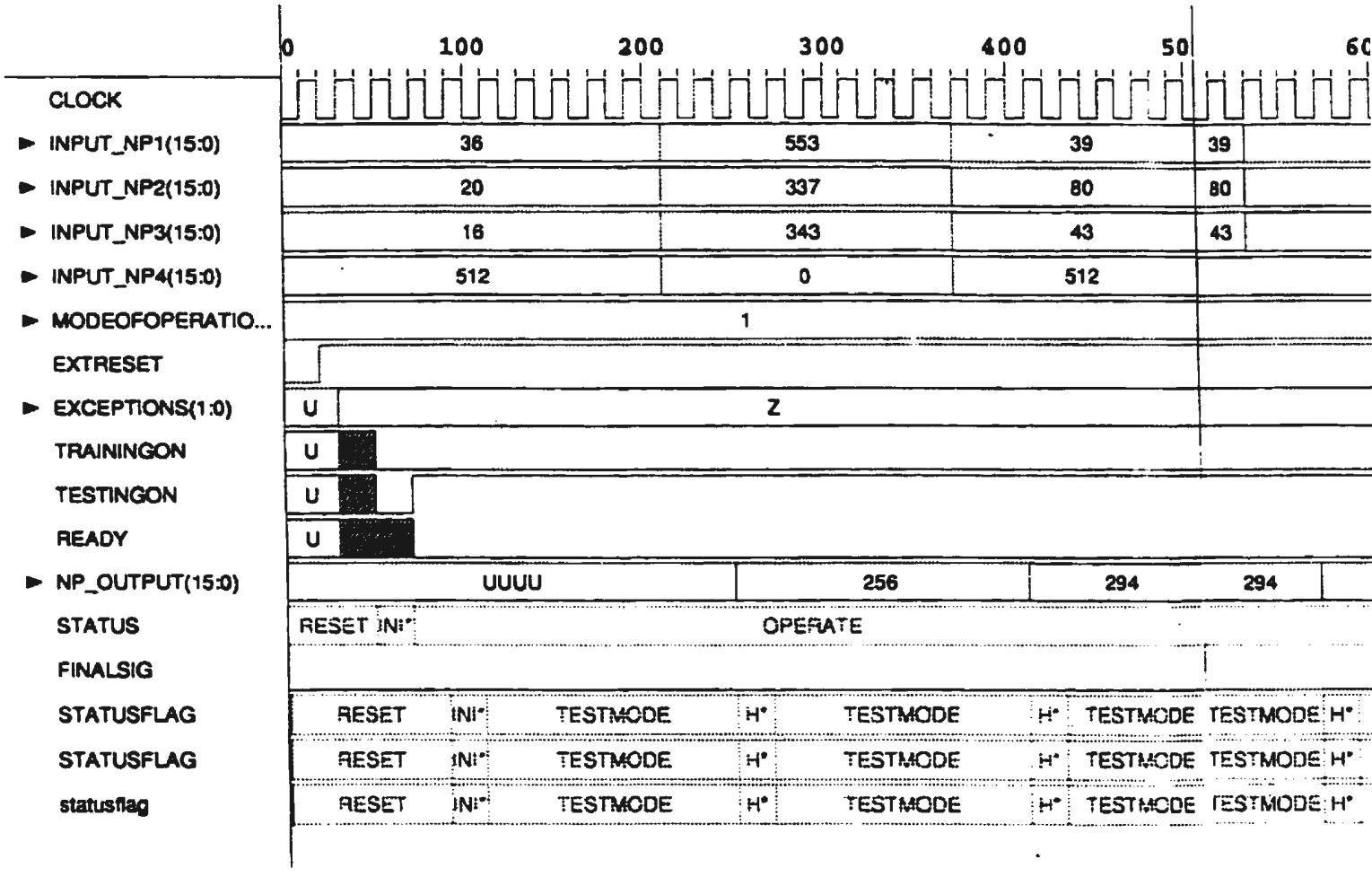
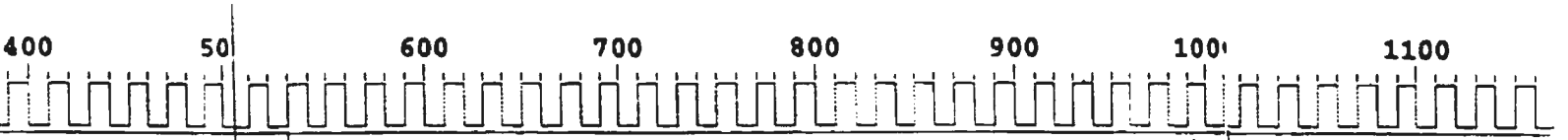


Figure 4.34: Simulation results of complete integrated test



39	39	728
80	80	429
43	43	441
512	512	
	1	
	Z	

	294	294	3	492	3	3	5

OPERATE

H*	TESTMODE	TESTMODE	H*	TESTMODE	H*	TESTMODE	H*	TESTMODE	TESTM*	H*	TESTMODE
H*	TESTMODE	TESTMODE	H*	TESTMODE	H*	TESTMODE	H*	TESTMODE	TESTM*	H*	TESTMODE
H*	TESTMODE	TESTMODE	H*	TESTMODE	H*	TESTMODE	H*	TESTMODE	TESTM*	H*	TESTMODE

are values due to initializing of accumulators and other registers. This is because of the pipelined nature of the processor. The output neuron receives the actual inputs only in third cycle. From then on, outputs are obtained every eight clock cycles. These eight clock cycles are due to the multicycle implementation of the neurons. The operation of the pipeline was explained in an earlier section. The inputs are shown as decimal values but in fact they are fixed point representations. The reason is that the waveform viewer does not support viewing of custom representations.

4.6.3 Exceptions Testing

This method of testing is to observe the function of the processor under exceptions. One of the conditions is the asynchronous reset condition at the global control unit. This should generate a global reset and clear all the registers and bring the processor to a fresh start state. The results are shown in Figure 4.35. Another exception condition is the occurrence of overflow in any of the neurons. This should generate a local reset and should send a signal to the global control unit about the problem. The results of the test are shown in Figure 4.36. The following section discusses the main features of the design and the speed of operation.

4.7 Features of the Design

The main features of the design are the on-chip preprocessor and the on-chip training function. This design also allows multimodal operation, meaning that more than one

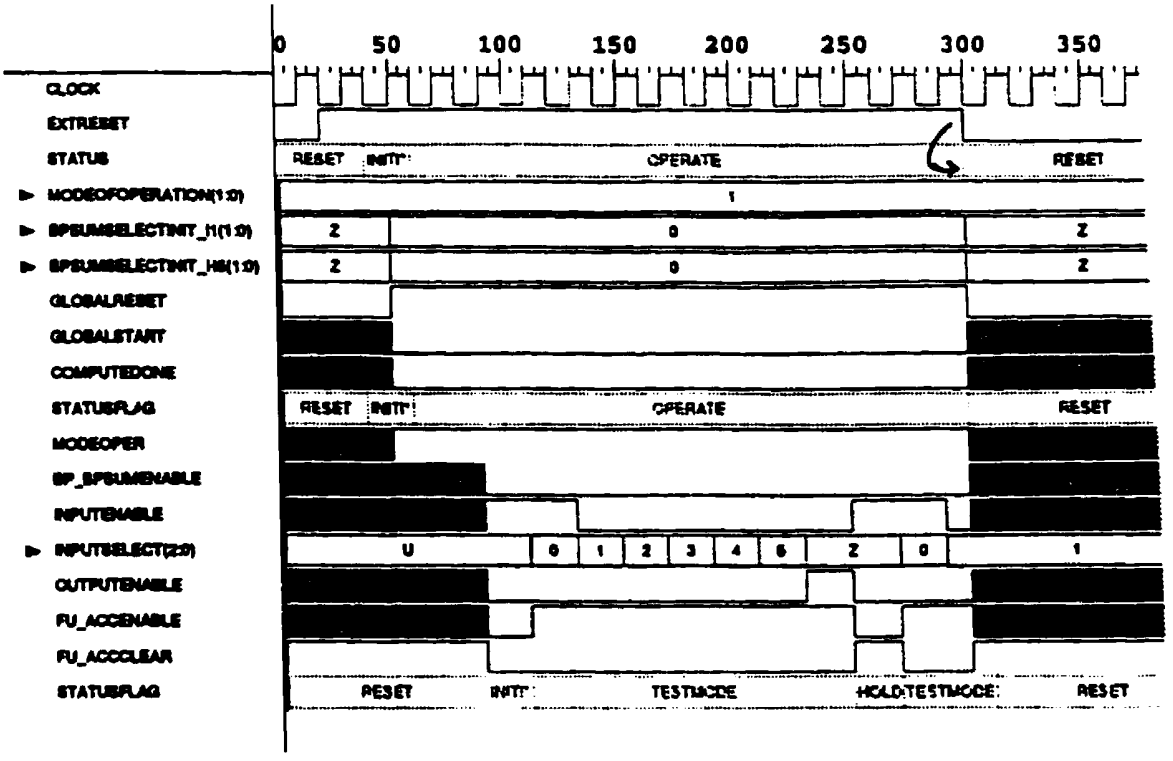


Figure 4.35: Simulation results under RESET condition

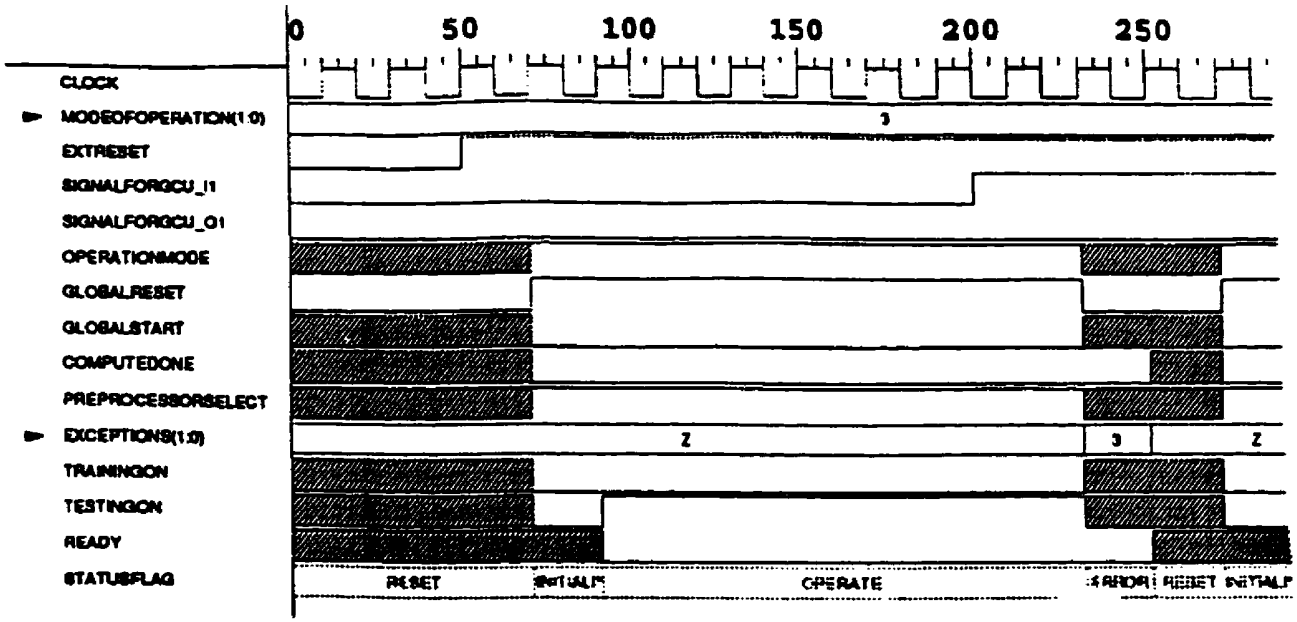


Figure 4.36: Simulation results under OVERFLOW condition

mode of operation is possible. The design allows operation of the neural processor unit with or without the use of preprocessors. The design also supports training and test modes of operation. The design supports concurrent forward and backward pass operations which is not supported in most of the designs reported to date [46, 40, 39]. Some of the design features are the speed of operation and the gate count. The input data range for the processor is $\approx$ -63.99,...,+63.99 with a resolution as 6 bits of exponent and 9 bits of fraction. As mentioned in the earlier sections the design was carried out using CMOSIS5 technology which is CMOS 0.5μ design technology. With this technology the gate count of the whole design is close to 260,000 gates. The large number of gates is due to the large number of arithmetic components like adders and multipliers used in the design. The gate count with respect to each component is given in the following table.

Component	Gate Count
forward unit	5632
back pass unit	16800
register file	2808
local control unit	720
global control unit	620
Preprocessors	10000

With the CMOSIS5 technology and a 40 MHz clock speed, the connection updates per second of the design is 2G i.e. speed of operation is 2 GCUPS. This favourably compares to most of the reported speeds of operation.

4.8 Summary

This chapter discussed the hardware design of DIANNE in a detailed manner. The design issues and the decision trade-offs related to the design were explained in detail. Each block and subunit of the processor was explained for its features and its functionality. The design cycle and the features of the design were discussed. The testing of the processor was discussed elaborately. The results of the testing were presented and discussed. Some features of the design specific to the technology of implementation were presented.

Chapter 5

Conclusion and Suggested Future Work

In the earlier chapters, the design and implementation process of a digital neural processor, DIANNE, for detection applications was discussed. A survey of similar designs was presented and related works were presented and analyzed. An elaborate discussion and description of the problem was presented in Chapter 3. The method of solution was discussed in detail. The design of the preprocessors for the application and the justification of the design method was also presented. The previous chapter discussed the hardware design aspects of the neural processor and the testing of the design. The main features of the design and the results of hardware simulation were also explained. This chapter summarizes the work and concludes the thesis. The following sections discuss some of the main contributions of the thesis and possible future work in different aspects of the thesis are mentioned. A critical assessment of the work done is presented and the thesis is concluded in the final section.

5.1 Contributions of the Thesis

There are three main contributions of the thesis as well as several minor contributions and novel design ideas. The main contributions are

- The hardware design of the digital neural processor, DIANNE. A complete multimodal 16 bit integer arithmetic digital neural processor with 11 neurons was designed based on the results of the simulation.

- The preprocessors are the main contribution towards the application. The method reduces the size of the artificial neural network required for the application and it enables real-time operation unlike other published solutions.

- A software simulator which supports flexible construction of an artificial neural network with backpropagation training algorithm has been developed. The simulator resembles the hardware design to identify the hardware requirements of the design through simulation. The simulator is object oriented and also supports different training methods. The simulator also supports fixed point simulation for studying quantization effects and identifying the bus width requirements.

Some other minor contributions are some of the design ideas like the back pass sum sequencer and the hold registers that would eliminate the sequential nature of the backpropagation algorithm. The simulator and the design are modular and flexible

so that future additions can be done with minor modifications to the code.

5.2 Improvements over the Hardware design

Hardware design of DIANNE was done as efficiently as possible with the current resource constraints. Still there is room for modifications and improvements to the current design which would make DIANNE a better neural processor. One of the design components that can be improved is the interconnection and interface between similar chips or other chips that support similar algorithms. Current design is self-contained and cannot be connected to other similar chips except for expanding the size of the network. Moreover, it has only eleven neurons that are enough for this application but may not be suited for similar detection applications, as the processor can be used for other detection applications. So the design could be improved to accommodate interconnections through interconnection buffers. The reason for not supporting this part in the design is the availability of limited input/output pins. This can be overcome by serial in parallel out or parallel in serial out sort of interconnections. This would allow more chips to be connected on a board level design for applications that would require neural networks larger than eleven neurons. The current design does not support loading of registers with pre-determined weights except for initialization. The training has to be done online. This can be modified to read weights from external sources. But the input/output pins could be a limitation in this case as well.

There could be some improvement over the design of the single neuron too. The current design is a multicycle implementation with a mix of interleaved pipelining. This could be modified to yield a completely pipelined design for better performance. The current design had hardware complexity limitations which restricted the complete pipelining. With the new technologies such as the CMOSP35 which are available now, the hardware complexity limitations could be overcome. Another area for improvement is the exception handling. In the current design exceptions only reset the system in case of errors. This could be modified to stall the processor in case of exceptions and correct the error or prompt manual intervention. Hardware complexity is the limitation here as well.

5.3 Future Work on the Software Design

The current software design is object-oriented but it has some constraints. The object oriented nature of the simulator could be improved further to accommodate more training algorithms, more user friendliness and different types of neurons. The current version supports fixed point simulation as a separate module. Current design of the software simulator was designed taking into account the specific application and hence not optimized for the use of memory. This could be modified to accommodate different applications. The commercial versions of ANN simulators do not facilitate fixed point simulation or flexibility over the structure or size of the network. Therefore an enhanced version of this simulator could prove to be useful to other researchers.

5.4 Critical Assessment and Conclusion

The earlier sections discussed some of the possible improvements over the current design. The main objective of the thesis is to design a digital neural processor for detection applications. The application chosen for analysis is the protection of transmission lines that has been explained in earlier chapters. A survey of known methods in solving the distance protection using artificial neural network reveals the use of complex filters and external fault identifiers. Upon proper analysis of the data, the preprocessors were designed which reduced the hardware complexity to a minimum and eliminated the use of external fault identifiers. From the design of the preprocessors, it can be stated that, if properly processed, the solution for distance protection would not require a large neural network. It can even be concluded that neural networks could be avoided in solving the problem.

A single transmission line was simulated to obtain the fault data. Strictly speaking, the whole power grid should be simulated, using complex simulation softwares, and the resulting fault data should be used to design the detector. Based on the nearly disjoint clusters obtained after preprocessing the data (derived from a single transmission line) it can be projected that an artificial neural network is not necessary for this application. Instead a well-designed preprocessor followed by a simple comparator could accomplish this task. Even if this cannot be concluded as stated above, it is very much clear that complex filters and large neural networks are not required

for this application. Fault location has not been studied in this thesis. However, it may be predicted that only a simple ANN will be, if at all, required for locating the fault, provided a good preprocessor is introduced. But the use of neural networks may be justified for larger problems like control or estimation.

References

[1] N. B. Karayiannis and A. N. Venetsanopoulos, *Artificial Neural Networks: Learning Algorithms, Performance Evaluation and Applications.* Kluwer Academic Publishers, 1993.

[2] S. Haykin, *Neural Networks: A Comprehensive Foundation.* IEEE Press, Macmillan College Publishing Company, Inc., 1994.

[3] J. A. Freeman and D. M. Skapura, *Neural Networks: Algorithms, Applications and Programming Techniques.* Addison-Wesley Publishing Company, Inc., 1992.

[4] J. A. K. Suykens, J. P. L. Vandewalle, and B. L. R. De Moor, *Artificial Neural Networks for Modelling and Control of Non-Linear Systems.* Kluwer Academic Publishers, 1996.

[5] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence.* Prentice-Hall, Inc., 1997.

[6] D. Guo and G. Parr, "Applying Neural Networks to ATM Cell Scheduling in Multistage Switches," in *The Proceedings of the 1998 Symposium on Performance Evaluation of Computer and Telecommunication Systems(SPECTS '98)*, pp. 37–41, 1998.

[7] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, 1962.

[8] T. Cornu and P. Ienne, "Performance of Digital Neuro-Computers," in *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pp. 87–93, 1994.

[9] P. Ienne, "Quantitative Comparison of Architectures for Digital Neuro Computers," in *Proceedings of the International Joint Conference on Neural Networks*, vol. II, pp. 1987–1990, 1993.

[10] L. M. Reyneri and E. Filippi, "An Analysis on the Performance of Silicon Implementations of Backpropagation Algorithms for Artificial Neural Networks," *IEEE Transactions on Computers*, vol. 40, pp. 1380–1389, December 1991.

[11] R. P. Gorman and T. J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 36, pp. 1135–1140, 1988.

[12] J. Hwang and J. Holt, "Finite Precision Error Analysis of Neural Network Electronic Hardware Implementation," in *The Proceedings of the 1991 International Joint Conference on Neural Networks(IJCNN '91)*, vol. I, pp. 519–526, 1991.

[13] P. P. Gandhi and V. Ramamurthy, "Neural networks for signal detection in non-Gaussian noise," *IEEE Transactions on Signal Processing*, vol. 45, pp. 2846–2851, November 1997.

[14] N. Muthuswamy and R. S. Blum, "Neural detectors for medical signal processing," in *The Proceedings of the First Regional Conference of IEEE Engineering in Medicine and Biology Society*, pp. 4/31–4/32, 1996.

[15] F. Vaz and J. C. Principe, "Neural networks for EEG signal decomposition and classification," in *The Proceedings of the IEEE 17th Annual Conference of Engineering in Medicine and Biology Society*, vol. 1, pp. 793–794, 1995.

[16] G. Sylvestri, F. B. Verona, M. Innocenti, and M. Napolitano, "Fault detection using neural networks," in *The Proceedings of the 1994 IEEE International Conference on Neural Networks*, vol. 6, pp. 3796–3799, 1994.

[17] D. Patel, I. Hannah, and E. R. Davis, "Soft contaminant detection using neural networks: techniques and limitations," in *The Proceedings of the 1994 IEEE International Conference on Neural Networks*, vol. 7, pp. 4316–4320, 1994.

[18] D. V. Coury and D. C. Jorge, "Artificial Neural Network Approach to Distance Protection of Transmission Lines," *IEEE Transactions on Power Delivery*, vol. 13, pp. 102–108, January 1998.

[19] F. Zahra, B. Jeyasurya, and J. E. Quaicoe, "Artificial Neural Network Based Transmission Line Protective Relaying," in *Proceedings of the 30th North American Power Symposium*, October 1998.

[20] T. Cornu, P. Ienne, D. Niebur, and M. A. Viredaz, "A Systolic Accelerator for Power System Security Assessment," *Proceedings of the International Conference on Intelligent System Application to Power Systems*, vol. 1, pp. 431–438, September 1994.

[21] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, vol. 1, pp. 4–27, March 1990.

[22] R. Safaric, K. Jezernik, M. Pec, and I. J. Rudas, "Implementation of neural network sliding-mode controller for DD robot," in *Proceedings of the IEEE International Conference on Intelligent Engineering Systems (INES '97)*, pp. 83–88, 1997.

[23] R. J. T. Morris and B. Samadi, "Neural network control of communications systems," *IEEE Transactions on Neural Networks*, vol. 5, pp. 639–650, July

1994.

[24] U. Ramacher and U. Rückert, *VLSI Design of Neural Networks.* Kluwer Academic Publishers, 1991.

[25] K. W. Przytula and V. K. Prasanna, *Parallel Digital Implementations of Neural Networks.* Prentice-Hall, Inc., 1993.

[26] B. J. Sheu and J. Choi, *Neural Information Processing and VLSI.* Kluwer Academic Publishers, 1995.

[27] A. König, "Survey and Current Status of Neural Network Hardware," in *Proceedings of the International Conference on Artificial Neural Networks*, pp. 391–410, October 1995.

[28] Y. Hirai, "Recent VLSI Neural Networks in Japan," *Journal of VLSI Signal Processing*, vol. 6, pp. 7–18, 1993.

[29] H. P. Graf, E. Sackinger, and L. D. Jackel, "Recent Developments of Electronic Neural Nets in North America," *Journal of VLSI Signal Processing*, vol. 5, pp. 19–31, 1993.

[30] P. Ienne, T. Cornu, and G. Kuhn, "Special-Purpose Digital Hardware for Neural Networks: An Architectural Survey," *Journal of VLSI Signal Processing*, vol. 13, pp. 5–25, 1996.

[31] C. S. Lidsey and T. Lindblad, "Review of Hardware Neural Networks: A User's Perspective," *International Journal of Neural Systems*, vol. 6, pp. 215–224, 1995.

[32] C. Mead, *Analog VLSI and Neural Systems*. Addison-Wesley Publishing Company, Inc., 1989.

[33] P. Ienne, "Digital Connectionist Hardware: Current Problems and Future Challenges," *Biological and Artificial Computation: From Neuroscience to Technology, Lecture Notes in Computer Science*, vol. 1240, pp. 688–713, 1997.

[34] B. E. Boser, E. Sackinger, J. Bromley, Y. LeCun, and L. D. Jackel, "Hardware Requirements for Neural Network Pattern Classifers: A Case Study and Implementation," *IEEE Micro*, pp. 32–40, February 1992.

[35] N. Manduit, M. Duranton, J. Gobert, and J.-A. Sirat, "LNeuro1.0: A piece of hardware LEGO for buliding neural network systems," *IEEE Transactions on Neural Networks*, vol. NN-3, pp. 414–422, May 1992.

[36] M. Duranton, "L-Neuro 2.3: a VLSI for Image Processing by Neural Networks," in *Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro '96)*, pp. 157–160, February 1996.

[37] D. Hammerstrom, "A Highly Parallel Digital Architecture for Neural Network Emulation," in *VLSI for Artificial Intelligence and Neural Networks* (J. G.

Delgado-Frias and W. R. Moore, eds.), pp. 357–366, Plenum Press, New York, 1991.

[38] K. Asanović, B. E. D. Kingsbury, N. Morgan, and J. Wawrzynek, "HiPNeT-1: A Highly Pipelined Architecture for Neural Network Training," *Technical Reports of International Computer Science Institute, University of California at Berkeley, California, USA*, October 1991.

[39] D. D. Caviglia and M. Marchesi, "A Neural ASIC Architecure for Real-Time Classification," in *Proceedings of the 21st EUROMICRO Conference (EUROMICRO '95)*, pp. 632–638, September 1995.

[40] V. Tryba, "Neuro-ASIC for Low Cost Supervision of Water Pollution," in *Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, pp. 111–116, August 1996.

[41] U. Ramacher, J. Beichter, and N. Brüls, "A General-Purpose Signal Processor Architecture for Neurocomputing and Preprocessing Applications," *Journal of VLSI Signal Processing*, vol. 6, pp. 45–56, 1993.

[42] W.-C. Fang, G. Yang, B. Pain, and B. J. Sheu, "A Low Power SMART Vision system Based on Active Pixel Sensor Integrated with Programmable Neural Processor," in *Proceedings of the IEEE International Conference on Computer*

Design: VLSI in Computers and Processors (ICCD '97), pp. 429–434, October 1997.

[43] K. Asanović, J. Beck, B. E. D. Kingsbury, and P. Kohn, "SPERT: A Neuro-Microprocessor," in *VLSI for Neural Networks and Artificial Intelligence* (J. G. Delgado-Frias and W. R. Moore, eds.), pp. 103–108, Plenum Press, New York, September 1994.

[44] W. Fornaciari and F. Salice, "A Low Latency Digital Neural Network Architecure," in *VLSI for Neural Networks and Artificial Intelligence* (J. G. Delgado-Frias and W. R. Moore, eds.), pp. 81–92, Plenum Press, New York, September 1994.

[45] J. G. Delgado-Frias, S. Vassiliadis, G. G. Pechanek, W. Lin, S. M. Barber, and H. Ding, "A VLSI Pipelined Neuroemulator," in *VLSI for Neural Networks and Artificial Intelligence* (J. G. Delgado-Frias and W. R. Moore, eds.), pp. 71–80, Plenum Press, New York, September 1994.

[46] L. Larsson, S. Krol, and K. Lagemann, "NeNEB - An Application Adjustable Single Chip Neural Network Processor for Mobile Real Time Image Processing," in *Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, pp. 154–162, August 1996.

[47] A. Wright and C. Christopoulos, *Electrical Power System Protection.* Chapman & Hall Inc., 1991.

[48] The Electricity Training Association, ed., *Power System Protection*, vol. 2 & 4. London, United Kingdom: The Institution of Electrical Engineers, 1995.

[49] N. R. Shanbhag and K. K. Parhi, *Pipelined Adaptive Digital Filters.* Kluwer Academic Publishers, 1994.

[50] T. W. Parks and C. S. Burrus, *Digital Filter Design.* John Wiley & Sons, Inc., 1987.

[51] F. Zahra, "Artificial Neural Network Approach to Transmission Line Relaying," M.Eng. thesis, Memorial University of Newfoundland, 1998.

[52] S. Agaian, J. Astola, and K. Egiazarian, *Binary Polynomial Transforms and Nonlinear Digital Filters.* Marcel Dekker, Inc., 1995.

[53] H. Schildt, *C++: The Complete Reference.* McGraw-Hill, Inc., 1995.

[54] B. Stroustrup, *The C++ Programming Language.* Addison-Wesley Publishing Company, Inc., 1991.

[55] Canadian Microelectronics Corporation, Ontario, Canada, *Basic Digital IC Design Flow Instruction*, November 1997.

[56] Z. Navabi, *VHDL: Analysis and Modeling of Digital Systems.* McGraw-Hill, Inc., 1998.

[57] Synopsys, Inc., California, United States of America, *VHDL Compiler Reference Manual*, November 1992.

[58] Cadence Design Systems, Inc., California, United States of America, *Integrated IC Design System: Design Synthesis Reference Manual*, March 1989.

[59] D. A. Patterson and J. L. Hennessey, *Computer Architecture A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1996.