2005

Optimization of a Parallel Cordic Architecture to Compute the Gaussian Potential Function in Neural Networks

Nanditha Chandrasekhar

THE FLORIDA STATE UNIVERSITY

COLLEGE OF ENGINEERING

OPTIMIZATION OF A PARALLEL CORDIC ARCHITECTURE TO COMPUTE THE GAUSSIAN POTENTIAL FUNCTION IN NEURAL NETWORKS

By

NANDITHA CHANDRASEKHAR

A Thesis submitted to the
Department of Electrical and Computer Engineering
in partial fulfillment of the
requirements for the degree of
Master of Science

Degree Awarded:
Spring Semester, 2005

The members of the Committee approve the thesis of Nanditha Chandrasekhar defended on 7th of April 2005.

Anke Meyer Baese
Professor Directing Thesis

Uwe Meyer Baese
Committee Member

Simon Foo
Committee Member

Approved:

Leonard J. Tung Chair, Electrical and Computer Engineering

Ching-Jen Chen, Dean, College of Engineering

The office of the Graduate studies has verified and approved the above named committee members.

Dedicated to my Parents,

Chandrasekhara and Kamala Prabha.

ACKNOWLEDGEMENT

With a deep sense of gratitude I wish to express my sincere thanks to all who helped me to complete my thesis. I am deeply indebted to my advisor Dr. Anke Meyer Baese whose constant suggestions and enlightening guidance was of valuable help in the time of research for and writing this thesis.

I want to thank Dr. Uwe Meyer Baese for his unconditional support in directing me during the research on my thesis and Dr.Simon Foo for all the help he has offered during the course of my studies.

I would like to thank all the faculty members and staff of the Electrical and Computer engineering department for their support during my entire graduate studies.

I would also like to thank my family for their constant support, love and encouragement.

TABLE OF CONTENTS

LIST OF ACRONYMS

AM: Auto Association Memory

ANN: Artificial Neural Networks

BAM: Bidirectional Associative Memory

CORDIC: Coordinate Rotation Digital Computer

FSM: Finite State Machine

GPFU: Gaussian Potential Functional Unit

HSOL: Hierarchically Self Organizing Learning

LC: Logic Cells

MAC: Multiplication and Accumulation

MLP: Multi Layer Perceptron

MNN: Mapping Neural Network

PFU: Potential Function Unit

PSP: Post Synaptic Potential

RBF: Radial Basis Function

RBNN: Radial Basis Neural Network

VHDL: VHSIC Hardware Description Language

VHSIC: Very High Speed Integrated Circuit

VLSI: Very Large Scale Integration

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Many pattern recognition tasks employ artificial neural networks based on radial basis functions. The statistical characteristics of pattern generating processes are determined by neural networks. The Gaussian potential function is the most common radial basis function considered which includes square and exponential function calculations.

The Coordinate Rotations Digital Computer, CORDIC algorithm which is used to compute the exponential function and the exponent was first derived by Volder in 1959 for calculating trigonometric functions and conversions between rectangular and polar co-ordinates. It was later developed by Walther, the CORDIC is a class of shift-add algorithms for rotating vectors in a plane. In a nutshell, the CORDIC rotator performs a rotation using a series of specific incremental rotation angles selected so that each is performed by a shift and add operation.

This thesis focuses on implementation of new parallel hardware architecture to compute the Gaussian Potential Function in neural basis classifiers for pattern recognition. The new hardware proposed computes the exponential function and the exponent simultaneously in parallel thus reducing computational delay in the output function. The new CORDIC is synthesized by Altera's MAX PLUS II software for FLEX 10 K device and improvised for calculation of Radix 4. Case studies are presented and compared on the performance of Radix 2 and Radix 4 design based on the speed and the size occupied respectively. It is observed that though the area occupied by Radix 4 is more as compared to Radix 2 there is speed improvement which is desirable.

INTRODUCTION

A more superior method to the traditional neural networks - perceptron, in terms of interpolation, processing speed and pattern analysis and classification is the radial basis neural networks (RBNN).Pattern recognition tasks utilize artificial neural networks based on radial basis functions.

Implementation of Radial basis network can be done by two methods:

The distance between an input vector and each of the stored prototypes can be calculated sequentially in the serial method.
The parallel method is based on parallelism in neural computing where the distance between an input vector and each of the stored prototypes can be calculated at the same time.

The Radial basis neural network requires more memory and an exponential unit e^x computed by a paradigm based on an optimized CORDIC co – processor.

Thus the statistical characteristics of a pattern generating process is estimated by neural networks based on the Gaussian potential function, a most commonly used radial basis function.

The Gaussian potential function is defined as:

$$\phi(x) = e^{1/2.(x-\mu)^T C^{-1}(x-\mu)}$$

x denotes the feature vector of the pattern and C is the covariance matrix. A maximum is assumed by the Gaussian at the center μ and continuously falling with increasing distance, bounded by 0. Hence a relation is established between the hidden Gaussian potentials (GPFUs) and the training data. A fast digital GPFU is designed to compute the Gaussian potential for an overall classifier as compared to the existing efficient other learning algorithms.

Hence C, a covariance matrix is considered in the form $E.\frac{1}{2}\sigma^2$, where E is the unit matrix and σ is the standard deviation. Hence the exponent of the GPFU is considered as:

$$\varnothing(x) = \sum_{n=1}^{N} k_n (x_n - \mu_n)^2$$

where $K_n = \frac{1}{2}\sigma_n^2 \geq 0$. Thus based on the learning algorithms the centers μ_n and the coefficients k_n are calculated. Initially an approximation of the exponential function is computed, and then the square of a number is computed by separate units. The summations of the above equation are processed in parallel and fed bit by bit into the exponential unit.

CHAPTER 1

Neural Networks

The human brain is a non-linear, complex parallel computer able to perform computations such as motor control, pattern recognition by organizing its structural constituents, the neurons. This has motivated work on artificial intelligence proving that the human brain functions differently from the conventional digital computer. Extended research on artificial intelligence has led to neural networks a network, implemented by electronic components or simulated by software is used to design a machine, which models the way in which the brain performs a particular task.

Biological Aspect of Neural Networks

The brain consists of a large number of interconnected neurons, capable of propagating an electrochemical signal. The neuron has a branching input and output structure and a cell body. The output structure (axon) of one cell is connected to the input structure (dendrites) of another cell by a synapse. A neuron is activated if the total signal received at the cell body from the dendrites exceeds a threshold value and releases an electromagnetic signal along the axon.

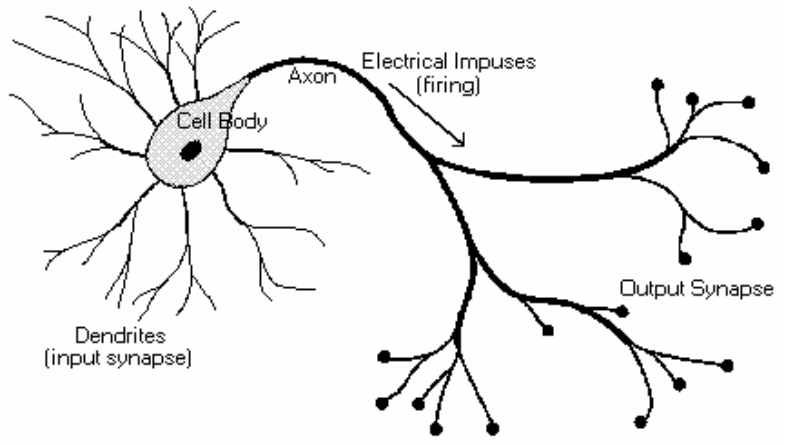

Fig: 1.1 The Biological neuron

Basic Artificial Model

An artificial neuron consists of a number of inputs from the original data or from an output of another neuron in a neural network. These inputs are based on a weight corresponding to synaptic efficacy in a biological neuron and each neuron has a single threshold value. The activation of the neuron (post synaptic potential, PSP) is composed by the weighted sum of the input subtracted from the threshold; this activation signal is passed through an activation function to produce the output of the neuron.

A neural network is defined as a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:

1) Knowledge is acquired by the network from its environment through a learning process
2) Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

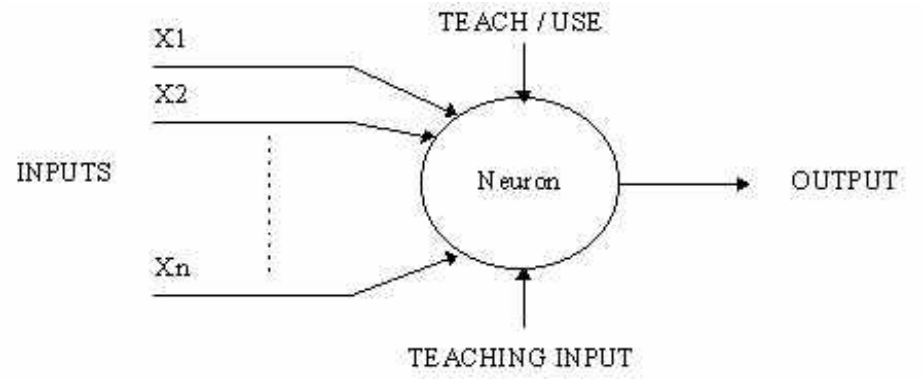

Fig 1.2: A simple neuron

Artificial Neural Networks

Artificial neural networks unlike traditional computing are considered as computational paradigms based on mathematical models having a structure and operation similar to that of the human brain.

Artificial neural networks or neural networks are composed by a series of interconnected processing elements that operate in parallel and hence are also called as connectionist systems, parallel distributed systems or adaptive systems.

The main aim of artificial neural networks was to understand and shape the functional characteristics and the computational properties of the brain when it performs cognitive processes such as sensorial perception, concept categorization, concept association and learning.

Model for Artificial Neural Network

A set of highly interconnected processing elements called neurons which process information as a response to external stimuli is defined as a generic artificial neural network. The stimuli are transmitted from one processing element to another via a synapses or interconnections which can be excitatory or inhibitory.

Modes of behaviour

Learning (training) and testing are considered as the two modes in which an artificial neural network performs. During the training process, the network guesses the output for each example, and then the network modifies itself internally until a stable state is reached at which the outputs proposed are satisfactory. Hence learning is an adaptive process during which the weights associated to all the interconnected neurons change in order to provide the best possible response to all the observed stimuli. Neural networks learn in two ways, supervised and unsupervised learning.

Supervised Learning

The aim of supervised learning is to be able to teach the network to identify the given input with the desired output. Hence the network trains a set of input-output pair. The network receives an input and produces an actual output which is compared with the desired output after each trial and if any difference exists between the desired output and the actual output, it is rectified by adjusting the weights.

Unsupervised Learning

In unsupervised training the network is trained using the input signal only. Thus the network is internally organized to produce an output that is consistent with particular stimulus or group of similar stimuli. Clusters are formed by the inputs in the input space where each cluster represents a set of elements of the real world with some common features.

Structure of Artificial Neural Networks

Neural networks are arranged in layers, where each layer in a layered network is an array of processing elements or neurons. Here information flows from the input to the output through hidden layers, these three layered networks are called as multi layer perceptron (MLP).The input layer receives the external stimuli and propagates it to the next layer, the hidden layer which processes these input units by means of activations functions. The hidden units send an output signal towards the neurons in the next layer, the output layer. Information is continuously propagated forward until an output is produced.

Network Architectures

The learning algorithms used to train the network in neural networks are structured (network structures), two network architectures are identified.

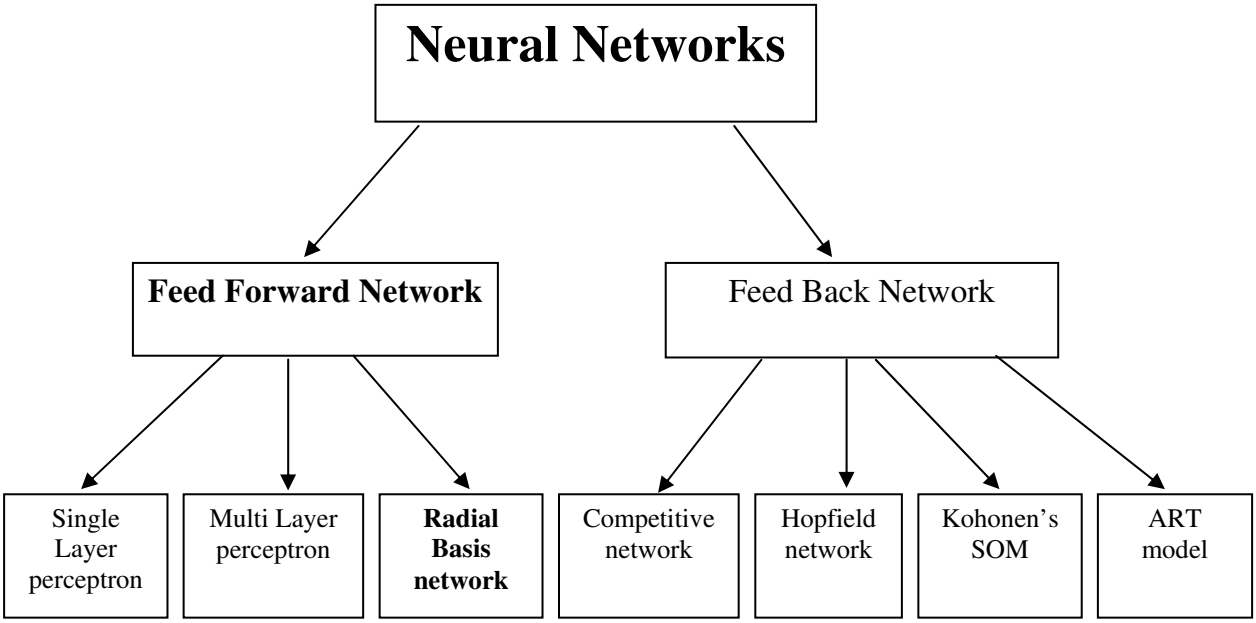

Fig 1.3 Artificial Neural Networks

Feed Forward Network

Feed forward networks are straight forward networks allowing signals to travel only in one way, i.e., the perceptrons are arranged in layers with the first layer taking in an input and the last layer producing an output, thus information is constantly "fed forward" from one layer to the next. There is no sense of time or memory of previous layers. Feed forward networks are also called as bottom-up or top-down networks.

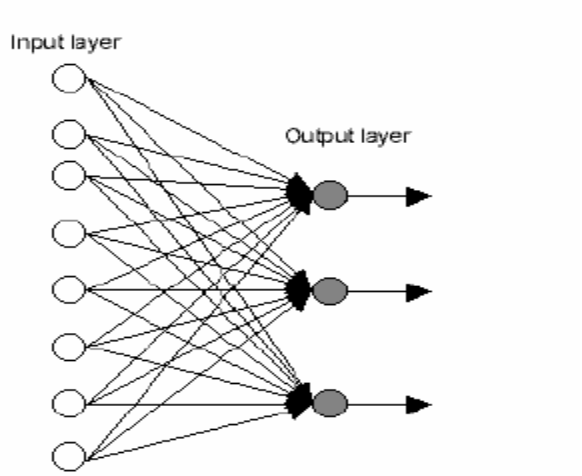

Fig 1.4 Simple feed forward network

Radial Basis Neural Network (RBNN)

It is a three layer network as shown in fig 1.5; the input layer is a fan-out and does not perform any processing. The patterns achieve linear seperability when the hidden layer performs a non linear mapping from the input space into a higher dimensional space. The output layer, the last layer calculates the weighted sum.

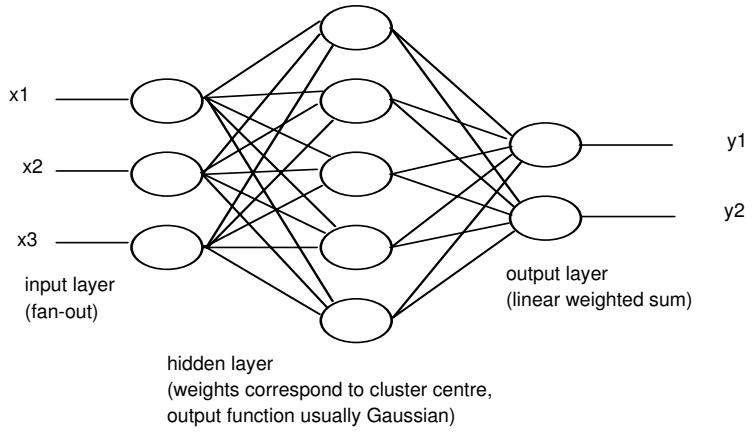

Fig: 1.5 Radial Basis Neural Network

A unique feature of RBF is the formation of clusters by the patterns in the input space. If the center of the cluster is known, then the distance from the cluster center can be determined, the distance measured is nonlinear, so that if a pattern is in an area close to the cluster center the value determined is 1, and beyond this area the value drops. Thus the area is radically

symmetrical around the cluster center so that the non linear functions are known as the radial basis function.

Thus the RBNN (radial basis neural network) is based on the radial basis function (RBF) and is of the form $\phi = (|| x - \mu ||)$ where ϕ denotes the non linear activation scalar function of the distance between the two vectors x and μ.

The Gaussian function is the most commonly used radial basis function represented as $\phi(x) = e^{1/2.(x-\mu)^T C^{-1}(x-\mu)}$ where x denotes the feature vector of the pattern and C is the covariance matrix. A maximum value is assumed by the Gaussian at the center μ and continuously falling with increasing distance, bounded by 0.

The hidden layer in the RBF network has units which have weights corresponding to the vector representation of the center of the cluster. These weights are determined by algorithms such as the K-means algorithm or the Kohonen algorithm. The radius of the Gaussian curves is determined by the P nearest neighbor algorithm. The output layer is trained using a standard gradient descent, the least mean square algorithm.

Radial basis neural network (RBNN) can be implemented in two ways by a digital computer. In the parallel method, parallely the distance between the input vector and the each stored prototyped is calculated at the same and can be performed by the Intel Ni 1000 chip time. In the serial method the distance is calculated sequentially thus having the lowest power consumption and least silicon area as compared to the parallel method. Here a parallel implementation of the RBNN based on an optimized CORDIC is considered.

Advantages of a RBNN over a MLP (multi- layer perceptron) are that the RBN trains faster and the hidden layer are easier to interpret.

Recurrent Networks (Feed back networks)

In a recurrent neural network the signal is transported back to the network, thus there exists a feedback loop. The learning capability and the performance of the network are improved due to the presence of the feedback loops. The recurrent networks can be self- feedback, where the output of a neuron is fed back to its own input or without self feedback too.

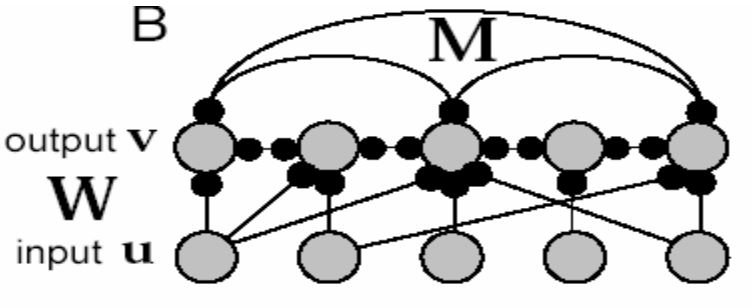

Fig 1.6 A Recurrent network

Advantages of Neural Networks

- Adaptive learning: capability to learn to perform tasks based on the given data for training or initial experience.
- Mapping: As neural networks are based on learning algorithms, by training samples. Thus these samples consist of a unique input signal and a corresponding desired response or output. Training of the network is repeated for many examples, until a stable state is reached by the network. Hence an input –output map is constructed based on the examples by which the network learns the examples.
- Real time operation: It is advantageous as artificial neural networks can be performed parallelly.

- Fault tolerance: Partial destruction of a network leads to degradation in performance, however the network has the capability to retain some of the features.
- VLSI implement ability: Due to neural networks massive parallelism nature they are implemented on VLSI.(Very Large Scale Integration)
- Neurobiological analogy: Used as a research tool for interpretation of neurobiological phenomena.
- Noise reduction: Capable of recognizing various patterns in the input and thus produce a noiseless output.

Applications of Neural Networks

- **Neural network in practice:**
  Neural Networks have found a broad application in real world business problems as neural networks are capable of identifying patterns or trends suited for prediction or forecasting for e.g., sales forecasting, data validation, risk management, target marketing.
- **Neural network in medicine:**
  Neural networks are capable of recognizing diseases based on scans as they learn by examples. Research is underway by modeling various parts of the human body and recognizing diseases from the scans (e.g. ultrasonic scans, cardiograms, CAT scans)
- **Neural networks in business:**
  Neural networks are used in resource allocation and scheduling, and data mining where implicit patterns are searched explicitly the stored information in the database. Thus neural networks are used in marketing and credit evaluation too.

CHAPTER 2

The Gaussian Potential Function Network (GPFN)

An input – output mapping got by efficient training of a given set of teaching patterns can be used to evaluate an artificial neural network which depends on selection of network configuration, network architecture, number of neurons, type of activation functions, and capability of a learning algorithm. A learning algorithm capable of self configuring a neural network by nonsigmoidal activation functions such as Gaussian, sinusoidal or sigmoidal and recruiting neurons itself by hierarchical learning is considered as compared to most artificial networks which are trained on parameters selected by the designer.

The Gaussian Potential Function Network(GPFN) a, nonsigmoidal MNN (Mapping Neural Network), is considered for arbitrary mapping, as the GPFN can approximate a many –to –one continuous function by a potential field synthesized over the domain of the input space by a number of Gaussian Potential Function Units (GPFU) based on Hierarchically Self Organizing Learning (HSOL) capable of selecting the necessary GPFUs based on hierarchical learning, by continuously adjusting the effective radii of individual GPFUs in the input domain.

A MNN (Mapping Neural Network) is a network performing mapping, ϕ from a set, I^n to an m dimensional Euclidean space, $R^m, \phi : I^n \rightarrow R^m$, based on the interconnection of neurons as basic non linear computational units in a parallel and distributed manner. Thus the artificial networks can be defined based on a MNN as:

1) **Associative Memory**: Here in an addressable memory an input – output mapping is stored where the network concurs to the stored memory depending on the dynamics embedded in the network structure. Example: Auto association memory(AM), Bidirectional Associative memory(BAM), Boltzman machine
2) **Competitive Network**: An output pattern is generated as the reference pattern of the chosen neuron based on the shortest distance between the reference patterns, represented by individual patterns and the given input pattern. Example: Hamming Network, Self-organizing feature map, Counter propagation network.
3) **Multilayer feed forward network**: An arbitrary input-output mapping is realized by reconstructing the input domain into progressively complex non linear manifolds in the upper layers through a series of interposed mappings. Example: Backpropagation Network, Neocognitrion, Athena.

Potential Function Network

A weighted sum of finite number of potential functions can symbolize a discriminant function, ϕ as

$$\phi(x) = \sum_{i=1}^{M} c_i k(x, x_i) \qquad (1)$$

$k(x, x_i)$ is the ith potential function of x got by shifting K(x, 0) by x_i

c_i - real constant

It is observed that at $x = x_i$, the potential function $k(x, x_i)$ has maximum value and as $\| x - x_i \|$ approaches infinity $k(x, x_i)$ monotonically decreases to zero.

Considering binary classification on equation (1), a learning algorithm has been proposed:

$$\phi^{new}(x) =$$

$$\begin{cases} \phi^{old}(x) + k(x, x_k) if x_k is labelled +1 \\ and \phi^{old}(x_k) \le 0 \\ \phi^{old}(x) - (x, x_k) if x_k is labelled -1 \\ and \phi^{old}(x_k) \ge 0 \\ \phi^{old}(x) otherwise \end{cases} \qquad (2)$$

It is observed that equations (1) and (2) are similar to the non parametric estimation of a probability density function, Parzen window.

Here from the given input samples, $x_i s$ i=1 ……, n a probability density function is considered

$$p_n(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h_n^d} \psi \left(\frac{x - x_i}{h_n} \right) \qquad (3)$$

ψ - a bounded negative kernel function of the d dimensional input vector x,

h_n -a sequence of positive numbers

Considered to the number of input samples, the number of potential functions (kernel functions) needed for the implementation of an unknown function is very large. Thus a general form of equation (1) and (3) is considered by adjusting the shape parameters and self recruitment of potential functions as below:

$$\phi(x) = \sum_{i=1}^{M} c_i \psi(x, p_i) \qquad (4)$$

M- number of potential functions to be recruited

c_i - summation weight

p_i - a parameter vector representing the position shift and the shape parameters of the ith potential function

Here as a learning algorithm is developed for self recruiting or considering the necessary number of potential functions and for training the shape parameters and the summation weights, an activation function is needed.

Equation (4) represents a general form of a multilayer feed forward network, where an output vector of the input layer, x is directly used as an input vector to a hidden unit. Hence based on the condition that ψ is integrable, an arbitrary function can be approximated as below:

Consider $\psi(x)$ represented by N discrete samples points, $\varphi(x_1)$,….. $\varphi(x_N)$.

Then Z= [$\varphi(x_1)$,…., $\varphi(x_N)^T$

$$c = [c_1, c_2 ....., c_m]^T$$

$$y_i = [\psi(x_i, p_i), \psi(x_2, p_i) ....., \psi(x_N, p_i)]^T$$

z =Yc, represents the discrete form of equation (12) where Y= [y_1, y_2....., y_m]: NxM matrix.

When M≥N, an exact solution for c is got but it is an impracticable case as N value should be large to minimize errors.

When M<N, an over determined set of equations are observed, thus rank of [Y: z] =M, produces an exact solution for c which represents the state when z is fixed in the subspace, S_y represented by $y_1, ...y_m$ where z, y_i, i = 1.Thus there exists no exact solution for c, but a solution c^* exists, which minimizes the error as shown below:

$$c^* = Y^+ z \qquad (5)$$

$$E_{\min} = || (I - P) z ||^2 \qquad (6)$$

Y^+ - generalized inverse of Y

By accommodating p_i, the value of y_i is got such that z is fixed in S_y, hence a trajectory $R(y_i)$ of y_i is produced which is the same for all i, i=1…., M, if all the activation functions are of the

same form. M vectors can be formed by arbitrarily selecting M points from $R(y_i)$ with M PFUs. Thus a linear manifold $L(Y^k)$, is defined by the linear combination of the selected M vectors in S_N which proves a theorem stating that:

Theorem:

$z \equiv [\phi(x_1), \phi(x_2), ..., \phi(x_N)]^t$ is exactly realizable if $z \varepsilon \bigcup_{k \varepsilon K} L(Y^k)$ where K is a finite or infinite index set representing all the possible selections of M points from R(y_i).

Considering a basic network with three training samples $\phi(x_1), \phi(x_2), \phi(x_3)$, two PFUs $\psi(x, p_1)$ and $\psi(x, p_2)$ and analyzing the capability of z, $z \equiv [\phi(x_1), \phi(x_2), ..., \phi(x_N)]^T$ with y_1, $y_1 = [\psi(x_1, p_1), \psi(x_2, p_1), \psi(x_3, p_1)]^T$ and y_2, $y_2 = [\psi(x_1, p_2), \psi(x_2, p_2), \psi(x_3, p_2)]^T$ based on $z = [y_1, y_2]$ c.

By simulating the above for three cases, sigmoidal, Gaussian and sinusoidal PFUs, it is noted that the sinusoidal activation provides the best mapping and the Gaussian provides better mapping than the sigmoidal activation function. But the Gaussian activation function is chosen for a MNN with self recruitment as it generalizes a global mapping and also has the capability of promoting the already learned mapping without many changes.

Gaussian Potential Function Network

The building up of a potential function is performed by an un-normalized form of Gaussian density function, the Gaussian potential.

Gaussian potential function is defined as

$$\psi_i = \psi(x, p_i) = e^{-d(x, p_i)/2}$$

$$d(x, p_i) = d(x, m^i, k^i) = (x - m_i)^i K^i (x - m^i)$$

where

x – is input pattern

m^i - is mean vector

k^i - is shape matrix

Hence a three layer network model is considered, the input layer and output layer comprising of linear inputs and the hidden layer comprising of Gaussian potential function units.(GPFU) capable of producing Gaussian potential functions. Thus the GPFUs in the hidden layer combined with the three layer PFN comprises the Gaussian potential function network (GPFN)

CHAPTER 3

COORDINATE ROTATIONS DIGITAL COMPUTER (CORDIC)

Microprocessors with features such as single cycle multiply-accumulate instructions and special addressing modes are used widely in the field of digital signal processing. It is observed that algorithms optimized for these microprocessors do not compliment well with hardware as DSP algorithms also require MAC (multiplication and accumulation) units. Sophisticated DSP algorithms are used to enhance the performance for modern digital signal processing systems thus reducing the hardware costs. These sophisticated algorithms require the evaluation of elementary functions, such as trigonometric, exponential and logarithm functions which cannot be evaluated efficiently with MAC (multiplication and accumulation) units. Hence an alternative algorithm computing algorithm CORDIC (coordinate rotation Digital Computer) is used.

A class of iterative solutions used for trigonometric and other transcendental functions that perform based on only shifts and adds is one of the hardware-efficient algorithms used. CORDIC, an acronym for Coordinate Rotational Digital Computer is one such hardware – an efficient trigonometric algorithm, where the trigonometric functions are based on vector rotations.

CORDIC ALGORITHIMS

A multiplier free coordinate conversion between rectangular (x, y) and polar (R, θ) co-ordinates is performed by the original CORDIC algorithm by Volder. The CORDIC algorithm was generalized by Walther who introduced circular (m = 1), linear (m = 0) and hyperbolic (m = -1) transforms. Two rotation directions – vectoring and rotation, are identified for each of these modes.

In the vector rotation mode or vectoring mode or forwarding mode , a vector with starting co-ordinates (Xo, Yo) is rotated in such a way that the vector finally lies on the abscissa by converging Y_k iteratively to zero.

In the angle accumulation mode, rotation mode or Y – reduction mode or backward rotation mode, a vector with starting co-ordinates (Xo, Yo) is rotated by an angle θ_0 in such a way that the final value of the angle register, denoted by Z converges to zero. The angle θ_k is selected so that each iteration can be performed with an addition and a binary shift.

The CORDIC algorithm is defined by the following equations:

$$\begin{bmatrix} X_{K+1} \\ Y_{K+1} \end{bmatrix} = \begin{pmatrix} 1 & m\delta_K 2^{-K} \\ m\delta_K 2^{-K} & 1 \end{pmatrix} \begin{bmatrix} X_K \\ Y_K \end{bmatrix}$$

$$Z_{K+1} = Z_K + \delta_K \theta_K$$

$$Z_K \to 0, \quad Y_K \to 0$$

The CORDIC algorithm modes are defined as:

Table 3.1 CORDIC algorithm modes

Modus	Angle θ_k	Shift sequence	Radius- Factor
Circular m = 1	$\tan^{-1}(2^{-k})$	0, 1, 2……..	$K_1 = 1.65$
Linear m = 0	2^{-k}	1, 2….	$K_0 = 1.0$
Hyperbolic m = -1	$\tanh^{-1}(2^{-k})$	1, 2, 3, 4……	$k_{-1} = 0.80$

Hence it is observed that there are six operational modes as below:

Table 3.2 Modes m of operation for the CORDIC algorithm

m	$Z_k \to 0$	$Y_k \to 0$
1	$X_K = k_1(X_0\cos(Z_0) - Y_0\sin(Z_0))$ $Y_K = k_1(X_0\cos(Z_0) + Y_0\sin(Z_0))$	$X_K = K_1\sqrt{X_0^2 + Y_0^2}$ $Z_K = Z_0 + \arctan(Y_0 / X_0)$
0	$X_K = X_0$ $Y_K = Y_0 + X_0.Z_0$	$X_K = X_0$ $Z_K = Z_0 + Y_0 / X_0$
-1	$X_K = k_{-1}(X_0\cosh(Z_0) - Y_0\sinh(Z_0))$ $Y_K = k_{-1}(X_0\cosh(Z_0) + Y_0\sinh(Z_0))$	$X_K = K_{-1}\sqrt{X_0^2 + Y_0^2}$ $Z_K = Z_0 + \tanh^{-1}(Y_0 / X_0)$

Nearly all transcendental functions can be computed with the CORDIC algorithm. By proper selection of the initial values, functions such as X, Y, Y/X,
sin (Z), cos (Z), "tan inverse(Y),"sinh (Z), cosh (Z), tanh (Z) can be computed easily. Also by choosing appropriate initialization, combined with multiple modes of operation, additional functions as shown below can be determined.

$\tan(Z) = \sin(z)/\cos(z)$, Modes: m =1, 0

$\tanh(Z) = \sinh(z)/\cosh(z)$, Modes: m =-1, 0

$\exp(Z) = \sinh(z) + \cosh(z)$, Modes: m =-1, x = y =1

$\log_e(W) = 2\tanh^{-1}(Y/X)$, Modes: m =-1 with X = W+1, Y=W-1

$\sqrt{W} = \sqrt{X^2 - Y^2}$, Modes: m =1 with X = W+ ¼, Y= W-1/4

It is observed that with every iteration the length of the vectors changes as in Fig: 3.1(b) After K iterations the same change in length of the vector occurs and this does not depend on the starting angle.

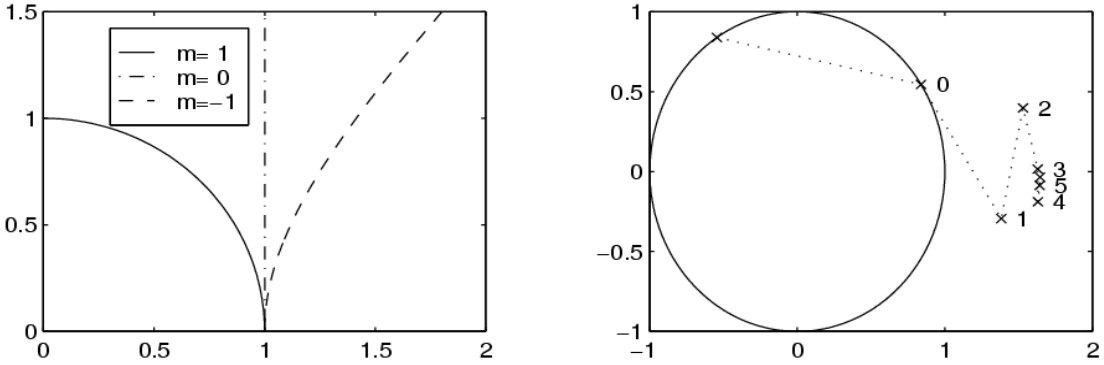

Fig: 3.1 CORDIC. (a) Modes (b) Example for circular vectoring

In Linear and Circular transforms, for the CORDIC algorithm to converge, the sum of all remaining rotation angles must be larger than the actual rotation angle. All iterations, $n_{k+1} = 3$ n_{k+1} have to be repeated for the hyperbolic mode.

A procedure developed by Hu (1992 u) gives an estimation on the output precision as shown in Fig: 3.2

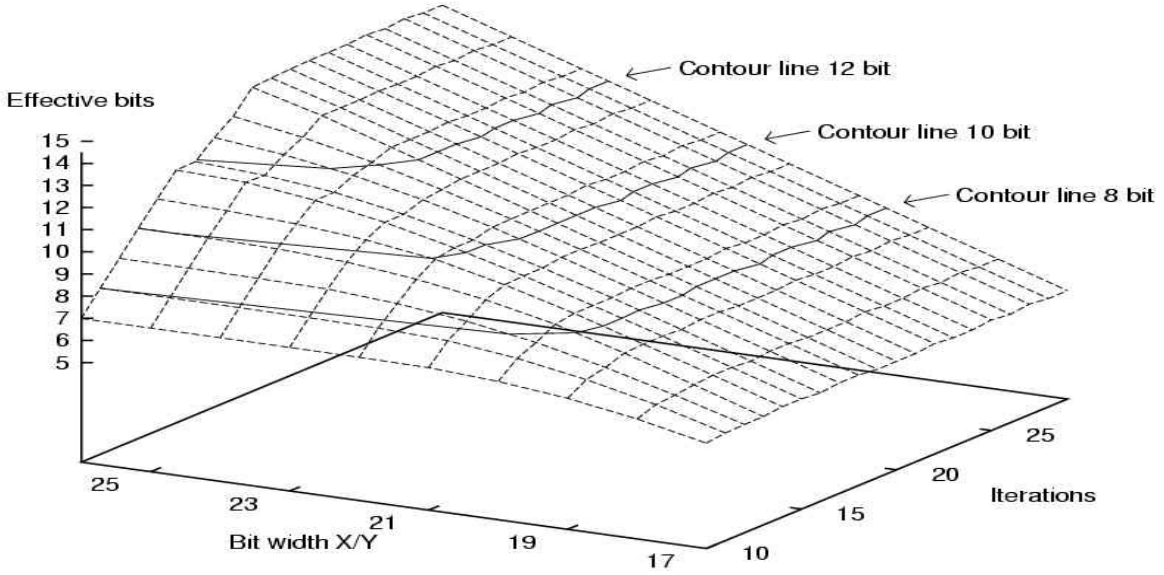

Fig: 3.2 Effective bits in hperbolic mode

Depending on the number of iterations and the X, Y path width, the bit precision for the circular mode is observed. If the desired output precision is b bits then the X, Y path should have $\log_2(b)$ additional guard bits.

In Hyperbolic CORDIC, the precision depends on the angular values of z(k) at iteration k and hence the effective resolution cannot be computed analytically. With the help of simulations the hyperbolic precision can be computed.

CORDIC ARCHITECTURES

The fully pipelined processor and the compact state machine are the two basic structures used to implement a CORDIC architecture.

If the required criteria are high speed then, a fully pipelined version as in Fig: 3.3 is used. A circular CORDIC with eight iterations is observed where x, y are the co-ordinates and θ is the rotation angle. A new output value is available after each cycle, after an initial delay of 'K' cycles.

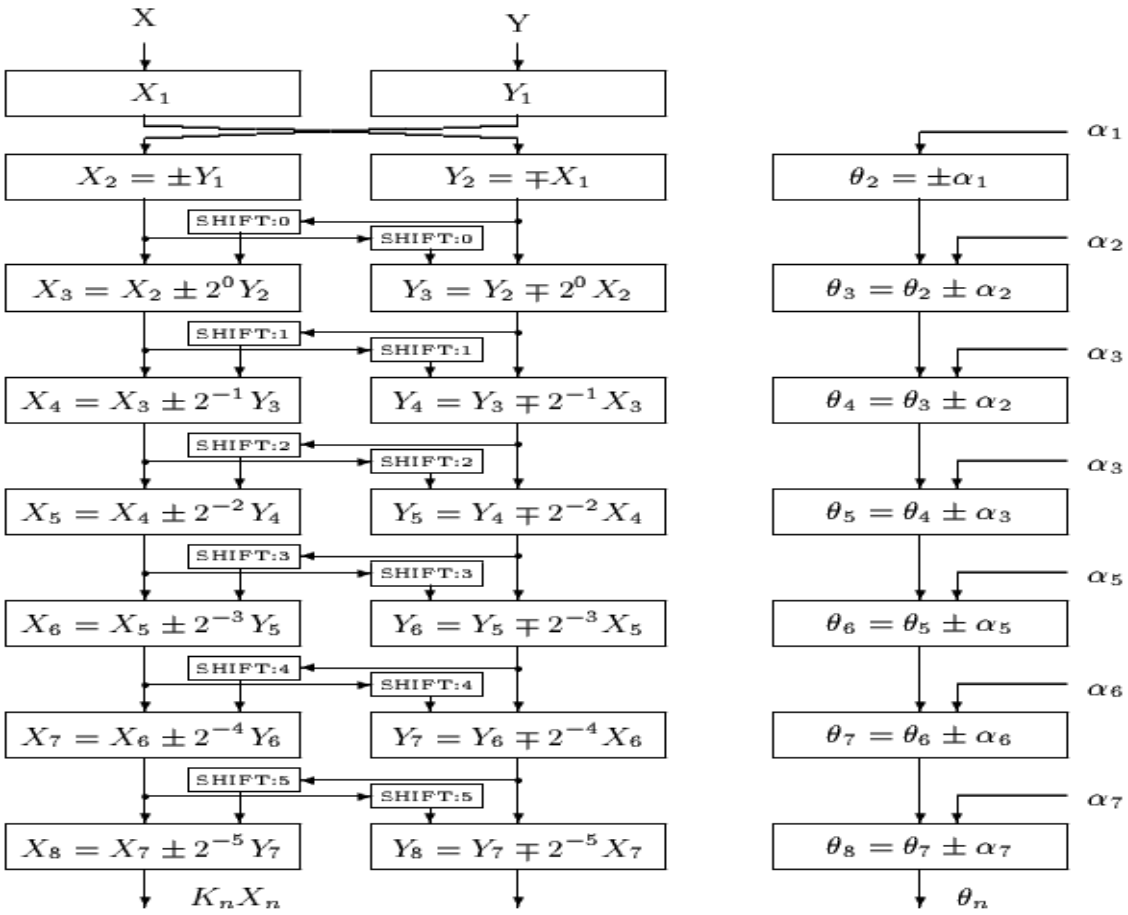

Fig: 3.3 Fast CORDIC pipelines

If the required criteria is chip size then a state machines as shown in Fig 3.4 is used. Only one iteration is computed in each cycle. As the two barrel shifters are the most complex part of the design, they can be replaced by a single barrel shifter using a multiplexer or a serial shifter.

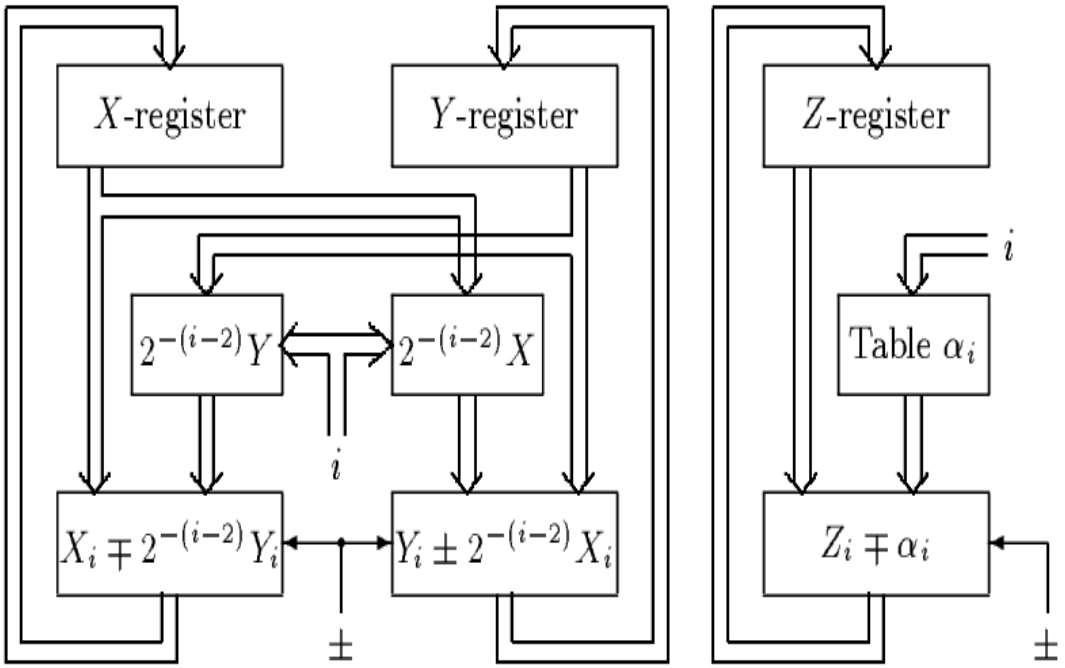

Fig: 3.4 CORDIC state machine

The New CORDIC architecture

Pattern recognition tasks utilize artificial neural networks based on radial basis function such as the Gaussian potential function:

$$\psi(x) = e^{1/2.(x-\mu)^T C^{-1}(x-\mu)}$$

x – Feature vector of the pattern presented

C – Co-variance matrix

A relation is observed between the hidden Gaussian potential function units (GPFU) and the training data as the Gaussian assumes a maximum at the centre μ and continuously decreases as the distance increases. Also the Gaussian is preferred to be used as the probability density of the pattern generating process can be structured by a linear combination of Gaussian normal distributions. There are numerable learning algorithms which allocate GPFUs, hence a digital GPFU is designed to compute the Gaussian potential function.

Considering that C is in the form of $E.\frac{1}{2}\sigma^2$, E – unit matrix and σ - standard deviation, the computing time can be reduced. If the above parameters are not related then the exponent of a GPFU is considered as:

$$\varphi(x) = \sum_{n=1}^{N} K_n \ (x_n - \mu_n)^2$$

K_n = ½ $\sigma_n^2 \geq 0$. Thus individual scaling for each factor can be done. Based on a learning algorithm the centers μ_n and the coefficients K_n are calculated. Initially an approximation of the exponential function is computed in one unit, and then later the square of the number is computed in another unit. All the summations of the above equation are parallel processed and fed bit by bit into an exponential function unit.

The Exponential Function Unit

A look up table can be used to implement the exponential function. But to calculate y = f(x) where x and y are numbers of length n and m bits, we require a table or ROM of size 2^n x m. Thus for n ≥ 20, the size is ≥ 16M.

Also the Taylor equation: $f(x_0) = \sum_{k=0}^{k} \frac{df^{(k)}(x - x_0)}{dx} x^k$ at $x = x_0$ can be used where the exponential function unit is reduced to a sequence of multiply and add operations. Though the Taylor series converges rapidly for small fractional arguments x, when x is close to one a number of steps are needed.

Thus a more efficient algorithm, the CORDIC algorithm is used for computing the exponential function.

The original CORDIC algorithm discussed previously is disadvantageous, as this algorithm requires knowledge of the complete exponent of the first step when implemented for online computations. It also requires logarithms tables or arctanh functions. Thus an algorithm where each iteration requires only 1 bit of the exponent is considered. This algorithm does not require any logarithm table but a shift control table.

The Binary representation of the exponent:

$$e^{-\beta} = e^{-\sum_{n=-p}^{M} b_n 2^{-n}} = \prod_{n=-p}^{M} e^{-b_n 2^{-n}} = \prod_{n=-p, b_{n=1}}^{M} F_n \qquad (7)$$

The exponential factors with terms $F_{-1} = 2^{-3} + 2^{-7}$ $b_n = 0$ are removed from the product as $e^0 = 1$.

For $n \geq 1$, $F_n = e^{-2^{-n}}$ is expressed using the Taylor series:

$$F_n = e^{-2^{-n}} = 1 - 2^{-n} + 2^{-(2n+1)} \ldots\ldots \qquad (8)$$

For $n \leq 0$, the precalculated values are used:

Table 3.3 Precalculated values for n

n	F_n (decimal)	F_n (Binary)
-2	0.01831564	000.0000010011
-1	0.13233528	000.0010001011
0	0.36787944	000.0101111001

With two terms the results in the above table can be approximated as follows:

$$F_{-2} = 2^{-6} + 2^{-8}$$

$$F_{-1} = 2^{-3} + 2^{-7} \qquad (9)$$

$$F_0 = 2^{-2} + 2^{-3} \qquad (10)$$

The multiplication term in equation (7) is implemented using hardware as shown below, as it comprises of a few shift and addition operations.

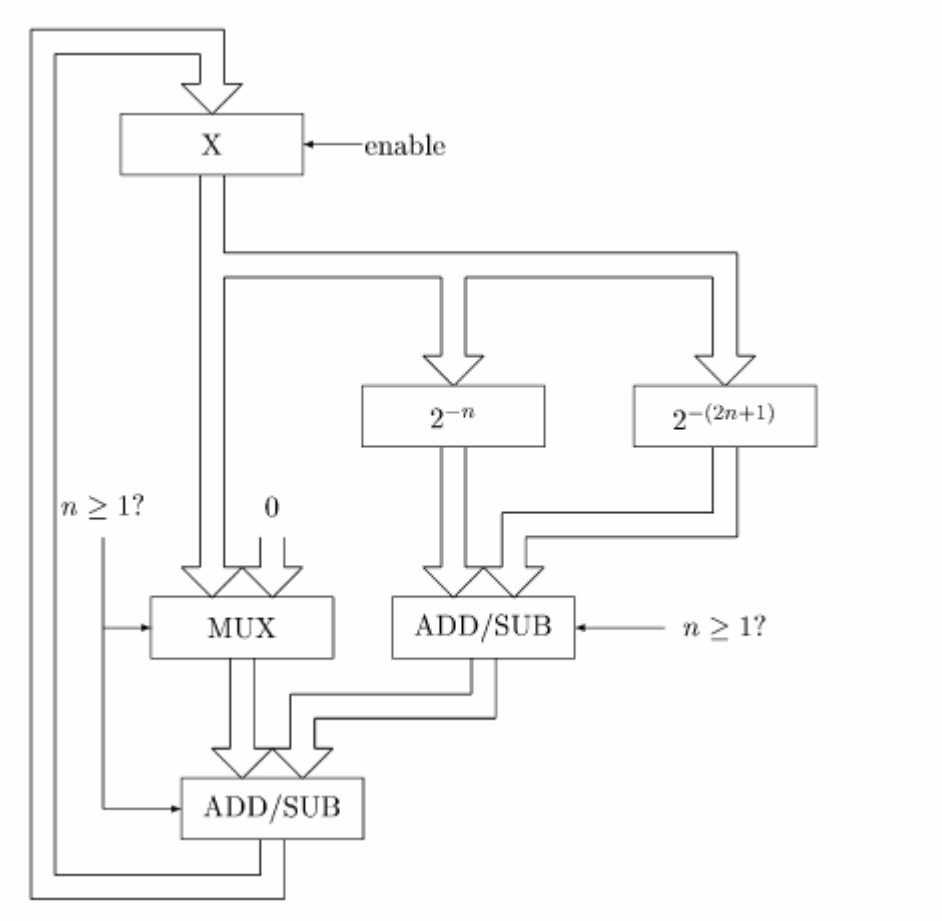

Fig: 3.5 Hardware to calculate the exponential function

The register X comprises of $X = \sum_{n=0}^{M} X_n 2^{-n}$ which is initialized with $X, X_0 = 1, X_n = 0$ for n>0.

The input bit b_n controls the enable signal of the register X in the n-th iteration, hence if b_n =0, the register does not perform any action.

If $n \geq 1$, X is selected by the multiplexer and the operation subtraction performed with the shifts derived from equation (8).

If $n \leq 0$, 0 is selected by the multiplexer and $\delta(i)$ the operation addition is performed with the shifts derived from a shift control table using equations (9) and (10).

For various pattern recognition tasks in neural networks, accuracy derived for the CORDIC approximations from three terms is considered sufficient. Accuracy can be improved by adding additional shifters to the structure.

The exponent can be processed from the least significant bit (LSB), as the product of equation (7) is commutative and hence the order of processing of the exponent bits is arbitrary.

Advantages of the new algorithm over the "lean CORDIC "architecture

- A table for $\theta(i)$ or $\beta(i)$ is not needed as the whole z path is reduced to a single register.
- The steps of the modified CORDIC can be easily determined by the binary representation of the exponent.
- If the barrelshifter is addressed appropriately, the bits of the exponent can be processed in any sequence.
- As only one bit is needed at a time, the computation of the exponential function can be performed concurrently with the computation of the exponent.
- All operations can be directly applied to the sign vector $\delta(i)$, when a GPFU has been performed onto the exponent.

Disadvantages of the new algorithm over the "lean CORDIC "architecture

- In this case an additional Barrel shifter and Adder/subtractor have to be implemented.
- The computation of the exponential functions is sufficient for the GPFU computations though it contains more errors than the "lean CORDIC" when three terms are used.

CHAPTER 4

RESULT AND CONCLUSION

Hardware Implementation

Serial Addition

The most used radial basis function, Gaussian potential function has been proved to be very useful in pattern recognition tasks. The probability density of the pattern generating process is based on a linear combination of Gaussian normal distributions. Efficient learning algorithms automatically allocate GPFUs, a fast hardware; digital GPFU is designed to compute the Gaussian potential function Exponent of the GPFU is considered as $\varnothing(x) = \sum_{n=1}^{N} k_n (x_n - \mu_n)^2$. Thus based on the learning algorithms the centers μ_n and the coefficients k_n In the new CORDIC architecture as the exponent is computed serially the square and the product are also computed serially as this requires less hardware thus a unit to compute the approximation of the exponential function is designed and another unit is designed for calculating the square of a number. As multiplication and squaring generate additional digits which do not contribute to the accuracy, hence they are masked. Several GPFUs in parallel can be monitored by a unit designed to control the masked signals. Hence a digital GPFU is built which includes units of serial addition, square and product computation.

To compute the sum s_n and the carry c_{n+1} x_n – a serial adder is used. Thus to calculate the addition of two numbers a and b, the serial adder calculates by

$$c_n = a_n b_n v a_n c_{n-1} v b_n c_{n-1}$$

$$s_n = a_n \oplus b_n \oplus c_{n-1}$$

A binary tree structure as shown is used if more than two numbers are to be added

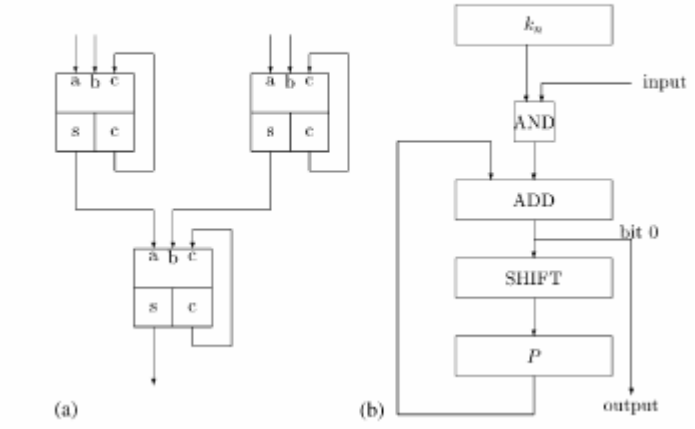

Fig: 4.3 (a) Serial addition for four operands (b) the conventional product unit

The adder output is the enable signal for the X register. By adding the two's complement of μ_n, adders of similar type are used to compute $x_n - \mu_n$.

Square and Product Computation

The square units compute one bit of the input in each iteration. The input is considered in the form, $a = \sum_{M=0}^{Q} a_m 2^m$, the serial addition of x_n and $-\mu_n$ produces a_m. The square of a is calculated as $a = \sum_{M=0}^{Q} a_m . a . 2^m$, where $a.2^m$ can be implemented in a shift register and the bits a_m mask the shift register's output. As a is shifted into the shift register, the mask bits are set, which determine whether bit m of the shift register will be added to the output or not. In case of a negative input the square algorithm is extended by a late sign decision unit.

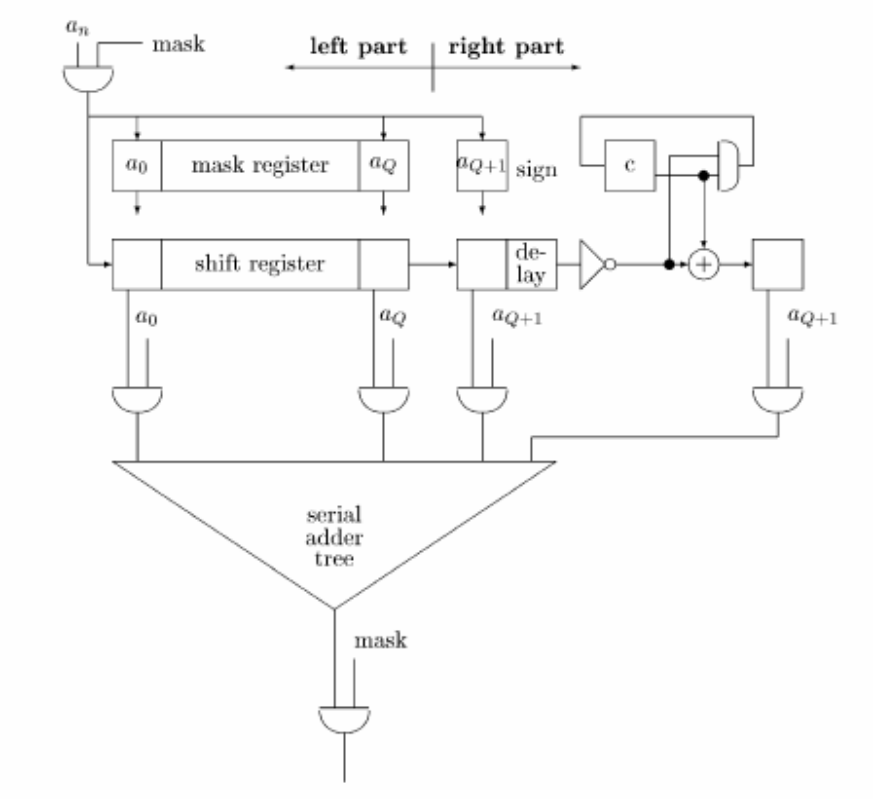

Fig: 4.4 Hardware structure for computing the square of a serial input

Digital GPFU

Hence a digital GPFU is constructed with the above discussed units as shown in below. Each feature performs independently, though they are processed simultaneously. The path of each feature includes a shift register for $-\mu_n$, which is fed back for the next iteration. x_n and $-\mu_n$ are added by a single elementary serial adder. The GPFU unit discussed can control several GPFU's in parallel and also the shifts in the exponential unit.

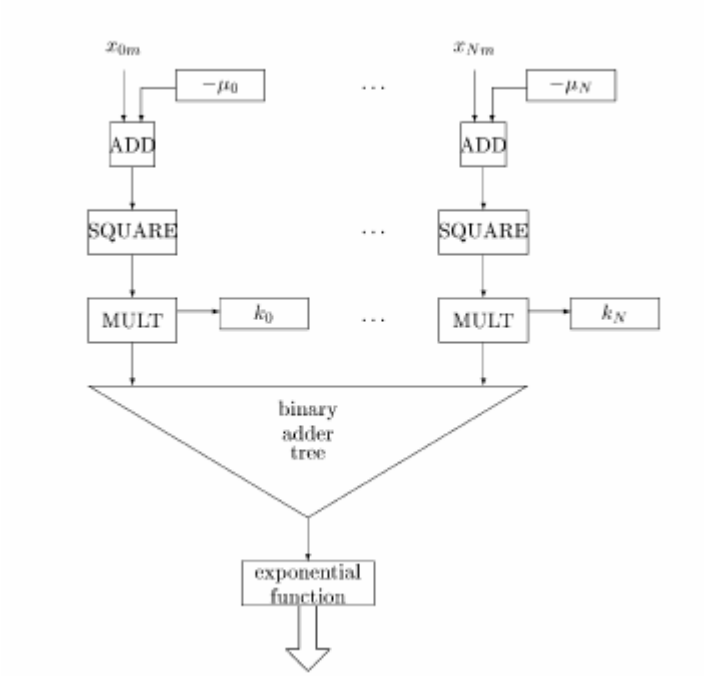

Fig: 4.5 The Complete digital GPFU

Synthesis Results

In Radix 2, where the data is entered one bit at a time the exponential unit processes one input in each iteration, hence the exponent is computed serially and the serial square unit is designed which has the capability of processing both negative and positive numbers. In Radix 4, the exponential unit processes two inputs at a time, i.e., two bits at a time. In the Radix 4 the case statements in the FSM are combined in an efficient way such that two bits are evaluated at a time. The main aim was to increase speed and minimize the area or space occupied as compared to Radix 2. For this the registered performance is performed and the results tabulated.

Software Implementation

The algorithm is synthesized on VHDL and optimized for speed and size with synthesis tools from Altera. A FPLD device EPF10K20RC208-4 from Altera is selected such that the largest design can be implemented on the device. The above FPLD supports fast carry arithmetic for 10K and 20K families and for sufficient number of logic cells (1152). Synthesis of the design is performed by Altera's Max plus 11 version 9.23 tools.

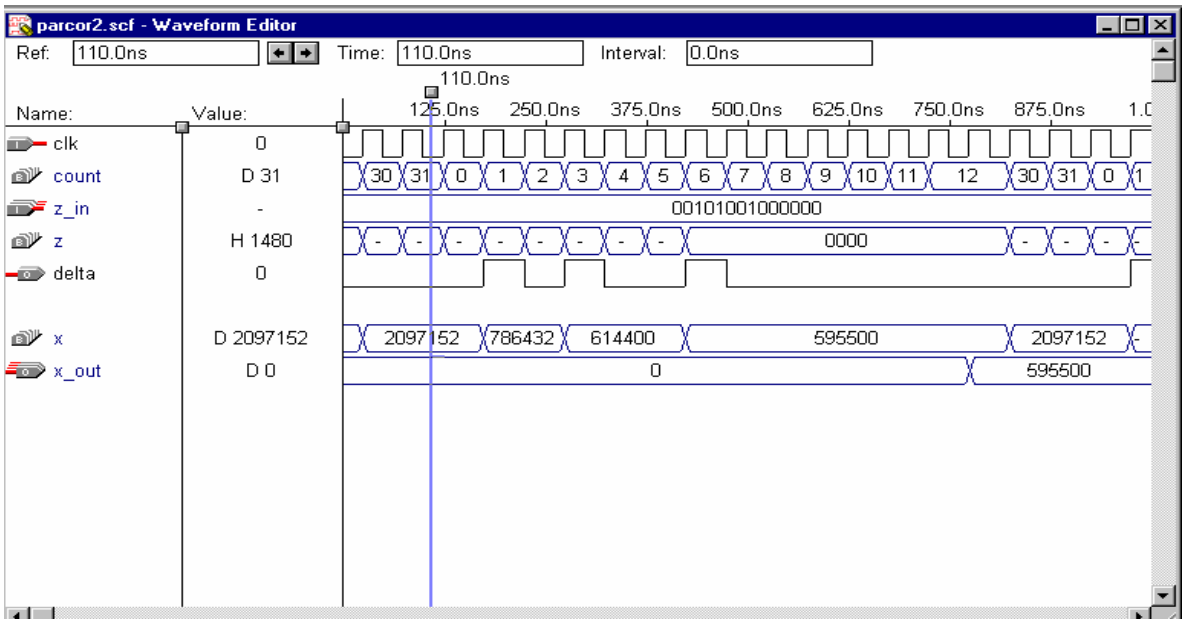

Fig4.1 VHDL simulation of the CORDIC algorithm with input one bit at a time

The data is entered in z_in, the CORDIC algorithm computes the output x_out after 750 ns. Depending on the input data based on the number of non- zero elements three scaling operations are required. Delta goes high whenever scaling exists. Here the algorithm is computed on one bit of the input at a time.

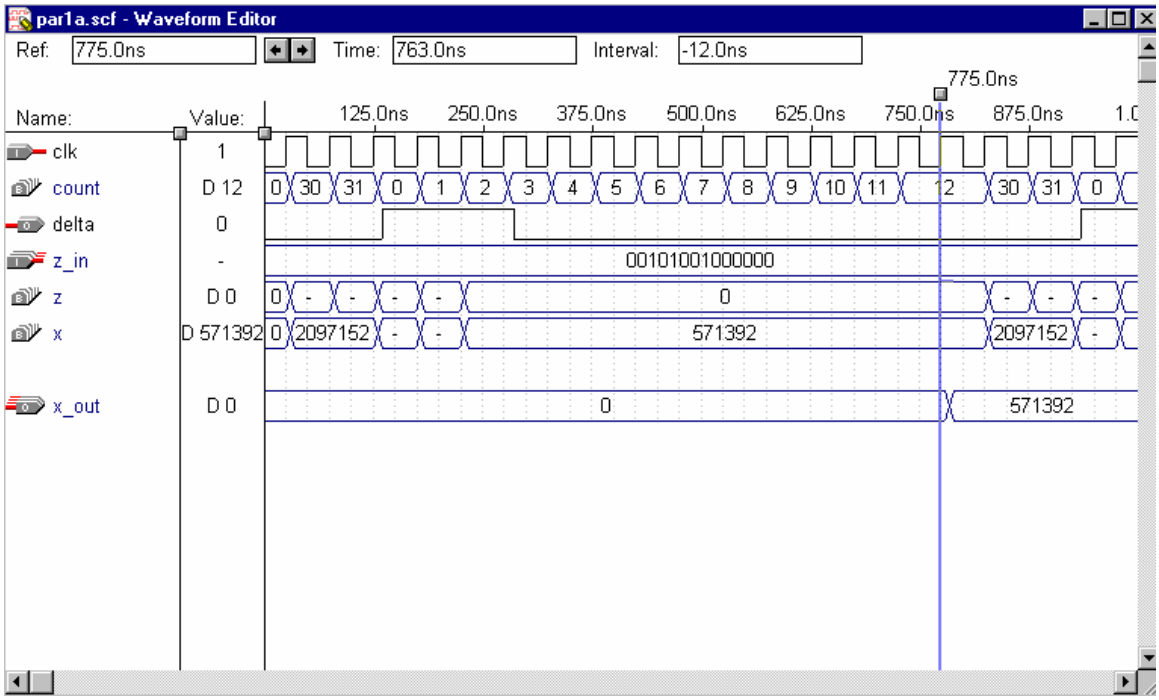

Fig4.2 VHDL simulation of the CORDIC algorithm with input two bits at a time

The same input data is considered for z_in, and the output is computed by the CORDIC algorithm after 750 ns. Here the algorithm computes two bits of the input data at a time thus reducing the time taken for the output to be computed.

Registered Performance Results

Table 4.1 Registered performance results

	Radix 4	**Radix 2**
Clock period	47.6ns	70.6ns
Latency	332.2	988.4
Frequency	21MHz	14.16MHz
Utilization of LC	89%	64%
A*T costs	342624	730132

There is speed improvement in Radix 4 as compared to Radix 2 and also the A*T cost are less in Radix 4 as compared to Radix 2. But the latency is comparatively high in Radix 2 as compared to Radix 4 as the number of clock cycles needed for Radix 2 evaluation is more. There will be a better speed improvement and A*T costs if three or higher bits are evaluated at a time.

Synopsis Software results

The design was optimized for area occupied and speed by the synopsis software which compiles with the LSI_10K ASIC elements to generate report files for area, timing and power dissipation.

Table 4.2 Result of synthesis of Radix 4

		Optimization		Area			
Rep #	Tmin	Area	Speed	Comb	Non comb	Total	Notes
1	100	X		2420	647	3067	Slack=11.97
1	5		X	7696	674	8370	Violated=-21.94
1	26		X	7332	669	8001	Violated=-0.66
1	26.5		X	6576	669	7245	Violated=-0.48
1	27		X	6722	668	7390	MET

Table 4.3 Result of synthesis of Radix 2

		Optimization		Area			
Rep #	Tmin	Area	Speed	Comb	Non comb	Total	Notes
1	100	X		1684	647	2331	Slack=0.76
1	5		X	8686	671	9256	Violated=-30.5
1	35		X	7882	667	8549	Violated=-2.11
1	35.5		X	7288	655	7953	Violated=-0.29
1	36		X	7456	667	8123	MET

From the above tables it is clear that the delay required in the Radix 4 design is less as compared to Radix 2, as in the case of Radix 4, the delay for which the slack is met is 27 ns whereas the delay in the case of Radix 2 is 36ns. Though the area occupied by the cells is more in Radix 4 as compared to Radix 2, the main goal of speed improvement is achieved.

REFERENCES

[1] A.Meyer-Baese, R.Watzel, U.Meyer-Baese, S.Foo, "A parallel CORDIC architecture dedicated to compute the Gaussian potential function in neural networks". Engineering Applications of Artificial Intelligence, 16 (2003) 595-605

[2] Sukhan Lee, Rhee M.Kil, "A Gaussian potential function network with hierarchically self organizing learning". Neural Networks, Vol 4, pp.207-224, 1991

[3] Jack E.Volder, "The CORDIC trigonometric computing technique". IRE Transactions on electronic computers.

[4] Ray Andraka, "A survey of CORDIC algorithms for FPGA based computers"

[5] J.S.Walther, "A unified algorithm for elementary functions". Hewlett-Packard company, Palo Alto, California

[6] Uwe Meyer-Baese, "Digital Signal Processing with Field Programmable Gate Arrays", Springer, Heidelberg 2001.

[7] Anke Meyer-Baese, "Pattern Recognition in Medical Imaging",

[8] Mark J.L Orr, "Introduction to Radial Basis Function Networks", Centre for Cognitive science, University Of Edinburgh

[9] U.Meyer-Baese, A.Meyer-Baese, J.Mellot, F.Taylor," A fast modified CORDIC-Implementation of Radial Basis Neural network", Journal of VLSI signal processing

[10] http://www.statsoft.com/textbook/stneunet.html

[11] http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

[12] http://www.clsp.jhu.edu/ws98/presentations/workshop/schuster/abstract.html

[13] http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html#algs

BIOGRAPHICAL SKETCH

I received my bachelor Of Engineering degree in Electronics and Communications from Vieshwariah Technological University in 2003. I worked in Four – C – Tron, Bangalore for about a year as a customer support engineer after completion of my graduate studies.

I came to FSU in August 2003 for pursuing a master's degree in Electrical and Computer Engineering and successfully completed master's degree in spring 2005.