Random Pulse Artificial Neural Network Architecture

by
Lichen Zhao B.Sc.

A thesis submitted to the school of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Applied Science
in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
Faculty of Engineering
University of Ottawa
May 1998

To my mom and dad

Abstract

This thesis presents a modular hardware artificial neural network architecture using the random pulse data representation and processing.

In random pulse machine, continuous variables can be represented as a probability of a pulse occurrence at a certain sampling time. Random pulse machines deal with analog variables while using digital technology to perform arithmetic and logic operations on binary pulses which are the information carries. Thus, large systems can be built to perform parallel analog computation on large amounts of input data using this technique. This is a good trade-off between the electronic circuit complexity and the computational accuracy and well suited for VLSI neural network.

Simulations have been conducted to validate the performance of both 1-bit and 2-bit random pulse neural network. Perceptron application for linear classification and autoassociative memory for digit recognition are finally presented to illustrate the behavior of the developed artificial neural network.

Acknowledgment

I would like to express my sincere gratitude to my supervisor Dr. Emil Petriu for his valuable advice, guidance and patience during my graduate studies. It has been an honor and a privilege to work with him.

I would also like to thank Mitel Corp. for providing me internship opportunity and financial assistance.

My deepest gratitude is reserved for my mom and dad, for their unselfish love, encouragement and understanding throughout all the stages of my study.

Table of Contents

List of Figures

List of Tables

List of Symbols

n	quantization level
b	quantization bit
Δ	quantization step
V_q	non-dither quantization output
$Q(\bullet)$	quantization function
e	non-dither quantization error (noise)
$q(\bullet)$	quantization error function
v	quantization input signal
r	dither signal
ε	dithered quantization error (noise)
x_q	dithered quantization output
$E[\bullet]$	expectation (ensemble average)
$\Phi(\bullet)$	one dimensional characteristic function
$\overline{f}(\bullet)$	probability density function of dither signal
$\overline{m}_{\varepsilon\|v}(\bullet)$	mean of quantization error
$R(\bullet)$	correlation function
FS	amplitude of dither signal
X_{MSB}	most significant bit in 2-bit quantization
X_{LSB}	least significant bit in 2-bit quantization
σ	variance

Chapter 1 – Introduction

1.1 Background

In the past ten years there has been strong and renewed interest in neural networks motivated largely by the need for better management of uncertain information, the availability of Very Large Scale Integrated (VLSI) implementation, and improved performance in wide areas of application over the existing methods.

The idea of a neural network was originally conceived as an attempt to model biophysiology of the brain, i.e. to understand and explain how the brain operates and functions. Later scientists and engineers who are concerned with how "artificial" neurons can be interconnected to form networks with interesting and powerful computational capabilities joined the research. Neural network is thus defined as :

> *A massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use.*
> *It resembles the brain in two respects:*
> *1. Knowledge is acquired by the network through a learning phase.*
> *2. Interneuron connection strengths known as synaptic weights are used to store the knowledge.* [by S. Haykin, "Neural networks, a comprehensive foundation"]

Neural networks are information processing systems. In general, neural networks can be thought of as "black box" devices that accept inputs and produce outputs. Some of the operations that neural networks perform include:

Classification
Pattern Matching
Pattern Completion
Noise Removal
Optimization
Control

Figure 1.1 shows the topology of a typical neural network. W and V stand for weight matrices. The processing elements (PE) in a layer form a vector and are denoted as F_x and F_y.

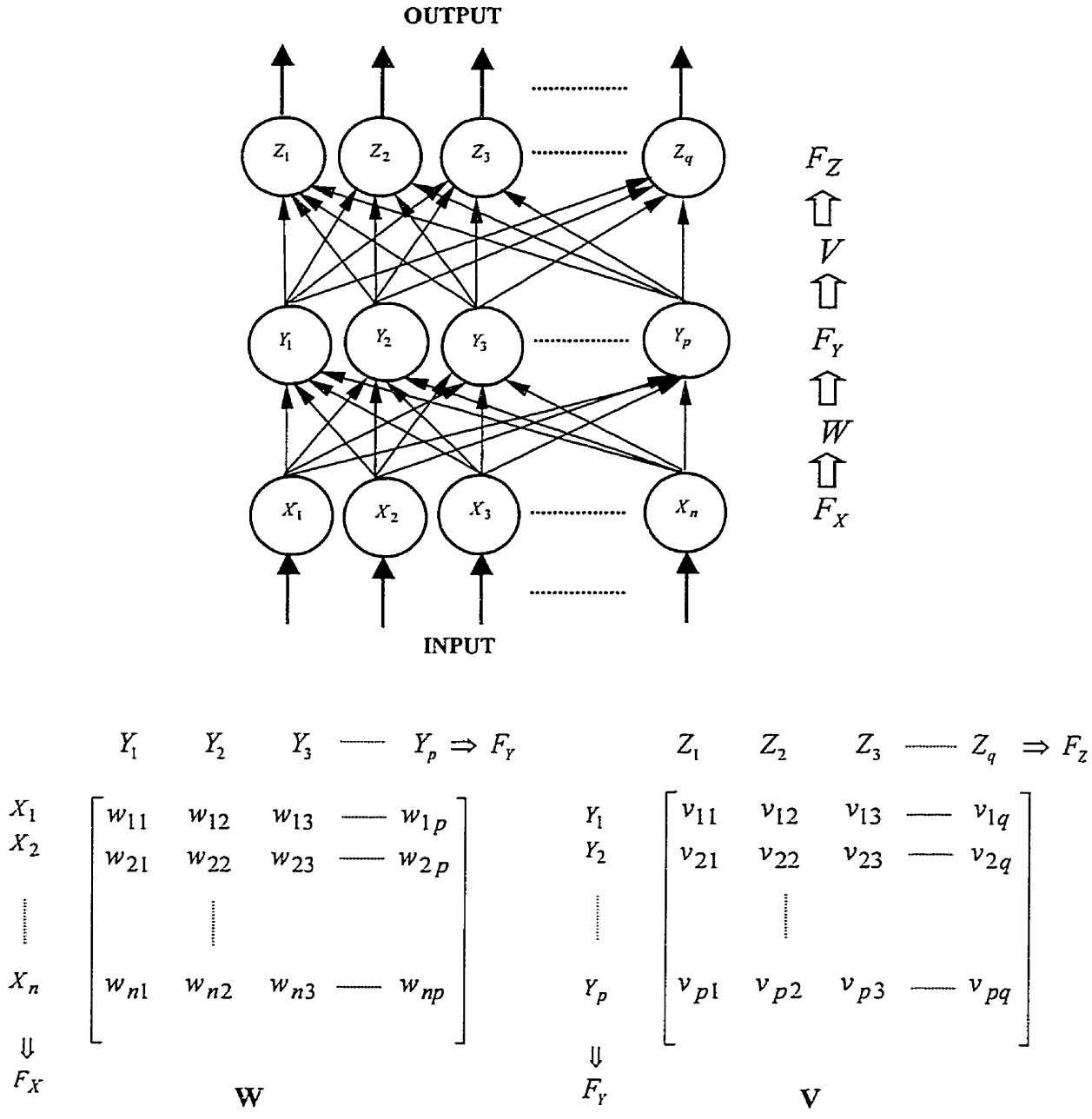


Figure 1.1 Neural network topology

1.2 Realization of Neural Networks

Modern general-purpose computers can be used to simulate almost any neural network architecture and learning algorithm, and there is little doubt that such simulations in many cases afford the easiest approach for designing/prototyping neural network applications. However, a neural network program has no parallelism beyond engaging integer and floating-point processor units simultaneously. The general-purpose processor calculates synaptic weight, activation function of each neuron and updates connection serially. There are major applications that require neural networks with a speed/throughput which can not be sustained on available computers, due either to very large network size or the need for short (real-time) learning or response intervals. Many custom digital implementations (and software implementations executing on a parallel computer system) offer a semi-parallel way: each processing element calculates a subset of the network's connections and neuron activations. Throughput is increased by sharing the task over multiple processors. However, a simple network used in a mass-produced commodity may only be cost-effective as a single-chip stand-alone neural system. Further gains may be achieved by designing specialized custom processors optimized for the neural tasks required, and the highest level of parallelism is achieved by implementing each synapse and each neuron as a distinct circuit. The rapidly developing technology of VLSI circuit makes it possible to fabricate tens of millions of transistors interconnected on a single silicon wafer to build such systems. Figure1.2 shows the spectrum of neurocomputer.

However, single-chip neural system are often limited in size, support only certain classes of architecture, and represent signals and weights with limited precision and with large noise components, compared to signals in general-purpose computer which can be represented with high precision and dynamic range by floating-point variables.

The success of a hardware implementation depends critically upon a judicious balance of the many trade-off involved in selecting a hardware technology, designing the circuits, fixing classes and sizes of architectures that will be supported, and crafting a compatible learning algorithm.

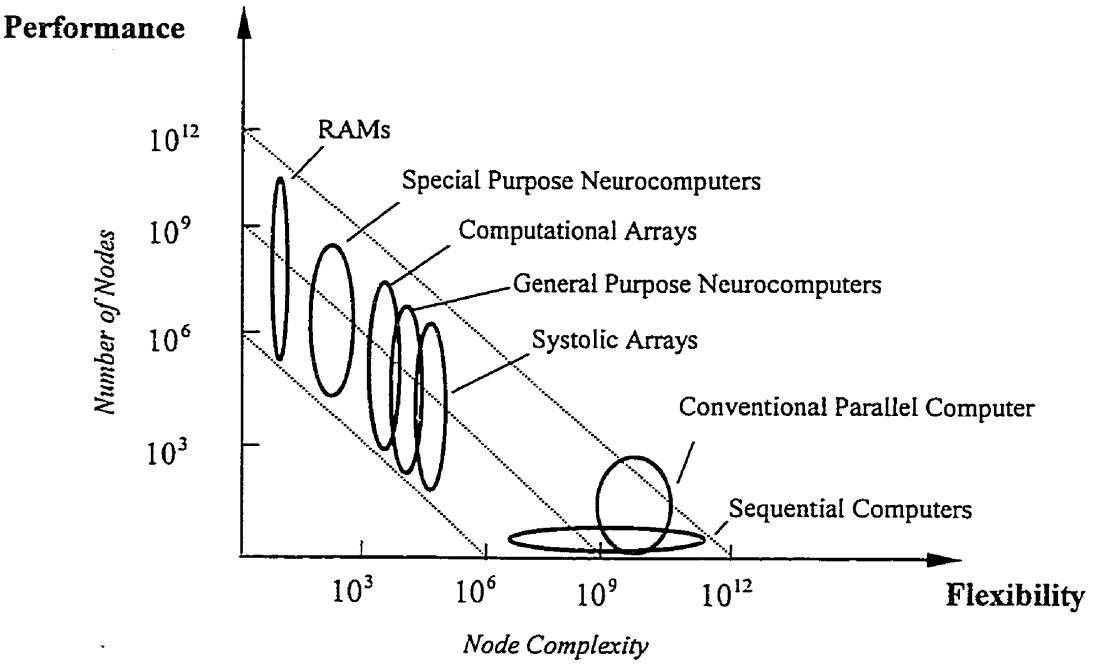


Figure 1.2 Neurocomputer implementations

(adapted from [6])

The most popular hardware implementation technologies are analog and digital CMOS implementations.

- Analog Structure

Analog systems can achieve a high packing density and are attractive for high-speed applications. However, being hardwired, they are more difficult to reconfigure and often implement only one fixed set of equations. The storage of analog weight values for synapse is difficult and the noise immunity is worse than in digital circuits.

- Digital Structure

Digital structures are straightforward to design and their testability is better than in the analog structure. Also the storage of the weight values is easy in the digital form and the interface to the digital coprocessors is also simpler. However, the overall area of digital structures is larger because of the computation and communication methods.

1.3 Random Pulse Neural Network Architecture

Hybrid and digital VLSI neural networks using pulse stream techniques have been studied to incorporate the merits of both digital and analog technology. There are two major approaches in this area. One is to encode signals using pulse amplitude, width, density, or frequency and to perform synaptic multiplication and signal integration using analog and (or) digital circuits combined with pulse modulation techniques. In the other approach, signals are represented by probabilities and encoded in random pulse streams by which algebraic operations with analog variables can be performed by logic gates.

The latter idea was first presented by von Neumann in 1956 [1]. Pursuing this idea, a number of similar stochastic data-processing concepts were reported in the 1960's: "noise computer" by Poppelbaum and Afuso [2], "random-pulse machine" by Ribeiro [19], and "stochastic computing" by Gaines [8]. Random-Pulse machine concept is employed in this research which deals with analog variables represented by the mean rate of random pulse streams. Such a representation can be viewed as the probability modulation of a random-pulse carrier by a deterministic analog variable. Simple digital gates can be used to perform arithmetic and logic operations. This concept presents a good tradeoff between the electronic circuit complexity and the computational accuracy. The resulting neural network architecture has a high packing density and is well suited for VLSI implementation. Dither technique is

employed in this research to reduce the effects of the quantization error in analog/random-pulse conversion.

1.4 Organization of This Thesis

Chapter 2 introduces the concept of random-pulse machine and random pulse data acquisition and representation. Effect of dither on quantization are analyzed based on sinusoidal signal. 1-bit & 2-bit Analog/Random-pulse conversions are studied and simulation results are given to compare the resolution. Different techniques of representing random pulse data and dither signal generation are also discussed.

Chapter 3 describes the arithmetic operations with random pulse data. Simulations are based on both 1-bit and 2-bit implementations for basic neuron operations: multiplication and addition. Random-pulse/Analog conversion, known also as moving average estimation, are presented. Finally, 1-bit synapse and neuron architectures are given.

Chapter 4 presents case studies of neural network applications based on 1-bit and 2-bit random pulse neural networks. It shows again 2-bit neural network achieves more precision.

The conclusions and contributions of this thesis, as well as a brief indication of further research are presented in chapter 5.

Chapter 2 – Random Pulse Data Representation

2.1 Problem Introduction

Implementing a neural network on a digital circuit yields the quantization of the synaptic weights and input signals.

Random pulse data representation concept was introduced by von Neumann as a model for representing the information in biological neural network [1]. Following the same idea, other similar stochastic data processing concepts were introduced: "noise computer" by Poppelbaum and Afuso [2], "stochastic computing" by Gaines [8] and "random-pulse machine" by Ribeiro [19].

In random pulse machine, analog variables are represented by the mean rate of random pulse streams. These random pulse streams are actually the result of dithering/random reference quantization (RRQ). Dithering/RRQ was used in many practical applications especially in 1960's to reduce the effects of noise quantization in speech and image processing. From a spectral point of view, it was found that a suitable dither choice allows the whitening of the quantization error spectrum. This means that a significant portion of the error power is located outside the signal band and can be easily eliminated by low-pass filtering the quantized data. In such a way the quantizer resolution and linearity are improved at the expense of both a reduced conversion rate and input range amplitude. In particular, while a small-amplitude dither mainly reduces the quantization error, large nonlinearity errors can be reduced when the dither amplitude covers a significant portion of the quantizer input range.

Random pulse machine presents a good tradeoff between electronic circuit complexity and the computational accuracy. The resulting neural network architecture has a high packing density and is well suited for VLSI implementation.

A time analysis of the dithering involving time-dependent data processing (e.g. correlation) as described further in this chapter is more suitable in applications.

2.2 Quantization

The notations used for the input and the output of the quantizer and for the quantization error are shown in Figure 2.1(a) and 2.1(b). And the quantizer input /output and error characteristic can be shown as in Figure 2.1(c) and 2.1(d) respectively.

In Figure 2.1, n=8 represents the number of quantization levels, and Δ is the quantization step. When the input is confined in the no-overload region $[-(n+1)\Delta/2,(n-1)\Delta/2]$, the analytical expression for quantizer input/output and error characteristics are, respectively,

$$v_q = Q(v) = \left\lfloor \frac{v+\frac{\Delta}{2}}{\Delta} \right\rfloor \Delta \qquad (2.1)$$

and

$$e = q(v) = \frac{\Delta}{2} - \Delta \left\langle \frac{v+\frac{\Delta}{2}}{\Delta} \right\rangle \qquad (2.2)$$

where $\lfloor a \rfloor$ represents the greatest integer less than or equal to a, and $\langle a \rangle = a - \lfloor a \rfloor$ is the fractional part of a. It is worth noticing that (2.2) is a periodic function of v with period Δ; thus it can be expressed by a Fourier series:

$$q(v) = \sum_{k=1}^{\infty} \frac{\Delta}{\pi k}(-1)^k \sin\left(2\pi k \frac{v}{\Delta}\right) \qquad (2.3)$$

Figure 2.2 illustrates the effects of different quantization levels for a sinusoidal signal. It is obvious that by increasing the quantization levels we can increase the resolution.

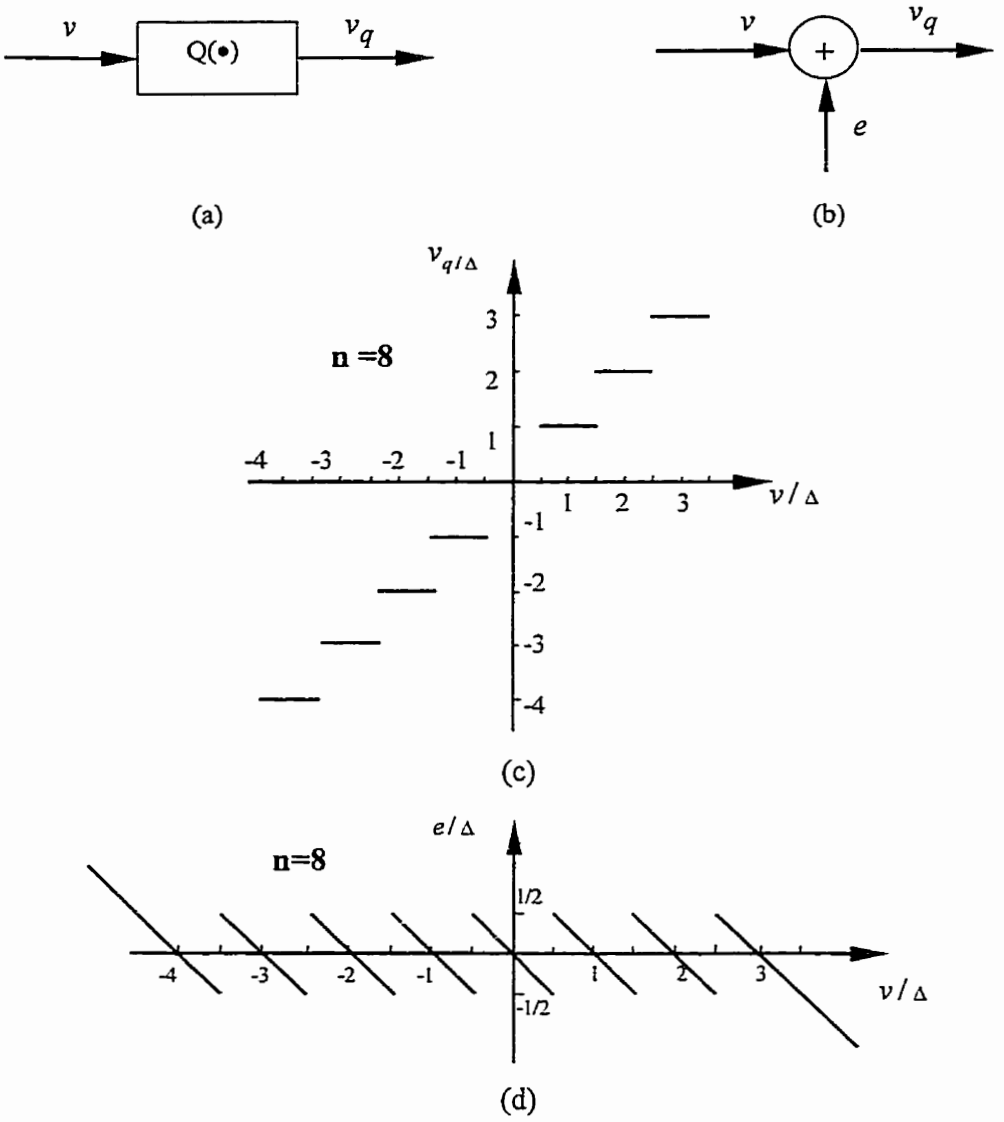


Figure 2.1 Quantization and quantization noise

(a) Block Diagram (b) Additive model (c) Quantizer Input/Output Characteristic Q(•)

(d) Quantization Error Characteristic q(•)

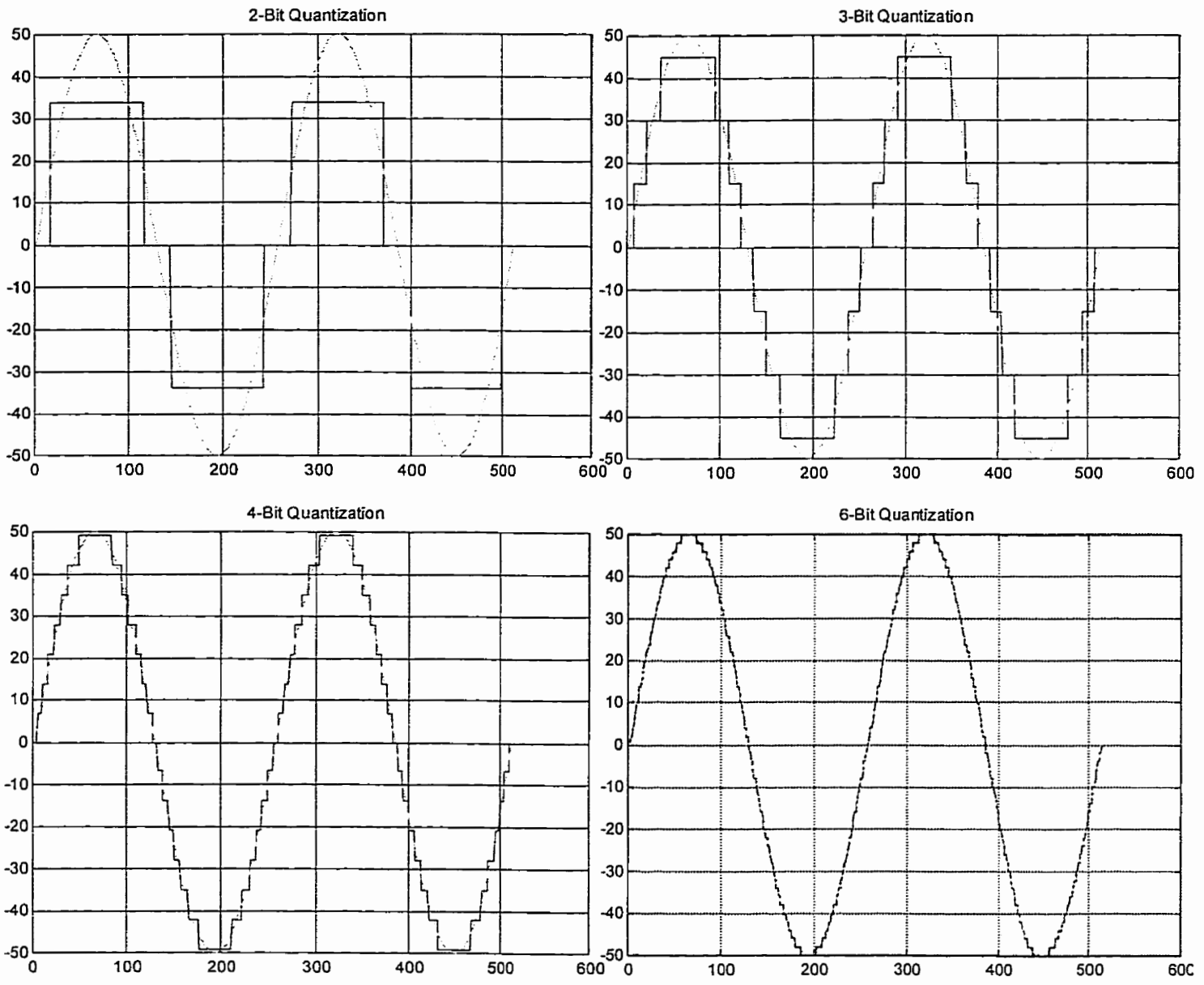


Figure 2.2 Quantization of a sinusoidal signal

2.3 Effect of the Dither on Quantization Error

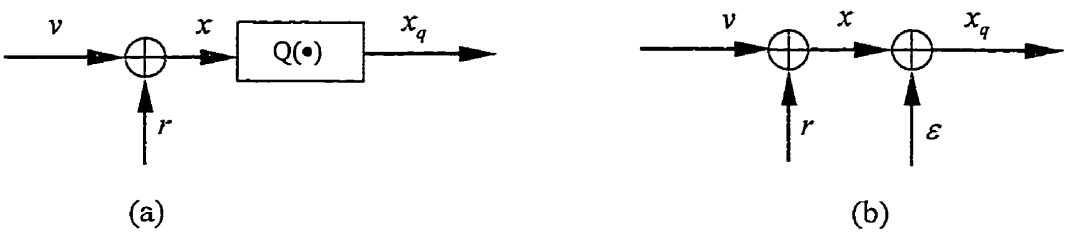


Figure 2.3 Quantizer with dither signal

(a) Block Diagram (b) Additive Model

Figure 2.3 shows both the block diagram and the model of a quantizer when a dither (RRQ) signal is added to its input; ε represents the error due to the quantization of the signal x. In order to quantify the effect of dither on the improvement of the quantizer resolution, it is of interest to find the average behavior of the quantization error when the slowly varying dithered input x is repeatedly quantized. According to quasi-stationary processes, the expectation of a process y is defined as :

$$\bar{E}[y] = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} E[y[n]] \qquad (2.4)$$

if the limit exists. The operator $E[\bullet]$ represents the usual ensemble average; therefore the expectation of (2.4) reduces to a time-average or to an ensemble-average in the special cases of purely deterministic or purely random processes, respectively. Other simple examples of quasi-stationary processes can be obtained by adding a deterministic signal and a stationary random noise, e.g., a sinusoid embedded in additive white noise.

One-dimensional characteristic function (CF) is defined as

$$\overline{\Phi_y}(u) = \overline{E}\left[e^{j2\pi uy}\right] \qquad (2.5)$$

and of its Fourier transform,

$$\overline{f_y}(y) = \int_{-\infty}^{\infty} \overline{\Phi_y}(u) e^{-j2\pi uy} du \qquad (2.6)$$

when the usual time average is considered, (2.6) represents the well-known amplitude density function (ADF) or probability density function (PDF).

Signals $v, x, r,$ and ε in Figure 2.3 are assumed to be continuous-amplitude, discrete-time quasi-stationary processes, while x_q is discrete both in amplitude and in time. The dither is assumed independent of the input signal; this means that $r[n]$ is independent of $v[k]$ for all times n and k. Moreover, the one-dimensional statistical description of r is assumed to be known. In particular, only dither signals with zero mean will be considered since this parameter does not affect the quantizer resolution.

Since uniform dither is of our most interest, following analysis is based on it.

A uniform dither of peak-to-peak amplitude R exhibits the following rectangle PDF:

$$\overline{f_r}(r) = \frac{1}{R} i(s), \quad S = \left\{ -\frac{R}{2} \le r \le \frac{R}{2} \right\} \qquad (2.7)$$

where $i(s)$ is the indicator function of the set S. Thus, the corresponding CF is given:

$$\overline{\Phi_r}(u) = \mathrm{sinc}(uR) \qquad (2.8)$$

in which $\mathrm{sinc}(x) = \lim_{k \to x} \sin(\pi k)/(\pi k)$. This distribution can be obtained both by using a random noise generator, or by sampling a sawtooth wave whose period T_p is not synchronized with the sampling period T. In fact, from (2.2) it follows that such a wave can be expressed as

$$r(t) = \frac{R}{2} - R\left\langle \frac{t + \frac{T_p}{2}}{T_p} \right\rangle \qquad (2.9)$$

in which the signal periodicity is related to the effect of the fractional-part operator. When (2.9) is asynchronously sampled, that is, when the ration between T and T_p is irrational, then the sequence of numbers $\{\langle nT/T_p + 1/2 \rangle\}$ uniformly fills the interval $\{0,1)$. Hence, since

(2.9) is a piecewise linear transformation, the sequence $r=\{r(nT)\}$ uniformly fills the interval $[-R/2, R/2)$.

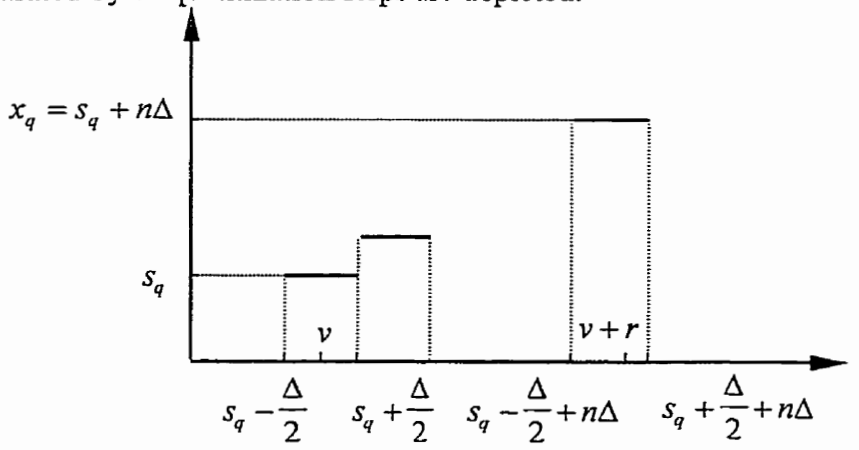
2.3.1 Mean of Quantization Error

Next, quantization error and quantizer output averages are analyzed in order to provide a lower bound on the resolution performance that can be achieved by a dithering strategy. This is of interest since the quantizer is usually followed by a low-pass filter whose output represents an estimation of the mean of the original input signal, e.g., the moving average filter whose output is the normalized sum of N successive quantized data.

Since slowly varying input signal are considered, the average of the quantization error $\overline{m}_{e|v}(\bullet)$, conditional on a given quantizer input v, will be derived. To this aim, two different approaches can be used. First, the average of the quantizer output x_q has to be evaluated first. It is obtained by observing that the quantizer input is given by

$$x = v + r = s_q - e + r \qquad (2.10)$$

where $e = s_q - v$ is the quantization error with no dither. Because of the presence of r, the quantizer output can assume different values. In particular, if it is $\left(n-\frac{1}{2}\right)\Delta + s_q \le v + r \le \left(n+\frac{1}{2}\right)\Delta + s_q$, then $x_q = s_q + n\Delta$. In figure 2.4, two generic output levels separated by n quantization steps are depicted.



Quantization of v and of v+r

Figure 2.4 Dithered quantizer model
(adapted from [37])

By generalizing this result for any dither value, we get

$$x_q = s_q + \Delta \sum_n ni(S_n),$$

$$S_n = \left\{ \left(n - \frac{1}{2} \right) \Delta + e \le r < \left(n + \frac{1}{2} \right) \Delta + e \right\} \qquad (2.11)$$

where the index n varies in the allowed range for the output codes. The intervals S_n are not overlapping so that for a given value of r only one term in the summation (2.11) is not zero. By using (2.11), the average of the quantizer input/output characteristic $\overline{m}_{x_q|v}(v) = \overline{E}[x_q \mid v]$ can be expressed as

$$\overline{m}_{x_q|v}(v) = \int_{-\infty}^{\infty} x_q \overline{f}_r(r) dr = s_q + \Delta \sum_n n \overline{P}(S_n) \quad (2.12)$$

in which

$$\overline{P}(S_n) = \overline{P}(r \in S_n) = \int_{S_n} \overline{f}_r(\alpha) d\alpha \qquad (2.13)$$

is the probability that the considered dither signal assumes a value in the set S_n. Finally, making use of (2.12) along with the following expression:

$$\varepsilon = x_q - v - r \qquad (2.14)$$

We obtain

$$\overline{m}_{\varepsilon|v}(e) = \overline{m}_{x_q|v}(v) - v = e + \Delta \sum_n n \overline{P}(S_n) \qquad (2.15)$$

which shows that $\overline{m}_{\varepsilon|v}(\bullet)$ depends on v only by means of e. Since (2.15) gives the deviation of the dither averaged input/output characteristic from the ideal straight line characteristic, it provides a bound on the resolution performances that can be achieved by low-pass filtering the quantizer output.

Alternatively to the previous approach, the mean of the quantization error ε can be directly evaluated by means of the following relationship:

$$\overline{m}_{\varepsilon|v}(v) = \overline{E}[q(v+r) \mid v] = \int_{-\infty}^{\infty} q(v+r) \overline{f}_r(r) dr \quad (2.16)$$

which by using (2.3) and (2.5),

$$\overline{m}_{\varepsilon|v}(e)=\sum_{k=1}^{\infty}\frac{\Delta}{\pi k}(-1)^{k+1}\overline{\Phi}_r^{\bullet}\left(\frac{k}{\Delta}\right)\sin\left(\frac{2\pi}{\Delta}ke\right) \qquad (2.17)$$

In (2.17) v has been substituted by $-e$ due to the periodicity in v with period Δ of the $\sin(\bullet)$ function.

From (2.17) and (2.8) we have

$$\overline{m}_{\varepsilon|v}(e')=\sum_{k=1}^{\infty}\frac{(-1)^{k+1}}{\pi k}\operatorname{sinc}(2kR')\sin(2\pi ke') \quad (2.18)$$

$$e'=\frac{e}{\Delta},\quad R'=\frac{R}{2\Delta},\quad \varepsilon'=\frac{\varepsilon}{\Delta},\quad v'=\frac{v}{\Delta}$$

It can also be written in simple form by using (2.15):

$$\overline{m}_{\varepsilon|v}(e')=\begin{cases}\dfrac{1}{R'}\left(\langle R'\rangle-\dfrac{1}{2}\right)\left[e'-\dfrac{1}{2}\operatorname{sgn}(e')\right], e'\in A\\ \dfrac{1}{R'}\left(\langle R'\rangle-1\right)e', e'\in A_{>}\cdot\\ \dfrac{1}{R'}\langle R'\rangle e', e'\in A_{<}\cdot\end{cases} \qquad (2.19)$$

Some calculations give for this function the following relationship:

$$\max_{e'}\left[\overline{m}_{\varepsilon|v}(e')\right]=\frac{\langle 2R'\rangle}{4R'}\left(1-\langle 2R'\rangle\right) \qquad (2.20)$$

The minimum of (2.20) occur for $R'=\frac{1}{2}n$, where n is an integer. In these cases, the mean of the quantization error becomes identically zero. This means that the quantizer input/output characteristic is no longer a stepwise function but, on the average, becomes a straight line, thus realizing an infinite resolution.

2.3.2 Correlation Between Quantization Error and Input

The correlation, R, between the quantizer input, y, and quantization error, e, is given:

$$R = E(y \cdot e) \tag{2.21}$$

where E is the expectation. For a bipolar ADC, the correlation for the *k*th quantization slot would be given by:

$$R(k) = \int_{(k-1/2)\Delta}^{(k+1/2)\Delta} (k-y)y \cdot f_y(y)dy \tag{2.22}$$

where Δ is the quantization step, and $f_y(y)$ is the PDF of y. By normalizing (2.22) with respect to y, which is equivalent to letting Δ=1, then

$$R(k) = \int_{k-1/2}^{k+1/2} (k-y)y \cdot f_y(y)dy \tag{2.23}$$

Take sinusoidal signal as an example:

A. With No Dither

The PDF of the undithered sinusoid, $y = A\sin\varpi t$, is given by

$$f_y(y) = \frac{1}{\pi\sqrt{A^2 - y^2}} \tag{2.24}$$

where A is the amplitude of the sinusoid. By plugging (2.24) into (2.23), then:

$$R(k) = \int_{k-1/2}^{k+1/2} \frac{(k-y)y}{\pi\sqrt{A^2 - y^2}} dy \tag{2.25}$$

It can be shown that:

$$R(k) = -\frac{1}{\pi}\left[\phi(x_2) - \phi(x_1)\right]$$

$$\phi(x_i) = k\sqrt{A^2 - x_i^2} + \frac{A^2}{2}\sin^{-1} x_i - \frac{x_i}{2}\sqrt{A^2 - x_i^2} \tag{2.26}$$

$$x_1 = k - \frac{1}{2}, x_2 = k + \frac{1}{2}$$

B. With Uniform Dither

A dither signal whose PDF is given by

$$f_r(r)=\begin{cases}\dfrac{1}{2c}, & -c\le r\le c\\ 0, & otherwise\end{cases} \qquad (2.27)$$

is added to the input signal whose PDF is given by (2.24). $f_y(y)$ is derived by convolving (2.24) and (2.27). Two cases are considered.

1) c<A: When the dither amplitude is smaller than the sinusoid amplitude, the correlation for the *k*th quantization slot is derived. Using mathematical tables for integration it can be shown that

$$R(k)=\frac{1}{2\pi}\left[\phi(x_2)-\phi(x_1)-\phi(x_4)+\phi(x_3)\right],$$
$$|k|\le A-c-\frac{1}{2} \qquad (2.28a)$$

where:

$$\phi(x_i)=-\left(\frac{A^3}{3c}\right)x_i^3\sin^{-1}x_i+A^2\left(\frac{k}{2c}+1\right)x_i^2\sin^{-1}x_i$$
$$-A(k+c)x_i\sin^{-1}x_i-A^2\left(\frac{k}{4c}+\frac{1}{2}\right)\sin^{-1}x_i$$
$$-\left(\frac{A^3}{9c}\right)x_i^2\sqrt{1-x_i^2}+A^2\left(\frac{k}{4c}+\frac{1}{2}\right)x_i\sqrt{1-x_i^2}$$
$$-A\left(k+c+\frac{2A^2}{9c}\right)\sqrt{1-x_i^2}$$

$$x_{1,2}=\frac{k\mp\frac{1}{2}+c}{A}, \qquad x_{3,4}=\frac{k\mp\frac{1}{2}-c}{A} \qquad (2.28b)$$

Also:

$$R(k)=\frac{1}{2\pi}\left[\phi(x_2)-\phi(x_1)\right],$$
$$A-c+\frac{1}{2}\le|k|\le A-c-\frac{1}{2} \qquad (2.29a)$$

where

$$\phi(x_i) = -\left(\frac{A^3}{3c}\right)x_i^3 \cos^{-1} x_i + A^2\left(\frac{k}{2c} - 1\right)x_i^2 \cos^{-1} x_i$$
$$+ A(k-c)x_i \cos^{-1} x_i - A^2\left(\frac{k}{4c} - \frac{1}{2}\right)\cos^{-1} x_i$$
$$-\left(\frac{A^3}{9c}\right)x_i^2\sqrt{1-x_i^2} - A^2\left(\frac{k}{4c} - \frac{1}{2}\right)x_i\sqrt{1-x_i^2}$$
$$-A\left(k - c + \frac{2A^2}{9c}\right)\sqrt{1-x_i^2}$$
$$x_{1,2} = \frac{k \mp \frac{1}{2} - c}{A} \qquad (2.29b)$$

It shows that as c increases, the quantization error is less correlated to the quantizer input, and advantage of larger amplitude dither.

2) c>A: When the dither amplitude is greater than the sinusoid amplitude, derivations utilizing mathematical tables lead to

$$R(k) = 0, \quad |k| \le c - A - \frac{1}{2} \qquad (2.30)$$

$$R(k) = \frac{1}{2\pi}\left[\phi(x_2) - \phi(x_1)\right], \quad c - A + \frac{1}{2} \le |k| \le c + A - \frac{1}{2} \qquad (2.31)$$

where $\phi(x_i), x_1$ and x_2, are given by (2.29b).

It shows that as c increases |R(k)| decreases.

2.3.3 Amplitude Spectra of Quantization Errors

We still take sinusoidal signal as an example.

The quantization error function is readily expanded as a Fourier series in x:

$$q(x) = \sum_{n=1}^{\infty} \frac{\Delta}{n\pi} \sin\left(\frac{2\pi n x}{\Delta}\right) \cdot (-1)^n \qquad (2.32)$$

When a dither signal r is added to the signal voltage x at the quantizer input, then the average observed error at the ADC output is:

$$\bar{q}(x) = \int_{-\infty}^{\infty} q(x+r) \cdot p(r) dr \quad (2.33)$$

Where $p(r)$ is the PDF of dither. Equation (2.33) can be recognized as a convolution integral and transformed to the frequency domain as:

$$\bar{Q}(f) = Q(f) \cdot P(f) \quad (2.34)$$

where $\bar{Q}(f), Q(f)$, and $P(f)$, are the Fourier transforms of $\bar{q}(x), q(x)$, and $p(r)$, respectively.

A. Without Dither

It can be shown that the positive frequency side of the dither-free amplitude spectrum is given by

$$|Q(\omega)| = \sum_{n=1}^{\infty} \frac{\Delta}{n} \delta(\omega - n\omega_0) \quad (2.35)$$

where $\omega_0 = \frac{2\pi}{\Delta}$.

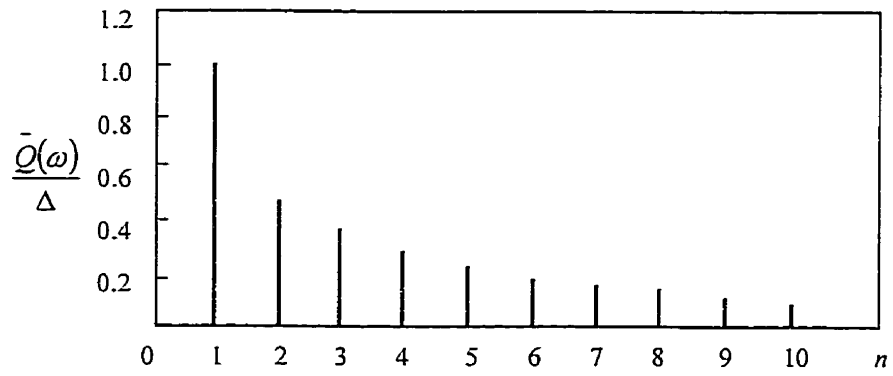


Figure 2.5 Spectrum of the average quantization error without dither.
(adapted from [38])

Figure 2.5 shows $|Q(\omega)| / \Delta$ versus n, the harmonic index, without dither.

B. Uniform Dither

The PDF of uniform dither is given by (2.27), and the Fourier transform is given by:

$$P(\omega) = \frac{\sin(\omega c)}{\omega c} \qquad (2.36)$$

from (2.34), (2.35) and (2.36), it can be shown that

$$\left|\bar{Q}(\omega)\right| = \sum_{n=1}^{\infty} \frac{\Delta}{n} \left| \frac{\sin(2\pi nc/\Delta)}{2\pi nc/\Delta} \right| \cdot \delta(\omega - n\omega_0) \qquad (2.37)$$

It is obvious that there are c values that nullify the spectrum of the averaged quantization error, e.g., $c = \Delta/2, \Delta, 2\Delta, \cdots$. These c values correspond to integral values of nc/Δ, or $2nc/\Delta$. Other c values do not nullify the spectrum.

2.4 Analog/Random-Pulse Conversion

2.4.1 Generalized Analog/Random-Pulse Conversion

Analog/Random-Pulse conversion serves as the interface between input signals and random pulse machine. It is accomplished by uniform dither quantizer as discussed in the previous section which can be viewed as a nonlinear mapping from the domain of continuous amplitude inputs onto one of a countable number of possible output levels.

The generalized analog/random-pulse converter is illustrated by Figure 2.6. The analog input V, supposed to have a relatively low variation rate, is mixed with an analog random dither signal R uniformly distributed between $-\Delta/2$ and $+\Delta/2$ which shall fulfill the following statistical requirements:

(i) zero mean;

(ii) independent of the input V;

(iii) characteristic function having periodic zeros.

The resulting analog signal VR is quantified with a b-bit resolution and then sampled by a clock to produce the random sequence VRP of b-bit data having amplitudes restricted to only two concatenated quantized values k-1 and k.

From the quantization diagram, we can calculate the ideal estimation over an infinite number of samples of the random data sequence VRP:

$$\begin{aligned} E[VRP] &= (k-1)p[(k-1.5)\Delta \le VR < (k-0.5)\Delta] + kp[(k-0.5)\Delta \le VR < (k+0.5)\Delta] \\ &= (k-1)b + kb = k - b \end{aligned} \qquad (2.38)$$

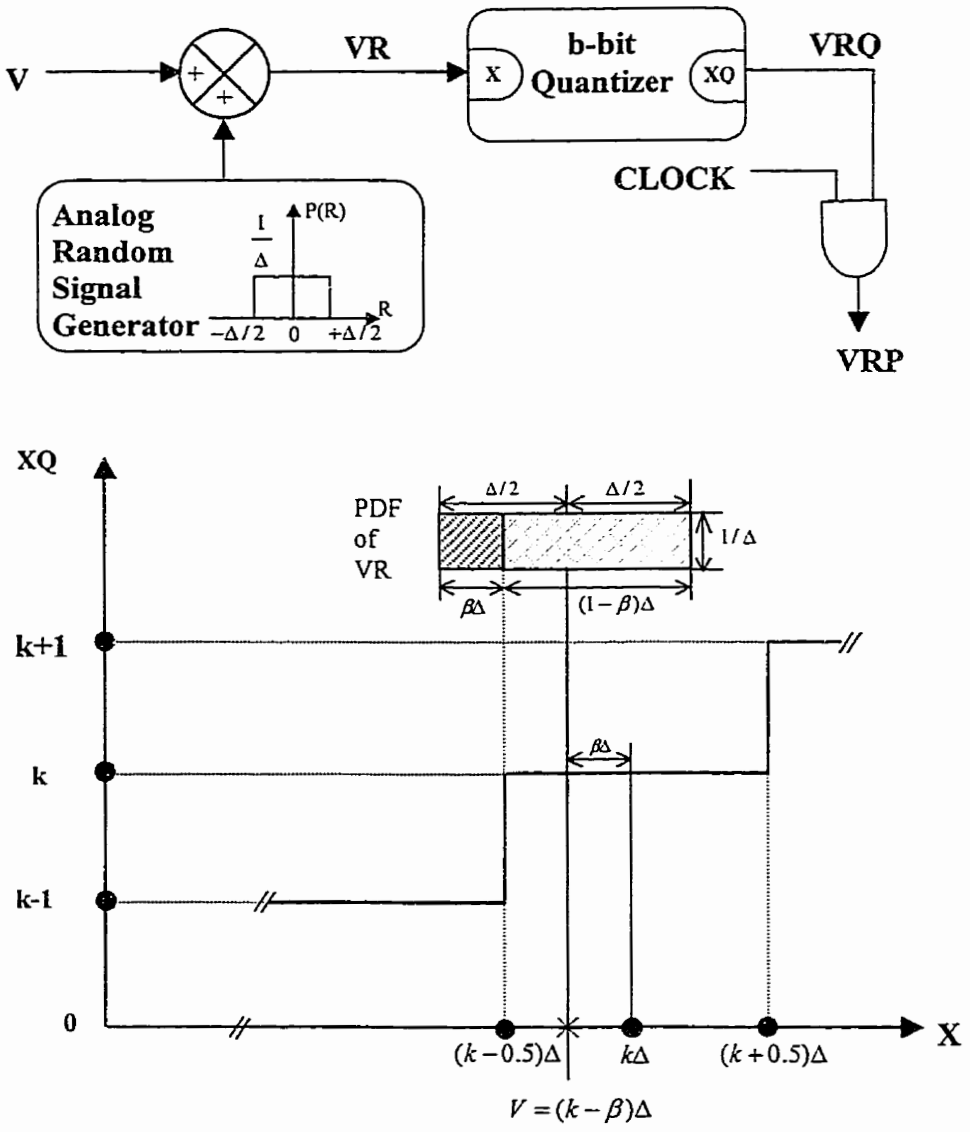
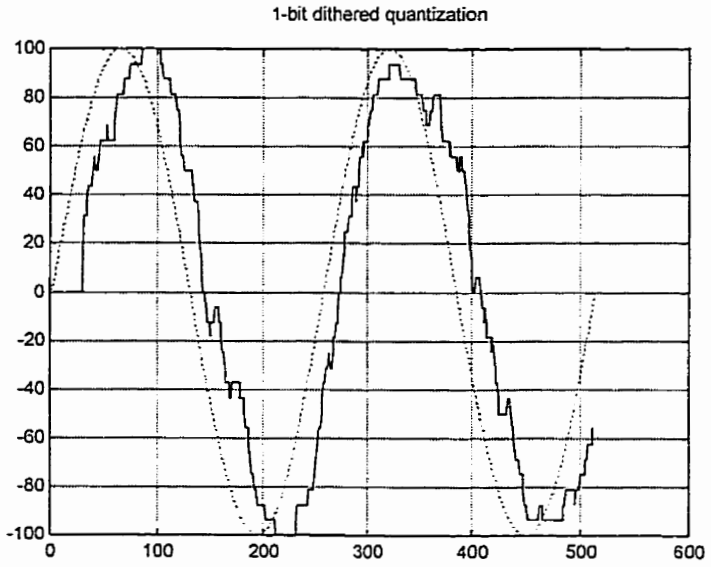
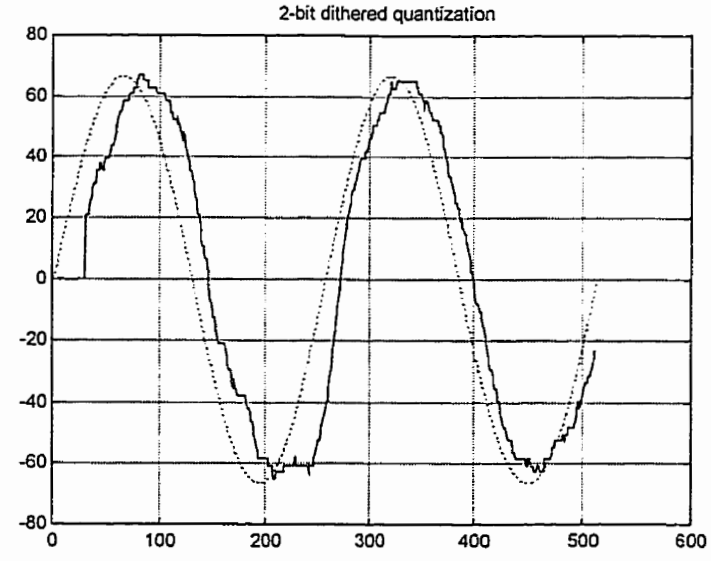


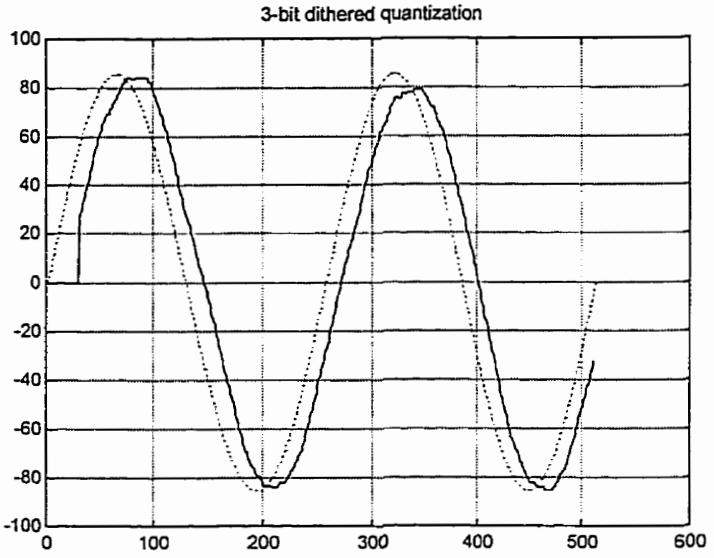
Figure 2.6 Generalized b-bit Analog/Random-pulse converter and its quantization characteristics

It should be noted that in case of dithered quantization the definition domain of the input signal v is trimmed by $\Delta/2$ to the left and to the right (i.e. an overall trimmed by Δ) of the domain for the non-dithered quantization. For instance for a dead-zone b-bit quantization the input domain is $\left[-(2^b+1)\frac{\Delta}{2}, (2^b+1)\frac{\Delta}{2}\right]$. If dithered quantization is used, the domain is trimmed to $\left[-2^b\frac{\Delta}{2}, 2^b\frac{\Delta}{2}\right]$.

Figure 2.7 shows the restored waveforms of a sinusoidal signal dithered quantized for different quantization levels. The restoring was done by moving averaging of the random pulse data over 32 pulse window size [23]-[24]. It can be seen that by increasing the initial quantization resolution the dithered quantized sinusoidal signal can be represented much more precisely. Compared with figure 2.2, with same quantization levels dithered quantization increases the resolution significantly. It is also worthwhile to note the phase shift of the reconstructed signal due to the delay introduced by the moving average of the pulses.







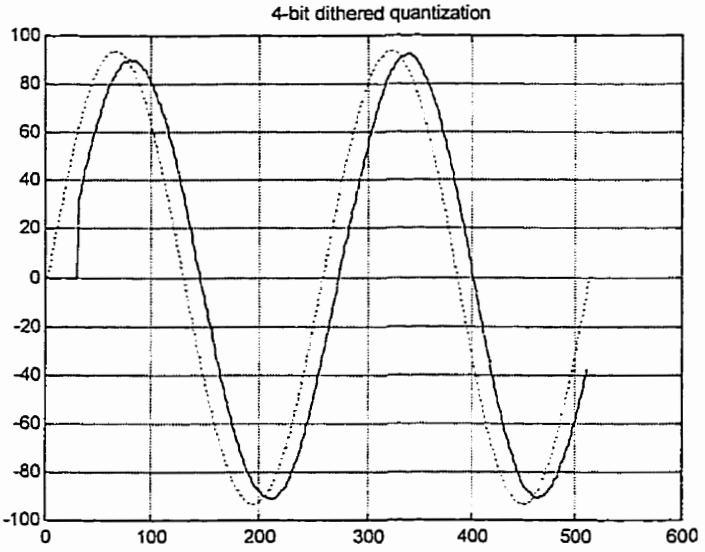


Figure 2.7 Dithered quantization of a sinusoidal signal
(moving average over 32 pulse window size)

2.4.2 1-bit & 2-bit Analog/Random-Pulse Conversion

Binary random-pulse representation is the simplest case of dither quantization, when b=1. The principle of this one-bit dither quantization is illustrated in Figure 2.8. A dither signal *R* uniformly distributed between *-FS* and *+FS* is added to the analog input *V* before quantization. The resulting analog random signal *VR* is then 1-bit quantified (using a simple comparator) to produce a random sequence of pulses *VRP* which will have the binary value +1 if $VR \geq 0$ or -1 if $VR < 0$.

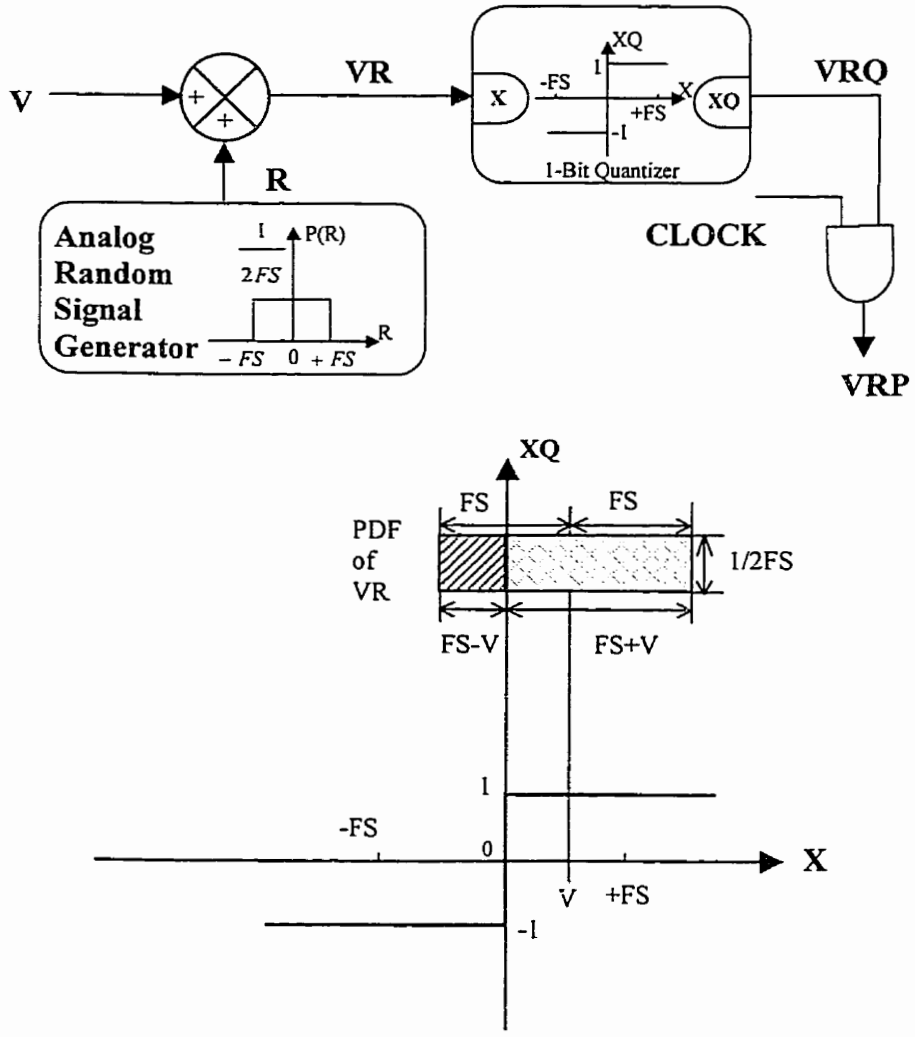


Figure 2.8 1-bit Analog/Random-pulse converter

A probabilistic estimation of the deterministic component of the random-pulse sequence can be calculated from the *VR* versus *VRP* quantization shown in Figure 2.8:

$$\begin{aligned} E[VRP] &= (+1)\cdot p[VR \geq 0] + (-1)\cdot p[VR < 0] \\ &= p(VRP) - p(VRP') \\ &= \frac{(FS+V)}{(2\text{-}FS)} - \frac{(FS-V)}{(2\text{-}FS)} \\ &= \frac{V}{FS} \end{aligned} \qquad (2.39)$$

which shows that the statistical mean value of the *VRP* sequence represents a measure of the deterministic analog input *V*. From (2.39) the deterministic analog value *V* associated with the *VRP* sequence is

$$V = [p(VRP) - p(VRP')]\cdot FS \qquad (2.40)$$

where the apostrophe sign (') denotes a logical inversion of the respective binary signal.

Figure 2.9 shows 2-bit deadzone-type quantizer. The "analog random signal generator" and "clocking" mechanism is the same as 1-bit analog/random-pulse converter.

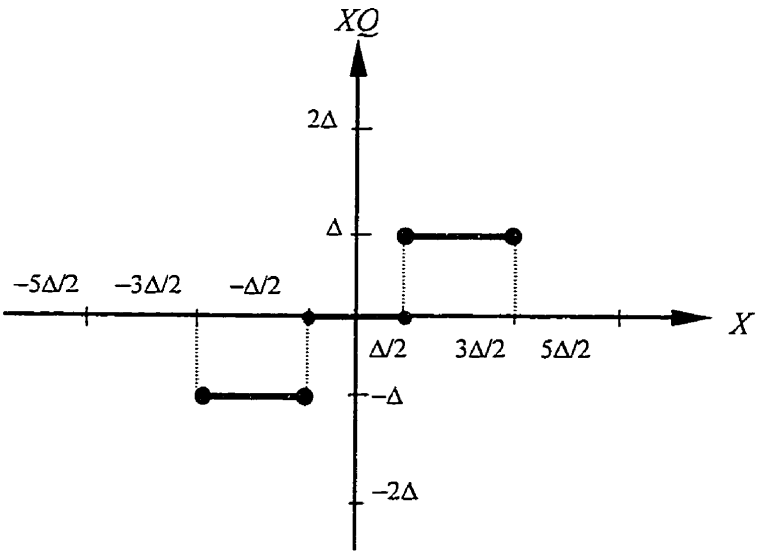


Figure 2.9 2-bit deadzone-type quantizer

Here again, we should note that the domains of input signal v in both 1-bit and 2-bit dithered quantizations are trimmed by $\Delta/2$ on both sides. Figure 2.10 shows the domains for both cases. In this figure, $\Delta = 2\Delta'$, for all the simulations in this thesis we take $\Delta = \Delta' = 1$.

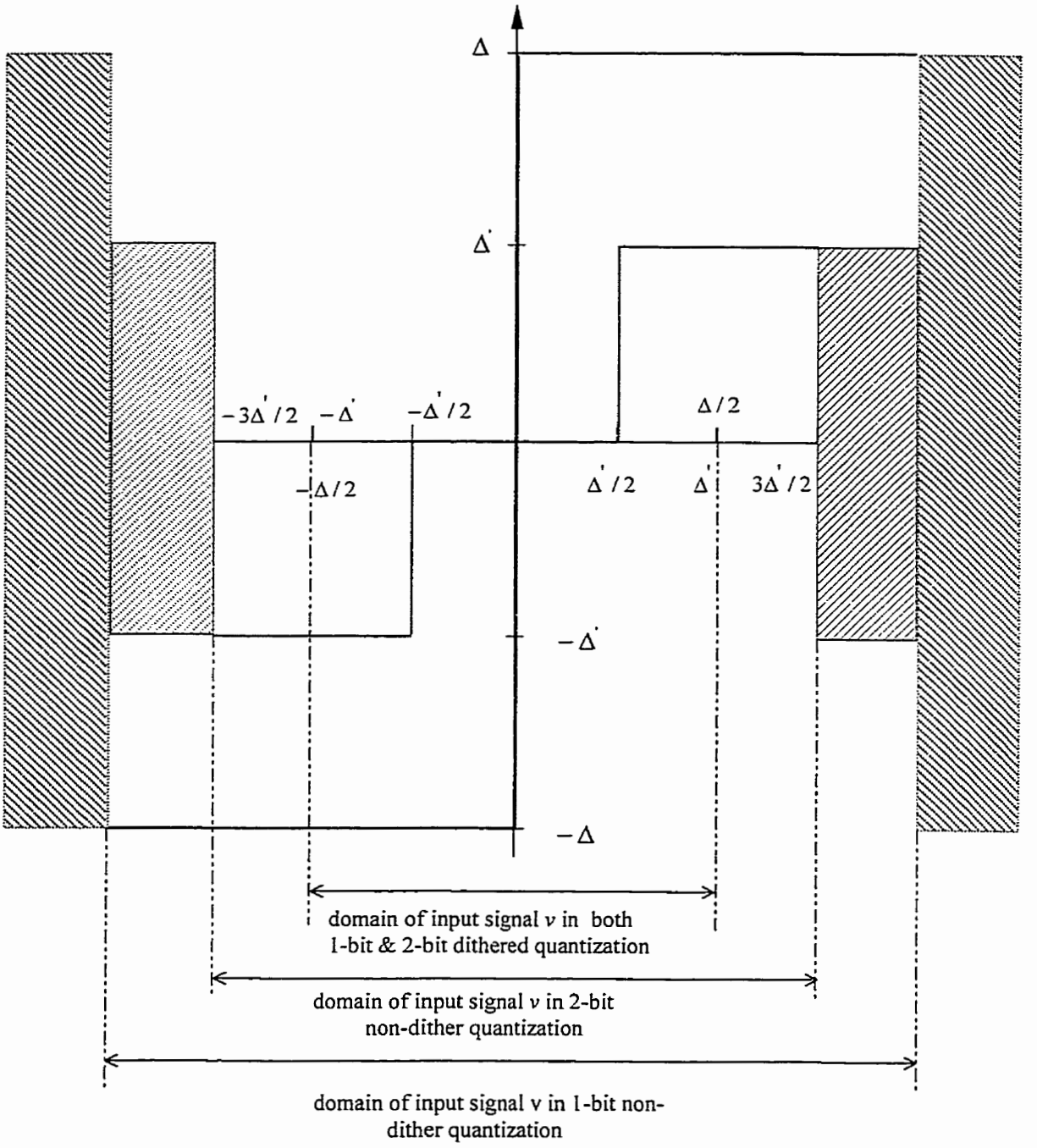


Figure 2.10 Domains of input signal in 1-bit & 2-bit non-dither and dithered quantizations

2.4.3 Simulation Results

Figure 2.11 shows the simulation result of 1-bit analog/random-pulse conversion of a step signal.

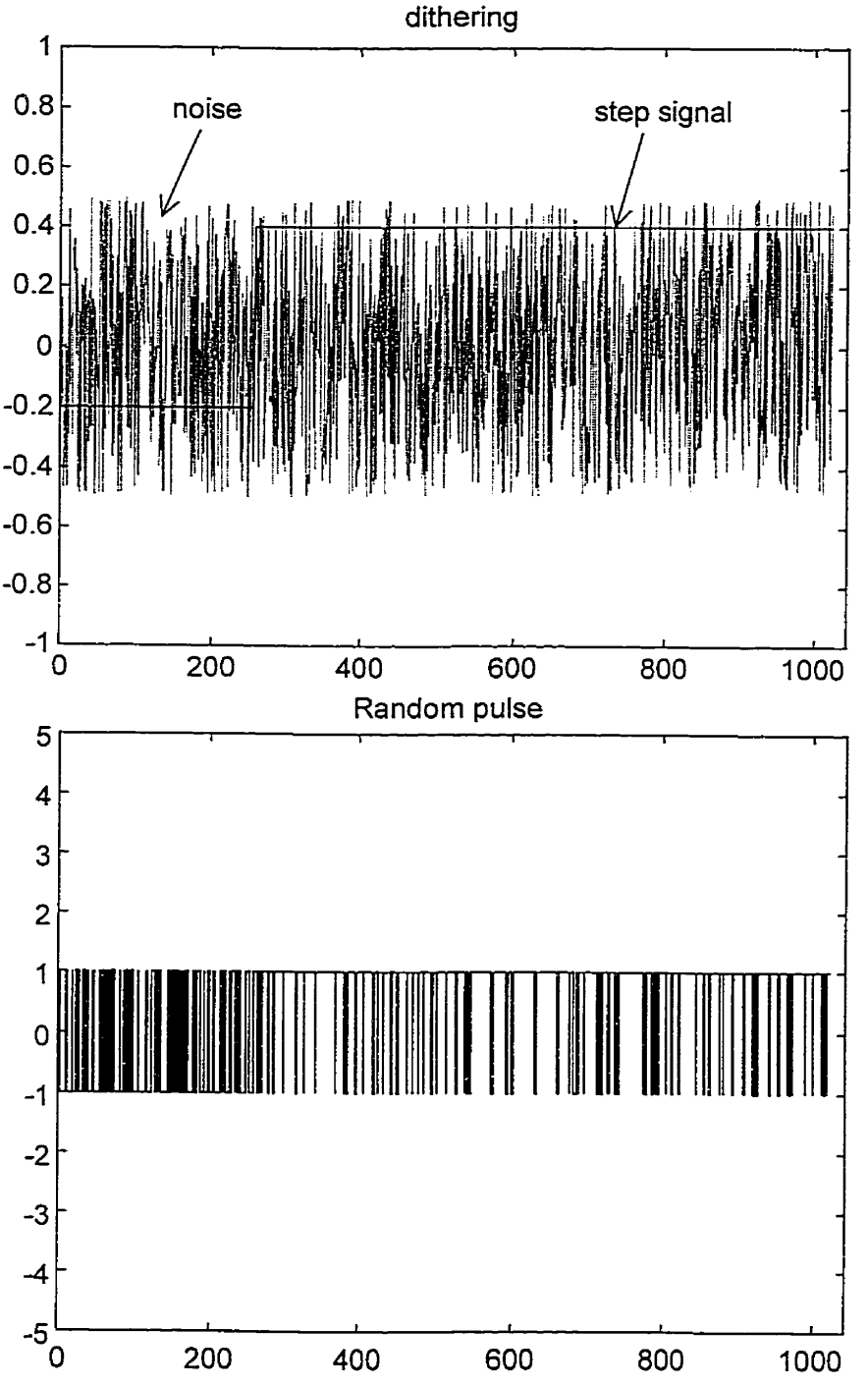


Figure 2.11 1-bit random pulse representation of a step signal

Figure 2.12 illustrates the 2-bit dithered quantization of the same step signal. Here we do not show the dithering for better clarity of the picture. The upper part is the most significant bit (MSB) and the lower part is the least significant bit (LSB).

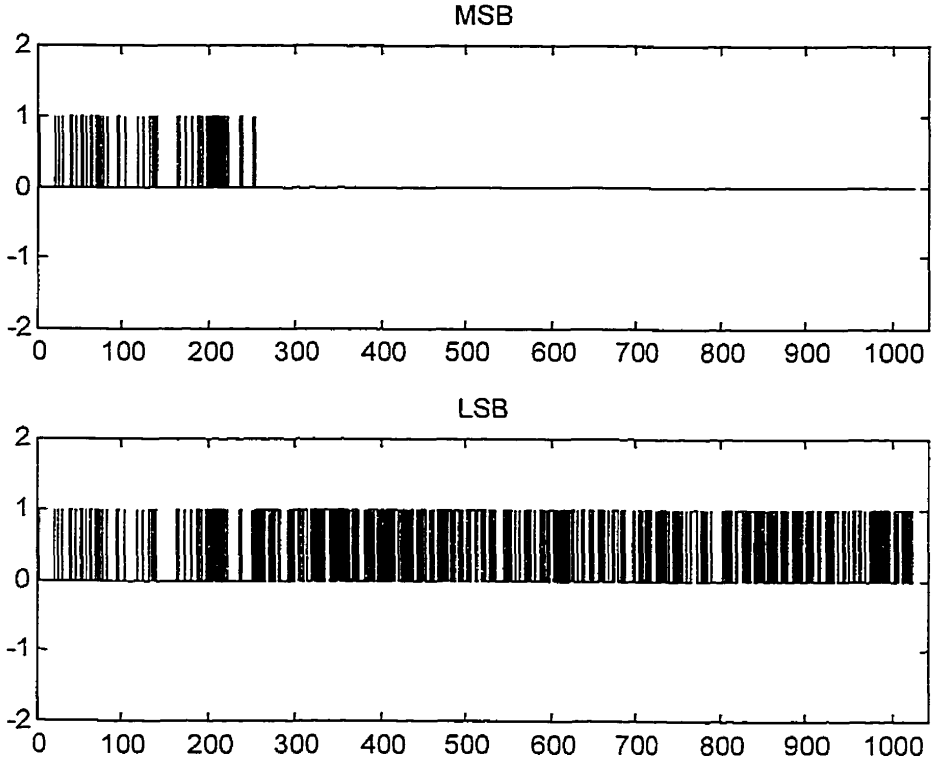


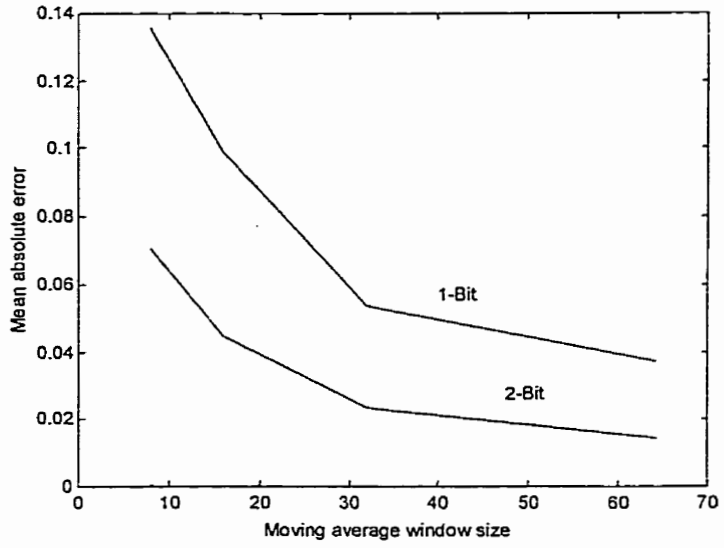
Figure 2.12 2-bit random-pulse representation of a step signal

Nav [bit]	Relative Mean Absolute Error (2-bit vs. 1-bit)	Relative Mean Square Error (2-bit vs. 1-bit)
8	0.5199	0.5219
16	0.4515	0.4461
32	0.4384	0.4588
64	0.3898	0.4112

MS=256 total samples;
Nav: moving average estimation bits;

Table 2.1 Relative errors of 1-bit & 2-bit Analog/Random-pulse conversions

Table 2.1 shows the relative errors (mean absolute and mean square error) of 2-bit & 1-bit analog/random-pulse conversions. Simulation is based on the different moving average window sizes (ranging from 8 to 64). Same white noises sources were used in order to compare the errors.



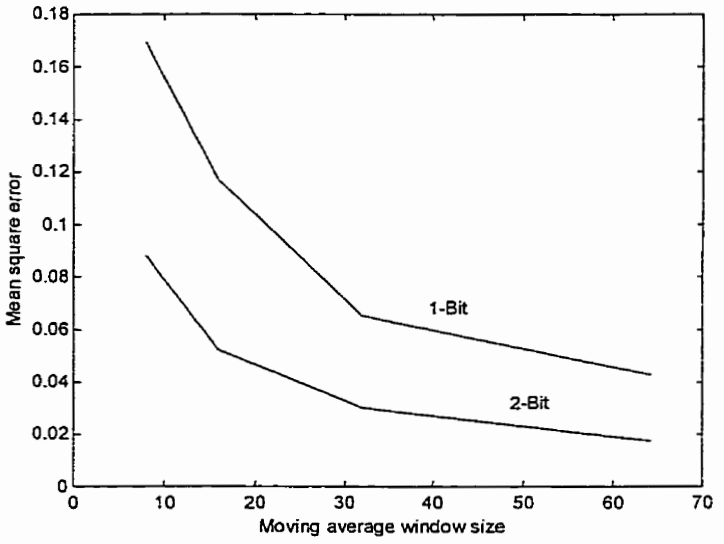


Figure 2.13 1-bit & 2-bit quantization errors

Figure 2.13 shows the comparisons of 1-bit and 2-bit quantization. It is obvious that 2-bit analog/random-pulse converter yields more precise results than 1-bit.

2.5 Dither Signal Generation

In neural network implementation, the technique used to generate noise for the neurons can be crucial. The noise sources must present uncorrelated noise simultaneously to all neurons in the system.

Noise often appears naturally and causes trouble in systems where it is not wanted. Paradoxically, when one attempts to generate artificial noise one often finds that it is difficult to obtain statistical randomness properties good enough for all envisioned application.

The requirements of the noise source can be summarized as follows:

(1) Each noise channel must output analog noise to mimic true thermal noise. The amplitude distribution is uniform of zero mean and the frequency distribution is flat (white noise) up to the operating frequency of the neuron.

(2) The pairwise cross-correlation of two noise channels must have zero mean and variance proportional to the correlation time window. This mimics true thermal noise. This implies that the bit streams from any two channels must not overlap.

A linear feedback shift register (LFSR) can generate a pseudorandom bit stream with good noise-like properties. An N-stage LFSR creates a PRBS of maximal length, $2^N - 1$, when the feedback taps are chosen carefully. Figure 2.14 shows an LFSR which makes an artificial analog noise source.

One useful property of such LFSR is that it has cross-correlation $-1/(2^N - 1)$ (effectively negligible) with a time shifted version of itself, assuming the cross-correlation is calculated after replacing each 1 of the binary bit stream with -1 and each 0 with 1. The time shift must be large enough for the network to settle sufficiently to "forget" the sequence during the anneal cycle before it sees another version of it later. More precisely, during the time that the network is being annealed, there will be a sequence of bits from each channel of the noise generator. Each sequence will start at a different time shift of the full PRBS.

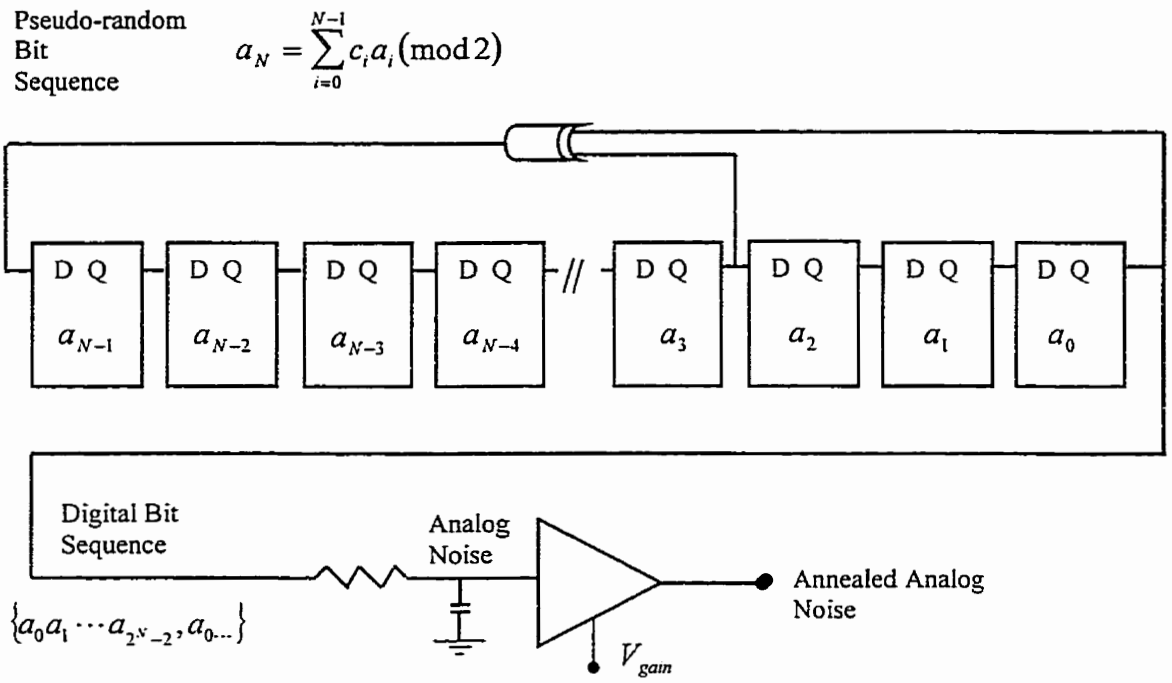


Figure 2.14 Artificial analog noise generation from LFSR
(adapted from [14])

There must be no overlap of any of these portions of the PRBS with any other shifted portion. This ensures that the partial period cross-correlation is unbiased. In practice, this no-over-lap condition is obtained easily with relatively small shift registers because the length of the sequence grows exponentially with the shift register size.

Using combinational logic (exclusive-or gates) to perform modulo 2 addition on the suitably chosen taps of a maximal length shift register can obtain proper shift from the main sequence. So only one shift register is needed for multi-pseudo-random bit streams.(see figure 2.15)

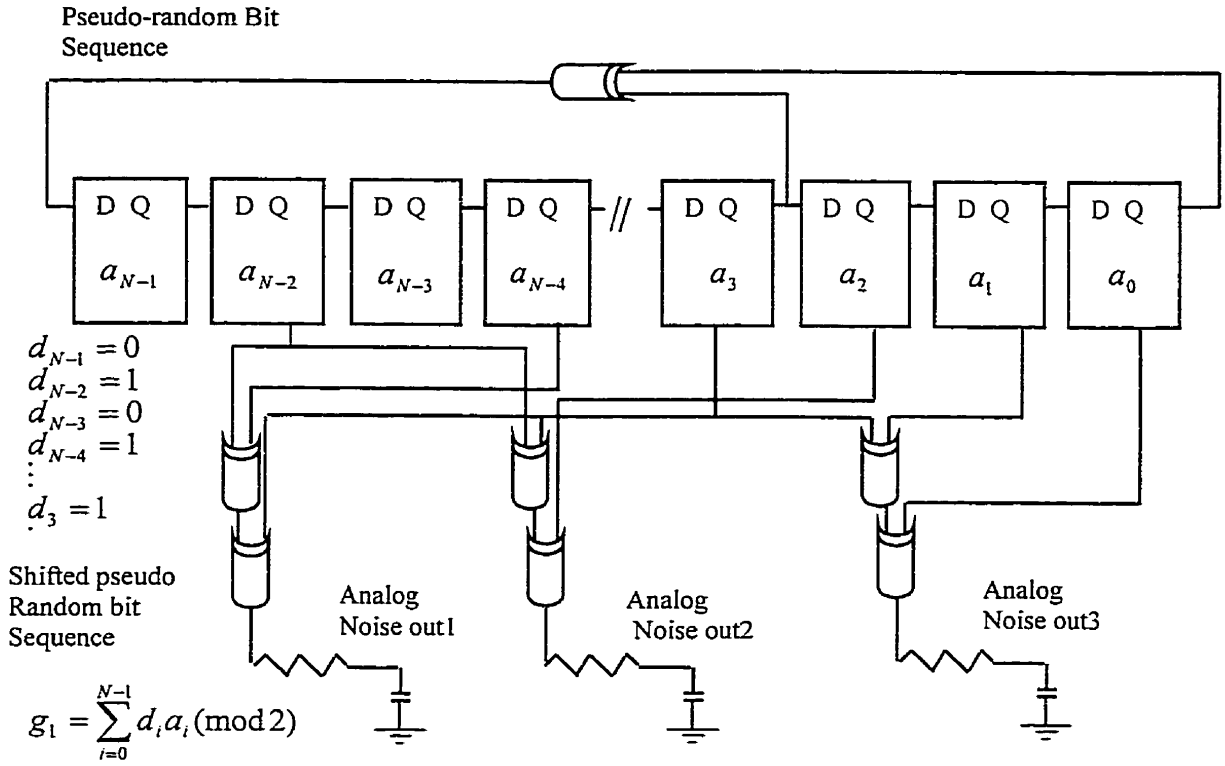
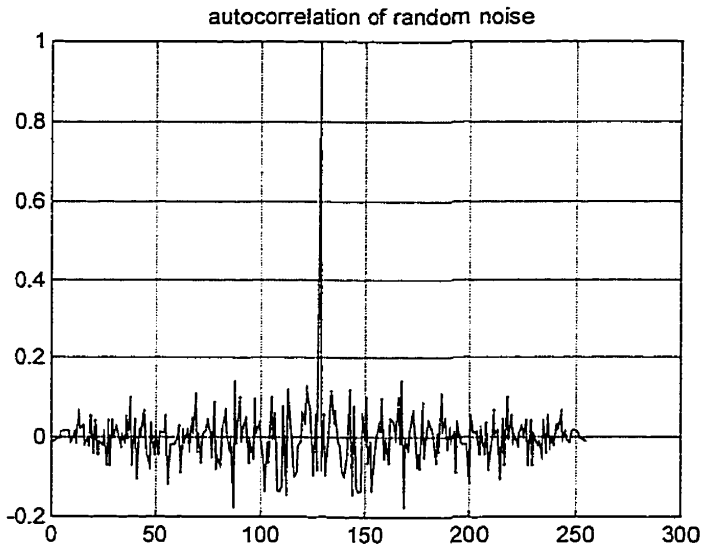
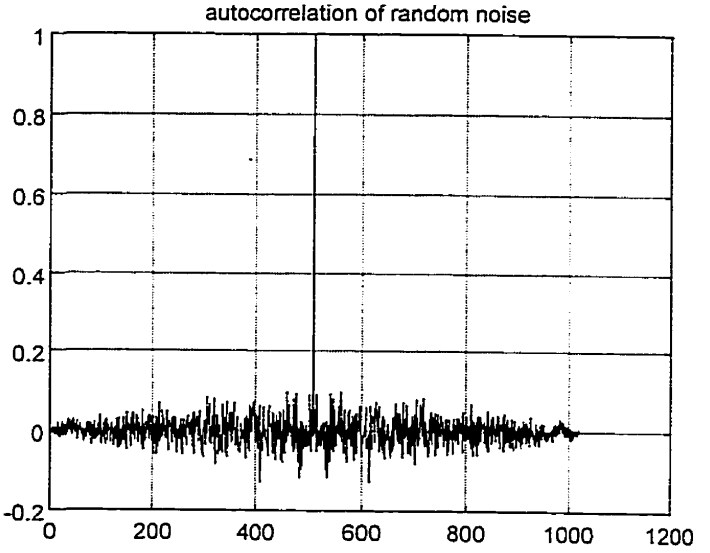


Figure 2.15 Creating multiple pseudo-random bit streams from one LFSR

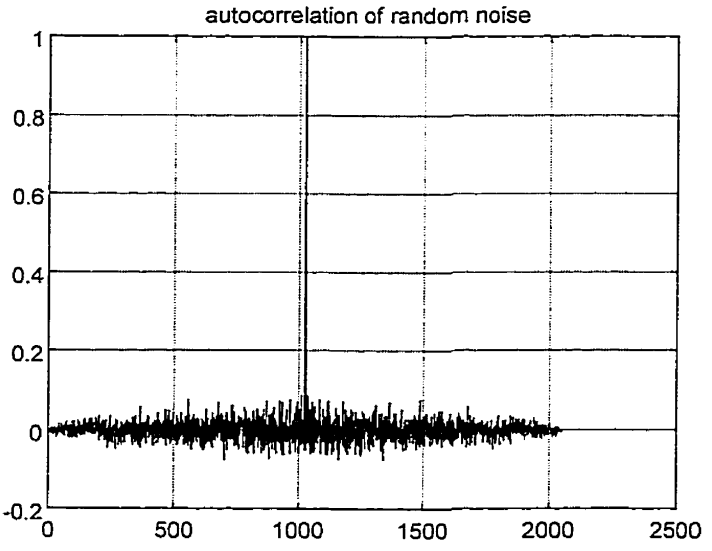
(adapted from [14])



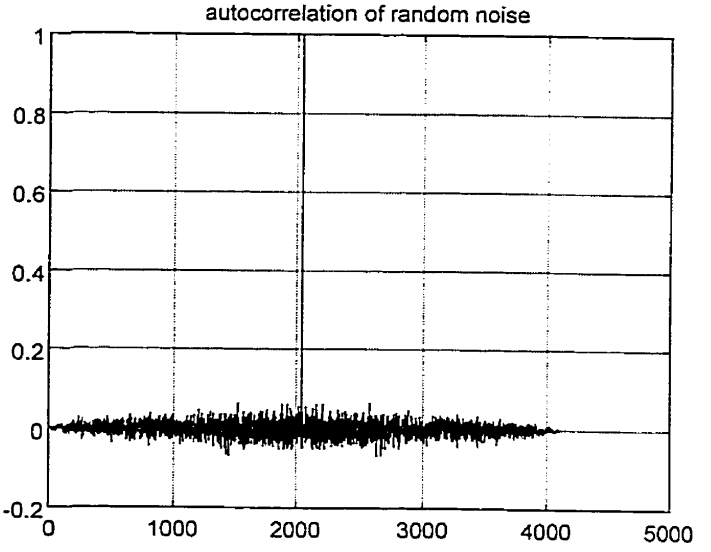
(128 sampling bits)



(512 sampling bits)



(1024 sampling bits)



(2048 sampling bits)

Figure 2.16 Autocorrelation of random noise sequences

Figure 2.16 shows the autocorrelation of random noise sequences used in this research. These noise sequences ranging from 128 sampling bits to 2048 bits are generated by M

atlab using linear congruential method. From the figure, we can see they are white (zero mean) and uniformly distributed. The autocorrelation decreases when sampling points increase, so the longer the total sample bits used in quantization, the better the precision.

2.6 Random Pulse Data Representation

A quantity within the random pulse machine is represented by the probability that a logic level in a clocked sequence will be **ON**. If many clock pulse positions are taken, then the quantity can be considered to be the fraction of **ON** logic levels in a clocked sequence of binary **ON** and **OFF** logic levels, with no pattern in the sequences of logic levels.

The probability that a logic level will be **ON** is an analog variable whose value ranges continuously from 0 to 1. In some computations, this is a convenient range to use, but generally the physical variables in a problem have to be scaled into the allowable range of computer variables. This is similar to the scaling operation in the analog computer, where the physical variables are related to voltages. The actual mapping will determine the hardware required to perform a particular computation. Although many mapping schemes are possible, three popular schemes are: unipolar, two-line bipolar and single-line bipolar.

- Unipolar

  If the quantities are always positive (or always negative), simple scaling is all that is required to bring them within range. Given quantity E in the range $0 \leq E \leq V$, it can be represented by the probability:

$$p(\mathbf{ON}) = \frac{E}{V} \qquad (2.41)$$

  So the maximum value of the range, $E=V$, is represented by a logic level always **ON**, $p(\mathbf{ON})=1$, zero value by its being always **OFF**, $p(\mathbf{ON})=0$, and an intermediate value by its fluctuating randomly, but with a certain probability that it will be **ON** at any particular instant.

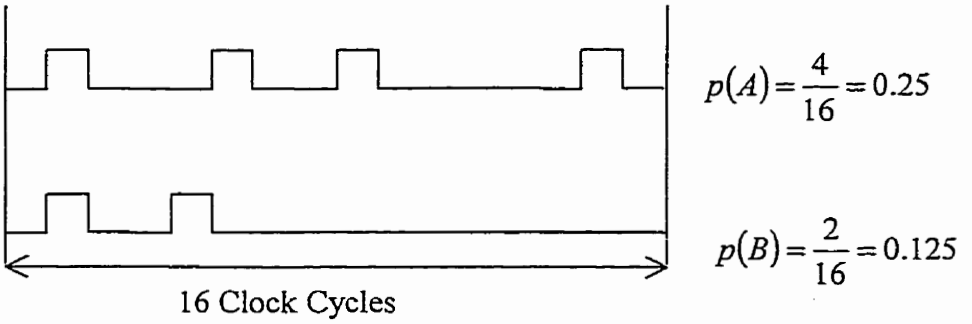


Figure 2.17 Random pulse encoding of an analog signal

- Two-lines Bipolar

It represents both positive and negative quantities by using two sequences of logic levels on separate lines, one representing positive values and the other negative. The line whose probability is weighted positively is called the **UP** line, and the line whose probability is weighted negatively is the **DOWN** line. For a quantity E in the range $-V \leq E \leq V$:

$$p(\text{UP=ON})-p(\text{DOWN=ON})=\frac{E}{V} \qquad (2.42)$$

So that UP line represents that maximum positive quantity is represented by the UP line always ON and the DOWN line always OFF , maximum negative quantity by the UP line always OFF and DOWN line always ON. For intermediate quantities there will be stochastic sequences on one or both lines. Zero quantity is represented by equal probabilities of an ON logic level for both lines.

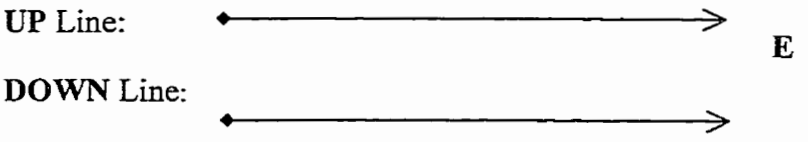


Figure 2.18 Two-lines bipolar representation

- Single-line Bipolar

It represents both positive and negative quantities by using a single line by setting:

$$p(\mathbf{ON})=\frac{1}{2}+\frac{1}{2}\frac{E}{V} \qquad (2.43)$$

So that maximum positive quantity is represented by a logic level always **ON**, maximum negative quantity by a logic level always **OFF**, and zero by a logic level fluctuating randomly with equal probability of being **ON** and **OFF**.

This representation is used in our application since it allows the simplest logical elements to carry out all the normal analog computing operations with both positive and negative quantities.

Chapter 3 – Random Pulse Neural Network Architecture

3.1 Biological Neuron

Understanding the ways in which the brain uses neural systems for information processing serves the key to the design of neural network.

The biological neuron is a complex analog computing device. Its shape and inter-connection characteristics seem to determine its function in brain activity. The neuron basically consists of a cell body, or soma, branching complex extensions that serve as inputs, or dendrites, and the output channel of the cell, or axon. The axon carries an electrical signal to other cells. It connects to the dendrites of these cells through specialized contacts called synapses that can change (positively or negatively) the axon potential. The traditional view holds that the neuron performs a simple threshold function on sum of the weighted input signals. If the result exceeds a certain threshold, a signal emits from the neuron.

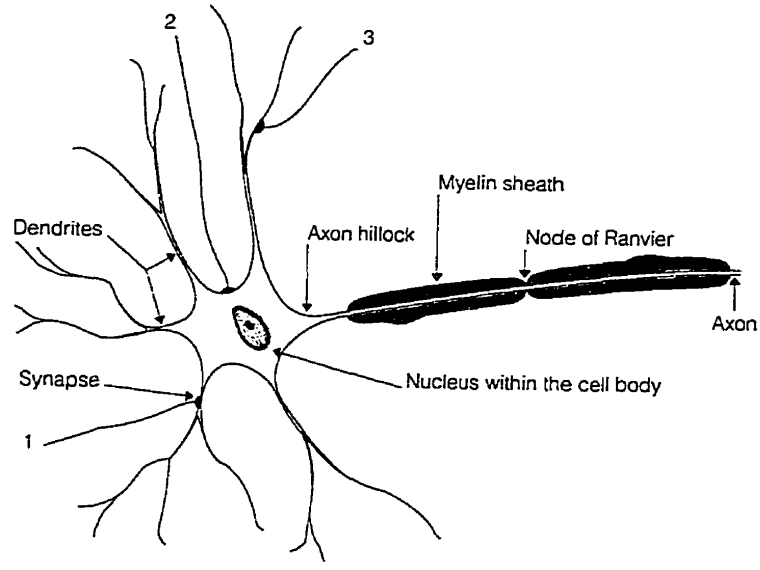


Figure 3.1 Conceptual structure of a biological neuron

(adapted from Katz, "Nerve, muscle and synapse")

1,2, and 3 are axons

The brain is a massively parallel natural computer composed of approximately 10^{11} neurons, with each neuron connected to approximately 10^{4} other neurons. The neural network is sparsely interconnected, but neuroscientists believe that signals can proceed from one neuron to any other neuron by taking a path through other connecting neurons. A neuron fires by sending an electrical impulse that leaves its cell body and reaches the next neuron through the synaptic junction. The next neuron then fires if enough energy is present at its inputs. Neuron's firing rates are very low (about 1,000 pulses per second), yet brain clearly can solve difficult problems of vision and language in less than a second.

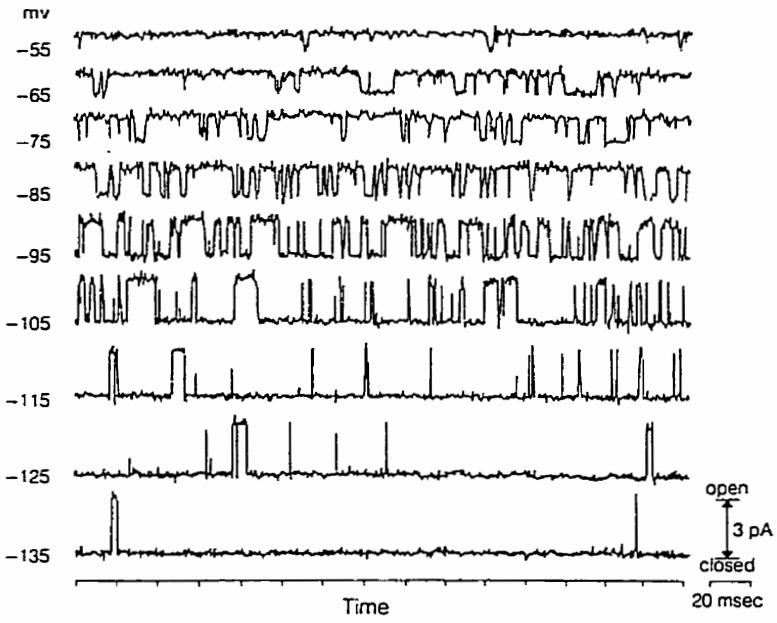


Figure 3.2 Current through nerve membrane as a function of time

(adapted from Keller, "Journal of general physiology",1986)

Figure 3.2 shows the current through nerve membrane for several membrane voltages. Upward steps are due to formation of sodium channels; downward steps are due to the channels' disappearance. The height of a single step is the current in a single channel, which increases approximately linearly with applied voltage, measured with respect to the sodium resting potential.

3.2 Pulse Stream Techniques

Motivated by the encoding schemes employed in biological neural systems, different pulse stream techniques are applied in artificial neural circuitry. From engineering stand point, pulse-train signal has inherent features of higher tolerance and reliability to noises as compared with the case using pure analog or digital signals, i.e., increasing or decreasing number of pulses owing to the noise may not at all be influenced for the amplitude of information. This is because the weight of a single pulse in the pulse-train signal is always unity. In addition, it is known that pulse-train signal processing systems are very simple in construction.

Pulse encoding techniques used in artificial neural networks include:

A. PAM (pulse amplitude modulation)

A_i ($V_i = A_i$ x constant frequency pulsed signal) is modulated in time, reflecting the variation in S_i. This technique, useful when signals are to be multiplexed onto a single line and can be interleaved, is not particularly satisfactory in neural nets. It incurs disadvantages in robustness and susceptibility to processing variations as information is transmitted as analog voltage levels.

B. PWM (pulse width modulation)

PWM represents the instantaneous value of the state S_i as the width of individual digital pulses in V_i. The constant frequency of signaling means that either the leading or trailing edges of neural state signals all occur simultaneously. Power supply lines must be oversized to cope with the high instantaneous currents involved.

C. PPM (pulse phase modulation)

The instantaneous value of the state S_i is represented by the phase difference between the two wave forms-- in other words, by modulating the delay between the occurrence of two pulsed on one or two wires.

D. PFM (pulse frequency modulation)

The instantaneous value of the state S_i is represented as the instantaneous frequency of digital pulses in V_i whose widths are equal. The advantage of pulse-stream technique is apparent, as no analog voltage is present in signal, with information coded as described along

the time axis. This signal is therefore robust, and furthermore can be decoded to an analog value by integration.

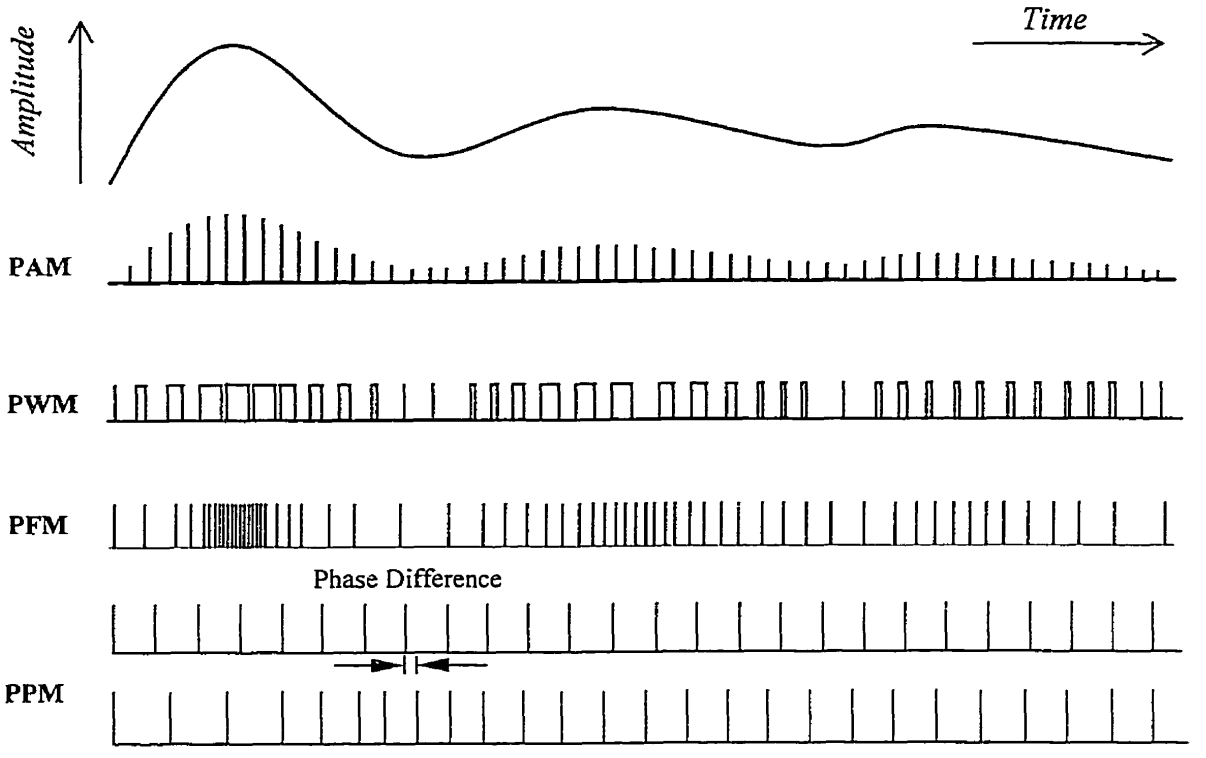


Figure 3.3 Methods for encoding a signal using pulse stream technique

(adapted from [20])

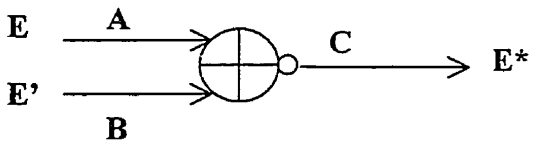
In this thesis, pulse frequency encoding method is employed for the artificial neuron model. Pulse frequency encoding permits:

- simple and rapid multiplication, addition and non-linear transaction using conventional digital circuit element;
- rapid and efficient implementation of learning rules for connection strengths;
- simple interface with standard digital circuitry off the chip;
- rapid, parallel operation;
- high reliability and noise insensitivity;
- power dissipation far lower than that in equivalent analog circuitry;
- reduced circuitry and hence greater number of neurons per chip;

3.3 Arithmetic Operations with PFM Random Pulse Data

3.3.1 Multiplication

1-bit random pulses multiplication is performed by an exclusive-nor gate.



$$C = A*B + \overline{A}*\overline{B} \qquad E* = \frac{EE'}{V}$$

Figure 3.4 Single-line bipolar (1-bit random pulse) multiplier

Its output is **ON** when its two inputs are the same, so that two positive quantities at the inputs, or two negative quantities at the inputs, represent a positive quantity at the output.

The multiplication can be confirmed by examining the relationship between input and output probabilities for the gates shown.

$$p(C) = p(A)p(B) + [1 - p(A)][1 - p(B)] \qquad (3.1)$$

$$p(A) = \frac{1}{2} + \frac{1}{2}\frac{E}{V}$$
$$p(B) = \frac{1}{2} + \frac{1}{2}\frac{E'}{V} \qquad (3.2)$$

so that:

$$p(C) = \frac{1}{2} + \frac{1}{2}\frac{EE'}{V^2} \qquad (3.3)$$

which is normalized multiplication of E by E'.

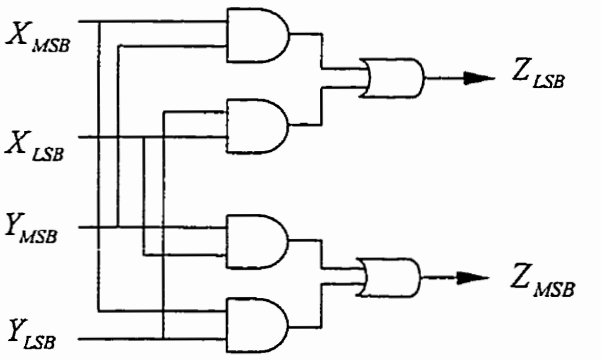
The circuitry for 2-bit random pulses multiplication can be built by combinational logic according to the truth table below . In our case, we use "00" to represent "0", "01" to represent "1", and "10" to represent "–1".

X \ Y		**0**	**1**	**-1**
		00	01	10
0	00	**0** (00)	**0** (00)	**0** (00)
1	01	**0** (00)	**1** (01)	**-1** (10)
-1	10	**0** (00)	**-1** (10)	**1** (01)

$$Z = X * Y$$

Table 3.1 Truth table for 2-bit random pulse multiplication

Figure 3.5 shows the circuitry of 2-bit random pulse multiplier. X_{MSB} stands for the most significant bit of variable X; X_{LSB} stands for the least significant bit of variable X; and Y_{MSB} and Y_{LSB} stand for the MSB and LSB for variable Y respectively.

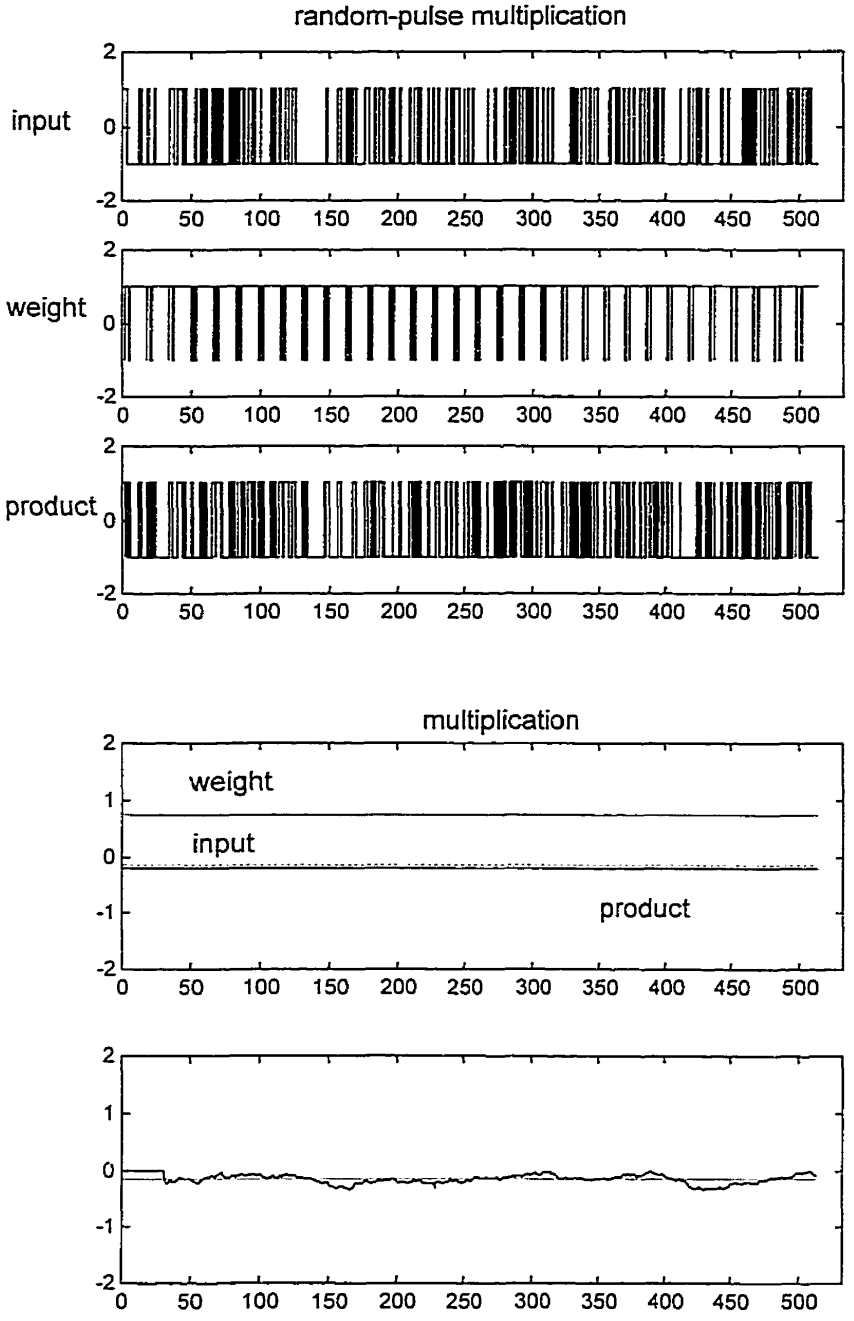


$$Z_{MSB} = X_{LSB} \cdot Y_{MSB} + X_{MSB} \cdot Y_{LSB}$$

$$Z_{LSB} = X_{MSB} \cdot Y_{MSB} + X_{LSB} \cdot Y_{LSB}$$

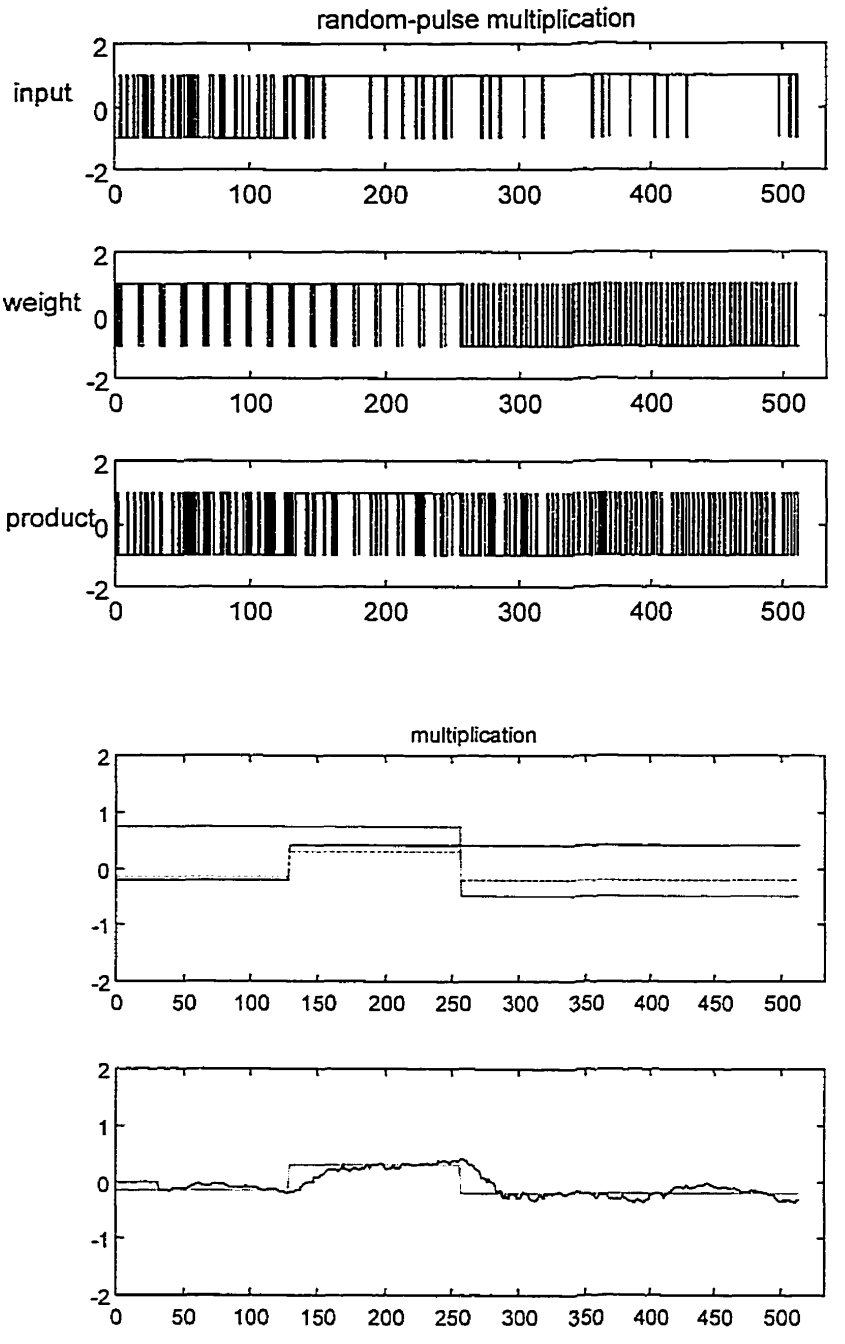
Figure 3.5 2-bit random-pulse multiplier

Figure 3.6 and 3.7 show the simulation results of both 1-bit and 2-bit multiplication. Moving average estimation over 32 bits caused the delays in the following figures.



MS=512 total sampling bits;
Nav=32 moving average window size

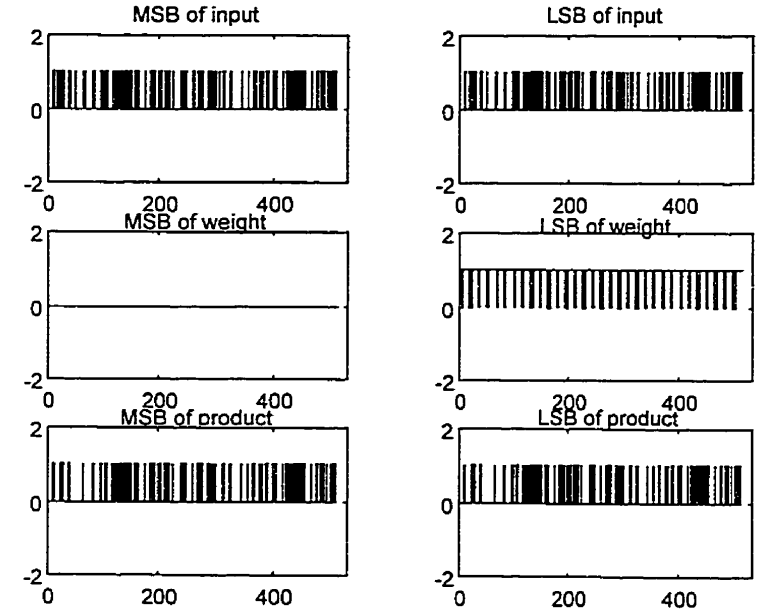
(a)
multiplication of constant signals

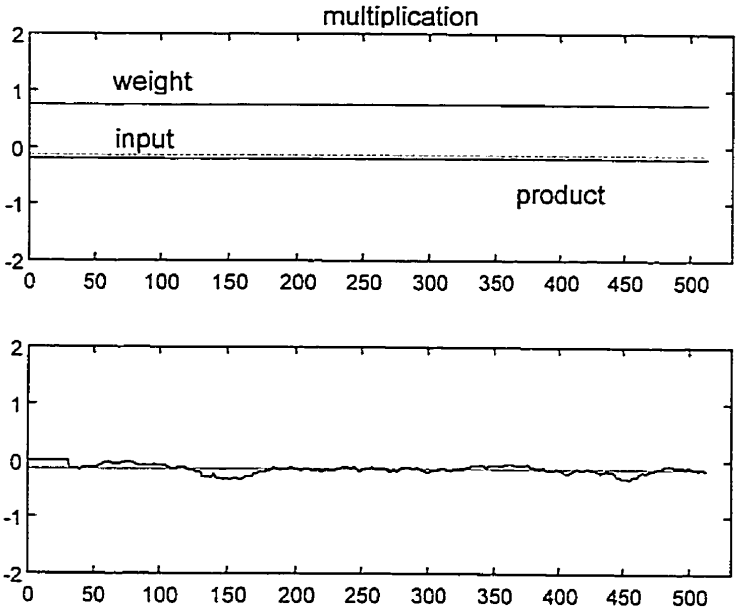


MS=512 total sampling bits;
Nav=32 moving average window size;

(b)
multiplication of variable signals

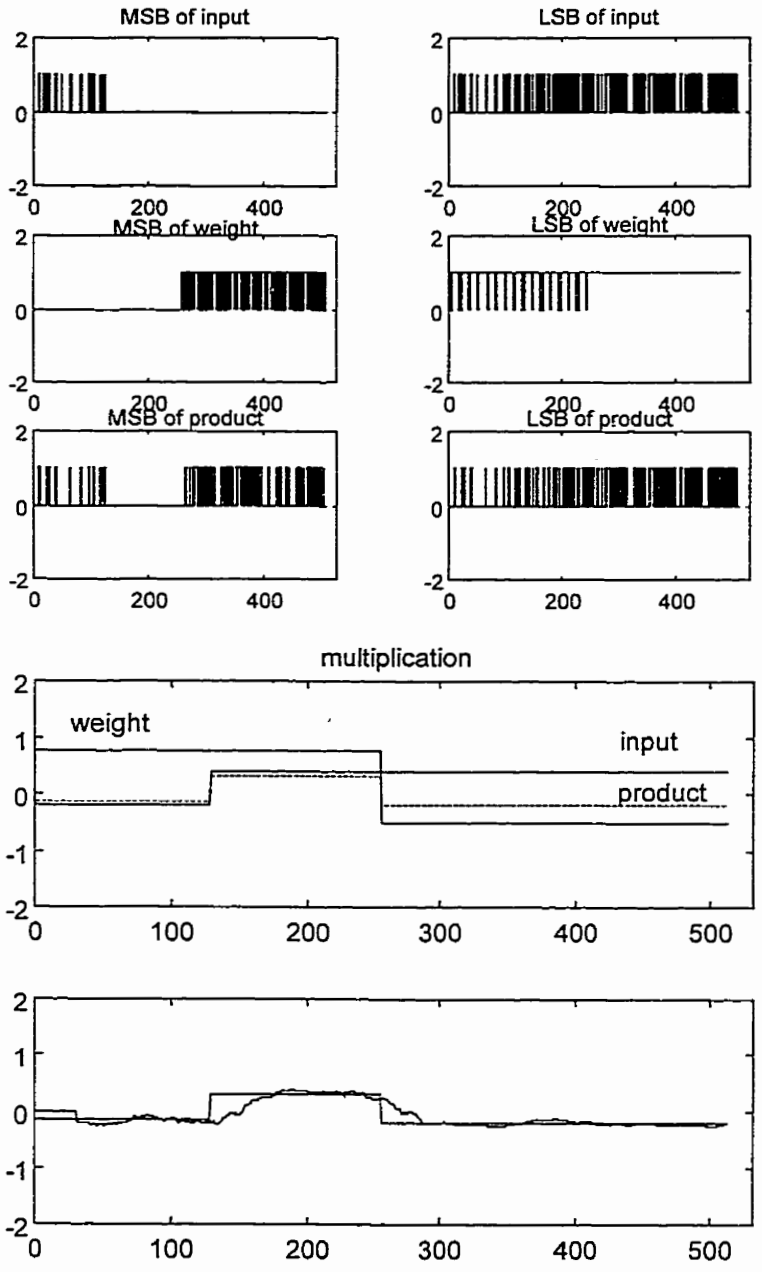
Figure 3.6 1-bit random-pulse multiplication





MS=256 total sampling bits;
Nav=32 moving average estimation bits;

(a)
multiplication of constant signals



(b)
multiplication of variable signals

Figure 3.7 2-bit random-pulse multiplication

Nav	Input (dithered quantization range)										
	-0.49	**-0.4**	**-0.3**	**-0.2**	**-0.1**	**0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.49**
8	0.0793	0.0985	0.1105	0.1254	0.1589	0.1637	0.1615	0.1329	0.1136	0.1126	0.0815
16	0.0148	0.0689	0.0704	0.0911	0.1152	0.1307	0.1227	0.0971	0.0719	0.0605	0.0109
32	0.0137	0.0395	0.0540	0.0702	0.0946	0.1071	0.0951	0.0792	0.0549	0.0448	0.0094
64	0.0080	0.0209	0.0382	0.0548	0.0808	0.0817	0.0697	0.0441	0.0321	0.0151	0.0077

MAE: mean absolute error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.2 Mean absolute error of 1-bit multiplication

Nav	Input (dithered quantization range)										
	-0.49	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4	0.49
8	0.0980	0.1273	0.1292	0.1562	0.1964	0.2054	0.1949	0.1659	0.1388	0.1312	0.1005
16	0.0240	0.0831	0.0859	0.1132	0.1513	0.1622	0.1537	0.1256	0.0885	0.0756	0.0163
32	0.0176	0.0459	0.0671	0.0866	0.1219	0.1299	0.1215	0.1008	0.0690	0.0544	0.0111
64	0.0084	0.0269	0.0534	0.0697	0.0956	0.0966	0.0896	0.0682	0.0527	0.0382	0.0078

MSE: mean square error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.3 Mean square error of 1-bit multiplication

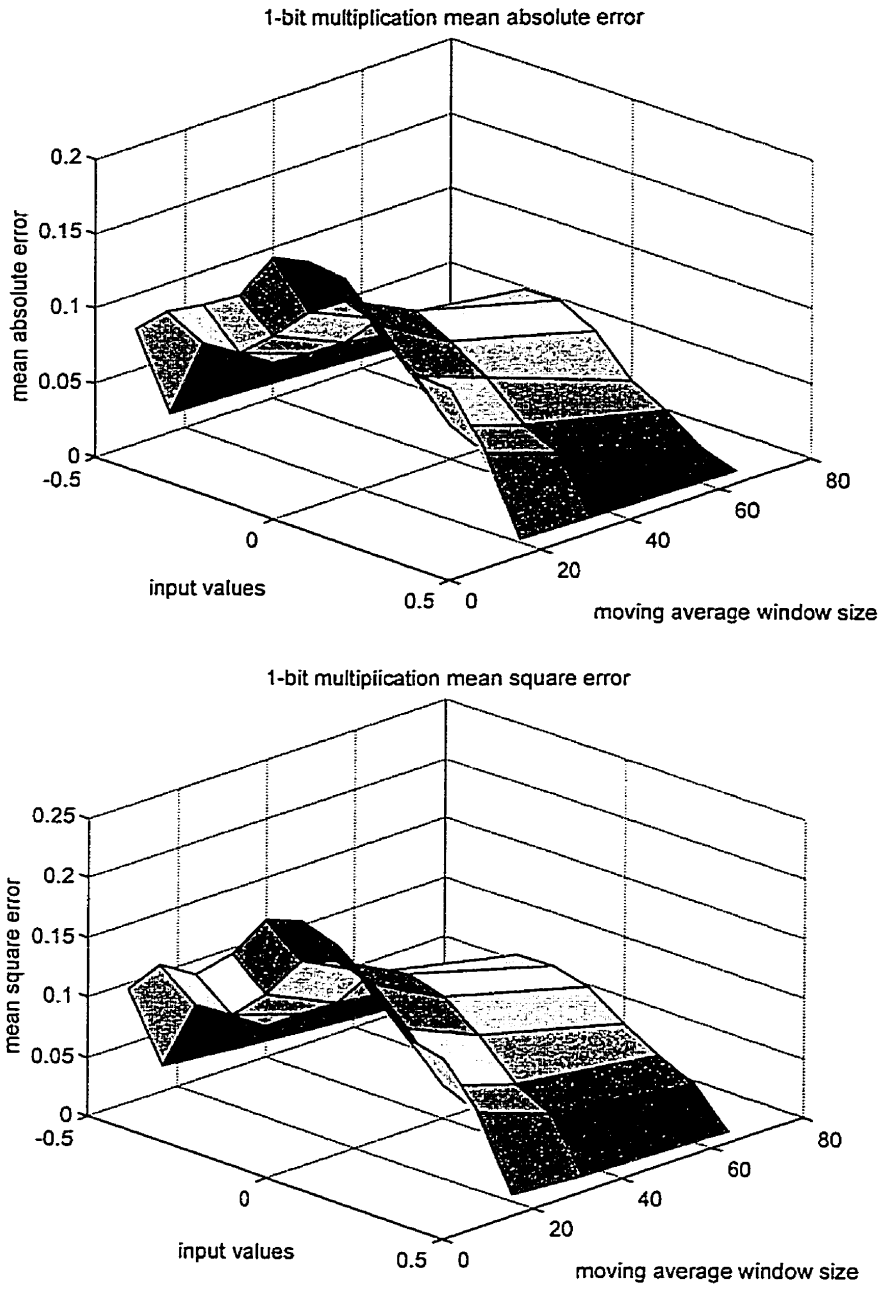


Figure 3.8 1-bit random pulse multiplication error

Nav	Input										
	-0.5	-0.4	-0.3	-0.2	-0.1	0.1	0.2	0.3	0.4	0.5	MAE
8	0.1421	0.1445	0.1329	0.1015	0.0687	0.0704	0.0981	0.1205	0.1465	0.1376	**0.1163**
16	0.0952	0.0990	0.0969	0.0684	0.0372	0.0373	0.0819	0.0938	0.1034	0.1076	**0.0821**
32	0.0843	0.0842	0.0676	0.0419	0.0297	0.0243	0.0577	0.0709	0.0790	0.0810	**0.0621**
64	0.0723	0.0767	0.0514	0.0313	0.0208	0.0117	0.0458	0.0648	0.0797	0.0555	**0.0510**

MAE: mean absolute error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.4 Mean absolute error of 2-bit multiplication

Nav	Input										
	-0.5	-0.4	-0.3	-0.2	-0.1	0.1	0.2	0.3	0.4	0.5	MSE
8	0.1832	0.1785	0.1681	0.1267	0.0742	0.0816	0.1296	0.1578	0.1798	0.1743	0.1453
16	0.1313	0.1295	0.1240	0.0873	0.0503	0.0489	0.1002	0.1119	0.1234	0.1283	0.1035
32	0.1040	0.1112	0.0965	0.0613	0.0351	0.0296	0.0718	0.0833	0.0929	0.0983	0.0784
64	0.0821	0.1001	0.0664	0.0403	0.0229	0.0155	0.0558	0.0704	0.0822	0.0699	0.0606

MSE: mean square error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.5 Mean square error of 2-bit multiplication

Simulation results in the above four tables are based on multiplications with a fixed weight 0.75. In order to compare, same white noise source and same input range are used for all occasions. From the above four tables, we can see 2-bit random pulse multiplication yields less error comparing with 1-bit multiplication.

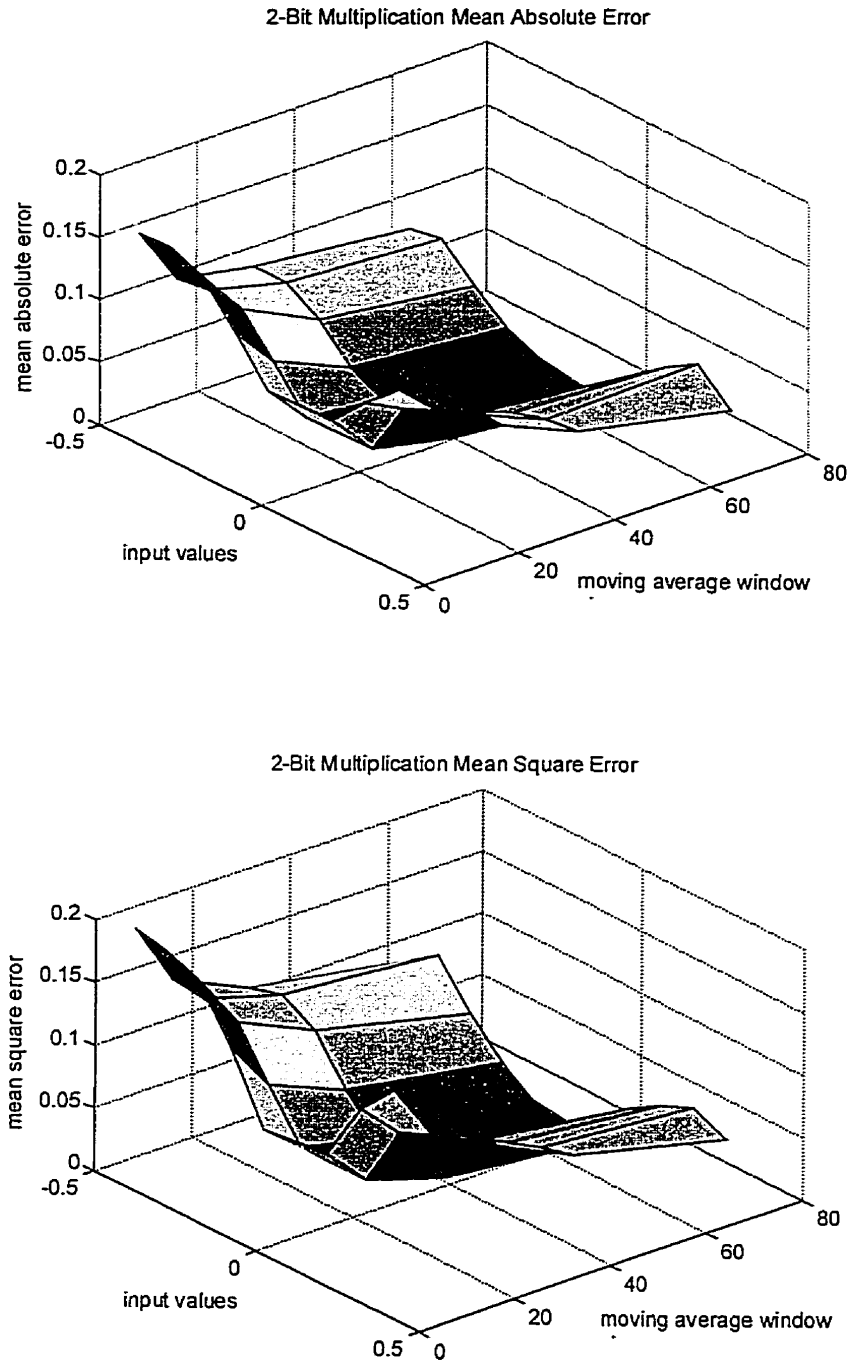


Figure 3.9 2-bit random pulse multiplication error

3.3.2 Addition

Stochastic summers may be regarded as switches which, at a clock pulse, randomly select one of the input lines and connect it to the output. The output line denotes the sum of the quantities represented by the input lines. The sum is weighted according to the probability (1/m in Figure 3.10) that a particular input line will be selected.

The random scanning acts as a "stochastic isolator" which removes unwanted correlations between sequences with similar patterns. The random scanning signals S_i are uniformly distributed having the same probability $p(S_i) = \frac{1}{m}$. Because of this scanning the multiplexed samples are statistically independent. The analog meaning of the output sequence y is :

$$Y = \frac{X_1 + X_2 + \cdots X_m}{m} \qquad (3.4)$$

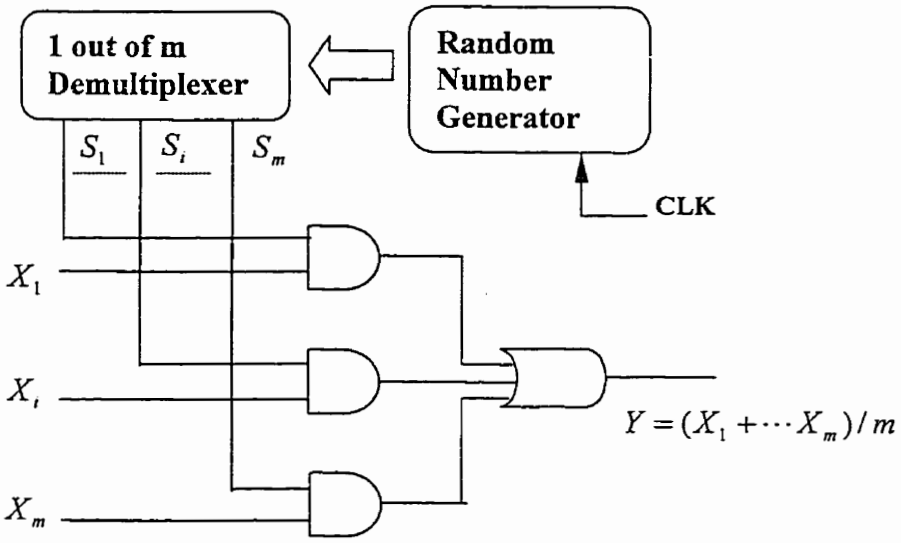


Figure 3.10 Circuitry of 1-bit random pulses addition

Figure 3.10 shows m-input stochastic summer with equal weighting. The addition can be confirmed by examining the relationship between input and output probabilities. Assuming symmetrically distributed digital noise, we have:

$$p(Y) = \frac{1}{m} p(X_1) + \cdots + \frac{1}{m} p(X_m) \qquad (3.5)$$

and hence:

$$\frac{E_Y}{V} = \frac{1}{m}\frac{E_{X_1}}{V} + \cdots + \frac{1}{m}\frac{E_{X_m}}{V} \qquad (3.6)$$

so that:

$$E_Y = \frac{1}{m}(E_{X_1} + \cdots + E_{X_m}) \qquad (3.7)$$

which is normalized addition for the single-line bipolar case.

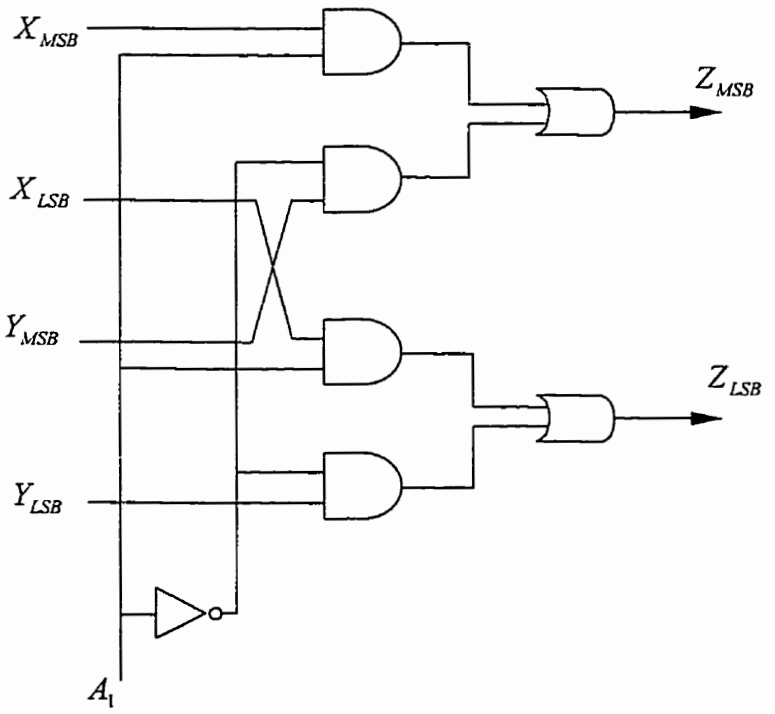
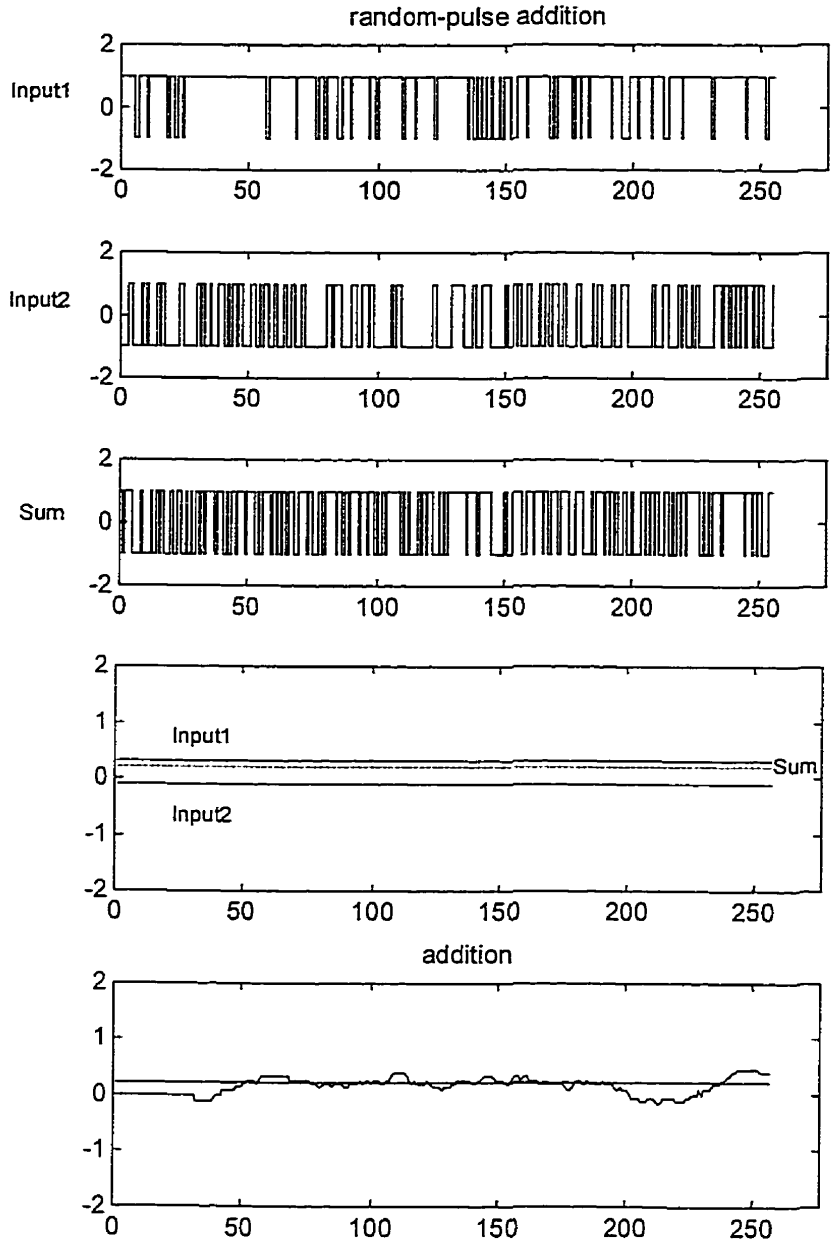


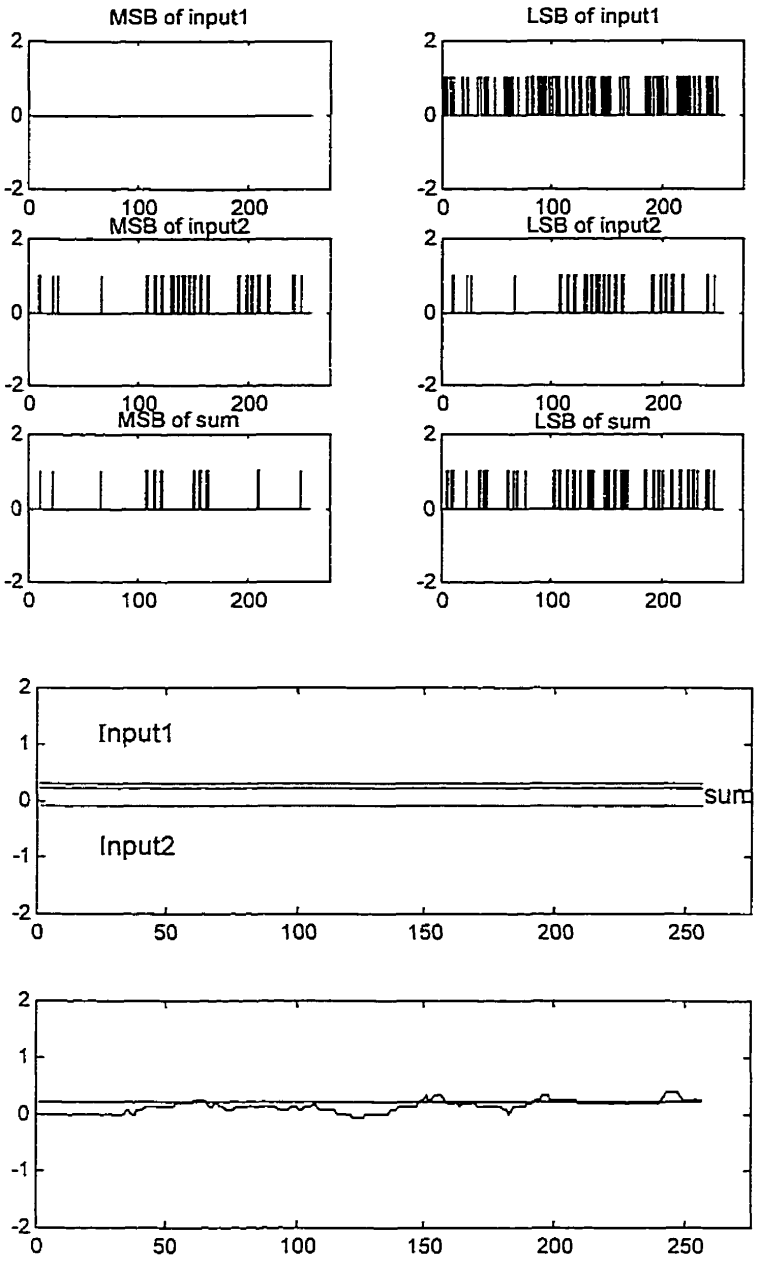
Figure 3.11 Circuitry of 2-bit random pulses addition

Figure 3.11 shows the circuitry of 2-bit random pulses addition. X_{MSB} stands for the most significant bit of variable X; X_{LSB} stands for the least significant bit of variable X; Y_{MSB} and Y_{LSB} stand for the MSB and LSB of variable Y respectively.



MS=256 total sampling bits;
Nav=32 moving average window size;
Input1=0.3,Input2=-0.1;

Figure 3.12 1-bit random-pulse addition



MS=256 total sampling bits;
Nav=32 moving average window size;
Input1=0.3,Input2=-0.1;

Figure 3.13 2-bit random-pulse addition

Figure 3.12 and 3.13 show the 1-bit and 2-bit random-pulse addition of two input values. The delays are caused by moving average estimation.

Nav	Input									
	-0.4	-0.3	-0.2	-0.1	0	0.1	0.2	0.3	0.4	MAE
8	0.2139	0.2751	0.3030	0.2998	0.3082	0.2982	0.3303	0.2861	0.2400	**0.2838**
16	0.1432	0.1837	0.2320	0.2261	0.2131	0.2044	0.2137	0.1983	0.1702	**0.1983**
32	0.0919	0.1254	0.1555	0.1444	0.1259	0.1131	0.1296	0.1067	0.0954	**0.1210**
64	0.0623	0.0899	0.1151	0.1040	0.0827	0.0908	0.1140	0.0953	0.0818	**0.0929**

MSE: mean square error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.6 Mean absolute error of 1-bit addition

Nav	Input									
	-0.4	**-0.3**	**-0.2**	**-0.1**	**0**	**0.1**	**0.2**	**0.3**	**0.4**	**MSE**
8	0.2882	0.3249	0.3602	0.3686	0.3702	0.3719	0.3940	0.3514	0.3031	**0.3481**
16	0.1811	0.2209	0.2662	0.2618	0.2495	0.2437	0.2590	0.2431	0.2029	**0.2365**
32	0.1125	0.1529	0.1881	0.1782	0.1549	0.1477	0.1691	0.1453	0.1191	**0.1520**
64	0.0781	0.1089	0.1390	0.1172	0.0978	0.1035	0.1345	0.1128	0.0966	**0.1098**

MSE: mean square error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.7 Mean square error of 1-bit addition

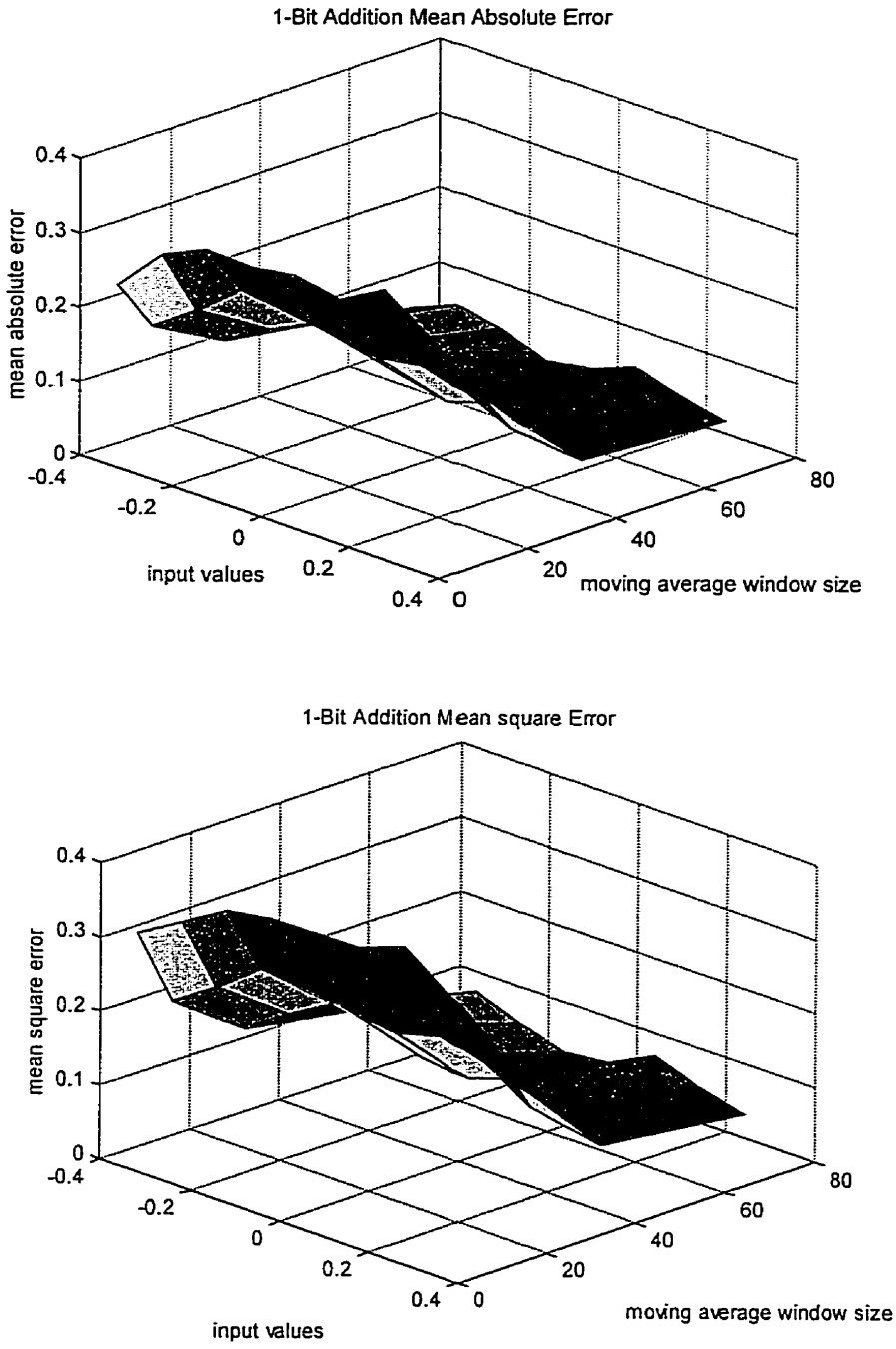


Figure 3.14 1-bit random pulse addition error

Nav	Input										
	-0.5	-0.4	-0.3	-0.2	-0.1	0.1	0.2	0.3	0.4	0.5	MAE
8	0.2564	0.2380	0.2596	0.2347	0.1624	0.1697	0.2442	0.2355	0.2363	0.2564	**0.2293**
16	0.1791	0.1634	0.1700	0.1738	0.1218	0.1385	0.1842	0.1856	0.1702	0.1791	**0.1666**
32	0.1429	0.1269	0.1197	0.1341	0.0839	0.1069	0.1479	0.1591	0.1359	0.1429	**0.1300**
64	0.0961	0.0865	0.0753	0.0950	0.0565	0.0748	0.0984	0.1024	0.0788	0.0961	**0.0860**

MAE: mean absolute error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.8 Mean absolute error of 2-bit addition

Nav	Input										
	-0.5	-0.4	-0.3	-0.2	-0.1	0.1	0.2	0.3	0.4	0.5	MSE
8	0.3159	0.3076	0.3148	0.2829	0.2011	0.2510	0.2917	0.3021	0.3098	0.3159	**0.2893**
16	0.2272	0.2159	0.2155	0.2109	0.1463	0.1862	0.2215	0.2314	0.2258	0.2272	**0.2108**
32	0.1721	0.1592	0.1452	0.1523	0.0975	0.1366	0.1712	0.1795	0.1714	0.1721	**0.1557**
64	0.1124	0.1092	0.0858	0.1164	0.0673	0.0892	0.1158	0.1160	0.0962	0.1124	**0.1021**

MSE: mean square error;
MS=256 total samples;
Nav: moving average window size;
Use 10 different pairs per calculation;

Table 3.9 Mean square error of 2-bit addition

Simulation results in the above four tables are based on addition with a fixed value –0.1. In order to compare, same white noise source and same input range are used for all occasions. From the above four tables, we can see 2-bit random pulse addition has better resolution than 1-bit addition. However additions cause bigger errors than multiplications.

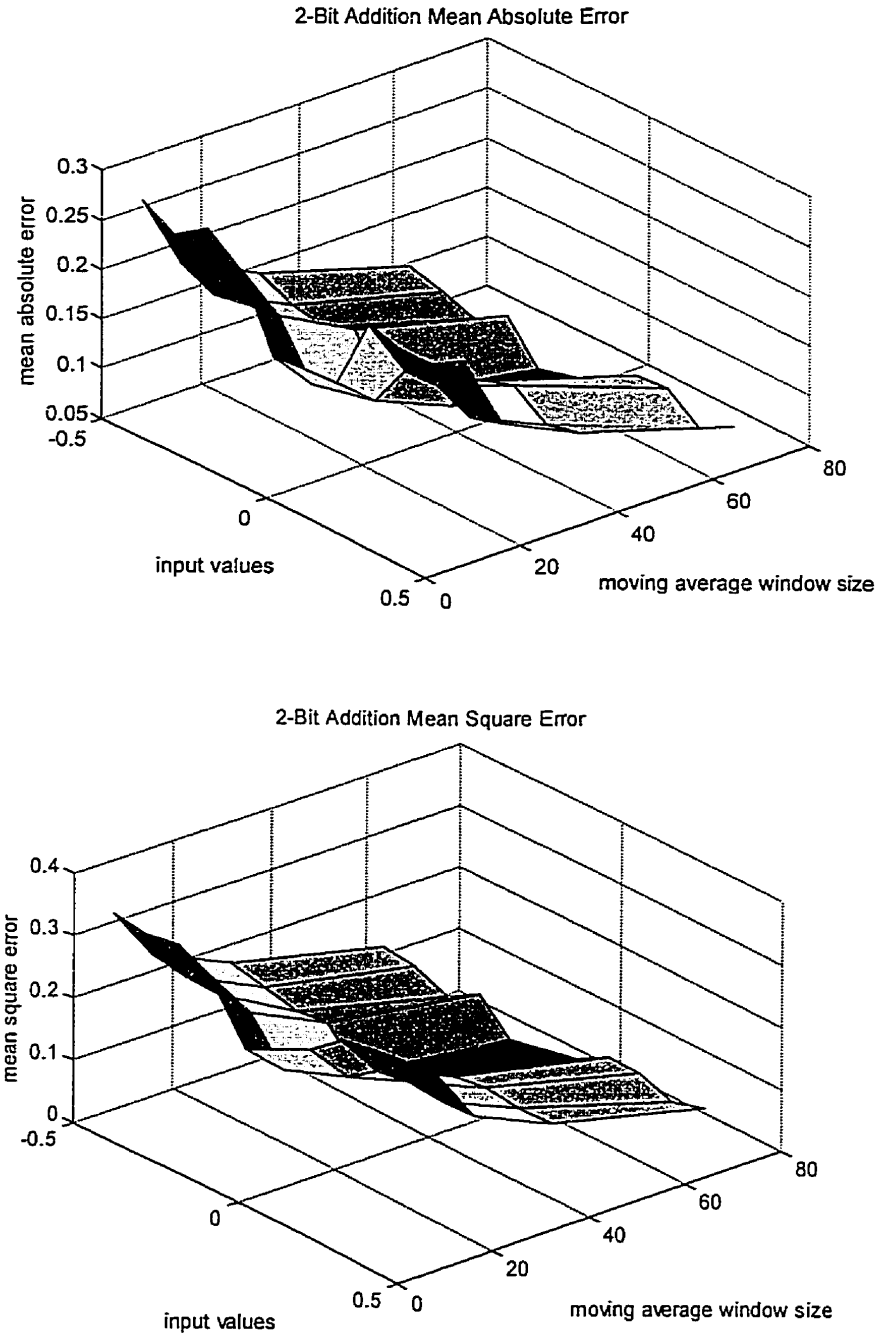


Figure 3.15 2-bit random pulse addition error

3.3.3 Inversion

A conventional logical inverter, with an output that is the complement of its input, performs the same function when used as a stochastic element.

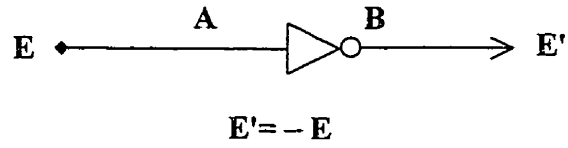


Figure 3.16 Inversion of random pulse data

Consider the relationship between the probability that its output (E') will be ON, p(B), and the probability that its input (E) will be ON, p(A); since the two cases are mutually exclusive, the sum of the probabilities is 1; or:

$$p(B) = 1 - p(A) \tag{3.8}$$

Thus the probabilities and the quantities they represent are:

$$p(A) = \frac{1}{2}\frac{E}{V} + \frac{1}{2}$$

$$p(B) = \frac{1}{2}\frac{E'}{V} + \frac{1}{2} \tag{3.9}$$

Hence: $$E' = -E \tag{3.10}$$

3.4 Random Pulse/Analog Conversion

The output of random pulse machine is required to be produced corresponding to the probability of occurrence of an input pulse. This is called Random-Pulse/Analog converter. A random-pulse stream can be viewed in probabilistic terms as a deterministic signal with superimposed noise. The output interface must be able to reject the noise component, and give a measure of the mean value of the sequences generating probability. Since the variation rate of the deterministic signal in neural network is relatively low, the signal can be substantially separated from the noise by low-pass filtering.

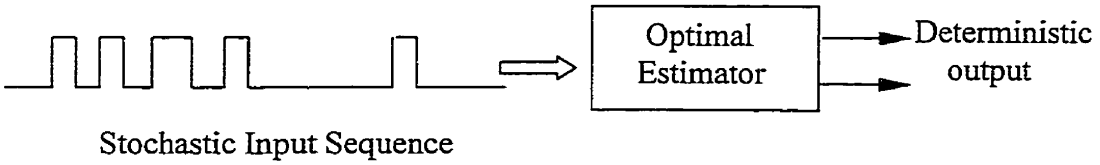


Figure 3.17 Basic estimation problem

If N Consecutive clock intervals of a Bernoulli sequence are examined, and the number of ON logic levels counted, then the ratio of the count to the number of clock intervals give an estimate of the sequences generating probability. When the sequence represents a fixed quantity (the corresponding Bernoulli sequence is stationary), the accuracy of the measurement can be increased by increasing the sample size, N. The price paid for the increase in accuracy is the length of time required to make the measurements. With a time-varying input signal the output interface must have the ability to track the signal continuously, or else the higher frequency components of the wave form will be lost. To do this requires the calculation of a short time of moving average.

To compute an average of the data requires taking N observations of data and then forms a summation of the weighted readings. This is described as the equation:

$$S_t = a_0 A_N + a_1 A_{N-1} + a_2 A_{N-2} + \cdots + a_{N-1} A_1 \quad (3.11)$$

where S_t is the computed average at time t, i.e.

$$S_t = \sum_{i=0}^{N-1} a_i A_{N-1} \quad (3.12)$$

For the average to be unbiased:

$$\sum_{i=0}^{N-1} a_i = 1 \quad (3.13)$$

If the subscript i is taken as referring to clock intervals in a digital system, the weighting coefficients can be plotted on a graph against time (see Figure 3.18), and the relative importance which the average assigns to consecutive readings can be readily assessed. When all the coefficients, a_i, are made equal, it can be described as:

$$\sum_{i=0}^{N-1} a_i = Na = 1 \qquad (3.14)$$

i.e.

$$a_i = \frac{1}{N} \text{ for all } I \qquad (3.15)$$

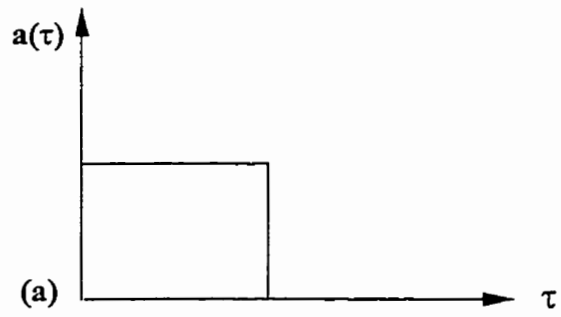


Figure 3.18 Weighting coefficient summing to unity

To form unbiased moving average of a stochastic sequence, the presence or absence of a pulse in all of N adjacent clock periods is recorded and stored, and the average pulse rate calculated. The next clock interval of the sequence is then interrogated and the average is recalculated over the N most recent clock periods. The information contained in the first interval is lost. If P_N is the estimate of the generating probability, p, over N clock intervals, and A_i, (0,1) is the value of the logic level at the ith clock pulse, then the short-time averaging technique can be described by the equation:

$$P_N = \frac{1}{N}\sum_{i=1}^{N} A_i = \frac{1}{N}(\sum_{i=1}^{N-1} A_i + A_N)$$

$$P_N = P_{N-1} + \frac{A_N - A_0}{N} \qquad (3.16)$$

The smaller the sample size, N, the more effect the new value of A_i has on the estimate. When the average is taken over a short time interval the estimated probability, P_N, responds quickly to change and is able to accommodate signals with a high harmonic content. When the value of N is increased the accuracy improves but the bandwidth is restricted.

The disadvantage of using this technique for filtering lies in the requirement to store all the logic levels present in the previous N clock intervals. However, since only the first and last levels are used in the calculation at any one time, a serial in/serial out shift register of length N can be chosen as the storage medium. Using registers, moving average estimation can be designed as the circuit below. Metal oxide semiconductor shift registers make this approach feasible for both high and low accuracy stochastic systems.

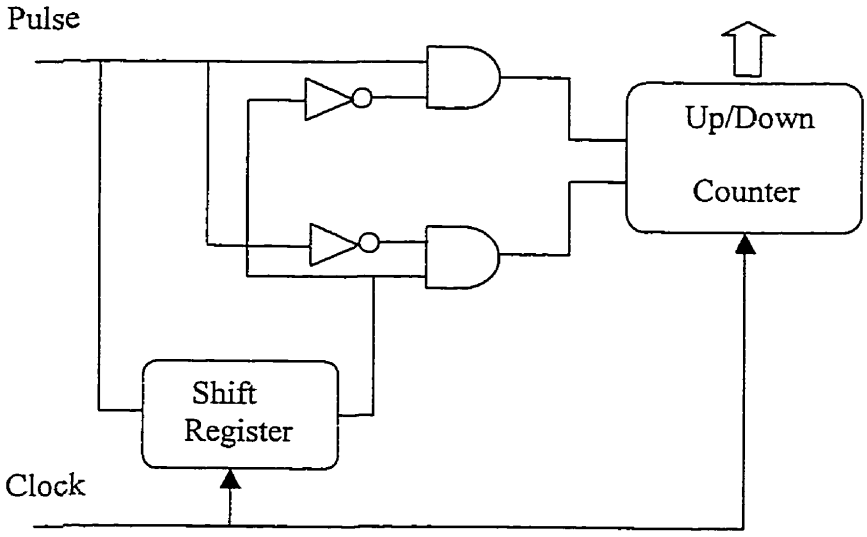


Figure 3.19 Circuitry for moving average estimation

The operation of averaging implies a smoothing out of any local changes in the data. If new input data were suddenly to jump in value, the effect on the average would be a slow change, with the output only reaching the new value after the new data had dominated the averaging sample space. The output at an intermediate time, t, clock periods later is given by:

$$S(t) = S(0) + p\sum_{i=0}^{t} a_i \qquad (3.17)$$

where p is the step change in the input value. The time-lag between input is a reciprocal measure of the speed of response of the particular averaging operation and is directly related to the weighting coefficients. The time-lag, L, can be defined in clock periods as

$$L = \frac{\sum_{i=0}^{\infty} i a_i}{\sum_{i=0}^{\infty} a_i} \qquad (3.18)$$

This formula states that the sum, *S(t)*, computed with certain weighting coefficients, is the average of the data at a time L clock periods ago.

The output interface of a stochastic computer is designed to estimate the probability of occurrence of a logic '1' at any clock interval. It does this by summing the outcome of a number of independent samples, where the samples are taken from successive clock intervals. The variance of a sum random variables is equal to the sum of the individual variances, i.e..

$$\sigma_s^2 = \sigma_1^2 + \sigma_2^2 + \sigma_3^2 + \sigma_4^2 + \ldots \qquad (3.19)$$

when each sample has its own weighting coefficient the equation becomes

$$\sigma_s^2 = \sum_{i=1}^{N} a_i^2 \sigma_i^2 \qquad (3.20)$$

As in our case, if the variances, σ_i^2, are all equal, it can be simplified to:

$$\sigma_s^2 = \sigma^2 \sum_{i=0}^{N} a_i^2$$

$$\sigma_s^2 = \sigma^2 \cdot N \cdot \frac{1}{N^2} = \sigma^2 \frac{1}{N}$$

$$\sigma_s = \sigma \cdot \sqrt{\frac{1}{N}} = \sqrt{\frac{p(1-p)}{N}} \qquad (3.21)$$

The accuracy of the output interface in estimating the input probability is directly governed by the sum of the squares of the weighting coefficients.

The series generated at the output of an averaging operation possesses statistics which are markedly different from the original input. The variance is decreased and a time-lag incurred.

The induced correlation is a function of the weighting coefficients of the averager. Taking the example of a short-term averager of sample size N and time delay m,

$$S_{t+m} = \sum_{i=1}^{N} a_i A_{t+1} \qquad (3.22)$$

It follows that

$$\mathrm{cov}(S_t S_{t+k}) = E\{a_1 A_{t+1} + a_2 A_{t+2} + \cdots\}\{a_1 A_{t+k+1} + a_2 A_{t+k+2} + \cdots\} \quad (3.23)$$

Assuming the A's are independent this reduces to:

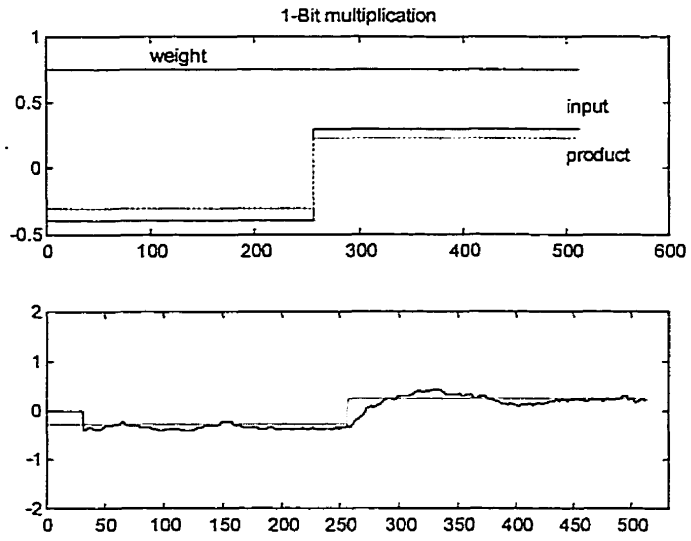
$$\mathrm{cov}(S_t S_{t+k}) = E[A^2]\{a_1 a_{k+1} + a_2 a_{k+2} + \cdots + a_{N-k} a_N\} \qquad (3.24)$$

the kth autocorrelation coefficient of the series is given by:

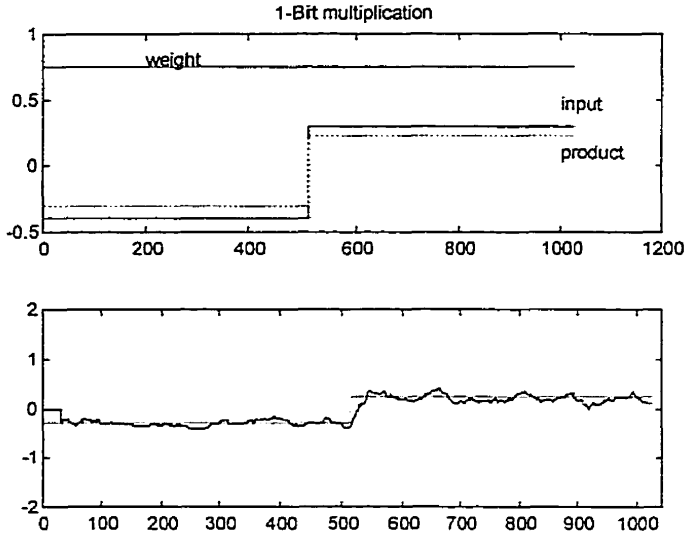
$$\begin{aligned} p_k &= \frac{E[S_{t+m} S_{t+m+k}]}{[ES_t^2]} \\ &= \frac{E[S_{t+m} S_{t+m+k}]}{E[A^2]\sum_{i=1}^{N} a_i^2} \\ &= \frac{\sum_{i=1}^{N-k} a_i a_{i+k}}{\sum_{i=1}^{N} a_i^2} \end{aligned} \qquad (3.25)$$

This equation shows that the averaged series possesses non-vanishing auto-correlations up to order N. Moreover p is always positive and can be quite for small values k. The moving average estimation we used has weights all equal to $\frac{1}{N}$. Applying this to equ (3.25) gives:

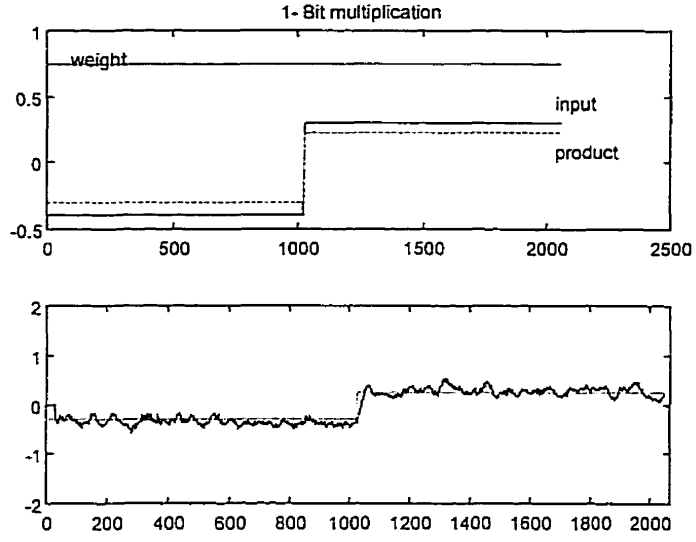
$$p_k = 1 - \frac{k}{N} \qquad (3.26)$$



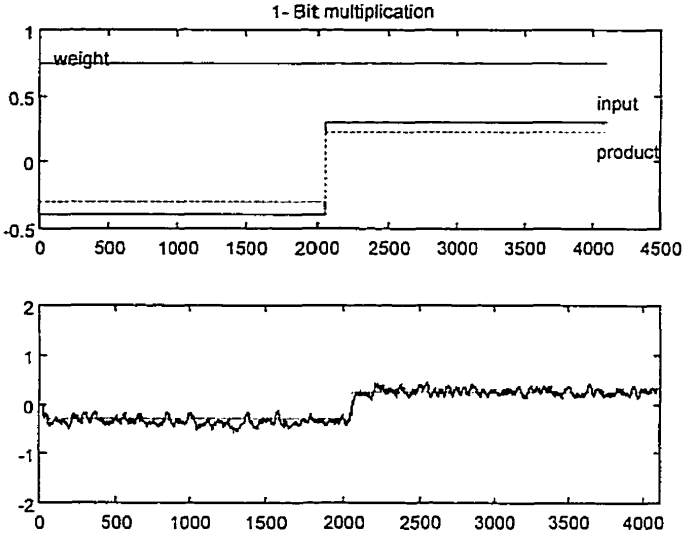
MS=512, Nav=32



MS=1024, Nav=32



MS=2048, Nav=32

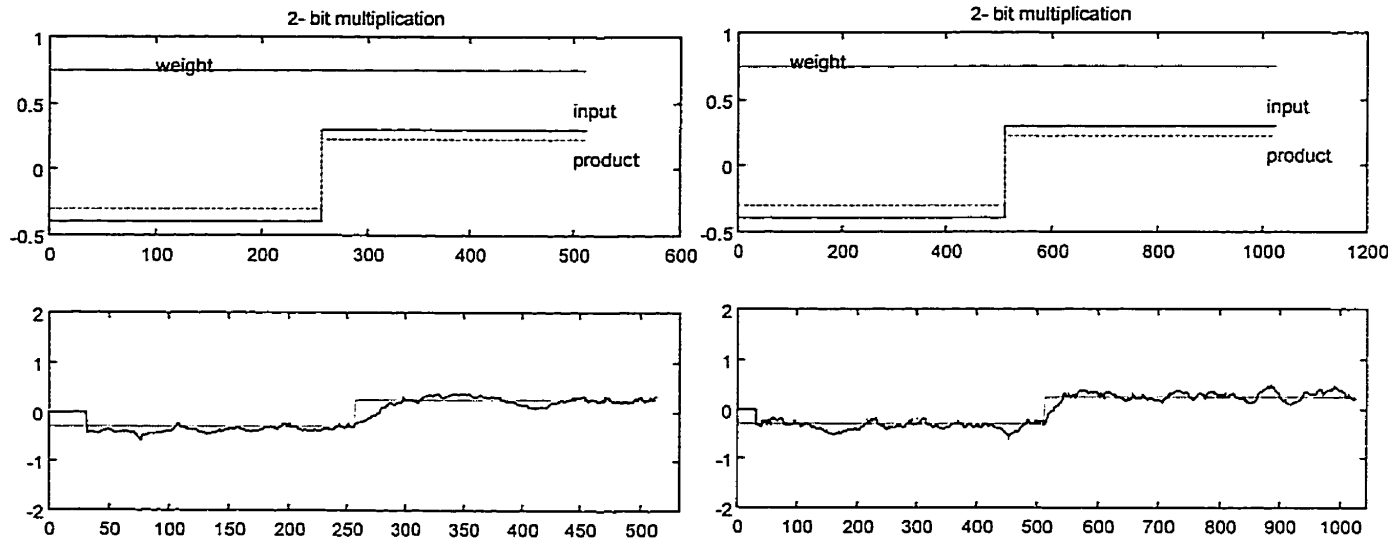


MS=4096, Nav=32

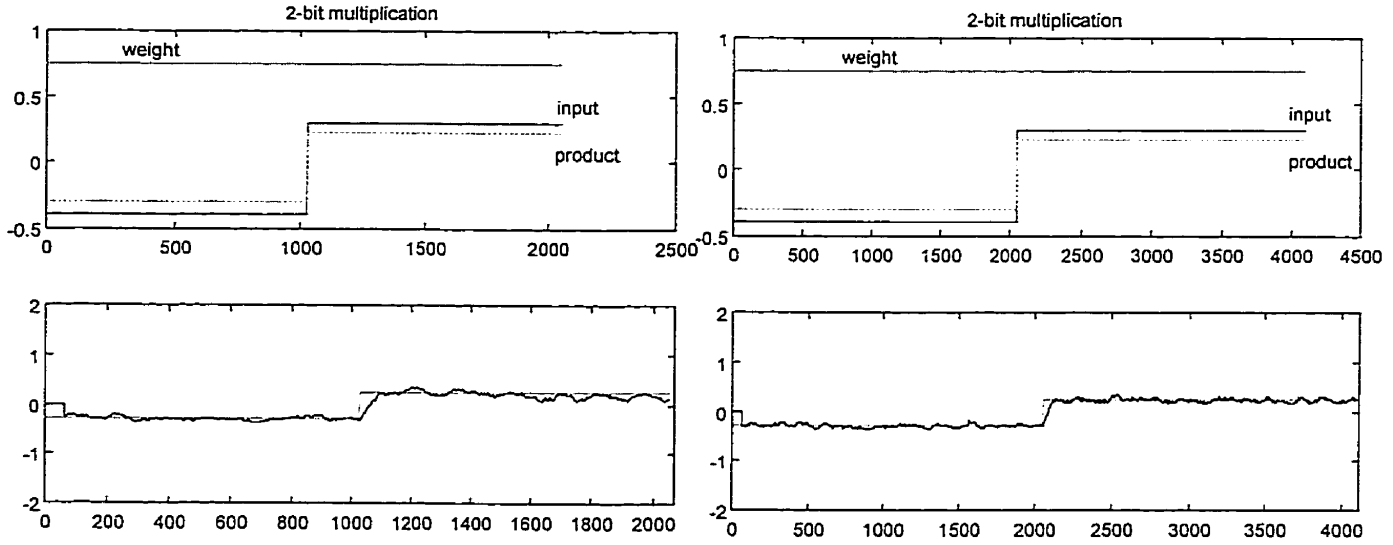
MS: total sampling bits;
Nav: moving average window size;

(a)

1-bit multiplication



MS=512, Nav=32 MS=1024, Nav=32



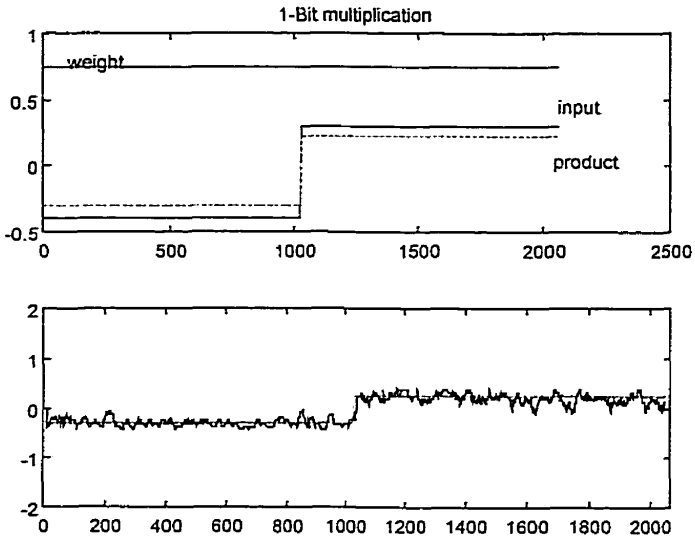
MS=2048, Nav=32 MS=4096, Nav=32

MS: total sampling bits;
Nav: moving average window size;

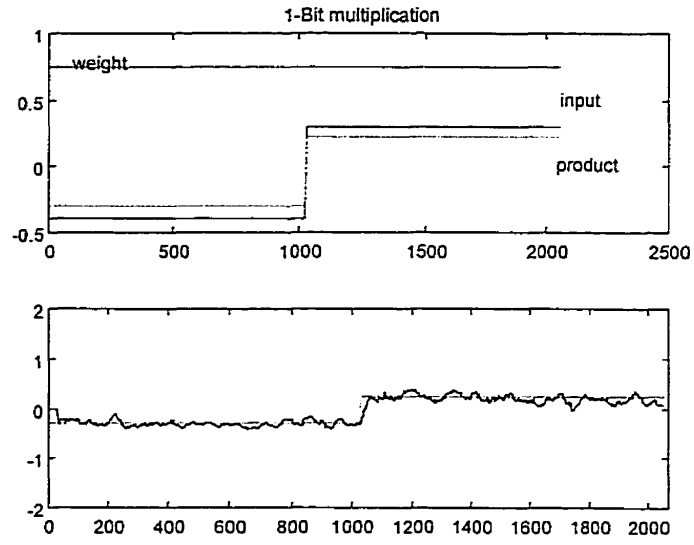
(b)

2-bit multiplication

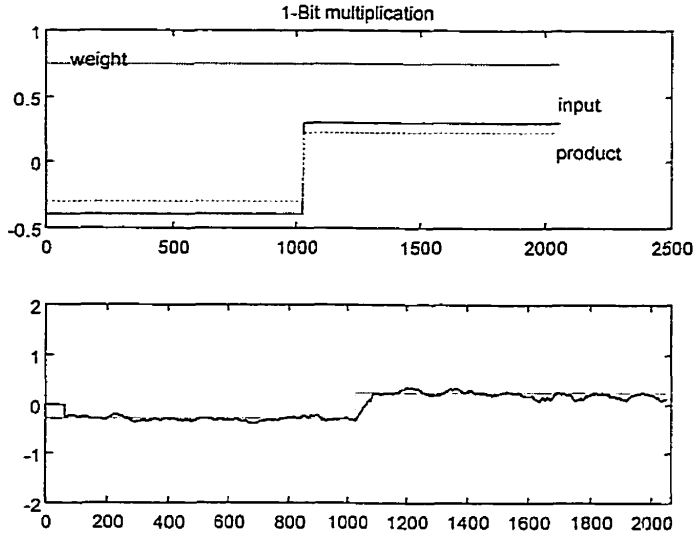
Figure 3.20 Moving average estimation of multiplication with different samples



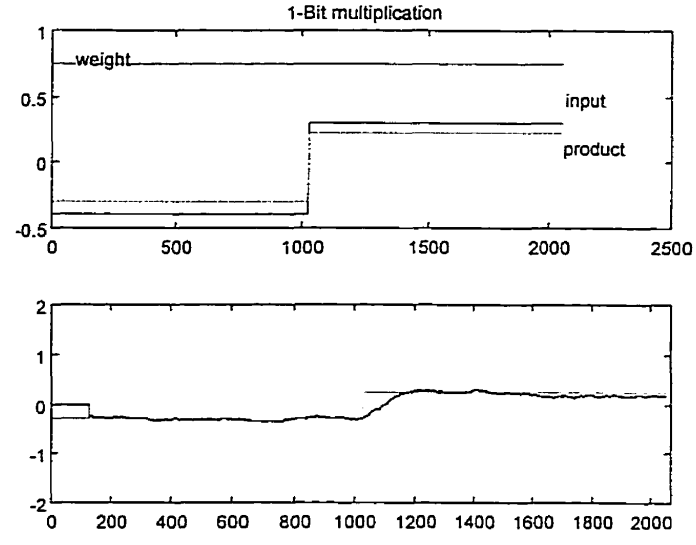
MS=2048, Nav=16



MS=2048, Nav=32



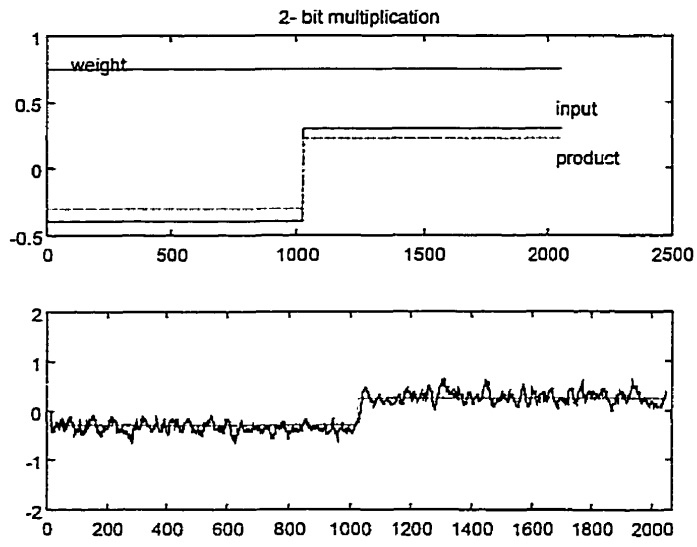
MS=2048, Nav=64

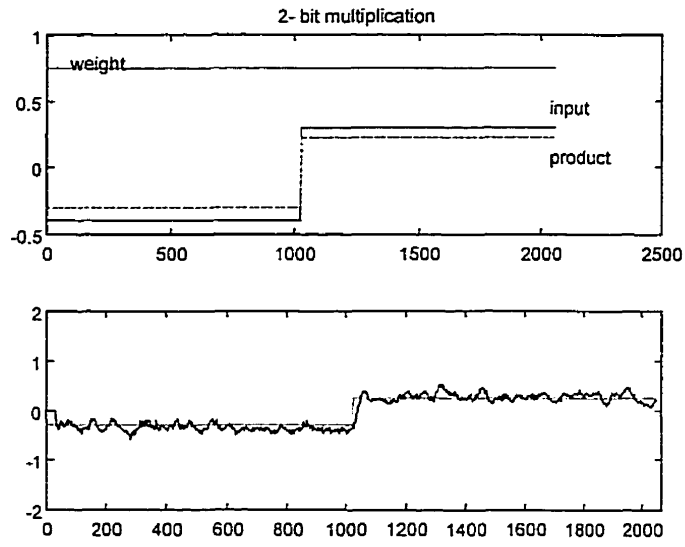


MS=2048, Nav=128

MS: total sampling bits;
Nav: moving average window size;

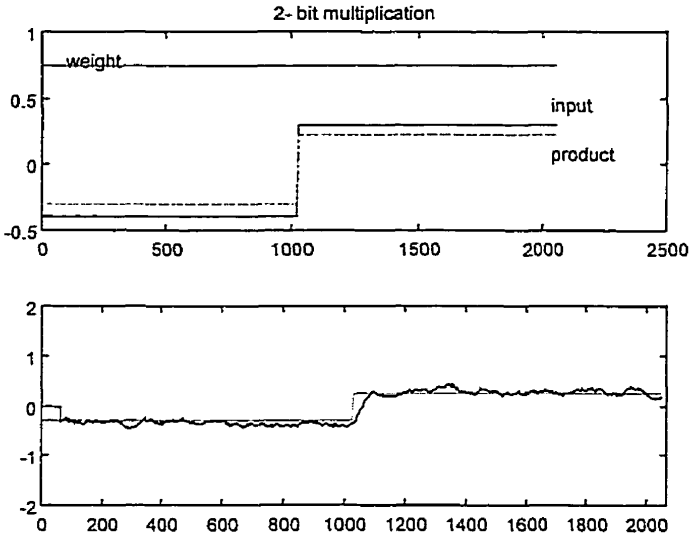
(a)
1-bit multiplication

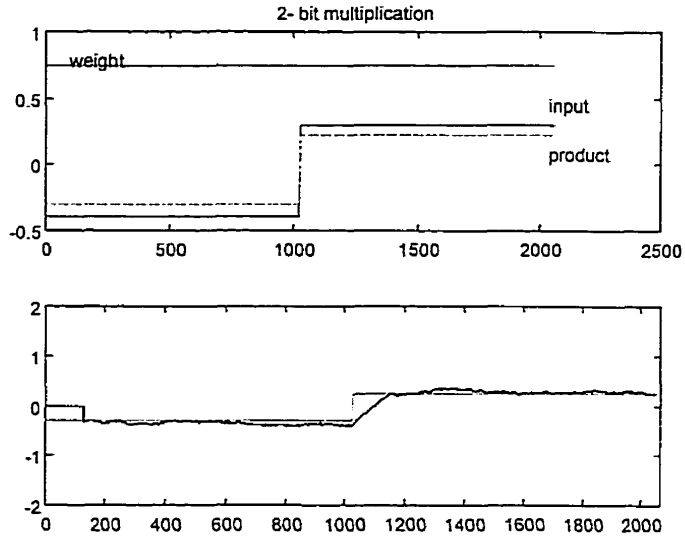




MS=2048, Nav=16

MS=2048, Nav=32





MS=2048, Nav=64

MS=2048, Nav=128

MS: total sampling bits;
Nav: moving average window size;

(b)
2-bit multiplication

Figure 3.21 Moving average estimation of multiplication with different window sizes

Figure 3.20 shows the moving average estimation of multiplication with different sample bits . We can observe that increasing the sample size can improve the accuracy. The disadvantage is the length of time required is also increased.

Figure 3.21 shows the moving average estimation of multiplication with different moving average window size (from 16 to128). Smaller window size yields better adaptation of the input signal, while bigger window size yields good estimation of static signals.

3.5 Random Pulse Artificial Neural Network

A typical neuron consists of more synapses and a neuron body, as shown in the following picture.

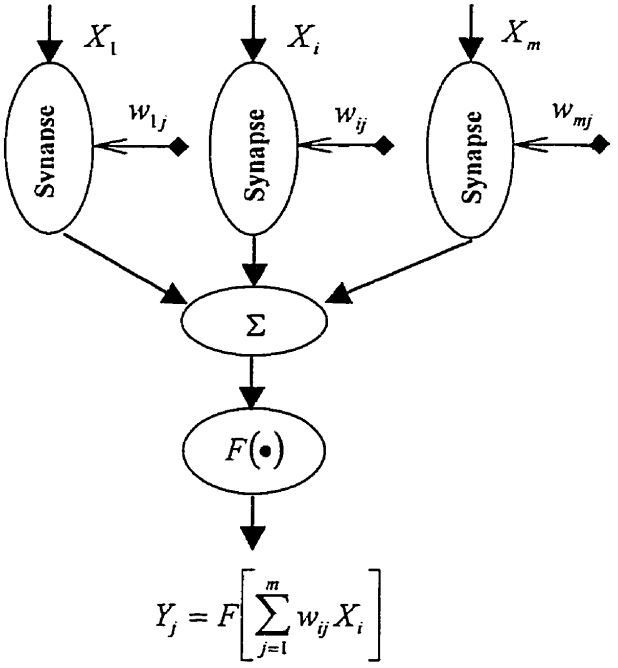


Figure 3.22 Random pulse neuron structure

Each synapse multiplies the incoming neural signal X_i, where $i = 1,2,\cdots m$, by synaptic stored variable weight values W_{ij}. The connection weights are adjusted during the learning phase. Connection weights which are positive-valued are "excitatory" connections, and those with negative values are "inhibitory" connections. The neuron body integrates the signals from all the post-synaptic channels (also called dendrites). The result of this integration is then submitted to an "activation function F" to produce the neuron's output signal Y_j.

Random pulse representation is used throughout the neural network, for synaptic weight storage as well as for arithmetic operations. Figure 3.23 shows the implementation of a synapse.

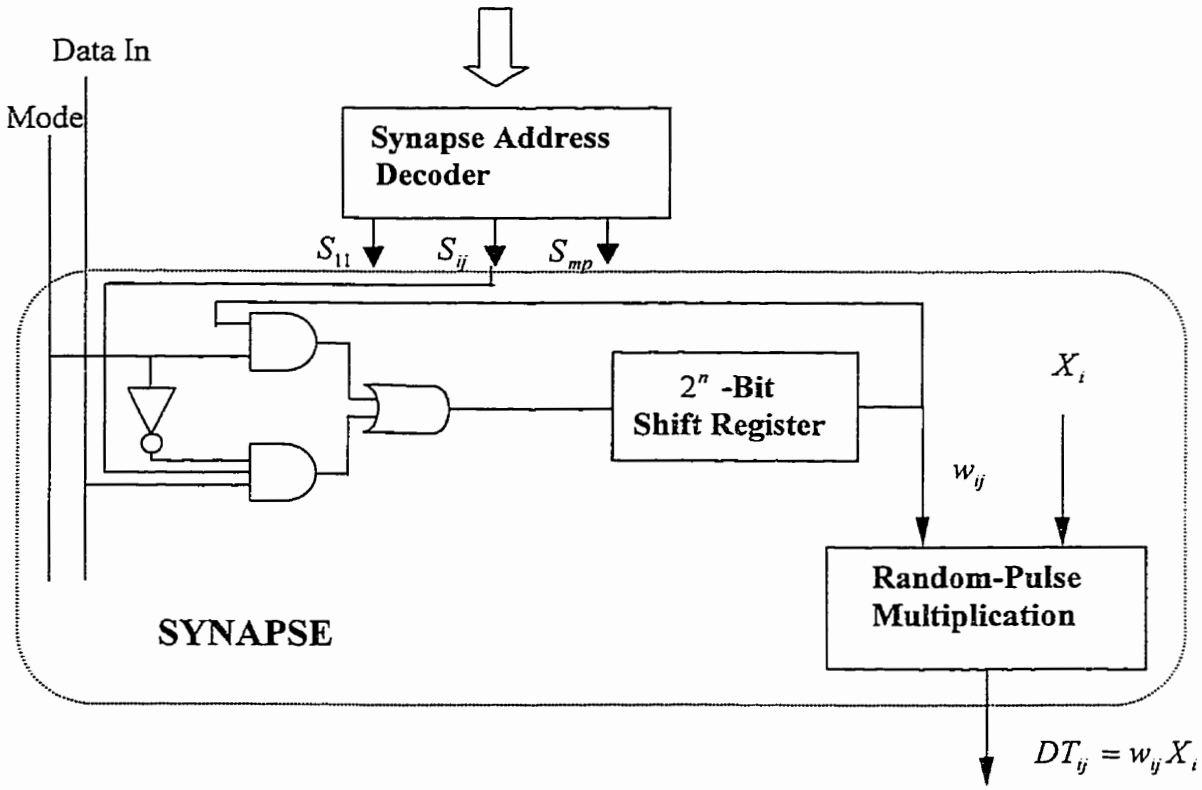


Figure 3.23 Random pulse implementation of a synapse

The synaptic weights are dynamically stored in a 2^n bit shift register. Loading weights values from the DATIN input into each register selected by the synapse address SYNADD is done serially when a low logic signal is applied to the control input MODE.

Figure 3.24 shows the random pulse implementation of the neuron body. The m input addition module collects the post synaptic data streams which are then integrated by a moving average random-pulse/digital converter. The internal neuron body clock CLK* has a pulse rate at least m times higher than the general clock CLK. There are only five activation functions types which are usually used: linear, step, ramp, sigmoid and Gaussian.

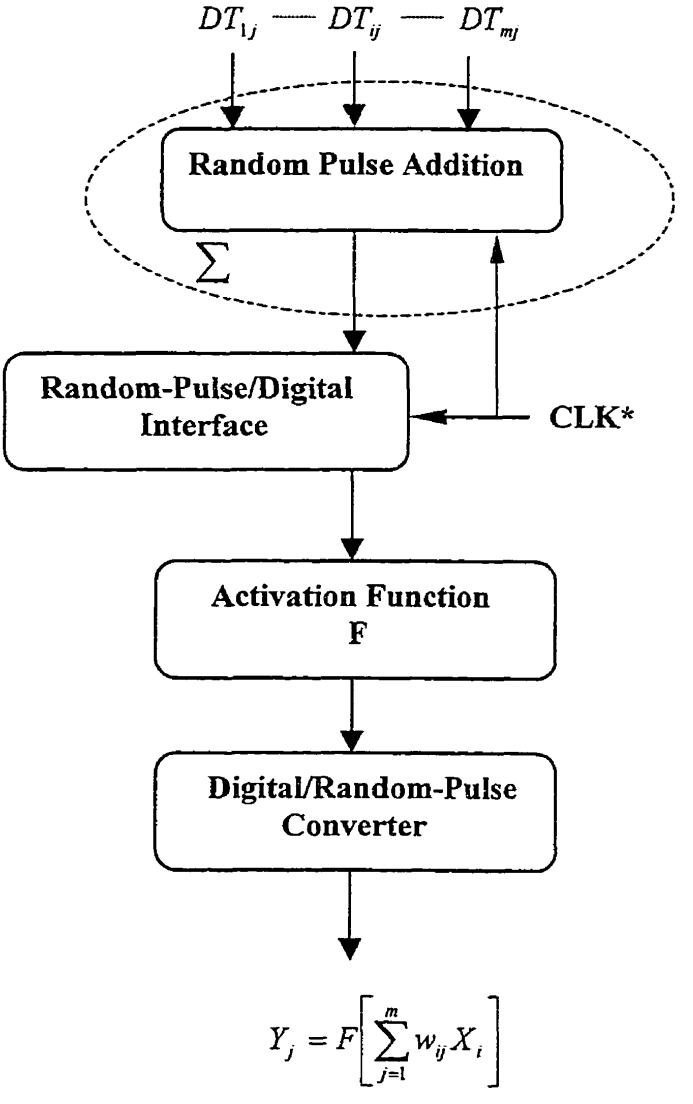


Figure 3.24 Random pulse implementation of the neuron body

During the learning phase, some algorithms like back propagation require a smooth nonlinear activation function which is differentiable everywhere. In general, any desired activation function can be implemented as a look-up table. Since the neuron output will be used as a synaptic input to other neurons, a final digital/random-pulse converter stage is used to restore the randomness of the signal Y_j. Like discussed in section 3.4, the counter of the random-pulse/digital converter is preloaded at each conversion cycle with a given threshold value. At any time when the counter content reaches the threshold, an overflow into the most significant bit is generated and used further as the neuron's random-pulse output.

The neural network architecture described can be interfaced directly with the environment or other computer via either analog/random-pulse or digital/random-pulse input interfaces and random-pulse/digital output interfaces.

Chapter 4 – Random Pulse Neural Network Applications

4.1 Perceptron

4.1.1 Introduction

In 1943, Warren McCulloch and Walter Pitts introduced one of the first artificial neurons. The main feature of their neuron model is that a weighted sum of input signals is compared to a threshold to determine the neuron output. When the output is greater than or equal to the threshold, the output is 1. When the sum is less than the threshold, the output is 0. They went on to show that networks of these neurons could, in principle, compute any arithmetic or logical function. In the late 1950s, Frank Rosenblatt and several other researchers developed a class of neural networks called perceptrons. The neurons in these networks were similar to those of McCulloch and Pitts. However, he introduced a learning rule for training perceptron networks to solve pattern recognition problems. He proved that his learning rule will always converge to the correct network weights, if weights exist to solve the problem. Examples of proper behavior were presented to the network, which learned from its mistakes. The perceptron can even learn when initialized with random values for its weights and biases.

The general perceptron network is shown in Figure 4.1.

The output of the network is given by :

$$a = \text{hardlim}\,(W * P + b) \qquad (4.1)$$

The network weight matrix is:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1R} \\ w_{21} & w_{22} & \cdots & w_{2R} \\ \vdots & \vdots & \vdots & \vdots \\ w_{S1} & w_{S2} & \cdots & w_{SR} \end{bmatrix} \qquad (4.2)$$

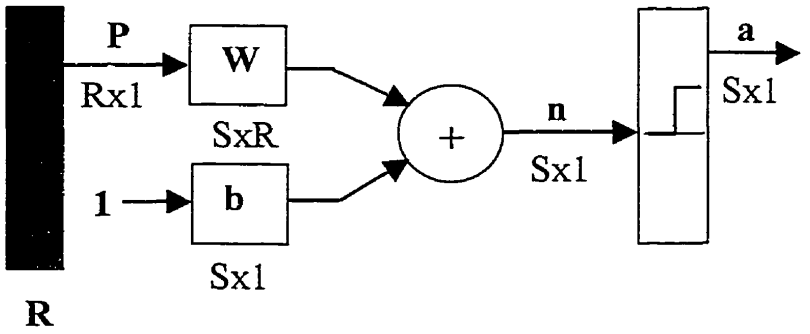


Figure 4.1 Perceptron Network

In the above figure, the solid vertical bar at the left represents the input vector **P**. The dimensions of **P** are displayed below the variable as *Rx1*, indicating that the input is a single vector of *R* elements. These inputs go to the weight matrix **W**, which has *S* rows and *R* columns. A constant **1** enters the neuron as an input and is multiplied by a scalar bias **b**. The net input to the transfer function (hard limit) is **n**, which is the sum of the bias **b** and the product **W*P**. The network's output is again a vector **a**.

4.1.2 Linear Classification by Perceptron

As we mentioned before, perceptron can solve pattern recognition problem. A single neuron perceptron can classify input vectors into two categories, since its output can be either 0 or 1. Figure 4.2 shows the input space of a 2-input hard limit perceptron. Two classification regions are formed by the dividing line $WP+b=0$ (**W**: weight matrix, **P**: input vector, **b**: bias). This line is perpendicular to the weight vector **W** and shifted according to the bias **b**. Input vectors above the line will result in a net input greater than 0, and therefore cause the hard limit neuron to output a 1. Input vectors below the line cause the neuron to output 0. The dividing line can be oriented and moved anywhere to classify the input space as desired by picking the weight and bias values.

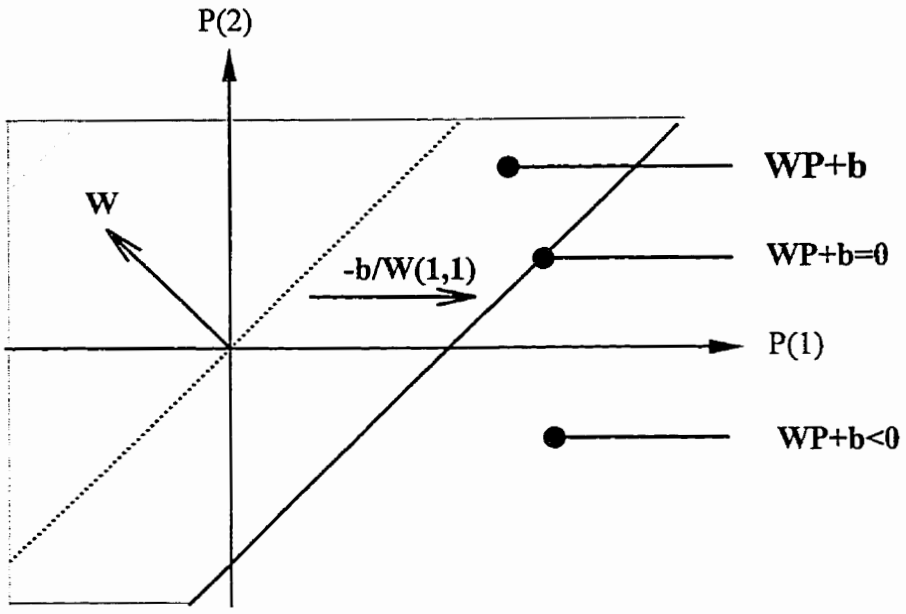
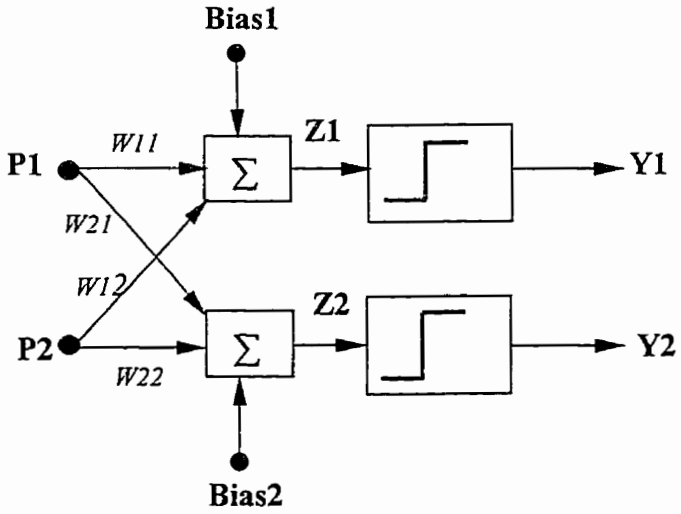


Figure 4.2 Analysis of perceptron classification

(adapted from Matlab neural network toolbox manual)

A multiple-neuron perceptron can classify inputs into many categories. Each category is represented by a different output vector. Since each element of the output vector can be either 0 or 1, there are a total of 2^S possible categories, where S is the number of neurons.

Followed is a two-neuron perceptron network for linear separable pattern classification.

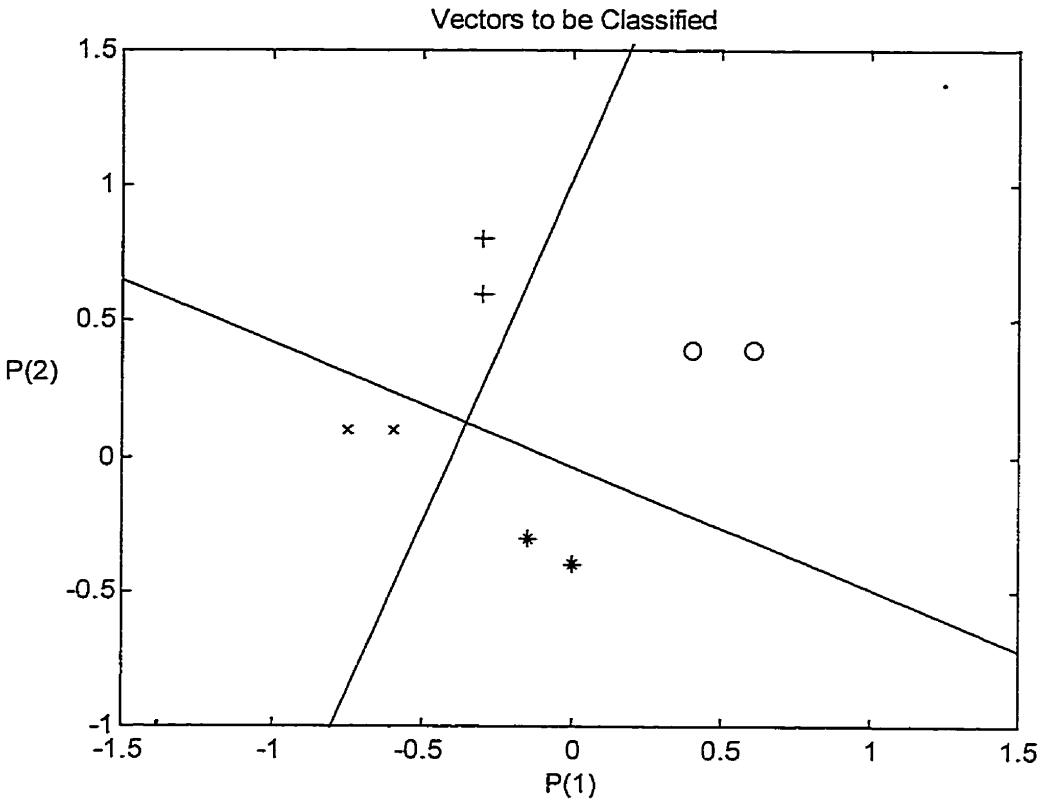


$$Y = hard\lim(W * p + b)$$

Figure 4.3 Two-input perceptron for linear separable classification

This Perceptron network can classify 2^2 different patterns.

Figure 4.4 shows perceptron linear classification solved by Matlab neural network toolbox. Four different patterns (marked with +, °, x, *) are presented to the network. The weight matrix and biases are obtained after the learning phase.



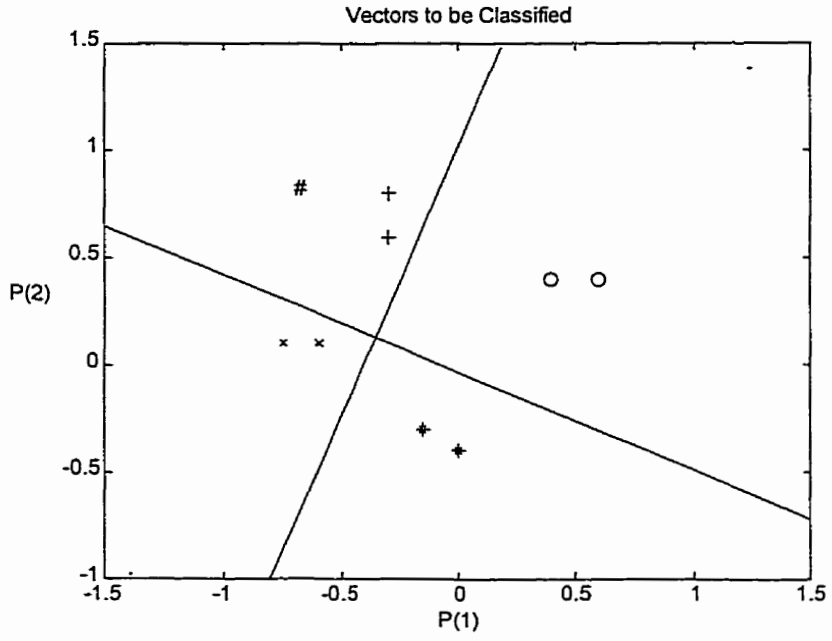
Input Patterns: $P = \begin{bmatrix} -0.3 & -0.3 & 0.4 & 0.6 & -0.6 & -0.75 & -0.15 & 0 \\ 0.6 & 0.8 & 0.4 & 0.4 & 0.1 & 0.1 & -0.3 & -0.4 \end{bmatrix}$

Targets: $T = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$

Weight Matrix: $W = \begin{bmatrix} -4.1391 & 1.6539 \\ -2.1077 & -4.6161 \end{bmatrix}$

Bias: $B = \begin{bmatrix} -1.6922 \\ -0.1680 \end{bmatrix}$

Figure 4.4 Linear classification by Matlab neural network toolbox



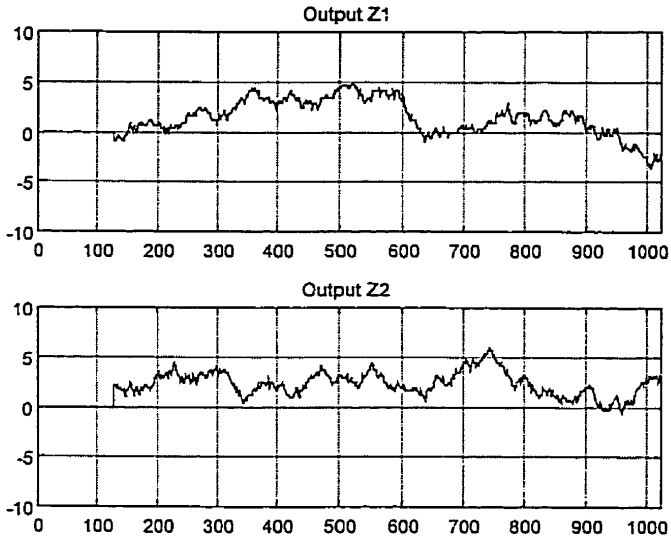
#: a new input vector $p = \begin{bmatrix} -0.7 \\ 0.8 \end{bmatrix}$

Figure 4.5 Perceptron classification for an input vector not used for training by Matlab toolbox

After learning phase, a new input vector (marked with #) is added to test the perceptron. Figure 4.5 shows the classification result by Matlab neural network toolbox. It shows this perceptron has the capability to decide the most closely associated pattern when a never-seen vector is presented.

The random pulse perceptron has the same structure like any generic perceptron (Fig 4.1), but the implementation of each functional block is done using the random pulse technique as described in section 3.5.

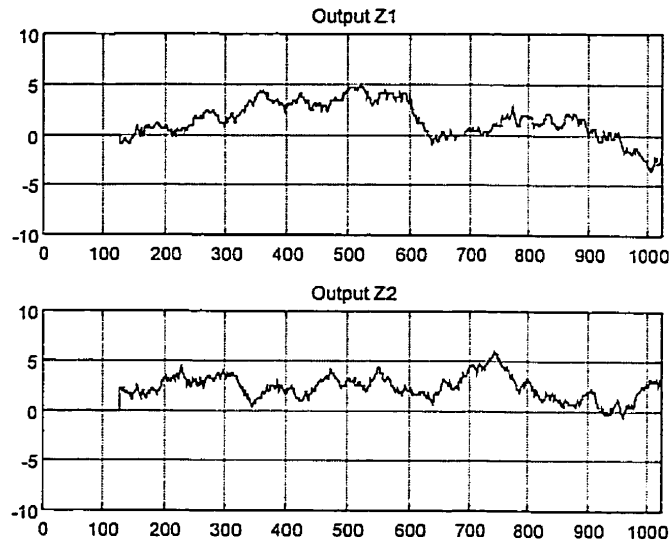
Figure 4.6 & 4.7 shows the simulation results of 1-bit and 2-bit random pulse perceptron respectively. 1-bit random pulse perceptron has much bigger error and can barely classify these four input patterns. Using 2-bit random pulse perceptron can get the same results as Matlab neural network toolbox does. Here, again we can conclude 2-bit random pulse neural network works more precisely than 1-bit system.



$p = (-0.3, 0.6)'$

correct output: $T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

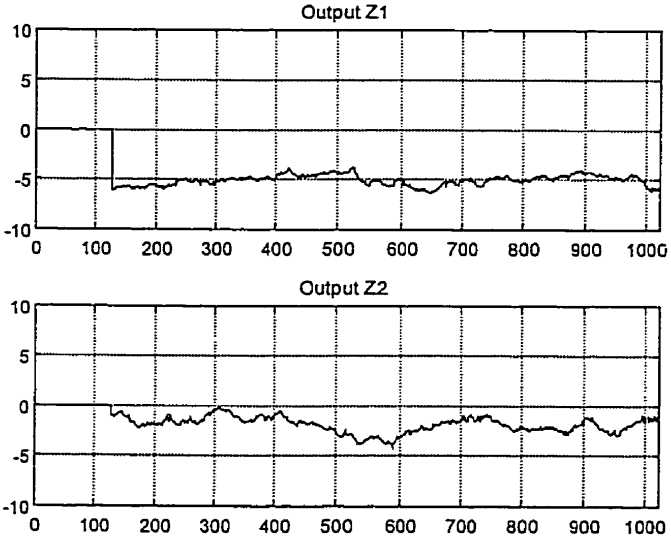
(unable to classify)



$p = (-0.3, 0.8)'$

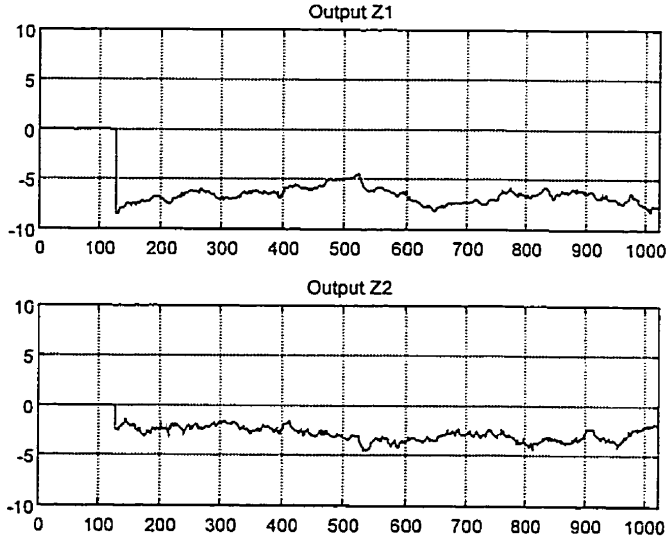
correct output: $T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

(unable to classify)



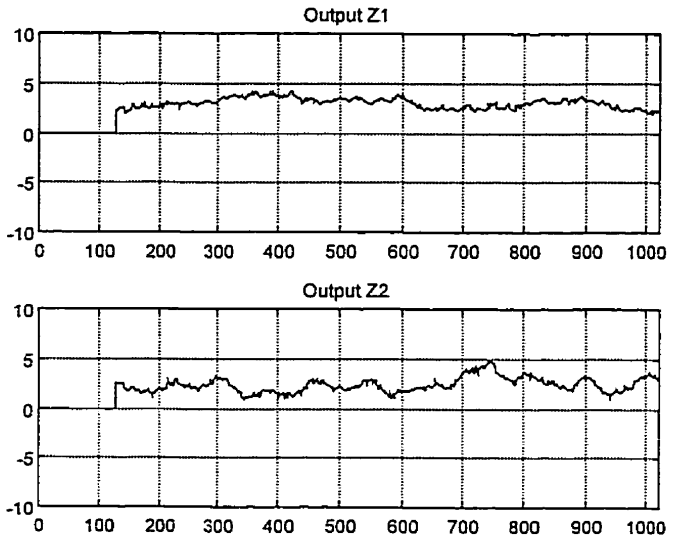
$p = (0.4, 0.4)'$

correct output: $T = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



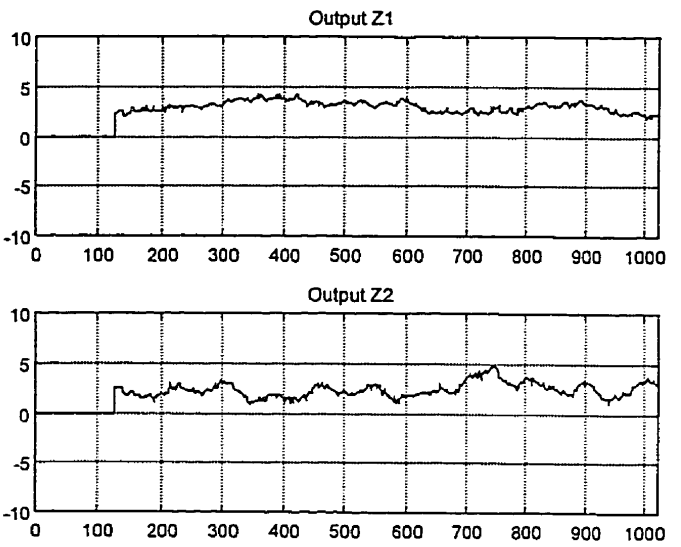
$p = (0.6, 0.4)'$

correct output: $T = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$



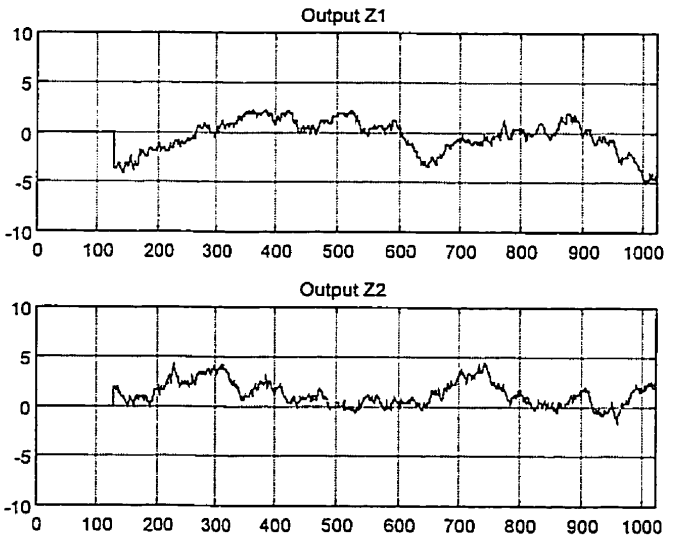
$p = (-0.6, 0.1)'$

correct output: $T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



$p = (-0.75, 0.1)'$

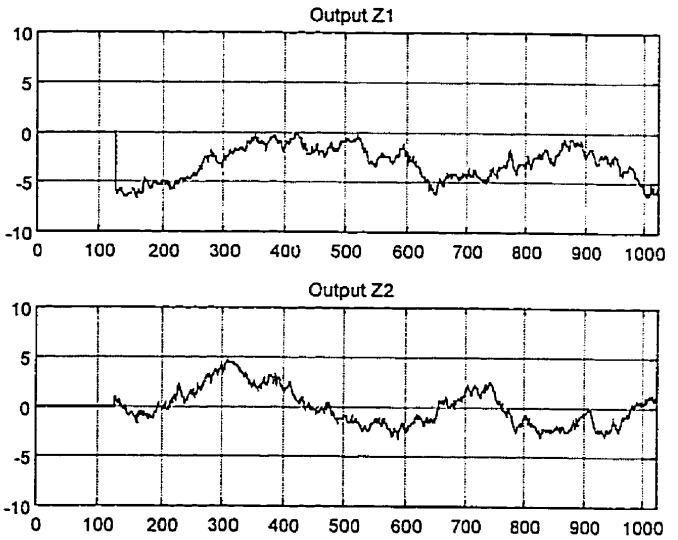
correct output: $T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



$p = (-0.15, -0.3)'$

Correct Output: $T = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

(unable to classify)



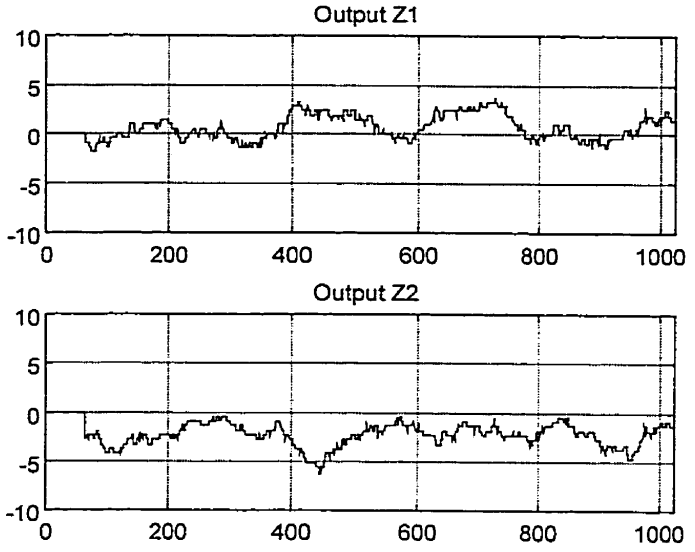
$p = (0, -0.4)'$

Correct Output: $T = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

(unable to classify)

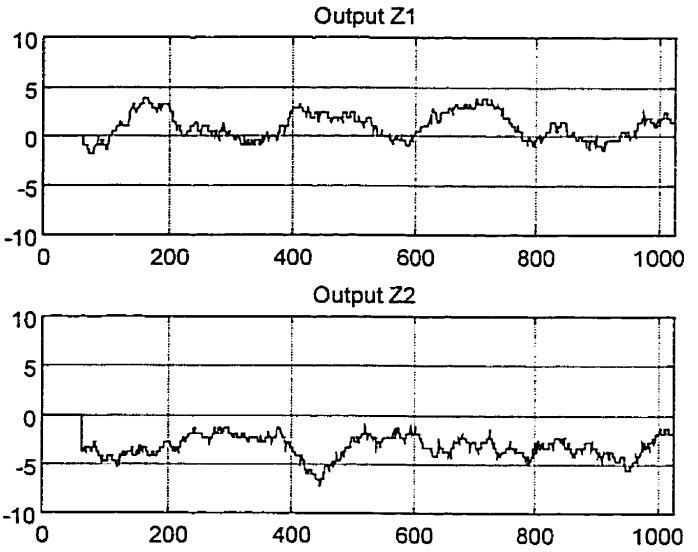
Figure 4.6 Linear classification using 1-bit random pulse neural network

Using 2-Bit hardware neural network can achieve the same results as Matlab toolbox does.



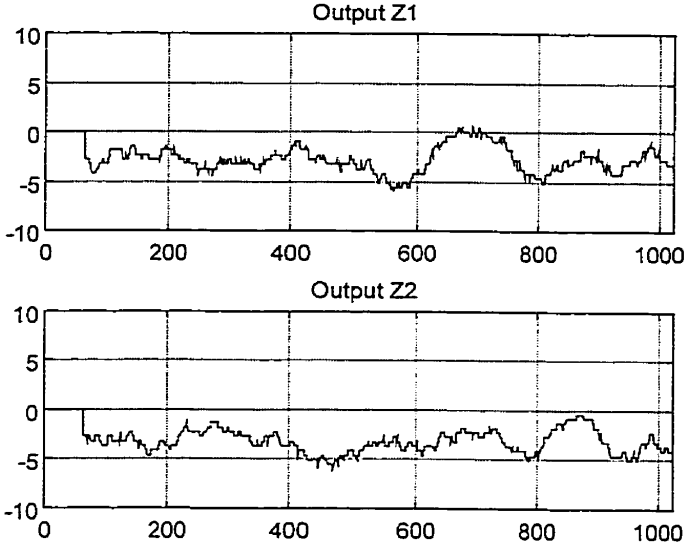
$p=(-0.3,0.6)'$

correct output: $T=\begin{bmatrix}1\\0\end{bmatrix}$



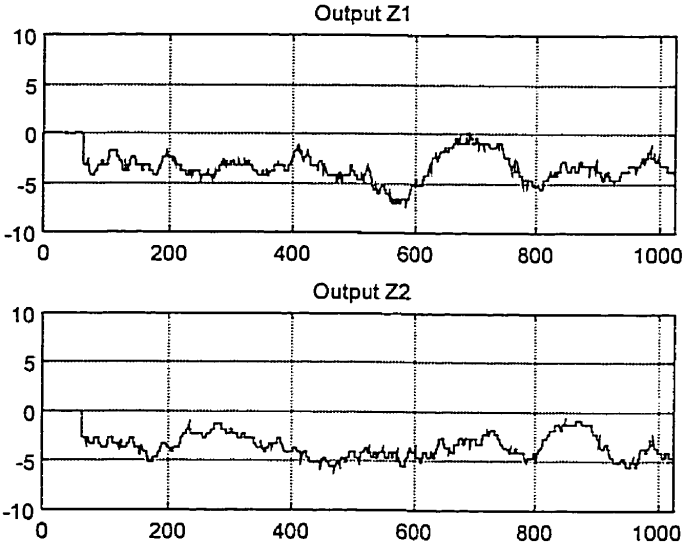
$p=(-0.3,0.8)'$

correct output: $T=\begin{bmatrix}1\\0\end{bmatrix}$



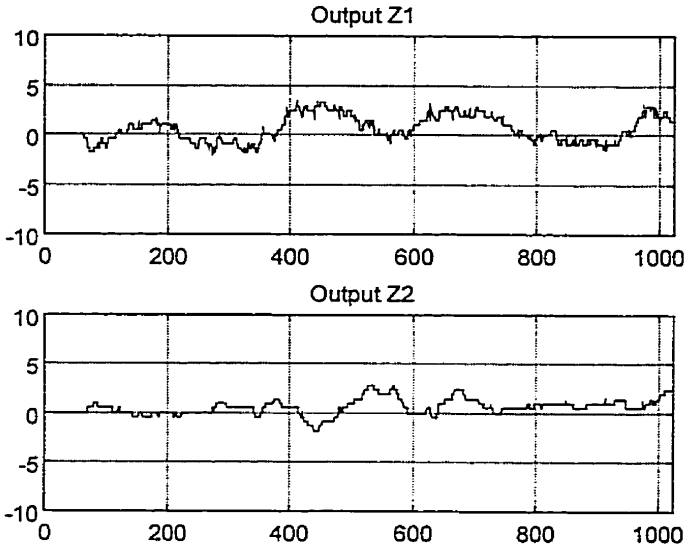
$p=(0.4,0.4)'$

Correct Output: $T=\begin{bmatrix}0\\0\end{bmatrix}$



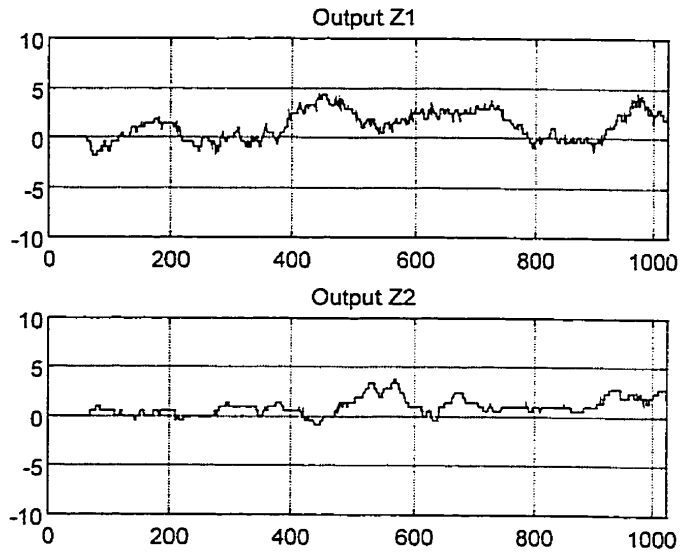
$p=(0.6,0.4)'$

Correct Output: $T=\begin{bmatrix}0\\0\end{bmatrix}$



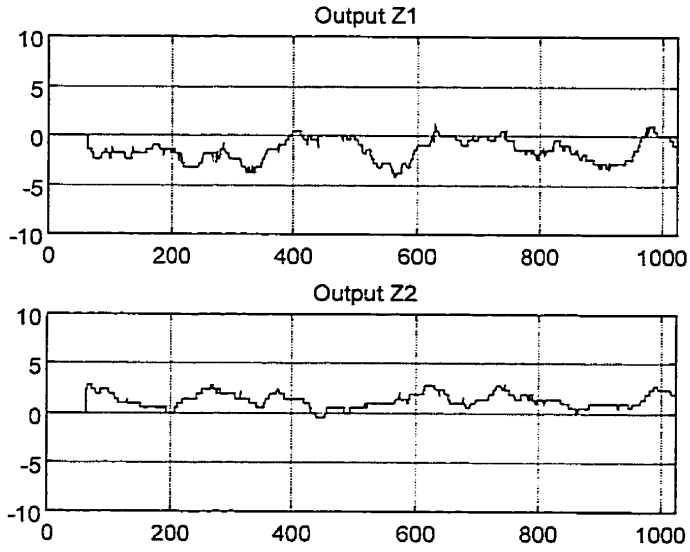
$p = (-0.6, 0.1)'$

Correct Output: $T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



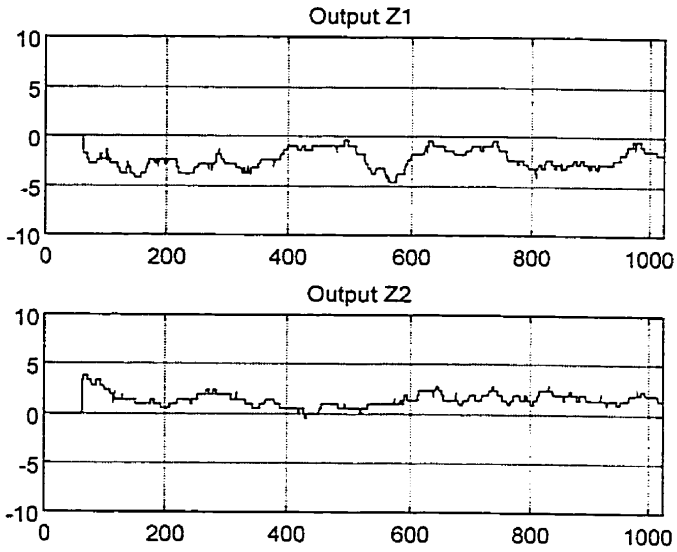
$p = (-0.75, 0.1)'$

Correct Output: $T = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



$p = (-0.15, -0.3)'$

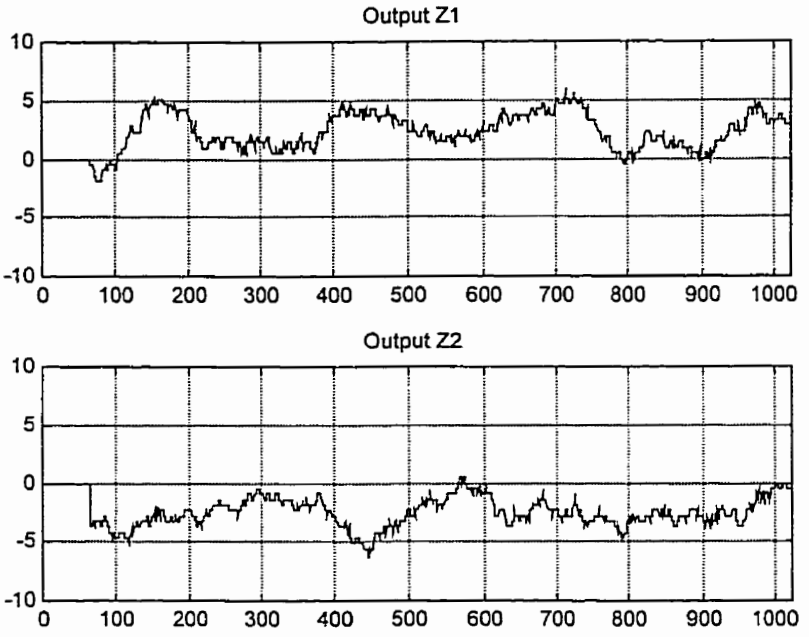
Correct Output: $T = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$



$p = (0, -0.4)'$

Correct Output: $T = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Figure 4.7 Linear classification using 2-bit random pulse neural network



Perceptron output for a new input vector $p = \begin{bmatrix} -0.7 \\ 0.8 \end{bmatrix}$

Correct output : $T = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Figure 4.8 2-bit random pulse perceptron classification for an input vector not used for training

Figure 4.8 shows that 2-bit random pulse perceptron can also classify a new input vector to the most close pattern. It can achieve the same result as Matlab neural network toolbox does. The fluctuation in Z1 and Z2 signals will be filtered out by the hard limitation at the perceptron output.

4.2 Autoassociative Memory

4.2.1 Introduction

Autoassociative memory is a type of neural network which can also handle the pattern recognition. The task of an autoassociative memory is to learn Q pairs of prototype input/output vectors:

$$\{P_1, t_1\}, \quad \{P_2, t_2\}, \quad \cdots \quad \{P_Q, t_Q\} \qquad (4.3)$$

In other words, if the network receives an input $P = P_Q$ then it should produce an output $a = t_q$, for $q = 1,2, \ldots, Q$. In addition, if the input is changed slightly (i.e., $P = P_Q + \delta$) then the output should only be changed slightly (i.e., $a = t_q + \varepsilon$).

In autoassociative memory, Hebb rule is used as learning algorithm. Hebb rule is based on Hebb's postulate which was presented by Donald O. Hebb as his study result from biological neural system.

> *When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's sufficiency, as one of the cells firing B, is increased.*
>
> Hebb's postulate

The mathematical interpretation of the postulate is:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha f_i(a_{iq}) g_j(p_{jq}) \qquad (4.4)$$

where p_{jq} is the jth element of the qth input vector p_q; a_{iq} is the ith element of the network output when the qth input vector is presented to the network; and α is a positive constant, called the learning rate. This equation indicates that the change in the weight w_{ij} is proportional to a product of functions of the activities on either side of the synapse. Equation 4.4 can be simplified as:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha a_{iq} p_{jq} \qquad (4.5)$$

The Hebb rule defined in (4.5) is an unsupervised learning rule. For the supervised Hebb rule we substitute the target output for the actual output. The resulting equation is:

$$w_{ij}^{new} = w_{ij}^{old} + t_{iq} p_{jq} \qquad (4.6)$$

where t_{iq} is the ith element of the qth target vector t_q. (α is set to one for simplicity.) (4.6) can be written in vector notation:

$$W^{new} = W^{old} + t_q p_q^T \qquad (4.7)$$

If we assume that the weight matrix is initialized to zero and then each of the Q input/output pairs are applied once to (4.7), we can write:

$$W = t_1 p_1^T + t_2 p_2^T + \cdots + t_Q p_Q^T = \sum_{q=1}^{Q} t_q p_q^T \qquad (4.8)$$

This can be represented in matrix form:

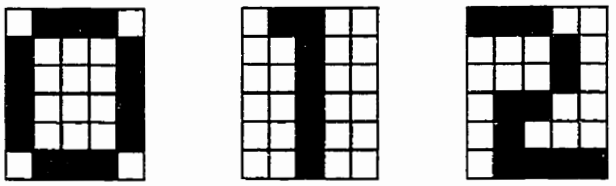
$$W = \begin{bmatrix} t_1 & t_2 & \cdots & t_Q \end{bmatrix} \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_Q^T \end{bmatrix} = TP^T \qquad (4.9)$$

where

$$T = \begin{bmatrix} t_1 & t_2 & \cdots & t_Q \end{bmatrix}, \qquad P = \begin{bmatrix} p_1 & p_2 & \cdots & p_Q \end{bmatrix} \qquad (4.10)$$

4.2.2 Digit Recognition by Autoassociative Memory

The patterns used for recognition are shown below. These patterns represent the input vectors and the targets. They represent the digits {0,1,2} displayed in a 6x5 grid. Each white square is represented by a "-1", and each black square is represented by a "1". To create the input vectors, we scan each 6x5 grid one column at a time.



P_1, t_1 P_2, t_2 P_3, t_3

Figure 4.9 Digit patterns for recognition

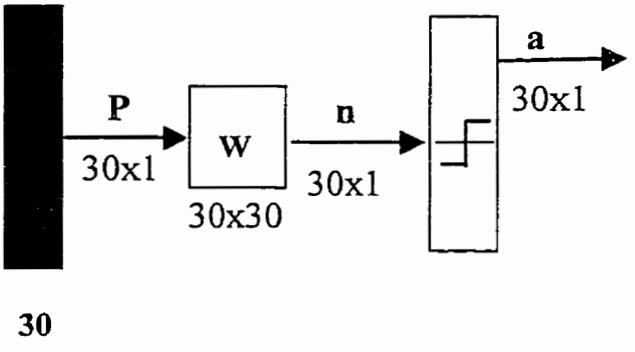
The first pattern is represented as a vector:

$$P_1 = [-1 \quad 1 \quad 1 \quad 1 \quad 1 \quad -1 \quad 1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad \cdots \quad 1 \quad -1] \quad (4.11)$$

The vector P_1 corresponds to the digit "0", P_2 to the digit "1", P_3 to the digit "2". Using the Hebb rule, the weight matrix is computed

$$W = P_1 P_1^T + P_2 P_2^T + P_3 P_3^T \qquad (4.12)$$

Figure 4.10 shows the autoassociative network for digit recognition.



$$a = \text{hardlims}(W * P)$$

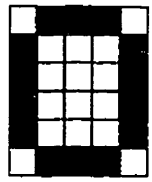
Figure 4.10 Autoassociative network for digit recognition

Using 2-bit random pulse neural network, when correct input vectors are given, the network can recover three digits as shown in figure 4.11.

For input vector :

$$P_1 = [\,-1\ \ 1\ \ 1\ \ 1\ \ 1\ \ -1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$-1\ \ 1\ \ 1\ \ 1\ \ 1\ \ -1\ \ ]'$$

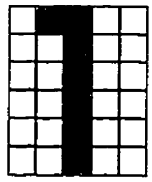
Random pulse neural network output is:



For input vector :

$$P_2 = [\,-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1\ \ ]'$$

Random pulse neural network output is:



For input vector :

$$P_3 = [\ 1\ -1\ -1\ -1\ -1\ -1$$
$$1\ -1\ -1\ 1\ 1\ 1$$
$$1\ -1\ -1\ 1\ -1\ 1$$
$$-1\ 1\ 1\ -1\ -1\ 1$$
$$-1\ -1\ -1\ -1\ -1\ 1]'$$

Random pulse neural network output is:

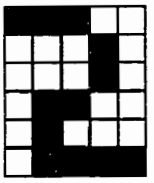


Figure 4.11 Digit recognition by 2-bit random pulse neural network

In addition to correct input vectors, autoassociative network is also able to deal with noise corrupted pattern recognition. Figure 4.12, 4.13 and 4.14 show the outputs of the networks when the input patterns are 10%, 20% and 30% occluded respectively. Figure 4.15, 4.16 and 4.17 show the outputs when the input patterns are 10%, 20% and 30% corrupted respectively. From simulation results, we can see the networks can restore the corrupted patterns successfully.

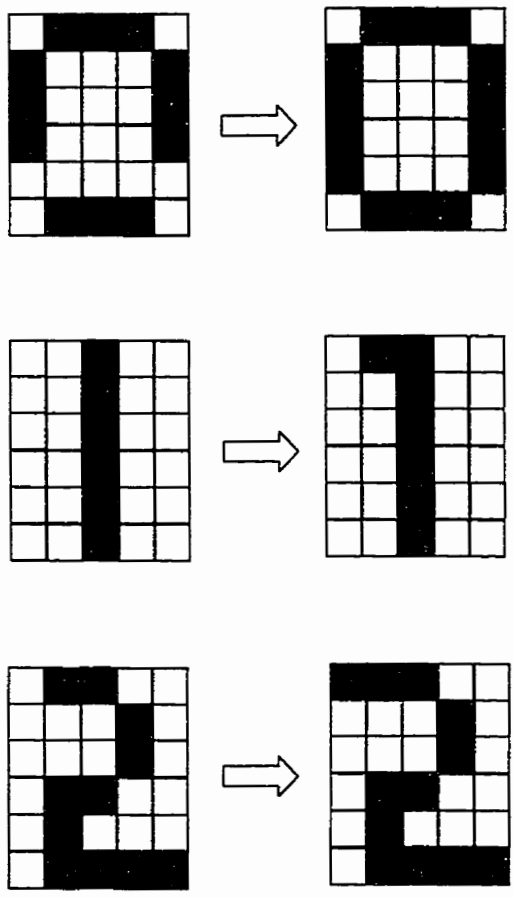


Figure 4.12 Recovery of 10% occluded patterns

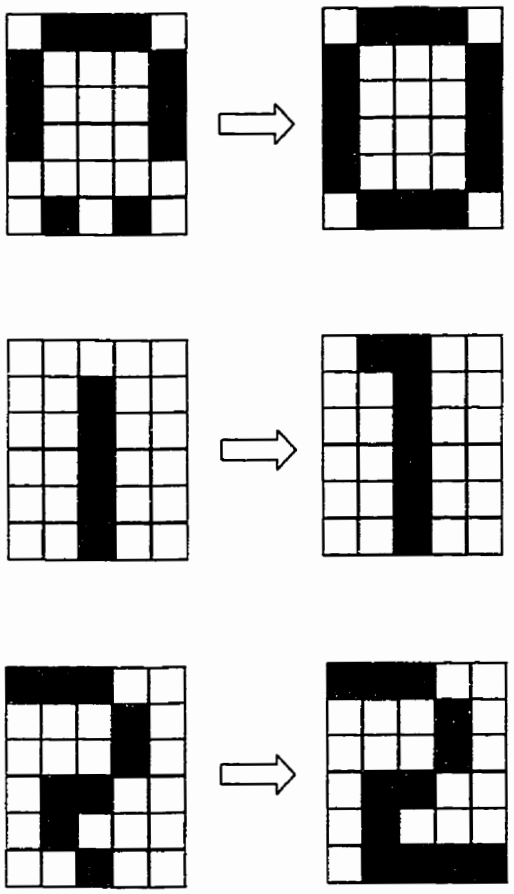


Figure 4.13 Recovery of 20% occluded patterns

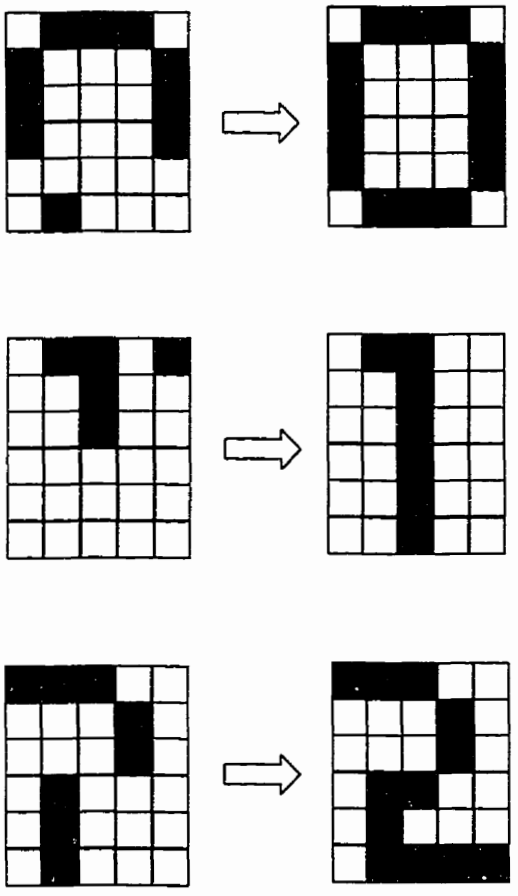


Figure 4.14 Recovery of 30% occluded patterns

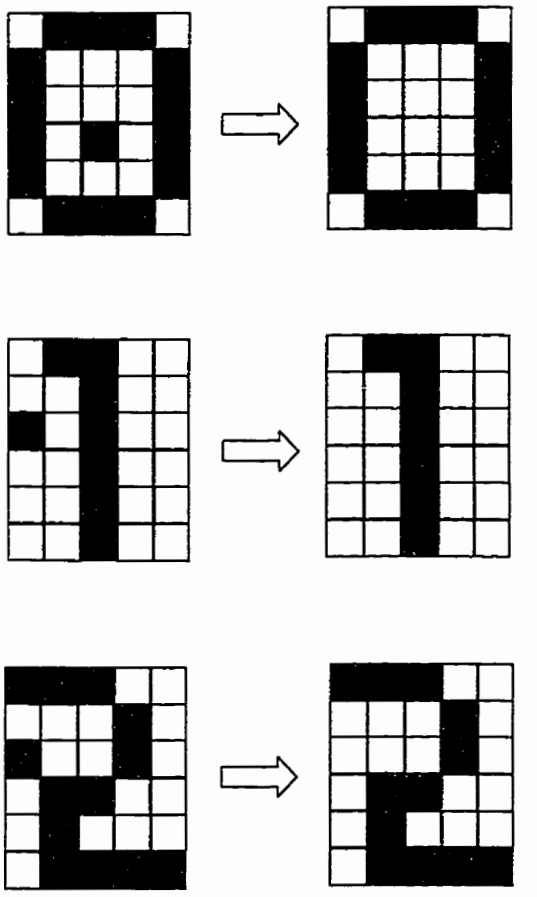


Figure 4.15 Recovery of 10% corrupted patterns

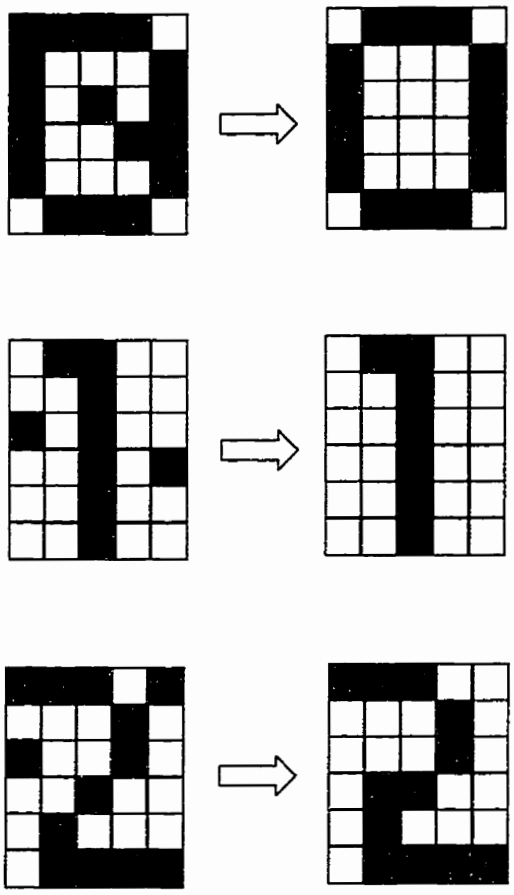


Figure 4.16 Recovery of 20% corrupted patterns

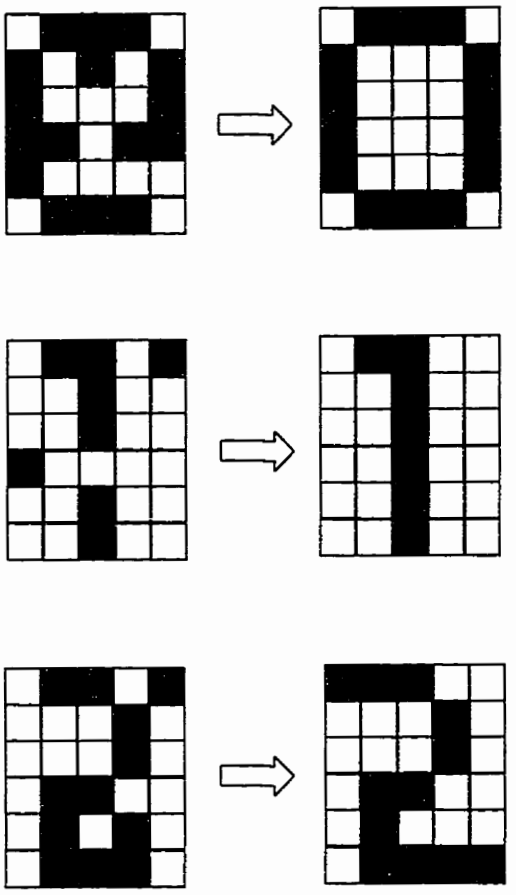
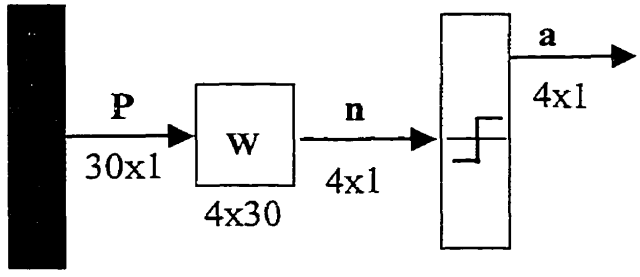


Figure 4.17 Recovery of 30% corrupted patterns

In order to recognize more digits, we use encoded output to represent different digits. The output has four bits which are binary encoded. The input patterns are from 0 to 9. Thus we reduce the network size from 30 output neurons to only 4 output neurons. Figure 4.18 shows the autoassociative network for encoded output.



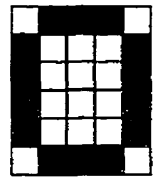
$$a = \text{hardlims}(W * P)$$

Figure 4.18 Digit recognition using encoded output

Followed are the simulation results of the recognition of 10 digits (from 0 to 9) by 2-bit random pulse neural network.

For input vector :

$$P_1 = [\,-1\ \ 1\ \ 1\ \ 1\ \ 1\ \ -1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$-1\ \ 1\ \ 1\ \ 1\ \ 1\ \ -1\ \ ]'$$

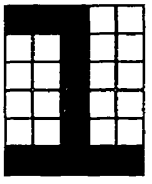


Random pulse neural network output is:

$$a = [0\ 0\ 0\ 0]$$

For input vector :

$$P_2 = [\,1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ 1\,]'$$

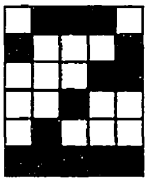


Random pulse neural network output is:

$$a = [0\ 0\ 0\ 1]$$

For input vector :

$$P_3 = [\,-1\ \ 1\ \ -1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ -1\ \ 1\ \ 1$$
$$1\ \ -1\ \ -1\ \ 1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$-1\ \ 1\ \ 1\ \ -1\ \ -1\ \ 1]'$$

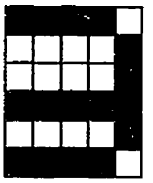


Random pulse neural network output is:

$$a = [0\ 0\ 1\ 0]$$

For input vector :

$$P_4 = [\ 1\ -1\ -1\ 1\ -1\ 1$$
$$1\ -1\ -1\ 1\ -1\ 1$$
$$1\ -1\ -1\ 1\ -1\ 1$$
$$1\ -1\ -1\ 1\ -1\ 1$$
$$-1\ 1\ 1\ 1\ 1\ -1]'$$

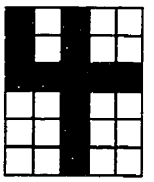


Random pulse neural network output is:

$$a = [0\ 0\ 1\ 1]$$

For input vector :

$$P_5 = [\ 1\ 1\ 1\ -1\ -1\ -1$$
$$-1\ -1\ 1\ -1\ -1\ -1$$
$$1\ 1\ 1\ 1\ 1\ 1$$
$$-1\ -1\ 1\ -1\ -1\ -1$$
$$-1\ -1\ 1\ -1\ -1\ -1]'$$

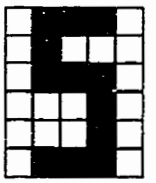


Random pulse neural network output is:

$$a = [0\ 1\ 0\ 0]$$

For input vector :

$$P_6 = [\ -1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ 1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1\ ]'$$

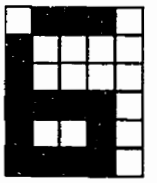


Random pulse neural network output is:

$$a = [0\ 1\ 0\ 1]$$

For input vector :

$$P_7 = [\ -1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$1\ \ -1\ \ -1\ \ 1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ 1\ \ -1\ \ 1$$
$$1\ \ -1\ \ -1\ \ 1\ \ 1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1]'$$

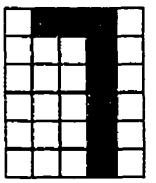


Random pulse neural network output is:

$$a = [0\ 1\ 1\ 0]$$

For input vector :

$$P_8 = [\ -1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$-1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1]'$$



Random pulse neural network output is:

$$a = [0\ 1\ 1\ 1]$$

For input vector :

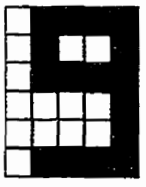
$$P_9 = [\ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1]'$$

Random pulse neural network output is:

$$a = [1\ \ 0\ \ 0\ \ 0]$$

For input vector :

$$P_{10} = [\ -1\ \ -1\ \ -1\ \ -1\ \ -1\ \ -1$$
$$1\ \ 1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ -1\ \ 1\ \ -1\ \ -1\ \ 1$$
$$1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1]'$$



Random pulse neural network output is:

$$a = [1\ \ 0\ \ 0\ \ 1]$$

Figure 4.19 Digit recognition by 2-bit random pulse neural network using encoded output

Chapter 5 – Conclusion

When an application requires real-time response that exceeds the capability of high-end workstations, or when a mass-produced system that is not computer-based requires nontrivial data processing, custom neural network hardware is the only cost-effective solution. Both analog and digital implementations have their place in today's neural milieu. Digital designs allow a greater flexibility in mapping arbitrary or extremely large problems onto a limited number of time-multiplexed processing circuits. Also, advances in fabrication technologies may benefit digital implementations more so than analog.

Among various digital implementation methods, random pulse machine concept provides a good tradeoff between electronic circuit complexity and the computational accuracy which leads to a high packing density well suited for VLSI implementation. Random pulse machine deals with analog variables represented by the mean rate of random pulse streams and use simple digital technology to perform arithmetic and logic operations. Analog variables are converted to random pulses by random reference quantization (RRQ). Rapid multiplication and addition operations can be achieved by conventional digital circuits elements.

Contributions

In this thesis, 1-bit and 2-bit random pulse neural network architectures are presented [47]. Simulation results of arithmetic operations for both systems show the computational accuracy for hardware implementation. Perceptron model for linear separable pattern classification and autoassociative memory for digit recognition were simulated. In both applications, 2-bit random pulse neural network did the pattern recognition successfully. Random pulse neural network architecture has the following advantages when compared with the pure analog and digital implementation:

- Signals are represented by continuous pulse streams
- Arithmetic operations can be implemented by simple logic gates
- Good noise immunity (robust)
- Trade off between circuit complexity and computational accuracy

Future development

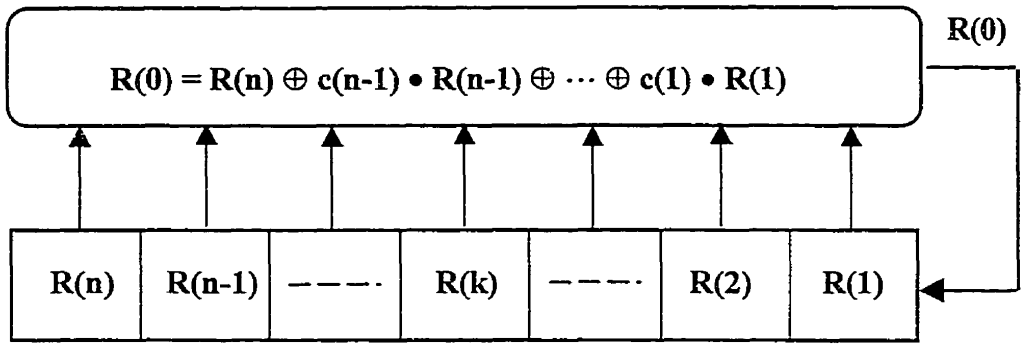
Further studies should involve simulations based on board-level design and eventually building a circuit board for certain application. The ultimate goal is to design a digital IC chip for real-time information processing such as recalling of 3-D electromagnetic field models used for computer aided design of very high speed circuits. The time-consuming learning phase will be kept off-line, and the recall phase will be integrated in the hardware and done in real time.

Appendix A

The Duality Between Boolean Operations with Individual Pulses and Algebraic Operations with the Variables Represented by their Respective Pulse Sequences

Truth Table		Boolean Operations with Pulses	Algebraic Operation with	
X Y	0 0 1 1 0 1 0 1	$Z = F(X,Y)$	Machine Variables $z = f(x,y)$	Problem Variables $z_0 = f_0(x_0, y_0)$
Z	0 0 0 0	0	0	-1
	0 0 0 1	$XY = \overline{\overline{X} \vee \overline{Y}}$	xy	$-\frac{1}{2}(1 - x_0 - y_0 - x_0 y_0)$
	0 0 1 0	$X\overline{Y} = \overline{\overline{X} \vee Y}$	$x(1-y)$	$-\frac{1}{2}(1 - x_0 + y_0 + x_0 y_0)$
	0 0 1 1	X	x	x_0
	0 1 0 0	$\overline{X}Y = \overline{X \vee \overline{Y}}$	$(1-x)y$	$-\frac{1}{2}(1 + x_0 - y_0 + x_0 y_0)$
	0 1 0 1	Y	y	y_0
	0 1 1 0	$X\overline{Y} \vee \overline{X}Y = X \oplus Y$	$x(1-y)+(1-x)y=x+y-2xy$	$-x_0 y_0$
	0 1 1 1	$X \vee Y$	$x+y-xy$	$\frac{1}{2}(1 + x_0 + y_0 - x_0 y_0)$
	1 0 0 0	$\overline{XY} = \overline{X \vee Y}$	$(1-x)(1-y)=1-(x+y-xy)$	$-\frac{1}{2}(1 + x_0 + y_0 - x_0 y_0)$
	1 0 0 1	$XY \vee \overline{X}\overline{Y} = \overline{X \oplus Y}$	$xy+(1-x)(1-y)=1-(x+y-2xy)$	$x_0 y_0$
	1 0 1 0	$\overline{Y}$	$1-y$	$-y_0$
	1 0 1 1	$X \vee \overline{Y} = \overline{\overline{X}Y}$	$x+(1-y)-x(1-y)=1-(1-x)y$	$\frac{1}{2}(1 + x_0 - y_0 + x_0 y_0)$
	1 1 0 0	$\overline{X}$	$1-x$	$-x_0$
	1 1 0 1	$\overline{X} \vee Y = \overline{X\overline{Y}}$	$(1-x)+y-(1-x)y=1-x(1-y)$	$\frac{1}{2}(1 - x_0 + y_0 + x_0 y_0)$
	1 1 1 0	$\overline{X} \vee \overline{Y} = \overline{XY}$	$(1-x)+(1-y)-(1-x)(1-y)=1-xy$	$\frac{1}{2}(1 - x_0 - y_0 - x_0 y_0)$
	1 1 1 1	1	1	1

Appendix B



Modulo-2 direct feedback shift register

Shift register length ***n***	**Feedback for direct PRBS** R(0) = R(n) ⊕ c(n-1) • R(n-1) ⊕ ··· ⊕ c(1) • R(1)	**Feedback for reverse PRBS** R(n+1) = R(1) ⊕ b(2) • R(2) ⊕ ··· ⊕ b(n) • R(n)
4	R(0) = R(4) ⊕ R(1)	R(5) = R(1) ⊕ R(2)
5	R(0) = R(5) ⊕ R(2)	R(6) = R(1) ⊕ R(3)
6	R(0) = R(6) ⊕ R(1)	R(7) = R(1) ⊕ R(2)
7	R(0) = R(7) ⊕ R(3)	R(8) = R(1) ⊕ R(4)
8	R(0) = R(8) ⊕ R(4) ⊕ R(3) ⊕ R(2)	R(9) = R(1) ⊕ R(3) ⊕ R(4) ⊕ R(5)
9	R(0) = R(9) ⊕ R(4)	R(10) = R(1) ⊕ R(5)
10	R(0) = R(10) ⊕ R(3)	R(11) = R(1) ⊕ R(4)

Feedback equations for PRBS generation

Appendix C

```
% neuron.m is one neuron
clear;
% Global Parameters
% MS is total samples
% Nav is moving average estimation bits
% NW bit shift register for weight
% DELTA

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DELTA=1;
MS=512;
Nav=32;
NW=16;
global MS DELTA Nav NW

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

input_no=2;
threshold=0.12;
x=[0.4 0.25];
w=[0.75 -0.5];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('total samples is:');
disp(MS);
disp('moving average estimation bits is:');
disp(Nav);
disp('shift register bit is:');
disp(NW);

sum=0;
sumrq=zeros(1,MS);

for i=1:input_no
   xw(i)=x(i)*w(i);
   xrq=quin(x(i));
   wrq=quw(w(i));
   XRQ(i,1:MS)=xrq;
   WRQ(i,1:MS)=wrq;
   mavx(i,1:MS)=movave(xrq);
   xwrq=mult(wrq,xrq);
   XWRQ(i,1:MS)=xwrq;
   mavxw(i,1:MS)=movave(xwrq);
```

```
   sum=sum+xw(i);
end

sumrq=add(XWRQ(1,1:MS),XWRQ(2,1:MS));

mavsumrq=movadd(sumrq);
Y=hardlim(mavsumrq);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%plot of the whole procedure
pl_quan(XRQ(1,1:MS),mavx(1,1:MS),x(1));  % plot the input signal
pl_mult(x(1),w(1),xw(1),mavxw(1,1:MS));    %plot the product
pl_add(xw(1),xw(2),mavsumrq);
err(x(1),mavx(1,1:MS),xw(1),mavxw(1,1:MS),sum,mavsumrq);  %plot the error


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xrq=quin(inx)

% input signal quantization
% inx is an input analog value
% xrq is random pulse stream carrying the value inx

global MS DELTA

for iav=1:MS
   noise(iav)=rand(1)-DELTA/2;              % generate white noise
   xdit(iav)=inx+noise(iav);                % dithered signal
   xrq(iav)=2*floor(xdit(iav)/DELTA)+1;     % quantization,generate pulse -1 or 1
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function wrq=quw(weight)

% ad hoc weight pulse streams
% choose from different weights
% weight is analog value for synaptic weight, wrq is pulse stream

global MS Nav NW


if weight ==0.75
```

```
    for wav=1:NW        % first NW bit
     if wav==2
      wrq(wav)=-1;
     elseif wav==5
      wrq(wav)=-1;
     else
      wrq(wav)=1;
     end
    end


   for i=1:(MS/NW-1)    % rotating the NW bit shift register
    for wav=(NW*i+1):(NW*(i+1))
       wrq(wav)=wrq(wav-NW*i);
    end
   end
end

if weight ==-0.5
   for wav=1:NW
    if wav==2
      wrq(wav)=1;
    elseif wav==5
      wrq(wav)=1;
    elseif wav==9
      wrq(wav)=1;
    elseif wav==14
      wrq(wav)=1;
    else
      wrq(wav)=-1;
    end
   end

  for i=1:(MS/NW-1)
   for wav=(NW*i+1):(NW*(i+1))
     wrq(wav)=wrq(wav-NW*i);
   end
  end

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function mavinrq=movave(inrq)

% moving average calculation
```

```
% inrq is radom pulse stream carrying an analog value
% mavinrq is the moving average estimate of the pulse stream

global MS Nav

in_j=0;
for iav=1:MS
   if iav>Nav
     mavinrq(iav)=mavinrq(iav-1)+(inrq(iav)-inrq(iav-Nav))/(2*Nav);
   else
     in_j=in_j+inrq(iav);
     mavinrq(Nav)=in_j/(2*Nav);
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xwrq=mult(wrq,xrq)

% multiplication of two random-pulse streams (xor)
% wrq is weight pulse stream ,xrq is input pulse stream
% xwrq is the product (also pulse stream)

global MS

for j=1:MS
   xwrq(j)=wrq(j)*xrq(j);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function   sumrq=add(xwrq1,xwrq2)

% addition of two random-pulse sequences , Samples <=1022
% xwrq1,xwrq2 are products of different weight and input pulse sequences
% sumrq is the sum of two random-pulse sequences,also pulse stream


global MS Nav

sr=zeros(1,1022);
scan=zeros(1,1022);
s1=zeros(1,1022);
s2=zeros(1,1022);

sr(1)=1;
```

```
scan(1)=1;
s1(1)=0;
s2(1)=1;

for it=1:1022

  tmp=floor(sr(it)/4)/2;

  if sr(it)>=512
    b=1;
  else
    b=0;
  end

  if ceil(tmp)>floor(tmp)
    c=1;
  else
    c=0;
  end

  a=rem(b+c,2);
  sr(it+1)=rem(2*sr(it)+a,1024);

  scan(it+1)=rem(sr(it+1),2);

  if scan(it+1)==0
    s1(it+1)=1;
    s2(it+1)=0;
  else
    s1(it+1)=0;
    s2(it+1)=1;
  end
end

for iav=1:MS
  sumrq(iav)=xwrq1(iav)*s1(iav)+xwrq2(iav)*s2(iav);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  err_quan(inx,mavxrq,xw,mavxwrq,sum,mavsumrq)

% Moving average error of input analog singal
% Moving average error of product
% Moving average error of addition of xw1 and xw2 (sum)
% inx is the input analog signal, xw is the analog product of weight and input
```

```
% mavxrq is the estimation of random-pulse stream of input signal
% mavxwrq is the estimation of product of two random-pulse streams
% mavsumrq is the moving average estimation of the sum
% of the product of two pulse streams

global MS Nav


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Moving average error of input analog singal
errx1=0;
errx2=0;
aerx=0;

for i=Nav:MS
    ERRx(i)=abs(inx-mavxrq(i));    % absolute error of input signal
   errx1=errx1+ERRx(i)^2;
   errx2=errx2+ERRx(i);
end
aerx=sqrt(errx1/(MS-Nav+1));        % mean square error of input signal
mnerx=errx2/(MS-Nav+1);             % mean absolute error
disp('mean absolute error of input signal is:');
disp(mnerx);
%disp('mean square error of input signal is:');
%disp(aerx);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Moving average error of product
errxw1=0;
errxw2=0;
aerxwr=0;

for i=Nav:MS
   ERRxw(i)=abs(xw-mavxwrq(i)); % absolute error of product
   errxw1=errxw1+ERRxw(i)^2;
   errxw2=errxw2+ERRxw(i);
end
aerxw=sqrt(errxw1/(MS-Nav+1));          % mean square error of product
mnerxw=errxw2/(MS-Nav+1);               % mean absolute error
disp('mean absolute error of product is:');
disp(mnerxw);
%disp('mean square error of product is :');
%disp(aerxw);
```

```
% Moving Average error of SOP
errxy1=0;
errxy2=0;
aersum=0;

for i=Nav:MS
  ERRsum(i)=abs(sum-mavsumrq(i)); % absolute error of addition
  errxy1=errxy1+ERRsum(i)^2;
  errxy2=errxy2+ERRsum(i);
end
aersum=sqrt(errxy1/(MS-Nav+1));          % mean square error of addition
mnersum=errxy2/(MS-Nav+1);               % mean absolute error
disp('mean absolute error of addition is:');
disp(mnersum);
disp('mean square error of addition is :');
disp(aersum);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot

figure;
subplot(3,1,1),
plot(ERRx);
axis([0 MS+20 0 1]);
title('absolute error of input quantization');

subplot(3,1,2),
plot(ERRxw);
axis([0 MS+20 0 1]);
title('absolute error of product');

subplot(3,1,3),
plot(ERRsum);
axis([0 MS+20 0 1]);
title('absolute error of addition');
```

Appendix D

```
% neuron.m is one neuron

clear;
clear global;

% Global Parameters
% MS is total samples
% Nav is moving average estimation bits
% NW bit shift register for weight
% DELTA

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

DELTA=1;
MS=1024;
Nav=64;
NW=16;
global MS DELTA Nav NW

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

input_no=2;
threshold=0;
x=[0.4 -0.65];
w=[0.75 -0.5];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('total samples is:');
disp(MS);
disp('moving average estimation bits is:');
disp(Nav);
disp('shift register bit is:');
disp(NW);


sum=0;
for i=1:input_no
   xw(i)=x(i)*w(i);
   xrq=quin(x(i),i);
   wrq=quw(w(i));
   mavx(i,1:MS)=movave(xrq);
   xwrq=mult(wrq,xrq);
   mavxw(i,1:MS)=movave(xwrq);
   XWRQ((2*i-1):2*i,1:MS)=xwrq;
   sum=sum+xw(i);
```

```
end

sumrq=nadd(XWRQ(1:2,1:MS),XWRQ(3:4,1:MS));
mavsumrq=movadd(sumrq);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%plot of the whole procedure
pl_quan(xrq,mavx(2,1:MS),x(2));
pl_mult(x(1),w(1),xw(1),mavxw(1,1:MS));
pl_add(xw(1),xw(2),mavsumrq);
err(x(1),mavx(1,1:MS),xw(1),mavxw(1,1:MS),sum,mavsumrq);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function xrq=quin(inx,i)

% input signal quantization
% inx is an input analog value
% xrq is random pulse stream carrying the value inx,2XMS;

global MS DELTA

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load c:\lichen\wnoi\1noi1024;
load c:\lichen\wnoi\2noi1024;
if i==1
  R=R1;
else
  R=R2;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for iav=1:MS
   xdit(iav)=inx+R(iav);                % dithered signal
   if xdit(iav)>DELTA/2
     xrq(1,iav)=0;                      % xrq are 2-bits,
     xrq(2,iav)=1;                      % xrq(1,iav)is MSB,xrq(2,iav) is LSB
   elseif  xdit(iav)<-(DELTA/2)
     xrq(1,iav)=1;
     xrq(2,iav)=0;
   else
     xrq(1,iav)=0;
     xrq(2,iav)=0;
   end
end
```

```
function wrq=quw(weight)

% ad hoc weight pulse streams
% choose from different weights
% MS/NW should be integer
% weight is analog value for synaptic weight, wrq is pulse stream,2XMS

global MS Nav NW

if weight ==0.75
  for wav=1:NW       % first NW bit
   if wav==2
     wrq(1,wav)=1;
     wrq(2,wav)=0;
   elseif wav==5
     wrq(1,wav)=1;
     wrq(2,wav)=0;
   else
     wrq(1,wav)=0;
     wrq(2,wav)=1;
   end
 end

   for i=1:(MS/NW-1)                % rotating the NW bit shift register
     for wav=(16*i+1):(16*i+16)
       wrq(1,wav)=wrq(1,wav-16*i);
       wrq(2,wav)=wrq(2,wav-16*i);
     end
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if weight ==-0.5
  for wav=1:NW
   if wav==2
     wrq(1,wav)=0;
     wrq(2,wav)=1;
   elseif wav==5
     wrq(1,wav)=0;
     wrq(2,wav)=1;
   elseif wav==9
     wrq(1,wav)=0;
     wrq(2,wav)=1;
   elseif wav==14
```

```
      wrq(1,wav)=0;
      wrq(2,wav)=1;
    else
      wrq(1,wav)=1;
      wrq(2,wav)=0;
    end
  end

   for i=1:(MS/NW-1)
    for wav=(16*i+1):(16*i+16)
      wrq(1,wav)=wrq(1,wav-16*i);
      wrq(2,wav)=wrq(2,wav-16*i);
    end
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xwrq=mult(wrq,xrq)

% multiplication
% wrq is weight pulse stream 2xMS ,xrq is input pulse stream 2xMS
% xwrq is the product (also pulse stream),2xMS

global MS

for i=1:MS
   if xrq(1,i)==1
     xxrq(i)=-1;
   elseif xrq(2,i)==0
     xxrq(i)=0;
   else
     xxrq(i)=1;
   end
end

for j=1:MS
   if wrq(1,j)==1
     wwrq(j)=-1;
   elseif wrq(2,j)==0
     wwrq(j)=0;
   else
     wwrq(j)=1;
   end
end
```

```
for k=1:MS
   product(k)=wwrq(k)*xxrq(k);
end

for i=1:MS
   if product(i)==1
      xwrq(1,i)=0;
      xwrq(2,i)=1;
   elseif product(i)==0
      xwrq(1,i)=0;
      xwrq(2,i)=0;
   else
      xwrq(1,i)=1;
      xwrq(2,i)=0;
   end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function   sumrq=nadd(xwrq1,xwrq2)

% addition of two random-pulse sequences
% xwrq1,xwrq2 are products of different weight and input pulse sequences,2XMS
% sumrq is the sum of two random-pulse sequences,also pulse stream


global MS

for k=1:MS
   scan1(k)=rem(k,2);
   scan2(k)=1-scan1(k);
end

sumrq(1,1:MS)=xwrq1(1,1:MS).*scan1+xwrq2(1,1:MS).*scan2;
sumrq(2,1:MS)=xwrq1(2,1:MS).*scan1+xwrq2(2,1:MS).*scan2;
```

Bibliography

[1] J. Von Neumann, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", *Automata Studies*, Princeton, NJ, 1956

[2] W. J. Poppelbaum, C. Afuso, "Noise Computer", University of Illinois, Urbana, *Technical Progress Reports*,1965

[3] K.Y.Chang, D.Moore, "Modified Digital Correlator and its Estimation Errors", *IEEE Trans. Information Technology*, vol.16,no.6, pp.699-706,1970

[4] Jouni E. Tomberg, Kimmo K.K. Kaski, "Pulse-Density Modulation Technique in VLSI of Neural Network Algorithms", *IEEE J. of Solid-State Circuits*, Vol. 25, No. 5, Oct 1990, pp.1277-1286

[5] Emil M. Petriu, Kenzo Watanabe, Tet H. Yeap, "Applications of Random-Pulse Machine Concept to Neural Network Design", *IEEE Transactions on Instrumentation and Measurement*, Vol.45, No.2, April 1996

[6] T. Yeap, G. Vukovitch, K. Watanabe, E. Petriu, "A VLSI Architecture of a Random Pulse Neural Network", *Canadian Conference on Electrical & Computer Eng.*, Sept,1995

[7] E. Pop, E. Petriu, "Influence of Reference Domain Unstability upon the Precision of Random Reference Quantizer with Uniformly distributed Auxiliary Source", *Signal Processing*, vol.5, pp.87-96,1983

[8] Brian R. Gaines , "Stochastic Computer Thrives on Noise", *Electronics* , July 10, 1967

[9] Alan F. Murray, "Pulse Arithmetic in VLSI Neural Networks", *IEEE Micro* , Dec 1989

[10] Philip Treleaven, Marco Pacheco, Marley Vellasco, "VLSI Architectures for Neural Networks", *IEEE Micro*, Dec.1989

[11] Mahmoud Fawzy Wagdy, Michael Goff, "Linearizing Average Transfer Characteristics of Ideal ADC's via Analog and Digital Dither", *IEEE Transactions on Instrumentation and Measurement* , Vol. 43, No. 2, April 1994

[12] Gyu Moon, Mona E. Zaghloul, Robert W. Newcomb, "VLSI Implementation of Synaptic Weighting and Summing in Pulse Coded Neural Type Cells", *IEEE Transactions on Neural Networks* , vol.3, no. 3, May 1992

[13] V.C.V Pratapa Reddy, "Moving-Average Output Interface for Digital Stochastic Computers", *Electronics Letters*, 17th April 1975, vol. 11, no.8

[14] Mavru Miura, Masaru Goishi, Katsuhiko Takahashi, "A Neural Network Structure Based on Pulse-Train Arithmetic", *IECON'91*, pp.1339-1342,1991

[15] F. Castanie, "Signal Processing by Random Reference Quantizing", *Signal Processing*, 1979, pp.27-43

[16] J. Alsepector, J. W. Gannett, S. Haber, M. B. parker, R. Chu, "A VLSI-efficient Technique for Generating Multiple Uncorrelated Noise Sources and its Application to Stochastic Neural Networks", *IEEE Transactions on Circuits and Systems*, vol.38, no.1, Jan 1991

[17] Peter J. W. Melsa, "Neural Networks : A Conceptual Overview", *TRC-89-08*, Tellabs Research Center,Aug,1989

[18] Patrick k. Simpson , "Foundations of Neural Networks", *Artificial Intelligence IEEE Press 1992*

[19] Sergio T. Ribeiro, "Random-Pulse Machines", *IEEE Transactions on Electronic Computer*, vol. EC-16, no.3, June 1967

[20] Alan F. Murray, Dante Del Corso, Lionel Tarassenko, "Pulse-Stream VLSI Neural Networks Mixing Analog and Digital Techniques", *IEEE Transactions on Neural Networks*, vol.2, no. 2, March 1991

[21] Alan F. Murray, Anthony V.W. Smith, "Asynchronous VLSI Neural Networks using Pulse-Stream Arithmetic", *IEEE Journal of Solid-State Circuits*, vol.23, no.3 , June 1988

[22] K. Goser, U. Hillerignmann, U. Rucckert, K. Schumacher, "VLSI Technologies for Artificial Neural Networks", *IEEE Micro*, Dec. 1989

[23] A.J.Miller, A.W. Brown, P.Mars, "Moving-Average Output Interface for Digital Stochastic Computers", *Electronic Letters*, Vol.10, No.20, pp.419—420, Oct.1974

[24] A.J.Miller, P.Mars, "Optimal Estimation of Digital Stochastic Sequences", *Int. J. Syst. Sci.* ,Vol.8, No.6, pp.683—696,1977

[25] T. Yeap, G. Vukovitch, G. Eatherly, E. Petriu, "An Experiment on a Random-Pulse Neural Network", *Proceeding of Emergent Technology for Instrumentation and Measurement Int. Workshop*, pp.18-25, Italy,1996

[26] A. J. Miller, A.W. Brown, P. Mars, "Adaptive Logic Circuits for Digital Stochastic Computers", *Electronics Letters*, 18th Oct 1973,vol.9,No.27

[27] Les. E. Atlas, Yoshitake Suzuki, "Digital Systems for Artificial Neural Networks", *IEEE Circuits and Devices Magazine*, Nov. 1989, pp.20—24

[28] A.P. Thakoor, A. Moopenn, John Lambe, S.K. Khanna, "Electronic Hardware Implementation of Neural Networks", *Applied Optics*, 1st Dec. 1987, Vol.26, No.23

[29] Pasi Kolinummi, Timo Hamalainen, Kimmo Kaski, "Designing a Digital Neurocomputer", *Circuits & Devices*, March 1997

[30] Young-chul Kim, Michael A. Shanblatt, "Architecture and Statistical Model of a Pulse-Mode Digital Multilayer Neural Network", *IEEE Trans. on Neural Networks*, Vol.6, No.5, Sept. 1995, pp.1109-1118

[31] C. H. Chen, "*Fuzzy Logic and Neural Network Handbook*", McGraw-Hill, 1996

[32] M. S. Tomlinson Jr., D. J. Walker, M. A. Sivilotti, "A Digital Neural Network Architecture for VLSI", *IJCNN* vol.2, pp.545-556, 1990

[33] Young-Chul Kim, Michael A. Shanblatt, "An Implementable Digital Multilayer Neural Network (DMNN)", *IJCNN*, vol.2,1992, pp.594-600,

[34] Gamze Erten, Rodney M. Goodman, "A Digital Neural Network Architecture Using Random Pulse trains", *IJCNN*, vol.1,1992, pp.190-201

[35] Holger Broman, "Quantization of Signals in Additive Noise: Efficiency and Linearity of the Equal Step-Size Quantizer", *IEEE Trans. on ASSP*, vol. 25, No.6, pp.572-577, 1977

[36] Anekal B. Sripad, Donald L. Snyder, "A Necessary and Sufficient Condition for Quantization Errors to be Uniform and White", *IEEE Trans. on ASSP*,vol.25, pp.442-448, Oct. 1977

[37] Mahmoud F. Wagdy, Wai Man NG, "Validity of Uniform Quantization Error Model for Sinusoidal Signals Without and With Dither", *IEEE Trans. on Instrumentation and Measurement*, vol.38, No.3, pp.718-722,1989

[38] Young Chul Kim, Michael A. Shanblatt, "Random Noise Effects in Pulse Mode Digital Multilayer Neural Networks", *IEEE Trans. on Neural Networks*, vol.6, no.1, pp.220-229, Jan 1995

[39] Paolo Carbone, Dario Petri, "Effect of Additive Dither on the Resolution of Ideal Quantizer", *IEEE Trans. on Instrumentation and Measurement*, vol.43, no,3, pp.389-396, June 1994

[40] Mahmoud F. Wagdy, "Effect of Various Dither Forms on Quantization Errors of Ideal A/D Converters", *IEEE Trans. on Instrumentation and Measurement*, vol.38 no.4, pp.850-855, Aug 1989

[41] Young Chul Kim, Michael A. Shanblatt, "Architecture and Statistical Model of a Pulse Mode Digital Multilayer Neural Network", *IEEE Trans. on Neural Networks*, vol.6, no.5, pp.1109-1118, Sept 1995

[42] Les E. Atlas, Yoshitake Suzuki, "Digital Systems for Artificial Neural Networks", *IEEE Circuits and Devices* , Nov. 1989, pp.20-24

[43] M. van Daalen, T. Kosel, P. Jeavons, J. Shawe Taylor, "Emergent Activation Functions from a Stochastic Bit-Stream Neuron", *Electronics Letters*, 17th Feb 1994, vol.30, no.4

[44] A. F. Murray, A.V.W. Smith, "Asynchronous Arithmetic for VLSI Neural Systems", *Electronics Letters*, 4th June 1987, vol. 23, no.11

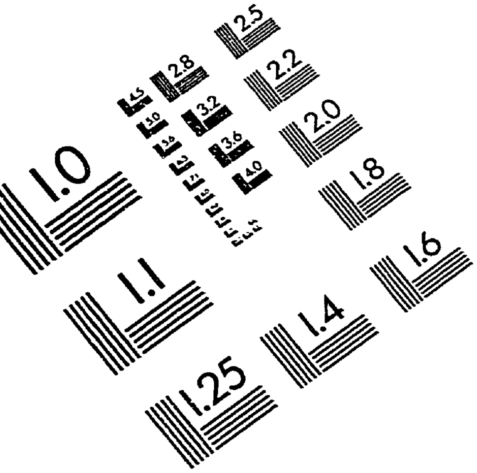
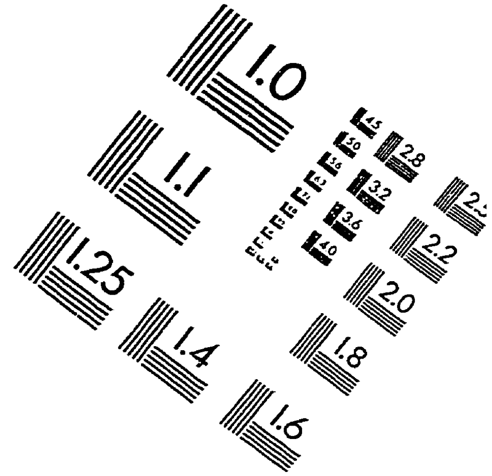
[45] Carver Mead, "*Analog VLSI and Neural Systems*", Addison-Wesley, 1989

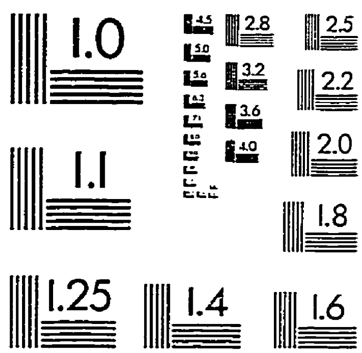
[46] S. Haykin, "*Neural networks, a comprehensive foundation*", Macmillan, New York, NY, 1994

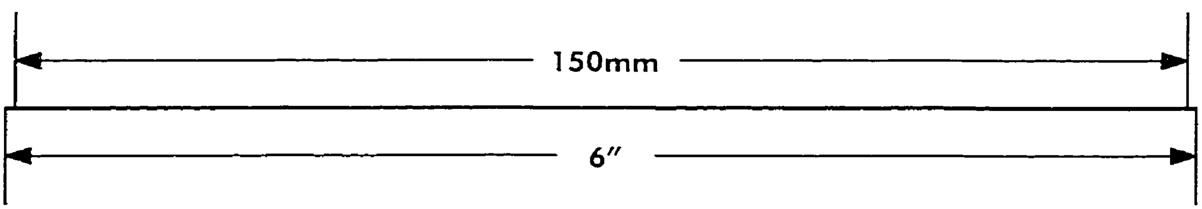
[47] E. Petriu, A. Guergachi, G. Patry, L.Zhao, D. Petriu, G. Vukovich, "Artificial neural architecture for real time modeling applications", *1997 3rd International Conference on Algorithms & Architecture for Parallel Processing, ICA*^{3}*PP97,* pp.639-650, Melbourne, Australia

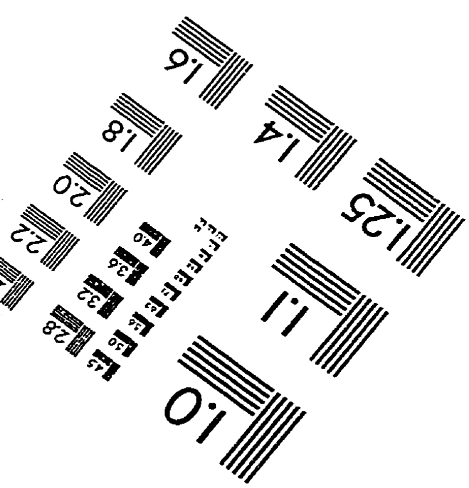
[48] M. T. Hagan, H.B. Demuth, M. Beale, "*Neural Network Design*", PWS publishing co., 1995

IMAGE EVALUATION TEST TARGET (QA-3)







APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989