USC-SIPI Report #152

VLSI Design of Adaptive Neural Systems

By

Bang Won Lee

May 1990

Signal and Image Processing Institute
University of Southern California
Department of Electrical Engineering-Systems
Powell Hall of Engineering
University Park, MC-0272
Los Angeles, CA 90089 U.S.A.

Copyright 1990 Bang Won Lee

Acknowledgment

I would like to express thanks towards my research advisor Professor Bing Sheu for his guidance and support throughout my Ph. D. study. I would also like to extend my gratitude towards Professor Irving Reed and Professor Wlodek Proskurowski for stimulating my research interests, serving on the dissertation committee, and giving valuable suggestions to dissertation writing.

I also wish to express appreciation to Professors John Choma, Jr. and Robert Gagliardi, who served on my qualifying examination along with Professors Irving Reed and Wlodek Proskurowski. Professor Mike Arbib, the Director of Center for Neural Engineering, Professor Rama Chellappa, the Director of Signal and Image Processing Institute, and Professor Bart Kosko have greatly stimulated the neural network research in USC, from which I benefit quite a lot. I thank ISI/MOSIS's Director George Lewicki, Deputy Director Cesar Pina, and Manager Vance Tyree for providing supplementary research assistantship and access to high-quality research facility.

Interaction with fellow colleagues in the Integrated Circuits and Microsystem field has been very beneficial. Sudhir Gowda and Ji-Chen Lee help to obtain some measurement results. Wen-Jay Hsu helps to manage the computing facility. I would like to thank other professors in USC and Kathie Fry, Wes Hansford, Sam Delatorre from ISI/MOSIS Service to provide great research environment. Many thanks to other fellow colleagues in Integrated Circuits and Microsystems Research Lab. The graduate-level neural network course from Professor Keith Jenkins in Spring 1988 gave me a great inspiration to this field.

I also appreciate deeply to many people in Samsung Semiconductor Co., especially Vice President Kwang-Ho Kim and R&D Center Head Kwang-Kyo Kim for their encouragement and support. Finally, I would like to thank my wife, Hae-kyung, children, Sun Oak and Won-Woo, and my parents for their

encouragement as well as the sacrifice through all these years.

This research was supported by Grants from AT&T Bell Laboratories, Powell Foundation, USC-Faculty Research and Innovation Fund, and National Institute for Health through USC-Biomedical Research Support Program.

Table of Contents

I. Introduction 1

1.1. Overview of neural architectures 4

1.2. VLSI neural network design methodology 14

II. VLSI Hopfield Networks 24

2.1. Circuit dynamics of Hopfield networks 26

2.2. Existence of local minima 28

2.3. Elimination of local minima 33

2.4. Neural-based A/D converter without local minima 36

2.4.1. The step function approach 44

2.4.2. The correction logic approach 52

2.5. Traveling salesman problem 60

2.5.1. Competitive-Hopfield network 63

2.5.2. Search for optimal solution 65

III. Hardware Annealing of Artificial VLSI Neural Networks 74

3.1. Simulated annealing in software computation 74

3.2. Electronic annealing 78

3.2.1. Starting temperature of the cooling schedule 78

3.2.2. Final temperature of the cooling schedule 84

3.3. Application to the neural-based A/D converter 87

3.3.1. Neuron gain requirement 87

3.3.2. Relaxed gain requirement using modified synapse weightings 93

IV. Programmable Synapses and Gain-Adjustable Neurons 100

4.1. Programmable neural chips 101

4.2 Medium-term and long-term storage of synapse weight 111

4.2.1. DRAM-style synapse weight 112

4.2.2. EEPROM-style synapse weight 116

V. System Integration of VLSI Neural Computing Machines 126

5.1. System module using the programmable neural chip 126

5.2. Application examples 130

5.2.1. Hopfield neural-based A/D converter 132

5.2.2. Modified Hopfield network for image restoration 144

VI. Conclusion and Suggested Future Studies 155

Appendixes 159

A. Program for neural-based A/D conversion and traveling salesman problems 159

B. Non-saturated input stage for large dynamic range in the synapse circuit 182

List of Tables

Table 1.1 Major neural network models 10

Table 1.2 Basic learning rules 13

Table 2.1 Analog input range for each digital code 41

Table 2.2 Analog range after generalized correction 51

Table 2.3 Logic table for the correction logic circuitry 53

Table 2.4 Analog range for adjacent-code correction 56

Table 4.1 Device geometries of programmable synapse and related neurons .. 109

Table 5.1 Computational speed of 4-bit A/D conversion 138

Table 5.2 Comparison of three image restoration systems 153

Table 6.1 Comparison of correction and annealing schemes 156

List of Figures

Fig. 1.1 Advanced neural computing system 3

Fig. 1.2 Neuron model 5
(a) Biological neuron model. (b) Artificial neuron model.

Fig. 1.3 Several neural networks 7
(a) Single-layer network. (b) Single-layer network with feedback.
(c) Three-layer network.

Fig. 1.4 A VLSI neuron with direct resistor implementation 15

Fig. 1.5 Network dynamic of a 10-city TSP 17
(a) Transient Behavior.
(b) Solution with adaptive time-step and with fixed time-step of 1.0e-4 and 2.0e-5.
(c) Solution with fixed time-step of 1.0e-3.

Fig. 2.1 A Hopfield neural network with inverting and non-inverting amplifiers as neurons 25

Fig. 2.2 Calculation of the characteristic parameter *GAP* 30
(a) $GAP_i < 0$. (b) $GAP_i = 0$. (c) $GAP_i > 0$.

Fig. 2.3 Modified Hopfield neural network without local minima 32
(a) Transfer characteristics of the Hopfield amplifier.
(b) Transfer characteristics of the correction amplifier.
(c) The complete neural network.

Fig. 2.4 A 4-bit neural-based A/D converter 37

Fig. 2.5 Digital output versus analog input characteristics of the A/D converter shown in Fig. 2.4 43

Fig. 2.6 Energy function of the original Hopfield A/D converter 46
(a) Energy surface with changing V_1 and V_2.
(b) The corresponding contour.

Fig. 2.7 Energy function of a modified Hopfield A/D converter 47
(a) Energy surface with changing V_1 and V_2.

(b) The corresponding contour.

Fig. 2.8 A 5-bit modified neural-based A/D converter 49
(a) Transfer characteristics of the Hopfield amplifier.
(b) Transfer characteristics of the correction amplifier.
(c) Complete circuit schematic.

Fig. 2.9 Measured transfer characteristics of 5-bit neural-based A/D converters 50
(a) Original Hopfield A/D converter.
(b) Modified Hopfield A/D converter.

Fig. 2.10 Circuit schematic of a modified 4-bit Hopfield A/D converter with the correction logic gates 54

Fig. 2.11 SPICE simulation results on the transfer characteristics of the original Hopfield A/D converter and the modified A/D converter .. 55

Fig. 2.12 Die photo of the modified A/D converter with correction logic gates 58

Fig. 2.13 Measured transfer characteristics of 4-bit neural-based A/D converters 59
(a) Original Hopfield network. (b) Modified Hopfield network.

Fig. 2.14. Transient responses 61
(a) A/D converter output. (b) Individual amplifier outputs.

Fig. 2.15 A competitive-Hopfield neural network 64

Fig. 2.16 One 10-city TSP 66

Fig. 2.17 Another 10-city TSP 67

Fig. 2.18 A 20-city TSP 68

Fig. 2.19 One 20-city TSP with averaged correction scheme 69

Fig. 2.20 Another 20-city TSP with averaged correction scheme 70

Fig. 3.1 Analogy between the annealing temperature of a Boltzmann machine and the amplifier gain of an electronic neuron 76

Fig. 3.2 A Hopfield neural network with non-inverting amplifiers as neurons 80

Fig. 3.3 Radius of eigenvalues with different amplifier gains 83

Fig. 3.4 Stiffness of the system function of a neural-based A/D converter.. 86

Fig. 3.5 Trajectory of the largest eigenvalue of Hopfield A/D converter against neuron gain 88

Fig. 3.6 Neural-based A/D converter outputs versus amplifier gain 90

Fig. 3.7 SPICE simulation results of a 4-bit Hopfield neural-based A/D converter in the time domain analysis with $V_{in} = 1.4V$ 91

Fig. 3.8 Trajectories of the A/D converter output versus the amplifier gain 92
(a) Amplifier D_1. (b) Amplifier D_2, D_3, and D_4.

Fig. 3.9 A 4-bit neural-based A/D converter with unity synapse weighting 94

Fig. 3.10 Transfer characteristics of a 4-bit Hopfield A/D converter 96
(a) Without hardware annealing.
(b) With hardware annealing.

Fig. 3.11 Different annealing styles 97
(a) Abrupt heating and slow cooling with $V_{in} = 0V$ and $3.3V$.
(b) Slow heating and abrupt cooling with $V_{in} = 0V$ and $3.3V$.
(c) Abrupt heating and slow cooling with $V_{in} = 0V$ and $2.1V$.
(e) Abrupt heating and abrupt cooling with $V_{in} = 0V$ and $2.1V$.

Fig. 4.1 A programmable synapse cell used in ETANN chip from Intel Inc. 102
(a) Circuit schematic. (b) Simulated transfer characteristics.

Fig. 4.2 Transconductance amplifier as the core for the programmable synapse cell 104
(a) Circuit schematic. (b) Equivalent model for binary inputs.

Fig. 4.3 Simulated synapse characteristics 106
(a) Synapse output current versus neuron input voltage.
(b) Equivalent synapse conductance versus programmed synapse voltage.

Fig. 4.4 Circuit schematic of the programmable synapse and related neurons 108

Fig. 4.5 Simulated charge retention characteristics in DRAM-style synapse cell 113

Fig. 4.6 DRAM-style synapse cell 114
(a) Die photo.
(b) Measured result of charge retention test.
(c) Measured result of the synapse characteristics.

Fig. 4.7 Die photos 115
(a) Input neuron. (b) Output neuron.

Fig. 4.8 Floating-gate transistor 117
(a) Top view. (b) Cross section.
(c) Equivalent large-signal model.

Fig. 4.9 Measured results of several programming schemes 119
(a) Threshold voltage change versus the number of fixed-width/magnitude pulses.
(b) Threshold voltage change versus a single pulse with variable width.
(c) Threshold voltage versus the number of fixed-width pulses with programming voltage being increased from 12 V or decreased from -20 V by 0.5 V after every 10 pulses.

Fig. 4.10 Measured charge retention characteristics of floating-gate transistors at an elevated temperature 121

Fig. 4.11 Die photos 123
(a) Programming device. (b) EEPROM-style synapse cell.
(c) 4-bit Hopfield A/D converter with EEPROM-style synapse cells.

Fig. 5.1 Neural computing board diagram using DRAM-style programmable synapses 127

Fig. 5.2 Neural computing board diagram using EEPROM-style programmable synapses 129

Fig. 5.3 Real-time neural computing system with a multi-layer network 131

Fig. 5.4 Circuit schematic of an 8-bit neural-based A/D converter with the programmable synapses and gain-adjustable neurons 133

Fig. 5.5 Die photo of the 8-bit neural-based A/D converter in Fig. 5.4 134

Fig. 5.6 Response time characteristics of neural-based A/D converters 135
(a) Original Hopfield A/D converter.
(b) Self-correction A/D converter.

Fig. 5.7 Transient response 136
(a) Each amplifier output of a Hopfield 4-bit A/D converter.
(b) 5-bit Hopfield A/D converter output.

Fig. 5.8 Software simulation results 139
(a) Adaptive time step.
(b) Fixed time step of 1.0×10^{-2} sec.
(c) Fixed time step of 1.0×10^{-3} sec.
(d) Fixed time step of 1.0×10^{-4} sec.

Fig. 5.9 Finite gain and input offset voltage effects 141
(a) Bias current for each amplifier is 2.5 μA.
(b) Bias current for each amplifier is 12.5 μA.

Fig. 5.10 Die photos of general-purpose neural chips using DRAM-style storage 143
(a) With multiplexing I/O pins. (b) Without multiplexing I/O pins.

Fig. 5.11 Architecture of image processing neural chip 145

Fig. 5.12 Logic schematic of one pixel programmable register 148

Fig. 5.13 Cifplot of a synchronous Hopfield network for image restoration .. 149

Fig. 5.14 System simulation results on the house image 150
(a) Blurred picture of girl image.
(b) Restored picture of girl image.
(c) Blurred picture of house image.
(d) Restored picture of house image.

Fig. 5.15 Schematic diagram of a multi-chip image restoration system 152

Fig. A.1 Conventional input differential pair in MOS amplifier 183

Fig. A.2 Improved CMOS non-saturated input differential pair 185

Abstract

VLSI neural circuits play an important role in exploring and exploiting the rich properties of neural networks associated with massively parallel processing using analog neurons and synapses. The algorithms for artificial neural networks can be mapped to the VLSI neural systems if the building blocks exist. Novel techniques which find and eliminate the local minima in Hopfield networks have been developed. With modified Hopfield networks using these techniques, the local minima in Hopfield neural-based A/D converters and traveling salesman problems are successfully eliminated. A competitive-Hopfield network, where the constraint functions are implemented in the competitive network, is very effective and efficient in finding a valid, near-optimal solution for the combinatorial optimization problem. The hardware annealing technique using the analogy between the annealing temperature and neuron gain is also described. Due to intrinsic continuity of the hardware annealing process, the computational time for searching the optimal solution is minimized. A compact and electrically programmable analog synapse cell in CMOS technologies is presented. Both DRAM-style and EEPROM-style weight storage mechanism are explored. By using the programmable synapses and gain-adjustable neurons, prototyping integrated-circuits which can operate in either asynchronous or synchronous modes are fabricated and evaluated. The neural-based A/D conversion and digital image restoration are used as system-level application examples. The VLSI neural chips can conduct network retrieving and learning processes concurrently. In the DRAM-style neural chips, weights of analog synapse cells are externally programmed and require dynamic refreshing. In the EEPROM-style neural chips, synapse weights are permanently stored in the floating gate. If the compact synapse and neuron cells are used in the industrial-level 1-μm VLSI technologies, a fully-connected general-purpose neural chip with 500 neurons can be achieved in 1 cm^2 silicon area.

Chapter 1

Introduction

Recent advances in neural sciences and microelectronic technologies have provided an excellent means to boost the computational capability and efficiency in complicated engineering tasks by several orders of magnitude [1]. Due to the optimized structure for artificial intelligence applications, a neural computer is considered as the most promising six-generation computing machine. The inter-disciplinary nature of neural network studies span many science and engineering fields including neuroscience, cognitive science, psychology, computer science, physics, mathematics, electrical engineering, and biomedical engineering. Many neural co-processors, which usually interface to personal computers and engineering workstations, are commercialized for accelerating neuro-computation. Typical products include ANZA from Hecht-Nielson Co., SAIC from Sigma, Odyssey from TI, and Mark III/IV from TRW Inc. [2]. However, the general-purpose neural co-processor is usually much slower than the special-purpose neural hardware which implements the neural networks in the optimal fashion.

Implementation technologies for the special-purpose neural machines can be classified into two categories. The first technology is the electronic implementation using analog, digital, or mixed-signal chips. Several significant achievements have been made including important results from AT&T Bell Labs. and California Institute of Technology. By using a state-of-the-art MOS technology, several hundreds of neurons and thousands of synapses can be realized in a single IC chip. However, the present silicon integration scale is still very limited

as compared with biological systems. The other implementation technology is to use optical and optoelectronic devices. Due to the inherent parallelism and spatial property of the optics, the interconnection and synapse weighting problems in multi-dimensional signal processing can be more efficiently addressed for certain applications. The high sensitivity problem on the portability of the opto-neural computer will diminish by combining electronics with the optics.

The secret of immense computational power in the neural networks is discovered as the parallel processing done by neurons and synapses. While each neuron performs simple analog processing at a low speed, the rich connectivity among neurons through synapses provides powerful computational capabilities for the large quantity of data. The data are processed asynchronously in the continuous time domain and spread globally into all network elements. In addition to the parallel processing nature, the neural network has self-learning capability which is done by changing the weights of synapses between neurons. The self-learning capability makes neural networks useful at the situation that training data are insufficient and that fault tolerance of a system is necessary. Moreover, the relatively imprecise network elements can be compensated with the self-learning scheme. The immense computational power and self-learning capability give neural networks excellent prospects in image processing, speech processing, artificial sensor processing, inexact knowledge processing, natural language processing, planning, forecasting, linear/nonlinear programming, and scientific optimization.

Figure 1.1 shows a block diagram of an advanced neural computing system. The real world signals are converted into the discrete form (mainly digital) at

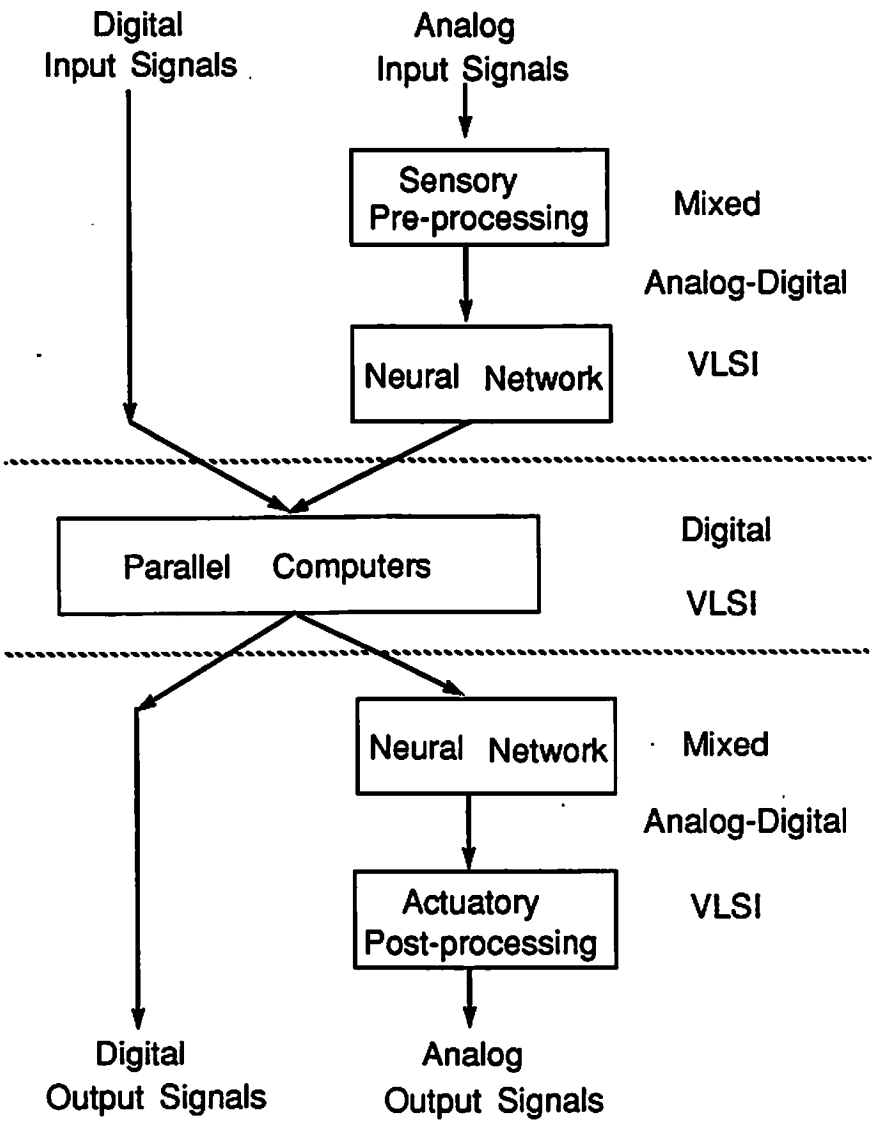
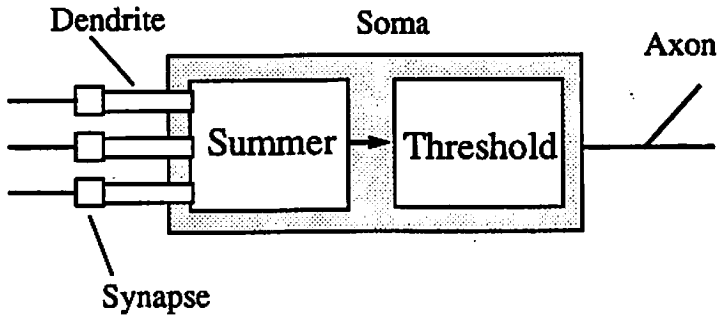


Fig. 1.1 Advanced neural computing system.

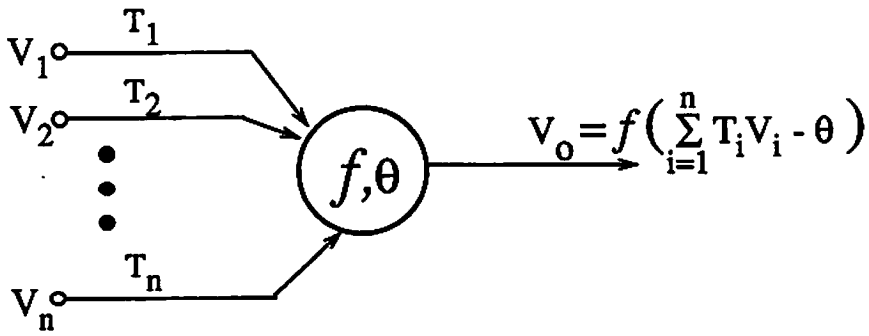
the interface block. The neural signal processing system handles the converted signals and the outputs can be transferred to digital computers for further data manipulation. The interface block might function as a data converter and conduct some early signal processing to the same level that human retinas can do. Due to the robustness of neural networks, outputs of the interface do not have to be completely precise which is in sharp contrast to the conventional interfacing devices. The signals inside a neural signal processing system are distributed throughout the whole network. Thus, a small amount of damage in the system does not produce noticeable degradation of the overall performance. Through self-learning procedures, the interconnection weights can be modified so that the original system performance is retained.

1.1. Overview of Neural Architectures

A biological neuron basically consists of a cell body, dendrites, and axons as shown in Fig. 1.2. The cell body, which is called *soma*, performs complicated chemical processes, such as summation and firing with respect to a threshold level. The input signals for a cell body are transmitted through the dendrite, while the output signals are carried to other cells through axons. The electrical signal of an axon connects to a dendrite through a specialized contact, which is called the synapse. In general, the neuron performs a simple threshold function. When the potential inside the cell body is larger than the threshold value, the neuron fires. Here, the normal firing rate is quite low which typically requires a few milli-second. It is known that the human brain has approximately 10^{11} neurons and 10^4 synapses. In the artificial neural network, the neuron and



(a)



(b)

Fig. 1.2 Neuron model.
(a) Biological neuron model.
(b) Artificial neuron model.

synapse are configured as processing element and connection strength, respectively. The various features of the artificial neural network are determined by the function of neuron and the interconnection patterns.

The artificial neural network can be characterized by the following properties:

network topology,
retrieving process,
training procedure, and
input values.

The network topology gives the most distinguished feature. The grouped neurons which are arranged into a disjointed structure is called *layer*. Figure 1.3 shows several topologies which include single-layer/multi-layer and feedforward/feedback networks. The neuron transfer function and threshold voltage characterize the retrieving process of an artificial neural network. Specific mathematical functions including sigmoid, step, Gaussian, and Boltzmann functions are widely used to model the neuron transfer function. The nonlinear transfer function decides the information propagation properties at the neural retrieving process. The retrieving process can operate in either synchronous or asynchronous mode. In the synchronous mode, all neuron outputs are updated simultaneously. On the other hand, the neuron updating process in the asynchronous mode is random and independent of the other neurons. Most artificial neural networks in software computation operate synchronously, while the biological neural network operates in the fully asynchronous mode. The training

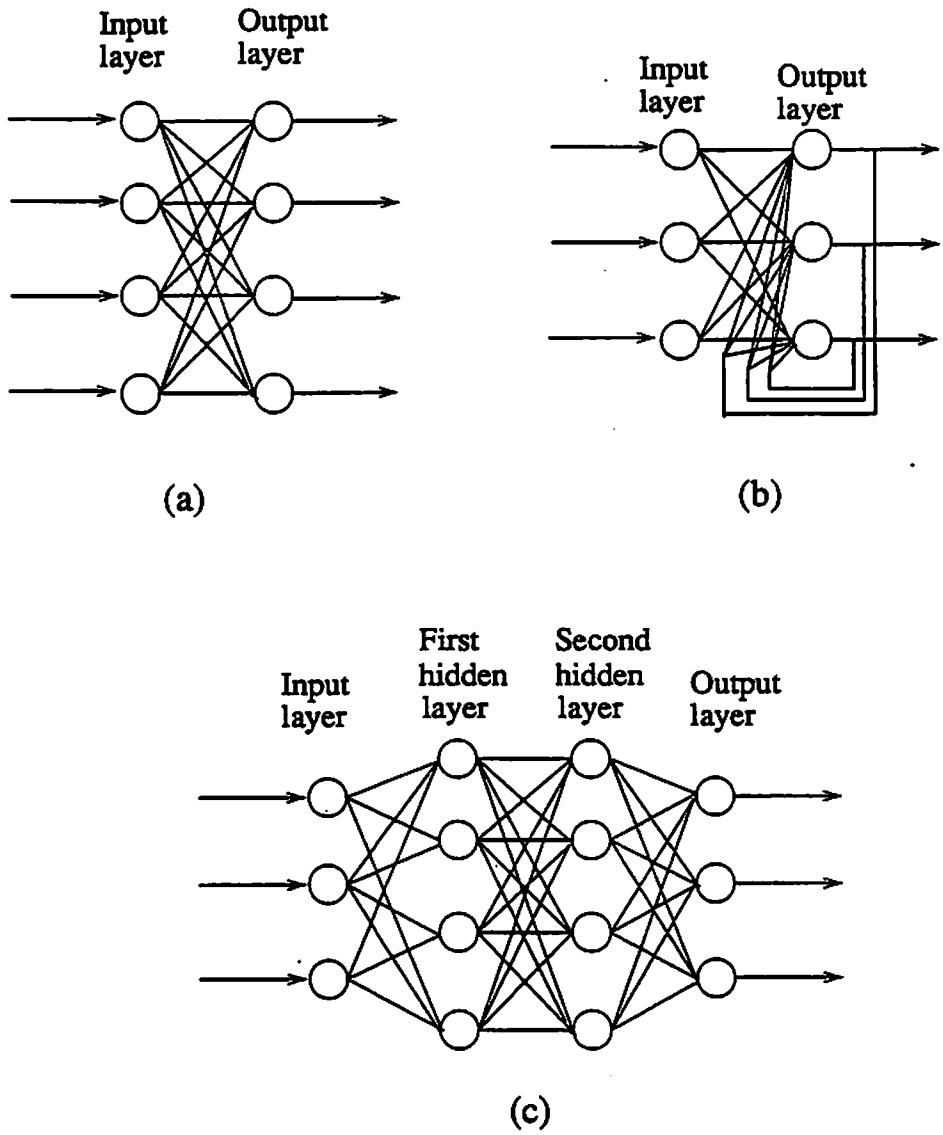


Fig. 1.3 Several neural networks.
(a) Single-layer network.
(b) Single-layer network with feedback.
(c) Three-layer network.

procedures are divided into *supervised learning* and *unsupervised learning*. In the supervised learning, synapse weightings are tuned by the difference between the retrieving patterns and expected patterns. In the unsupervised learning, the network classifies the input without references. The neural networks using unsupervised learning can detect the pattern regularities and group for each input patterns. The widely used learning rules include Hebbian rule, Delta rule, competitive learning rule, Boltzmann learning rule, Hopfield energy minimizing rule, and their derivatives [3,4]. In general, the input signal for an artificial neural network can be discrete or continuous values.

After McCulloch and Pitts introduced the abstract neuron model for performing a simple task in 1943 [5], the neural network study has been started. The major neural networks and learning rules are listed in Table 1.1 and Table 1.2, respectively. F. Rosenblatt developed *Perceptron* which sparked a great amount of research interest in neural computing [6]. The Perceptron is a single layer feedforward network for pattern classifiers. Initially, the Perceptron demonstrated an optical pattern recognition. Inputs of the system were connected to a grid of photocells. The input signals were transferred to the neural layer with randomly weighted connections. The neural network performs successfully with the Hebb's learning rule. The major limitation, which was pointed out by Minsky and Papert, is that the Perceptron can not represent an exclusive-or function so that Perceptron can not classify the complex categories. Multi-layer perceptrons were developed in the 60's to overcome the limitation of the initial Perceptron by Rosenblatt.

In the late 50's, the first neural hardware, which is called *adaptive linear element* (ADALINE), was developed by B. Widrow [7]. The neurons were realized with vacuum-tube amplifiers, while the synapse weightings were manually adjusted with variable resistors. The ADALINE was improved to become MADALINE, which consists of ADALINE and a two layer variant. They were adapted to a variety of applications, such as speech recognition, character recognition, weather prediction, adaptive control, and echo cancellation.

Another major category in neural networks is associative memory. J. Anderson proposed 'Brain-State-in-a-Box Model' with his linear associator and Hebb's learning rule [8]. The network consists of a layer with feedback and one post-processing output layer. Due to the positive feedback topology and the learning rule, the output is the best-matched pattern from the stored memory at a given input.

J. Hopfield's paper presented to the National Academy of Science ignited the neural network study again [9]. The Hopfield network is basically a single layer with feedback. The condition for the synapse weighting is very restrict (being symmetric and no self-feedback terms), while that for the neuron transfer function is very relaxed (only monotonically increasing). Using the energy of Lyapunov function, J. Hopfield proved that the network always moves toward a low energy level. Due to the simple architecture and clearly proved dynamics of the Hopfield network, many hardware implementations and real-world applications have been done. This network has been applied to the associative memory and many engineering optimization problems.

Table 1.1 Major Neural Network Models

Name	Years introduced	Primary Applications
Perceptron	1957	Typed-character recog., Simple pattern classifier
ADALINE/ MADALINE	1960	Adaptive processing; (equalizer, modem), Hand-written char. Recog.
Brain-State-in-a-Box (Linear Associator)	1977	Associative memory
Hopfield Network	1982	Image processing, Assoc. memory, Problem solver
Multi-layer w/ Back-propagation	1974-1985	Wide ranges: Speech synthesizer, Image Processing
Boltzmann Machine	1985	Pattern recognition for radar, sonar
Bidirectional Associative Memory	1986	Associative Memory

The multi-layer neural networks are vitalized by the back-propagation learning rule. Before the learning rule was developed, the usefulness of the multi-layer neural network had been well-known, but the decision of synapse weightings was the main problem. The multi-layer neural network can be used for various applications including data encoding/decoding, data compressing, signal processing, noise filtering, pattern classifying and forecasting.

A Boltzmann machine has the same network topology as the Hopfield network, but differs in the stochastic update and learning capabilities. The stochastic update in retrieving and learning processes is based on the simulated annealing technique using the Boltzmann probability function. By decreasing the temperature of the probability function from a high value, the system always finds the global minimum in the energy surface.

The bi-directional associative memory (BAM) designed for optical computing is a generalized Hopfield model to hetero-associative network [10]. BAM has two fully connected central layers and input/output buffer layers. The synapses and neurons in central two layers are bi-directional. For a given input, the BAM layers oscillate until a stable state is reached. The final stable output is the closest association stored in BAM.

Learning is the process of adapting the synapse weightings in response to external stimuli. The learning rules were developed with the network architectures. The first learning rule named *Hebb's rule* which shows the neural network can learn for a certain function was presented in 1957 [11]. The rule requires that if an input and the output are activated at the same time, the weighting between the input and output is increased. T. Kohonen developed

competitive learning where each neuron competes with others at a given inputs and the winner adapts to get more strength [12]. This type of learning called *unsupervised learning* does not need the reference data. On the other hand, the desired outputs can be given in the *supervised learning* approach. The simple *delta rule* is to adjust the synapse weightings using the error between the desired output and actual network output. Many derivatives from the simple delta rule are used for efficient learning results. The famous application of the supervised learning is back-propagation for a multi-layer network. The root-mean-square error at the output layer propagates backward through the network and is used to update the synapse weighting between layers. The process continues until the input layer is reached. The delta rule or its derivatives are used for determining the synapse weighting modifications. Due to extremely long learning time, only one or two hidden layers are popularly used. The counter-propagation learning [13], which can be only applied to three-layer networks, combines the competitive learning and delta rule. Thus, the counter-propagation network select the nearest stored pattern due to the competitive operation inside the hidden layer.

Another type of learning falling between the unsupervised learning and supervised learning is *reinforcement learning* [14]. In this learning, external observer gives a response whether the network output is good or not. The learning rule of a Boltzmann machine [15] is based on the stochastic process, which constructs distributed representations of the reference patterns with the simulated annealing technique. Due to the cooling procedure, the retrieving and learning time of the Boltzmann machine is extremely long.

Table 1.2 Basic Learning Rules

Model	Basic Equation	Comments
Hebb's Rule	$\Delta T_{ij} = \alpha V_i \bullet V_j$	Unsupervised, Hopfield network
Competitive Learning	$\Delta T_{ij} = \alpha V_i (V_j - T_{ij})$	Unsupervised, Associative Memory
Delta Rule	$\Delta T_{ij} = \alpha V_i \bullet E_j$	Supervised, Back propagation
Boltzmann Machine	Boltzmann probability function w/ simulated annealing	Supervised, Boltzmann Machine

1.2. VLSI Neural Network Design Methodology

The special-purpose VLSI implementation of neural circuits and systems seems to be extremely attractive in terms of size, power, and speed [16]. Various circuit design styles including pure analog, digital, and mixed-signal circuitries can be utilized to construct the VLSI neural chips. Since all network elements in the VLSI technology should be laid out on a two-dimensional plane, a significant limitation of the VLSI implementation is caused by a large number of interconnections and synapses. Thus, the existing programmable VLSI neural circuits are mainly restricted to compute only one dimensional input information and to produce one-dimensional outputs. When the input signals include two-dimensional information, a special mapping process should be considered during the implementation.

The neural network algorithms for the software computation can not be directly used to realize VLSI neural networks. Basic elements of the VLSI artificial neural networks consist of amplifiers as neurons and resistors as synapses. The neuron transfer function for the software computation is an exact mathematical function, while that for the VLSI implementation is an approximate function. Since the neuron transfer function is an important factor to decide the network characteristics, some neural networks, such as Boltzmann Machine and back-propagation, can not be directly realized in VLSI design. In addition, perfect match and wide dynamic ratio of the synapse weightings are difficult to obtain in VLSI technologies.

One important factor in VLSI neural network design is to increase the amplifier gain. Figure 1.4 shows the simple circuit realization of a neural model

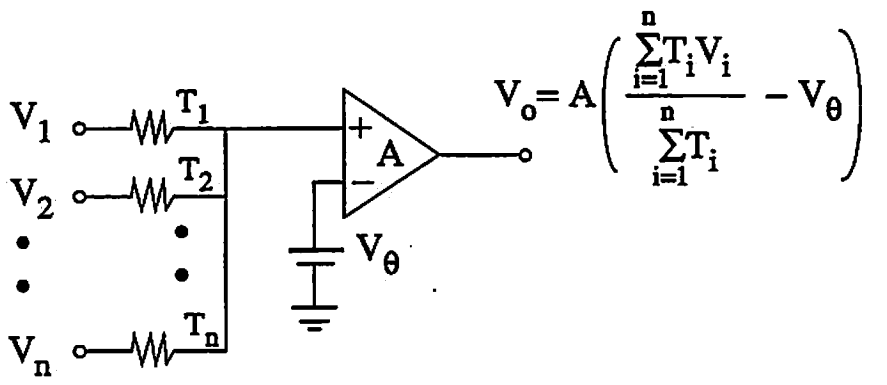


Fig. 1.4 A VLSI neuron with direct resistor implementation.

using resistors. The synapse weighting T_i is realized with a conventional resistor R_i. For a negative synapse weighting, a resistor and an inverted neuron output are used. Several VLSI chips [17,18] have been reported using this direct resistor realization. The amplifier output voltage is determined by

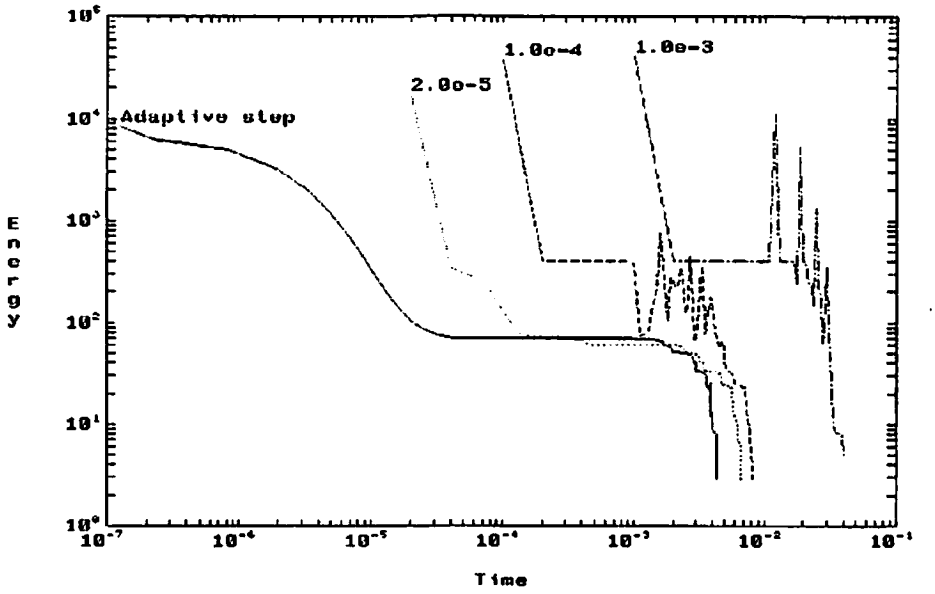
$$Y = f\left[\frac{\sum_{i=1}^{n} X_i/R_i}{1/R_{eq} + \sum_{i=1}^{n} 1/R_i}\right]. \tag{1.1}$$

Here, $f(.)$ is the transfer function of a neuron and R_{eq} is the input impedance of the neuron. On the other hand, the neuron output in software computation is determined by

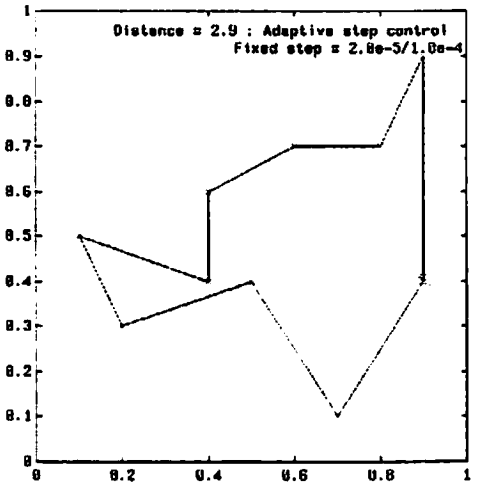
$$Y = f\left[\sum_{i=1}^{n} X_i/R_i\right]. \tag{1.2}$$

In the direct resistor realization method, large amplifier gain is needed due to the effect of the denominator in (1.1). In addition, the relative synapse values are more important than the absolute values in the VLSI implementation. In VLSI technologies, the relative tolerance in the device is much easier to be achieved than the absolute tolerance.

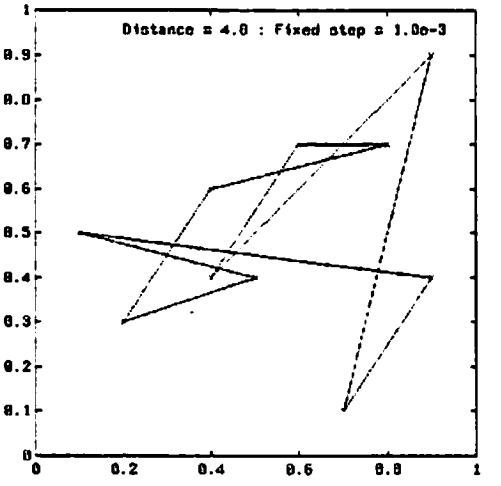
The chip area and number of package pins can be also the major limitations. The required chip area will be gradually decreased with the advances in the processing technologies and with the novel circuit design techniques. On the other hand, the number of package pins which is determined by the applications still remains the fundamental limitation. The physical package pins can share with several functional pins by multiplexing in the time domain. In this case,



(a)



(b)



(c)

Fig. 1.5 Network dynamic of a 10-city TSP.
(a) Transient behavior.
(b) Solution with adaptive time-step and with fixed time-step of 1.0×10^{-4} and 1.0×10^{-5}.
(c) Solution with fixed time-step of 1.0×10^{-3}.

the VLSI chip should operate in the synchronous mode. Notice that original neural networks were developed to operate asynchronously.

For many applications, the synchronous operation gives a significant difference from the asynchronous operation. One good example is to use a Hopfield neural network to solve a 10-city traveling salesman problem. Two versions of the forward-Euler integration methods were examined: the fixed time-step version and the adaptive time-step version with Richardson extrapolation [19]. The former version corresponds to synchronous operation and the latter version corresponds to asynchronous operation in VLSI neural circuits. For simplicity, the capacitances in Hopfield network [20] are set to be 1 and the neuron inputs are initially reset. Figure 1.5 shows the energy function change as a function of time. The CPU time for the fixed time-step being 1.0×10^{-4} is about 5 hours, while that for the adaptive time-step with 1.0×10^{-3} error tolerance is only 0.5 hour. Here, the software simulation is done in a SUN 3/60 workstation. With the fixed time-steps of 1.0×10^{-3} and 1.0×10^{-4}, the energy function is no longer continuously decreasing due to the numerical error from the synchronous operation. This result is exactly contradictory to the original Hopfield theory that the network always operates toward the direction of decreasing the energy function. This example shows that the asynchronous operation gives more advantages over the synchronous operation in terms of computational time and solution accuracy.

Several research projects address the issues of novel placement and routing of neuron and synapse cells [21]. One such activity is to design a silicon retina and cochlea [22]. MOS transistors in these chips operate in the subthreshold

region to achieve high voltage gain and low power dissipation. The VLSI chips perform the sensory functions with preliminary signal processing similar to human organs. These electronic neural networks can be characterized with high locality that one neuron is tied to just the neighboring neurons. The input signals are processed locally and gradually spread to the whole network in a decayed format. The high locality property is seldom explored in other neural networks.

One neural network designed for signal processing purpose is a VLSI chip from Bell Laboratories [17]. The chip has 256 neurons and 100,000 synapses pre-coded during the fabrication stage. The total chip size is 5700 μm x 5700 μm in 2.5-μm CMOS technology. The weightings of synapses, which are made from amorphous-Si, are defined by electron beams during wafer fabrication. The main application of this chip is known to compress the bandwidth of video image for telephone line transmission. Even though the synapse cell is very compact, the synapse weighting is fixed after chip fabrication. Another way to obtain the high resistivity material for synapse weightings in the standard CMOS technology is to use well resistors [18].

A limited programmability for a content-addressable memory can be obtained using switching MOS transistors [23]. The on/off-switch control data stored in the on-chip memory determine the fixed synapse weight. The synapses have only two states: inhibitory and excitatory. The chip has 54 neurons and 2916 synapses and occupies 6700 μm x 6700 μm area in a 2.5-μm CMOS technology. More synapse weighting selections can be made by switching certain ratioed current mirrors with the current-steering structure [24].

An analog multiplier with EEPROM cells is used to build a fully programmable synapse [25]. The synapse weighting is decided by the device aspect ratios and the bias current. Since the EEPROM-injected electronic charge at the floating gate can change threshold voltage of the transistor in the current mirror, bias currents of the analog multiplier are programmable. However, voltage range V_{IR} for a linear conductance is limited by the smaller dynamic range of the two differential pairs. For a proper input voltage range, the device W/L ratio should be very small. The similar circuit cell with DRAM cells is also developed [26]. Due to continuous charge leakage in the DRAM cell, periodic refreshing should be done. MOS transistors are used to store the synapse weighting information.

The digitally programmable synapse weighting can be made by controlling the RC charging time [27]. The synapse weighting data stored in the digital memory are converted into the switching pulse width. In this approach, the charge summation and neural decision are done in the analog fashion. The fully digitally operated architecture [28] includes on-chip random signal generator. The synapse weighting is coded into the pulse width and the summation function in neuron is realized with the AND logic. Since the mark time and occurrence on the pulse train is determined by a random number generator, such a sigmoid function in the neuron is performed by a stochastic process. In this architecture, the dynamic range of the synapse weighting is limited by the resolution of the pulse width so that very long processing time is necessary for the applications with a large ratioed synapse weighting.

In order to explore the rich properties of neural networks associated with massively parallel processing using analog neurons and synapses, compact and

programmable VLSI neural circuitries are needed.

References

[1] R. P. Lippman, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[2] R. Hecht-Nielsen, "Neural-computing: picking the human brain," *IEEE Spectrum*, vol. 25, no. 3, pp. 36-41, Mar. 1988.

[3] J. A. Anderson and E. Rosenfeld, *Neuralcomputing - Foundation of Research*, the MIT Press, 1988.

[4] S. Grossberg, *Neural Network and Natural Intelligence*, the MIT Press, 1988.

[5] McCulloch, S. Warren, Pitts, and H. Walter, "A logical calculus of the idea immanent in neural nets," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[6] F. Rosenblatt, *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*, Spartan Books, Washington, D. C., 1961.

[7] B. Widrow, Bernard, Hoff, and Marcian, "Adaptive switching circuits," *1960 IRE WESCON Convention Record*, Part 4, pp. 96-104, Aug. 23-26, 1960.

[8] Anderson, A. James, "A simple neural network generating an interactive memory," *Mathematical Biosciences*, vol. 14, pp. 197-220, 1972.

[9] J. J. Hopfield, "Neural network and physical systems with emergent collective computational abilities," *Proc. Natl. Acad., Sci. U.S.A.*, vol. 79, pp. 2554-2558, Apr. 1982.

[10] B. Kosko, "Adaptive bidirectional associative memories," *Applied Optics*, vol. 36, pp. 4947-4960, Dec. 1, 1987.

[11] D. Hebb, *Organization of behavior*, Science Editions, New York, 1961.

[12] T. Kohonen, *Self-organization and Associative memory*, Springer-Verlag, Berlin 1984.

[13] R. Hecht-Nielsen, "Counter-propagation networks," *IEEE First Inter. Conf. on Neural Networks*, vol. 2, pp. 19-32, 1987.

[14] A. H. Klopf, "A drive-reinforcement model of single neuron function: an alternative to the Hebbian neural model," *Proc. of Conf. on Neural Networks for Computing*, Snowbird, Utah, pp. 265-270, Apr. 1986.

[15] G. E. Hinton and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cognitive Science*, vol. 9, pp. 147-169, 1985.

[16] Y. P. Tsividis, "Analog MOS integrated circuits - certain new ideas, trends, and obstacles," *IEEE Jour. Solid-State Circuits*, vol. SC-22, no. 3, pp. 317-321, June 1987.

[17] H. P. Graf and P. deVegvar, "A CMOS implementation of a neural network model", *Proceedings of the 1987 Stanford Conference*, pp. 351-362, the MIT Press.

[18] B. W. Lee and B. J. Sheu, "Design of a neural-based A/D converter using modified Hopfield network," *IEEE Jour. of Solid-State Circuits*, vol. SC-24, no. 4, pp. 1129-1135, Aug. 1989.

[19] Dahlquist, Björck, and Anderson, *Numerical Methods*, pp. 269-273, Prentice-Hall Inc., 1974.

[20] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad., Sci. U.S.A.*, vol. 81, pp. 3088 -3092, May 1984.

[21] P. Treleaven, M. Pacheco, and M. Vellasco, "VLSI architectures for neural networks," *IEEE Micro*, vol. 9, no. 6, pp. 8-27, Dec. 1989.

[22] C. A. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley Publishing Co. Inc, 1989.

[23] R. E. Howard, D. B. Schwartz, J. S. Denker, R. W. Epworth, H. P. Graf, W. E. Hubbard, L. D. Jackel, B. L. Straughn, and D. M. Tennant, "An associative memory based on an electronic neural network architecture," *IEEE Trans. on Electron Devices*, vol. ED-34, no. 7, pp. 1553-1556, July 1987.

[24] P. Mueller, J. V. D. Spiegel, D. Blackman, T. Chiu, T. Clare, C. Donham, T. P. Hsieh, M. Loinaz, "Design and fabrication of VLSI components for a general purpose analog neural computer," *Analog VLSI Implementation of Neural Systems*, Kluwer Acad. Pub., pp. 135-169, 1989.

[25] M. Holler, S. Tam, H. Castro, R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'Float gate' synapses," *Inter. Joint Conf. on Neural Networks*, vol. 2, pp. 191-196, June 1989.

[26] T. Morishita, Y. Tamura, and T. Otsuki, "A BiCMOS analog neural network with dynamically updated weights," *IEEE Inter. Solid-State Circuits Conf.* pp. 142-143, Feb. 1990.

[27] A. F. Murray, "Pulse arithmetic in VLSI neural network," *IEEE Micro*, vol. 9, no. 6, pp. 64-74, Dec. 1989.

[28] D. E. Van den Bout and T. K. Miller III, "A digital architecture employing stochasticism for the simulation of Hopfield neural nets," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 5, pp. 732-746, May 1989.

[29] Professor I. Reed, University of Southern California, *Private communication*, 1987.

Chapter 2

VLSI Hopfield Networks

The Hopfield networks [1] are very popular for electronic neural computing due to the simplicity in the network architecture and the fast convergence property. A Hopfield network composed of one-layer neurons and fully connected feedback synapses can be used to realize associative memory, pattern classifier, and optimization circuits. The network always operates along a decreasing path for the energy function, so that the final output represents one minimum of the energy function. Due to the complexity of the energy function, there could exist several local minima. The minima in the energy function of Hopfield networks, which are decoded in the resistive network, are used for the exemplar patterns in associative memory and pattern classifier applications. However, the existence of local minima is not desirable for a great variety of other optimization applications.

Another major reason why Hopfield neural networks are widely used for VLSI implementation is the relaxed requirement on the network elements. The key requirement on neuron transfer function for the Hopfield network is to achieve monotonic functions, while that for other neural networks is usually an exact mathematical expression like sigmoid, Boltzmann, or Gaussian function. Since the amplifier gain in VLSI neutral circuits can not follow an exact function, the simpler neuron transfer function requirement makes Hopfield networks a popular choice for VLSI implementation. The rich local minima can help to operate without a severe performance degradation in the situation that the device

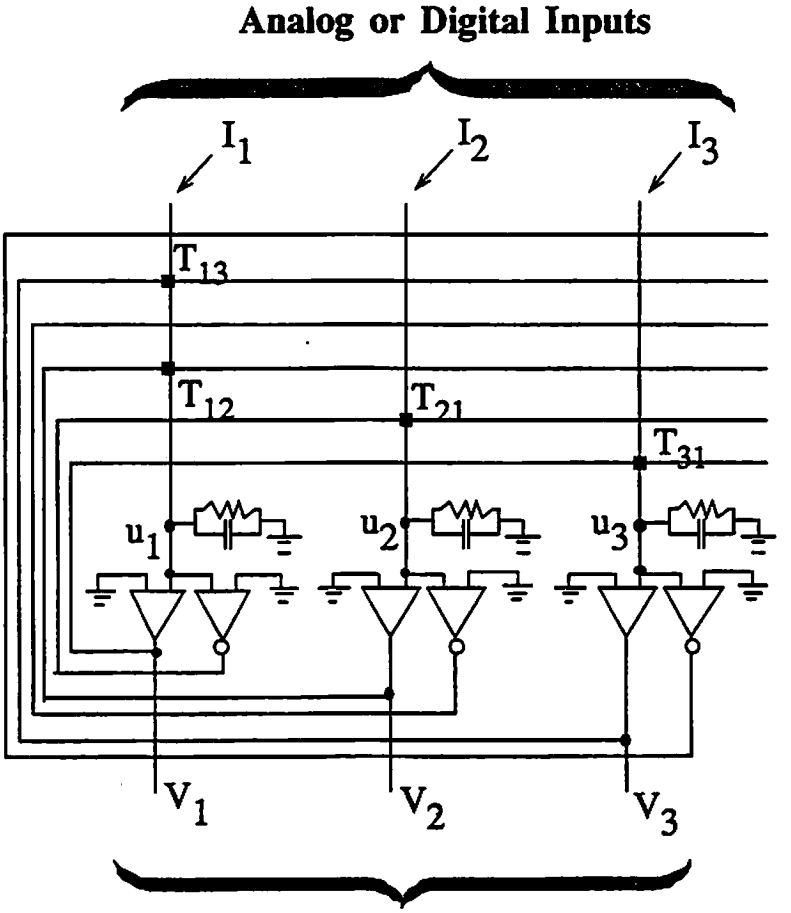


Fig. 2.1 A Hopfield neural network with inverting and non-inverting amplifiers as neurons.

variations are large.

Hopfield networks can be used as an effective interface between the real-world of analog transmission media and the digital computing machines. Inputs to the Hopfield network can be analog signals. Outputs are usually discrete values as shown in Fig. 2.1. The Hopfield network not only converts analog signals into the digital format but also can conduct the first-level signal processing, such as associative recalling, signal estimation, and combinatorial optimization in a way similar to the human retina. Due to the robustness of neural processors, output of the Hopfield neural network needs not to be with high precision, which is in strong contrast to conventional interface circuits. Key advantages of the VLSI neural interfaces over conventional interfaces are derived from the learning capabilities of Hopfield networks [2]. By adjusting the conductance values between the amplifiers with a learning rule, the adaptive characteristics can be made. The adaptability of a neural-based A/D converter [3], for example, will be useful not only to compensate for initial device mismatches or long-term characteristic drifts but also to provide a greater processing capability in the image and signal processing system.

2.1. Circuit Dynamics of Hopfield Networks

In a Hopfield network where the resistive network T_{ij} is symmetric and without self-feedback terms, i.e. $T_{ij} = T_{ji}$ and $T_{ii} = 0$, the energy function E can be expressed as

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,\ j\neq i}^{n} T_{ij}V_iV_j - \sum_{i=1}^{n} I_iV_i + \sum_{i=1}^{n} T_i \int_0^{V_i} g_i^{-1}(V)\,dV \ . \qquad (2.1)$$

Here, T_{ij} is the conductance between the i-th amplifier input and the j-th amplifier output, V_i is the i-th amplifier output voltage, I_i is the input current signal to the i-th amplifier, $g(\ .\ )$ is the amplifier transfer function, and n is the number of neurons. By using the energy function, the transient behavior of the Hopfield network can be explained in the following way. Input voltage of the i-th amplifier u_i is governed by Kirchoff's Current Law,

$$C_i \frac{du_i}{dt} = \sum_{j=1,\ j\neq i}^{n} T_{ij} V_j - T_i u_i + I_i \quad . \tag{2.2}$$

Here, C_i and T_i are the capacitance and the equivalent conductance at the i-th amplifier input node, respectively. Total input conductance for each amplifier, T_i, is given as

$$T_i = \sum_{j=1,\ j\neq i}^{n} |T_{ij}| \ . \tag{2.3}$$

In the VLSI circuit realization of a Hopfield network, simple decision-making amplifiers and a resistive network are used. Each neuron includes non-inverting and inverting amplifiers to handle both positive and negative synapse weightings. The output levels of a non-inverting amplifier are 0 V and 1 V, while that of an inverting amplifier are 0 V and -1 V. The amplifier outputs are fed back to the amplifier inputs through the densely connected resistive network. Capacitance C_i is the total capacitance at the input node of the i-th amplifier. The RC charging effect from the T_i and C_i in the original Hopfield network is equivalently obtained with the finite bandwidth of the amplifier in the VLSI implementation.

The time derivative of the energy function can be determined as

$$\frac{dE}{dt} = -\sum_{i=1}^{n} \frac{dV_i}{dt}\left[\sum_{j=1,\, j\neq i}^{n} T_{ij}V_j - T_i u_i + I_i\right]. \tag{2.4}$$

Substituting (2.2) into (2.4), we obtain

$$\begin{aligned}\frac{dE}{dt} &= -\sum_{i=1}^{n} \frac{dV_i}{dt} C_i \frac{du_i}{dt} \\ &= -\sum_{i=1}^{n} C_i g_i^{-1}(V_i)\left[\frac{dV_i}{dt}\right]^2. \end{aligned} \tag{2.5}$$

If $g_i(\,.\,)$ is monotonically increasing, $\frac{dE}{dt}$ is always negative. That is to say, the network moves in the direction of decreasing the energy function. When $\frac{dV_i}{dt} = 0$ for all i, the steady state is reached. Due to the nature of the energy function, the network output is highly dependent upon its initial state. Thus, the final output is usually the closest local minimum from the initial state of the network.

The energy function can be used to describe the macroscopic property of network dynamics. However, it is insufficient to explain such detailed behaviors of VLSI Hopfield circuits as the location of local minima, convergence speed, and the required amplifier gain for proper network operation.

2.2 Existence of Local Minima

Detailed properties of the local minima can be understood with the help of following analysis method [4]. At a stable output, the input voltage to each

neuron is governed by

$$\begin{cases} u_i > 0 & \text{when } V_i = 1V \\ u_i < 0 & \text{when } V_i = 0V \ . \end{cases} \tag{2.6}$$

Here, the input voltage of the i-th neuron, u_i, is determined by Kirchoff's Laws,

$$C_i \frac{du_i}{dt} + T_i u_i = \sum_{j \neq i,\ j=1}^{n} T_{ij} V_j + I_i \ . \tag{2.7}$$

From (2.6), the range of input current I_i for the corresponding stable state can be calculated,

$$\begin{cases} I_i > - \sum_{j \neq i, j=1}^{n} T_{ij} V_j & \text{for } V_i = 1V \\ I_i < - \sum_{j \neq i, j=1}^{n} T_{ij} V_j & \text{for } V_i = 0V \ . \end{cases} \tag{2.8}$$

During the transient period, the voltage V_i is changing in the direction that the energy function E decreases. When all amplifiers reach the stable condition in (2.6), the searching process is terminated. From (2.8), a characteristic parameter GAP can be defined as the input current difference of the lower limit for $V_i = 1V$ and the upper limit for $V_i = 0V$,

$$GAP_i \equiv - \sum_{j \neq i,\ j=1}^{n} T_{ij} (V_j^u - V_j^l) \ , \tag{2.9}$$

where V_j^u and V_j^l are the output voltage of the digital codes to the j-th amplifier output whose i-th bit from the least significant bit (LSB) are logic 1 and logic 0, respectively. Here, the unit of GAP is ampere. Since the parameter GAP_i represents the overlapped range of the i-th input current, two digital codes can

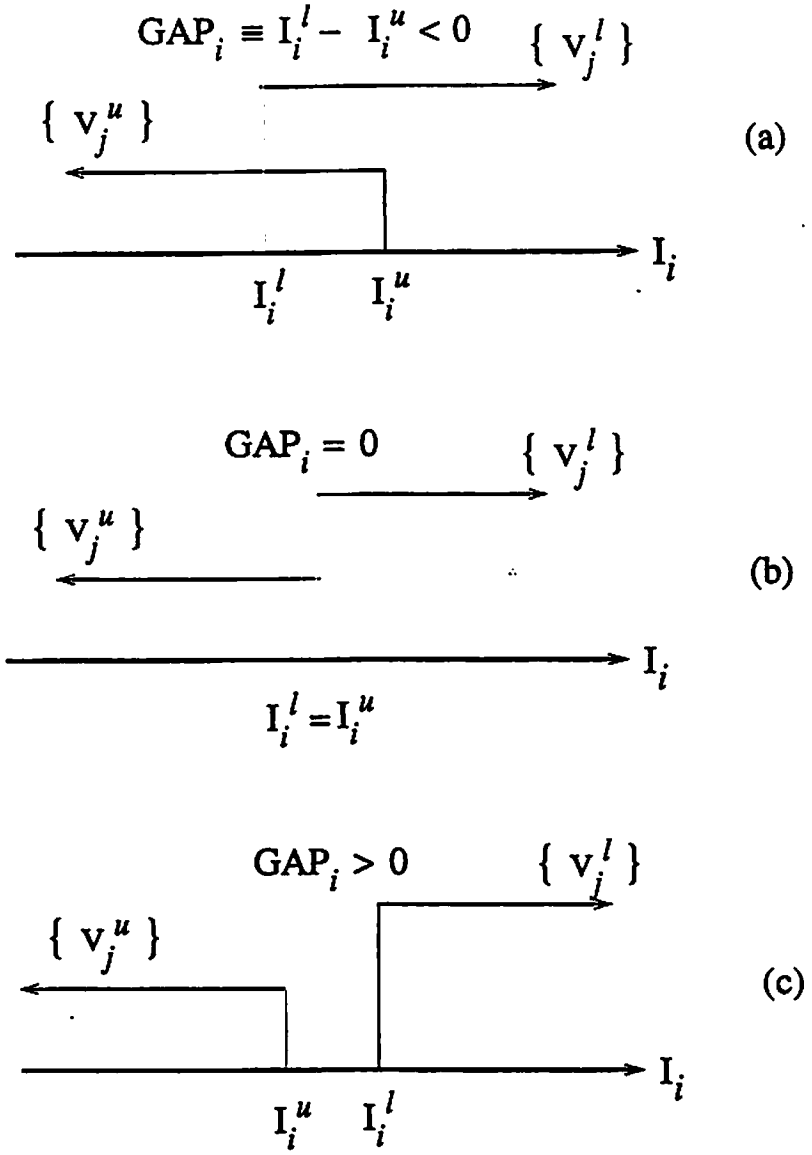


Fig. 2.2 Calculation of a characteristic parameter GAP.

(a) $GAP_i < 0$.

(b) $GAP_i = 0$.

(c) $GAP_i > 0$.

be stable for a given input current when $GAP_i < 0$ for every i as shown in Fig. 2.2. Therefore, the condition when the two digital codes can not be local minima in the energy function is simultaneously $GAP_i \geq 0$ for every i.

Let us examine the special case that $\{ V_i^u \}$ and $\{ V_i^l \}$ are adjacent digital codes. The parameter GAP_k can be simplified as

$$GAP_k = \begin{cases} \sum_{j=1}^{k-1} T_{kj} & \text{if } k > 1 , \\ 0 & \text{if } k = 1 . \end{cases} \tag{2.10}$$

Here, $\{ V_i^u \}$ and $\{ V_i^l \}$ are

$$\begin{cases} V_i^u = V_i^l & \text{if } i > k \\ V_i^u = 1V \text{ and } V_i^l = 0V & \text{if } i = k \\ V_i^u = 0V \text{ and } V_i^l = 1V & \text{if } i < k . \end{cases} \tag{2.11}$$

For a set of adjacent digital codes with the only different bit occurring in the LSB, the codes can not be minima at the same input. Even though $\sum_{j=1}^{k-1} T_{ij} \geq 0$ is just a necessary condition that both the adjacent codes can not be stable at a given input, it can be used as an indicator for the existence of local minima. For example, all feedback synapse weightings of the Hopfield neural-based A/D converter [3] are negative, so that the necessary condition can not be satisfied.

In a general case when two digital codes have different values at the k-th bit but have the same value above the k-th bit, the parameter GAP_i can be expressed as follows,

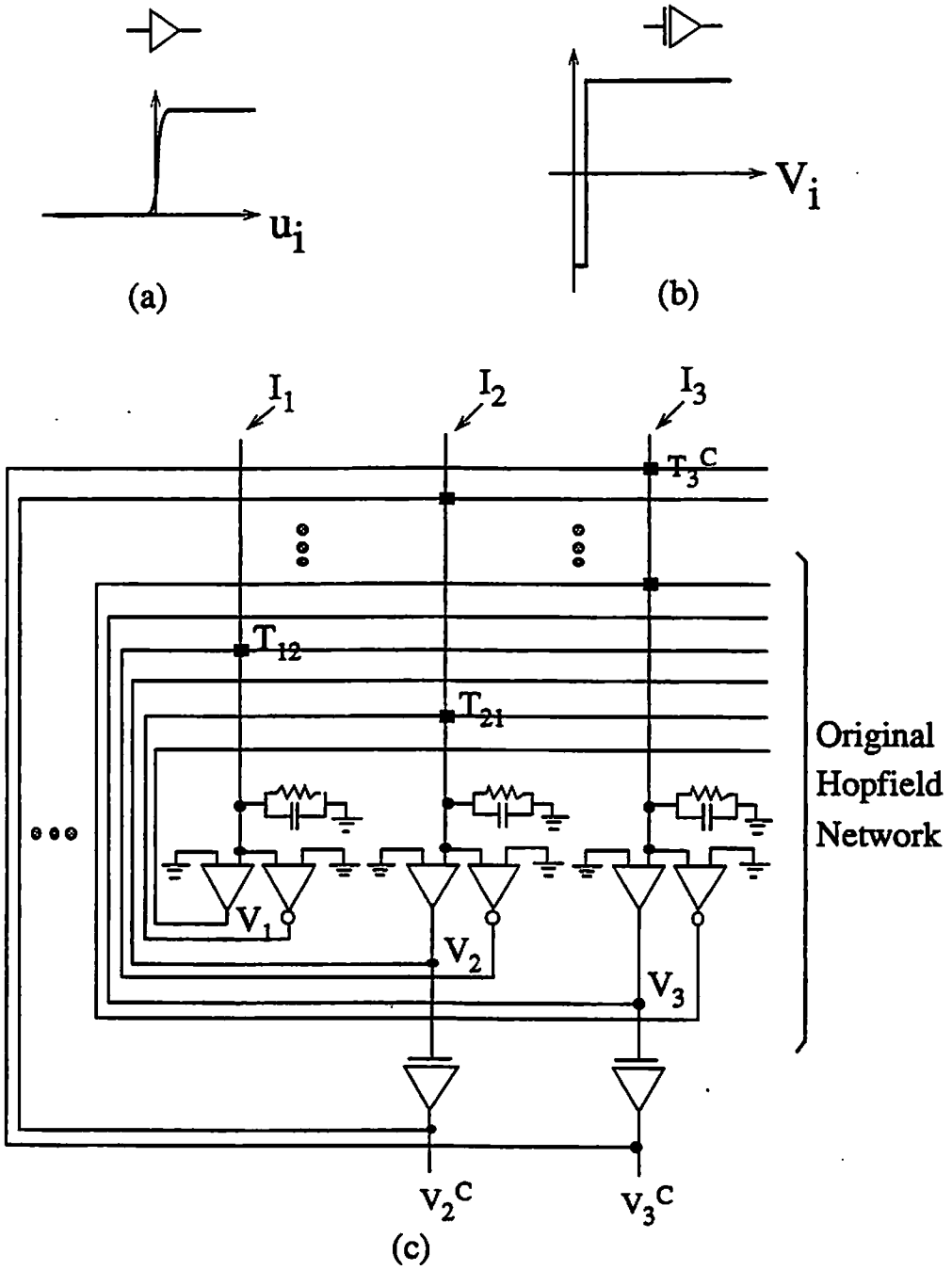


Fig. 2.3 Modified Hopfield neural network without local minima.
(a) Transfer characteristics of the Hopfield amplifier.
(b) Transfer characteristics of the correction amplifier.
(c) Modified Hopfield neural network.

$$GAP_i = \begin{cases} \text{undefined} & \text{if } i > k \ , \\ \sum_{j=1}^{k-1} T_{kj}(V_j^u - V_j^l) & \text{if } i = k \ , \\ (V_i^l - V_i^u)\left[T_{ki} + \sum_{j\neq i,\ j=1}^{k-1} T_{ij}(V_j^u - V_j^l)\right] & \text{if } i < k \ . \end{cases} \tag{2.12}$$

Here, the digital codes are given as

$$\begin{cases} V_j^u = 1V \text{ and } V_j^l = 0V & \text{for } j = k \\ V_j^u = V_j^l & \text{for } j > k \ . \end{cases} \tag{2.13}$$

The condition that two digital states can be stable at an input current is

$$GAP_i < 0 \qquad \text{for every } i \leq k \ . \tag{2.14}$$

2.3. Elimination of Local Minima

The overlapped input range between two digital codes can be eliminated by adding correction current $\{I_{iC}\}$ into the neuron inputs as shown in Fig. 2.3. The correction logic gates monitor the Hopfield network outputs and generates the correcting information. The correction currents are fed back into the neuron inputs through the extended conductance network. Here, the response time of the correction circuits is smaller than that for the rest of the network. The input voltage u_i with the correcting current is governed by

$$T_i u_i = \sum_{j\neq i,\ j=1}^{n} T_{ij} V_j + I_i + I_{iC} \ . \tag{2.15}$$

Therefore, the parameter GAP_i in (2.12) is changed to GAP_i^C as follows, If the

$$GAP_i^C = \begin{cases} \sum_{j=1}^{k-1} T_{kj}(V_j^u - V_j^l) - I_{kC}^u + I_{kC}^l & \text{for } i = k \\ (V_i^l - V_i^u)\left[\sum_{j=1}^{k-1} T_{ij}(V_j^u - V_j^l) + T_{ki} + I_{iC}^u - I_{iC}^l\right] + T_{ii} & \text{for } i < k \ . \end{cases} \tag{2.16}$$

The new energy function E_C is given as

$$\begin{aligned} E_C &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j\neq i,\ j=1}^{n} T_{ij}V_iV_j - \sum_{i=1}^{n}(I_i + I_{iC})V_i \\ &= E_O - \sum_{i=1}^{n} I_{iC}V_i . \end{aligned} \tag{2.17}$$

Here, E_C and E_O are the modified and original energy functions, respectively. The additional term in the modified energy function is introduced to fill up the wells of local minima, so that only the global minimum exists.

The general criteria for choosing the correction current to eliminate local minima in Hopfield networks can be described below.

Criterion-A: Avoid the condition that $GAP_i^C < 0$ for every $i < k$, which makes both the digital codes in (2.13) stable.

Criterion-B: Retain the same network dynamics of Hopfield networks so that the modified network moves in the direction of decreasing the energy function.

Criterion-C: Preserve the input current range for a global minimum.

The criterion-B requires that

$$\sum_{i=1}^{n} V_i \frac{\partial I_{iC}}{\partial V_i} = 0 \ . \tag{2.18}$$

Notice that the criterion-C is often difficult to satisfy in some optimization problems with no *a priori* knowledge of the solutions. For example, the solution of a Hopfield neural-based A/D converter at a given analog input voltage is well-defined, while that of a traveling salesman problem is not easy to know. The choices of the correction current for the two distinct cases will be described later in detail.

From criterion-A, $GAP_k^C = 0$ is chosen. Notice that the requirement in (2.14) is logic-AND condition for every $i < k$. By setting $I_{kC}^u = -I_{kC}^l$, the correction currents at the steady state are given as

$$I_{kC}^u = -I_{kC}^l = T_{kC} \ , \tag{2.19}$$

$$\text{where} \quad T_{kC} = \frac{1}{2}\sum_{j=1}^{k-1} T_{kj}(V_j^u - V_j^l). \tag{2.20}$$

The correcting currents during the searching process can be expressed as

$$\begin{aligned} I_{kC} &\equiv I_{kC}^l - I_{kC}^u \\ &= T_{kC} f^u(V_k) + T_{kC} f^l(V_k) \ . \end{aligned} \tag{2.21}$$

Here, the boundary conditions of $f^u(V_k)$ and $f^l(V_k)$ are determined from the steady-state condition in (2.13) as follows,

$$\begin{cases} f^u(0) = 0, \ f^u(1) = 1 \ ; \text{ and} \\ f^l(0) = 1, \ f^l(1) = 0 \ . \end{cases} \tag{2.22}$$

Functions $f^u(V_k)$ and $f^l(V_k)$ can be determined by the criterion-B,

$$V_k \frac{\partial I_{kC}}{\partial V_k} = T_{iC} V_k \left[\frac{\partial f^u(V_k)}{\partial V_k} - \frac{\partial f^l(V_k)}{\partial V_k} \right] = 0 \ . \tag{2.23}$$

functions are selected as

$$f^u(V) = U(V + \varepsilon) \tag{2.24}$$

and

$$f^l(V) = U(\varepsilon) - U(V + \varepsilon) \tag{2.25}$$

then, Eq. (2.23) becomes

$$V_k \frac{\partial I_{kC}}{\partial V_k} = T_{kC} 2V_k \delta(V_k + \varepsilon) \simeq 0 \ . \tag{2.26}$$

Here, $U(V)$ is a step function, $\delta(V)$ is a delta function, and ε is a very small positive number. Notice that the above selection is made by decreasing the input overlapped range by half from each side of the two digital codes.

In the following sections, the self-correction techniques are applied to eliminate the local minima for the neural-based A/D conversion and traveling salesman problems. The analog-to-digital conversion is a simple and straight-forward example to demonstrate the capabilities of the modified Hopfield networks because the optimal solution is clearly known. The traveling salesman problem is with the intrinsically unknown global minimum. Our modified Hopfield networks find excellent solutions to both practical problems.

2.4. Neural-Based A/D Converter without Local Minima

The function of an A/D converter is to find a digital word $\{ V_n V_{n-1} \cdots V_1 \}$ which is a best representation of the analog input signal V_S. The output of an A/D converter is to achieve the minimum of the following function

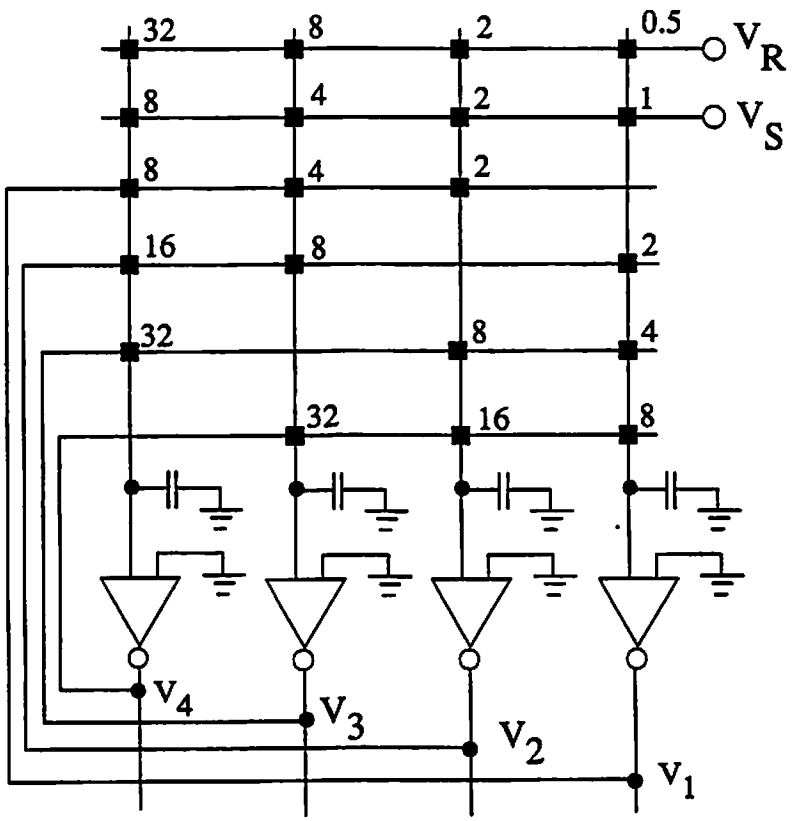


Fig. 2.4 A 4-bit neural-based A/D converter.

$$E_o = (V_S - \sum_{i=1}^{n} V_i 2^{i-1})^2 \ . \tag{2.27}$$

In order to construct a Hopfield-like energy function, some modifications have to be made,

$$E = \frac{1}{2}E_o - \frac{1}{2}\sum_{i=1}^{n}(2^{i-1})^2[V_i(V_i - 1)] \ . \tag{2.28}$$

Notice that the additional term, which eliminate the diagonal elements in the Hopfield network, does not affect the final correct solution [3]. By expanding (2.28), the objective function for the A/D converter can be expressed as

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,\ j\neq i}^{n}(-2^{i+j-2})V_iV_j - \sum_{i=1}^{n}(-2^{2i-3} + 2^{i-1}V_S)V_i \ . \tag{2.29}$$

The synapse weightings T_{ij} and input current I_i can be determined from (2.1) and (2.29),

$$T_{ij} = -2^{i+j-2} \tag{2.30}$$

and

$$I_i = -2^{2i-3} + 2^{i-1}V_S \ . \tag{2.31}$$

The circuit schematic of a 4-bit A/D converter is shown in Fig. 2.4. The input current I_i is also implemented using the conductances (T_{iR} and T_{iS}) and the reference voltage $V_R = -1V$. Since all actual conductance values are positive in the VLSI implementation, the polarity for output voltage V_i is changed to [−1V, 0V] by using inverting amplifiers. Thus, the values for the conductances in the A/D converter are

$$G_{ij} = 2^{i+j-2} , \tag{2.32}$$

$$G_{iR} = 2^{2i-3}, \tag{2.33}$$

and

$$G_{iS} = 2^{i-1}. \tag{2.34}$$

Here, G_{iR} is the conductance between the input terminal of the i-th amplifier and the reference voltage V_R, and G_{iS} is the conductance between the input terminal of the i-th amplifier and the analog input voltage V_S. Notice that T and G are used interchangeably to represent the conductances of the network elements.

The amplifier input voltage $\{u_i\}$ is governed by

$$\left[\sum_{j=1,\ j\neq i}^{n} G_{ij} + G_{iR} + G_{iS} \right] u_i = \sum_{j=1,\ j\neq i}^{n} G_{ij} V_j + G_{iS} V_S + G_{iR} V_R \ . \tag{2.35}$$

Since the input voltage $\{u_i\}$ is determined by the ratios of conductances, the scaling factor to realize absolute conductance values can be used as an integrated-circuit design parameter. The normalized conductances are

$$\hat{G}_{iS} = 1 , \tag{2.36}$$

$$\hat{G}_{ij} = 2^{j-1} , \tag{2.37}$$

and

$$\hat{G}_{iR} = 2^{i-2} . \tag{2.38}$$

The maximum conductance ratio for a n-bit A/D converter has been greatly reduced from 2^{2n-2} to 2^n. If the voltage level of the amplifier is scaled to (

0V, $-V_A$), the reference voltage is scaled to $-V_{ref}$, and the conversion step size is scaled to V_{step}, then the conductances are scaled to $\hat{G}_{ij}/V_A$, $\hat{G}_{iR}/V_{ref}$, and $\hat{G}_{iS}/V_{step}$, respectively.

For a specific digital output, the analog input signal V_S to the i-th amplifier can be expressed as

$$V_S \geq -\frac{\hat{G}_{iR}}{\hat{G}_{iS}}V_R - \sum_{j=1,\ j\neq i}^{n}\frac{\hat{G}_{ij}}{\hat{G}_{iS}}V_j \quad \text{when } V_i = -1\ V \tag{2.39}$$

and

$$V_S < -\frac{\hat{G}_{iR}}{\hat{G}_{iS}}V_R - \sum_{j=1,\ j\neq i}^{n}\frac{\hat{G}_{ij}}{\hat{G}_{iS}}V_j \quad \text{when } V_i = 0\ V\ . \tag{2.40}$$

Since

$$-\frac{\hat{G}_{iR}}{\hat{G}_{iS}}V_R - \sum_{j=1,\ j\neq i}^{n}\frac{\hat{G}_{ij}}{\hat{G}_{iS}}V_j = 2^{i-2} + V_O - 2^{i-1}|V_i|\ , \tag{2.41}$$

the upper limit and lower limit of the analog input voltage always increases with i. Here, $V_R = -1\ V$ and V_O is the digital output voltage which equals to $\sum_{i=1}^{n}2^{i-1}|V_i|$. To achieve a stable output, the input signal range is governed by the logic-AND operation of the range decided by each amplifier. Therefore, the lower limit of V_S at a given digital output is determined by the first logic-1 occurrence of the digital word from the least significant bit (LSB) and the upper limit is determined by the first logic-0 occurrence from the LSB as shown in Table 2.1. If a digital word has the first logic-0 at the i-th bit from the LSB, then the next adjacent digital word has the first logic-1 at the i-th bit. Thus, the

Table 2.1 Analog input range for each digital code

DIGITAL CODE				ANALOG INPUT RANGE			
D_4	D_3	D_2	D_1	V_1	V_2	V_3	V_4
0	0	0	0	**< 0.5**	< 1.0	< 2.0	< 4.0
0	0	0	1	**> 0.5**	**< 2.0**	< 3.0	< 5.0
0	0	1	0	**< 2.5**	**> 1.0**	< 4.0	< 6.0
0	0	1	1	**> 2.5**	> 2.0	**< 5.0**	< 7.0
0	1	0	0	**< 4.5**	< 5.0	**> 2.0**	< 8.0
0	1	0	1	**> 4.5**	**< 6.0**	> 3.0	< 9.0
0	1	1	0	**< 6.5**	**> 5.0**	> 4.0	< 10.0
0	1	1	1	**> 6.5**	> 6.0	> 5.0	**< 11.0**
1	0	0	0	**< 8.5**	< 9.0	< 10.0	**> 4.0**
1	0	0	1	**> 8.5**	**< 10.0**	< 11.0	> 5.0
1	0	1	0	**< 10.5**	**> 9.0**	< 12.0	> 6.0
1	0	1	1	**> 10.5**	> 10.0	**< 13.0**	> 7.0
1	1	0	0	**< 12.5**	< 13.0	**> 10.0**	> 8.0
1	1	0	1	**> 12.5**	**< 14.0**	> 11.0	> 9.0
1	1	1	0	**< 14.5**	**> 13.0**	> 12.0	> 10.0
1	1	1	1	**> 14.5**	> 14.0	> 13.0	> 11.0

i-th amplifier decides the upper limit and lower limit of the analog input voltage between two adjacent digital words. That is to say, if the upper limit of the input signal corresponding to a digital word is decided by the i-th amplifier, the lower limit of the upper adjacent digital word is also decided by the i-th amplifier.

To guarantee that only one global minimum corresponding to each analog input value (i.e. without local minima) exists, only one-to-one correspondence between the digital output and analog input should exist. The characteristic parameter *GAP* can be used to understand the existence of local minima. It can be expressed as

$$GAP = -\sum_{j=1}^{k-1} 2^{j-1}(V_j^l - V_j^u) \tag{2.42}$$

where $\{V_j^l\}$ and $\{V_j^u\}$ are the adjacent digital words in (2.11). Since input currents for the neural-based A/D converter are determined by the analog input voltage, only one characteristic parameter exists. If the *GAP* is negative, both the adjacent digital words can be the stable digital output at a given analog input voltage. On the other hand, if the *GAP* is positive, there can be a gap of analog input voltage where no stable digital output exists. For a proper A/D conversion, the *GAP* should be zero.

The characteristic parameter *GAP* becomes

$$GAP = -2^{k-1} + 1 \ . \tag{2.43}$$

The above equation shows that the *GAP* is always negative except when $k = 1$. Hence, there could exist more than two digital output words corresponding to a

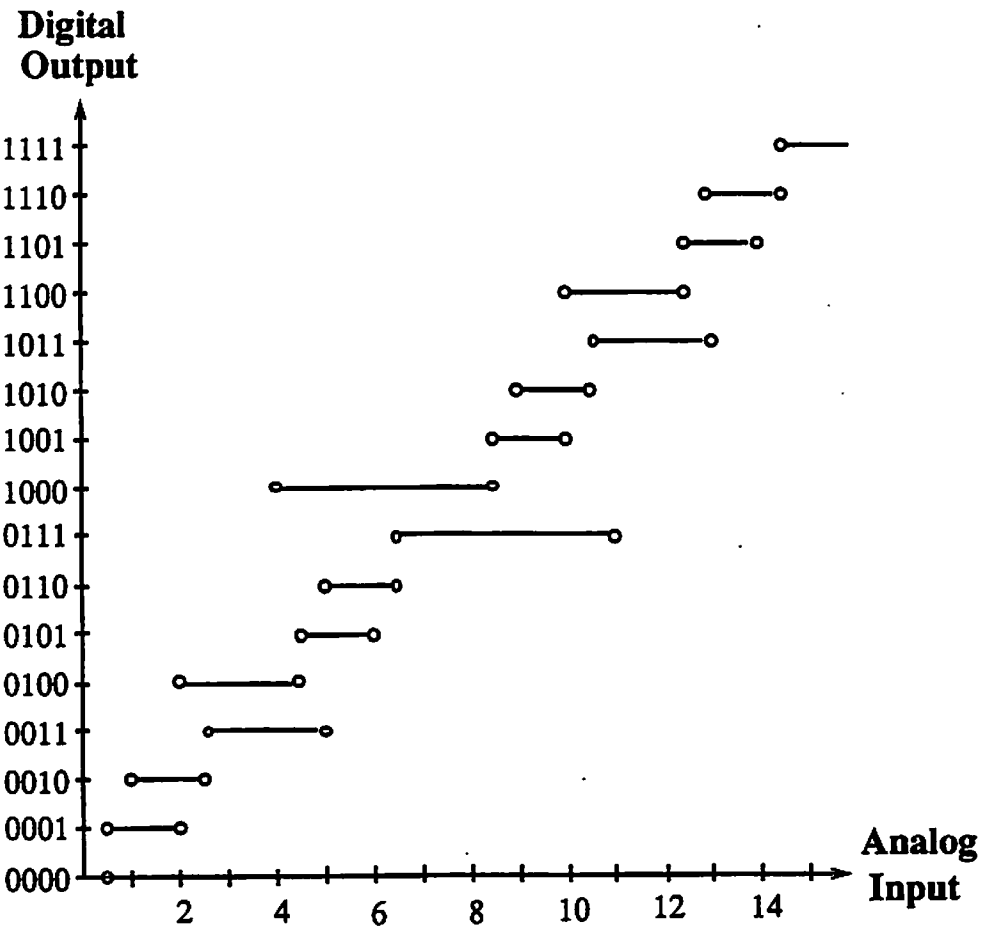


Fig. 2.5 Digital output versus analog input characteristics of the A/D converter shown in Fig. 2.4.

given analog input. In addition, the overlapped length is not the same for every output word in the Hopfield A/D converter. Figure 2.5 shows the input signal range corresponding to each digital word. The overlapped digital words for a given analog input value are caused by the local minima in the energy function. The largest overlap in the analog input voltage occurs at the word (0111) and (1000) where the 4-th amplifier decides the analog input range. This phenomenon is apparent because the characteristic parameter *GAP* increases with k. However, there is no overlapped input range between the adjacent digital words if these two words differ by the LSB.

The overlapped regions in Fig. 2.5 indicate the existence of local minima in the Hopfield neural-based A/D converter proposed by Tank and Hopfield [3]. For example, the words (1000), (0110), and (0101) can be the stable outputs when $V_S = 5.6\ V$. The converged output in VLSI chips is determined by the initial state of the network and device mismatches. The local minima give severe nonideal output characteristics. While nonlinearities can be remedied in software computation to some extent by resetting the network to the ground state prior to each conversion cycle, the circuit transfer characteristic in VLSI chips still has strong nonlinearity due to amplifier mismatches.

2.4.1. The Step Function Approach

The analog input voltage range of a digital code are determined as

$$\begin{cases} V_S \geq V_o - 2^{p-2} \\ V_S \leq V_o + 2^{q-2} , \end{cases} \tag{2.44}$$

where

$$V_o = \sum_{i=1}^{n} 2^{i-1} V_i \ . \tag{2.45}$$

Here, p and q are the first logic-1 bit and logic-0 bit from the LSB, respectively. After adding the correction current terms, the analog input voltage range and the characteristic parameter of the adjacent digital codes in (2.11) are changed as

$$\begin{cases} V_S^C \geq V_o^l - 2^{k-2} - \dfrac{T_{kC}}{2^{k-1}} f^l(V_k^l) \\ V_S^C \leq V_o^u + 2^{k-2} - \dfrac{T_{kC}}{2^{k-1}} f^u(V_k^u) \end{cases} \tag{2.46}$$

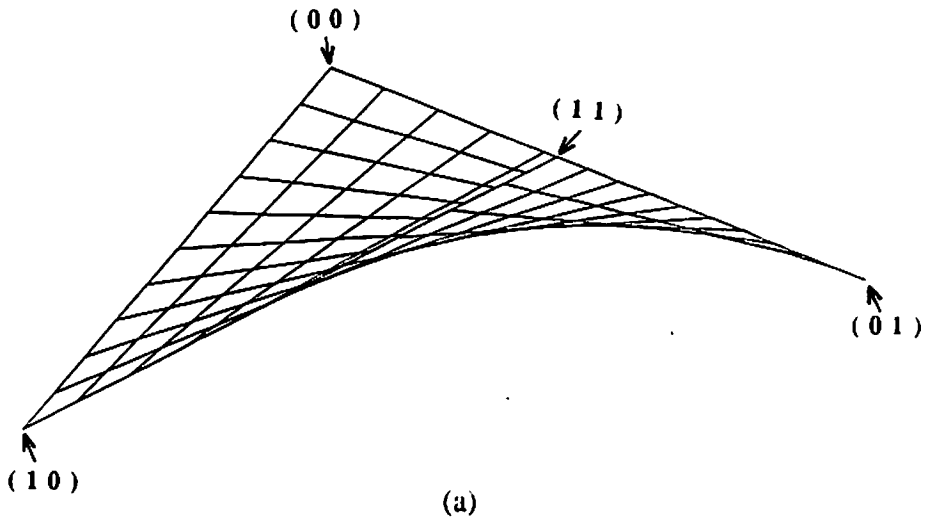
and

$$GAP^C = -2^{k-1} - 1 - \frac{T_{kC}}{2^{k-1}} \left[f^l(V_k^l) - f^u(V_k^u) \right] . \tag{2.47}$$

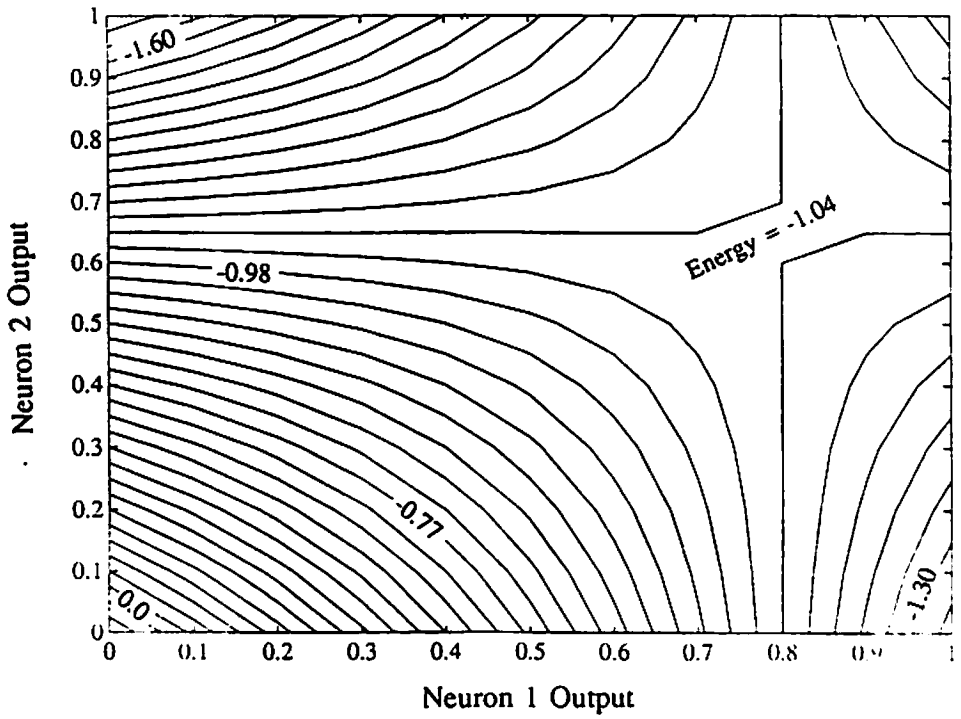
Here, V_S^C and GAP^C are the corrected input voltage and characteristic parameter. Since analog input ranges of a neural-based A/D converter without local minima are given as

$$V_o - 0.5 < V_S < V_o + 0.5 \ , \tag{2.48}$$

the analog input range should be corrected to be at the mid-point of the overlapped range. Notice that the choice of $\{T_{iC}\}$ for the neural-based A/D converter can keep the analog input voltage range of global minimum, because the voltage range can be well-defined at a given analog input. Therefore, the correction current for the codes is

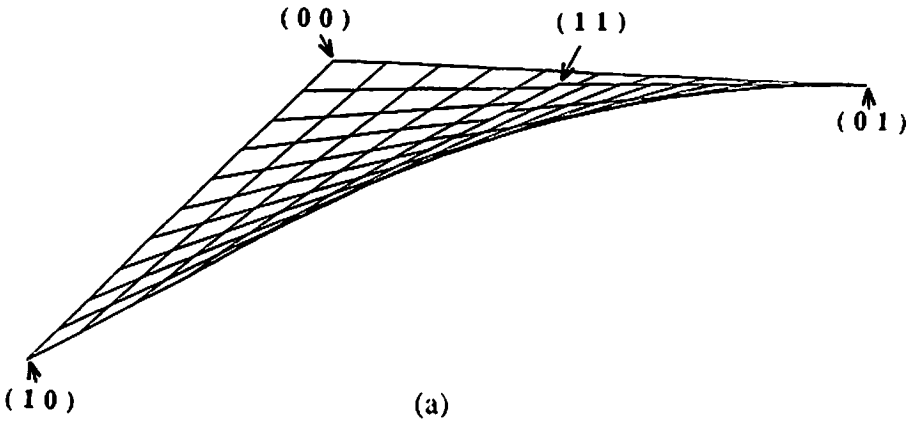


(a)

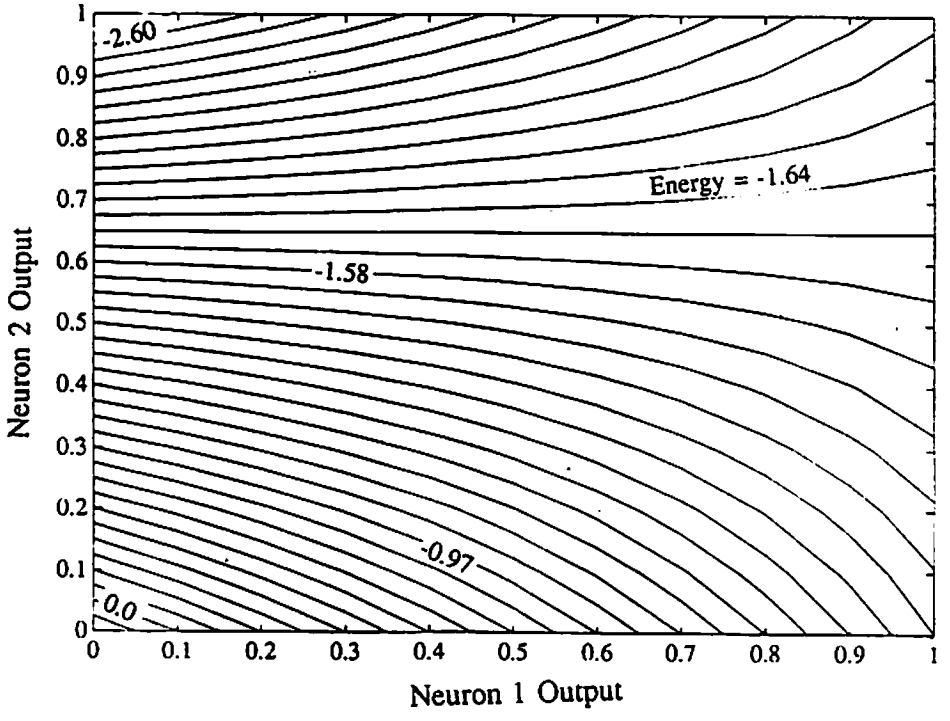


(b)

Fig. 2.6 Energy function of the original Hopfield A/D converter.
(a) Energy surface with changing V_1 and V_2.
(b) The corresponding contour.



(a)



(b)

Fig. 2.7 Energy function of a modified Hopfield A/D converter.
(a) Energy surface with changing V_1 and V_2.
(b) The corresponding contour.

$$I_{kC} = -\frac{2^{k-1}-1}{2}\left[f^u(V_k) - f^l(V_k)\right] . \tag{2.49}$$

Here, $f^u(V)$ and $f^l(V)$ are given in (2.24) and (2.25).

The energy function of a Hopfield neural-based A/D converter is shown in Fig. 2.6. The original Hopfield energy function has two local minima at $V_S = 1.8V$, which are (0001) and (0010). Due to the two energy wells, the final converged output is determined by the initial states of the network. Notice that the two digital codes are the local minima in the transfer curve shown in Fig. 2.5. The corrected energy surface shown in Fig. 2.7 has only one minimum. Therefore, the final output is always the global minimum. Here, the correction functions used in this simulation are

$$f^u(x) = \frac{1}{1 + \exp(-100000 \times (x - 0.001))} \tag{2.50}$$

and

$$f^l(x) = \frac{1}{1 + \exp(-100000 \times (0.999 - x))} . \tag{2.51}$$

Figure 2.8 shows the circuit schematic of a neural-based A/D converter with the correction technique. In the laboratory experiments, standard integrated-circuit parts were used to construct the 5-bit Hopfield neural-based A/D converter. Amplifiers function as electronic neurons and resistors function as electronic synapses. For the realization of the negative synapse weighting, the amplifier output polarity is reversed. Voltage comparators are used to build the correction circuit, whose reference voltage is a very small negative value of - 2 mV. The measured voltage transfer curves are shown in Fig. 2.9. With the

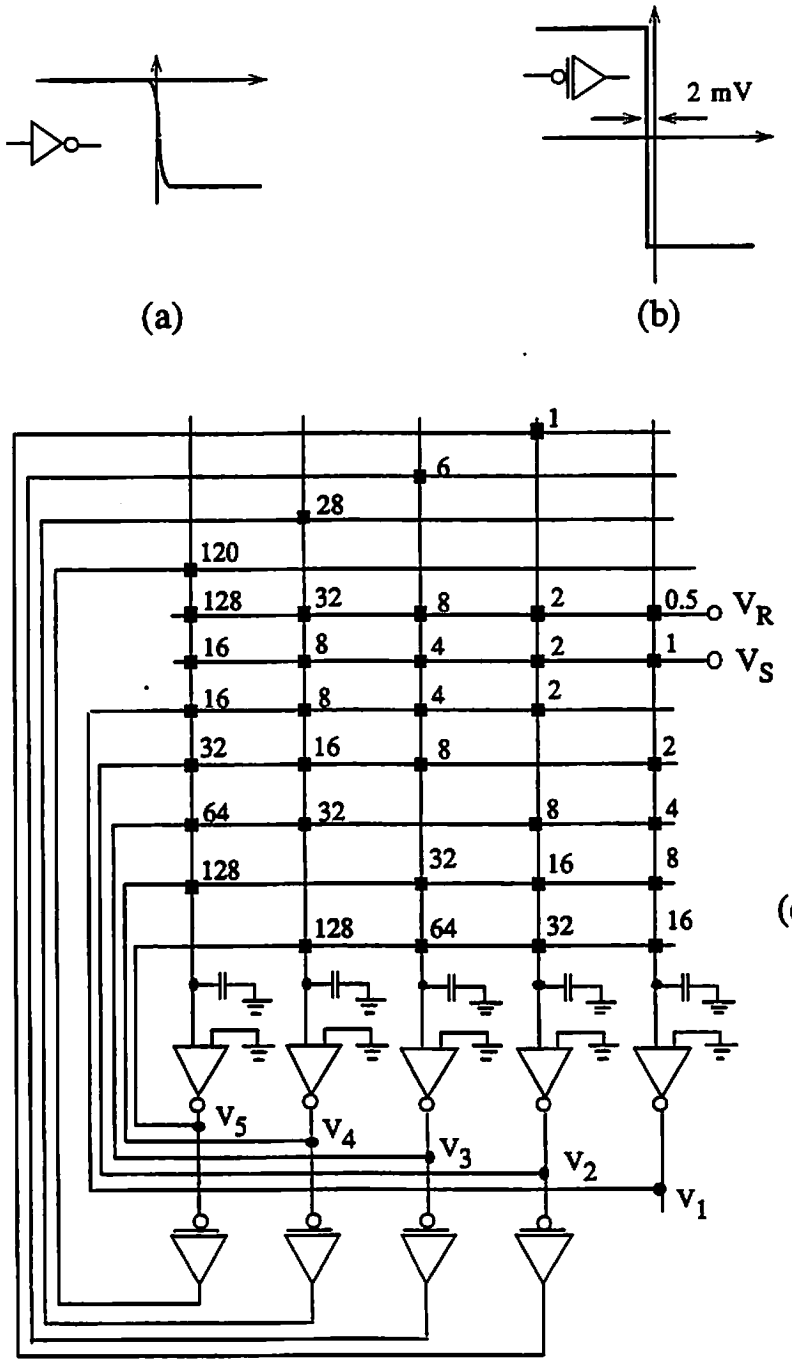
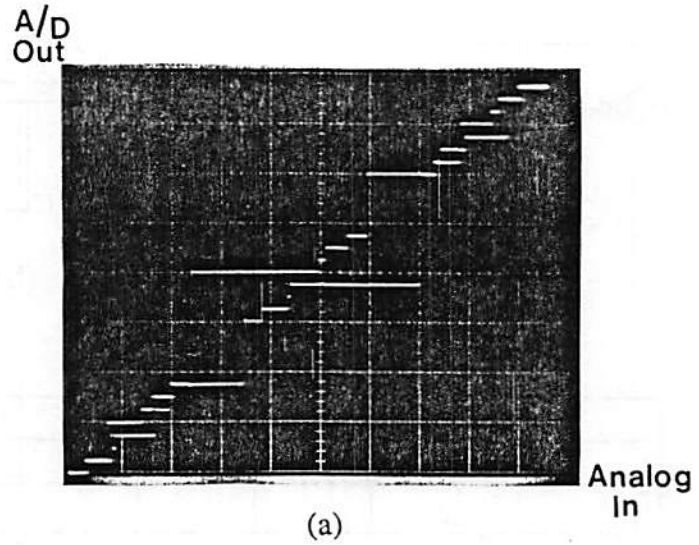
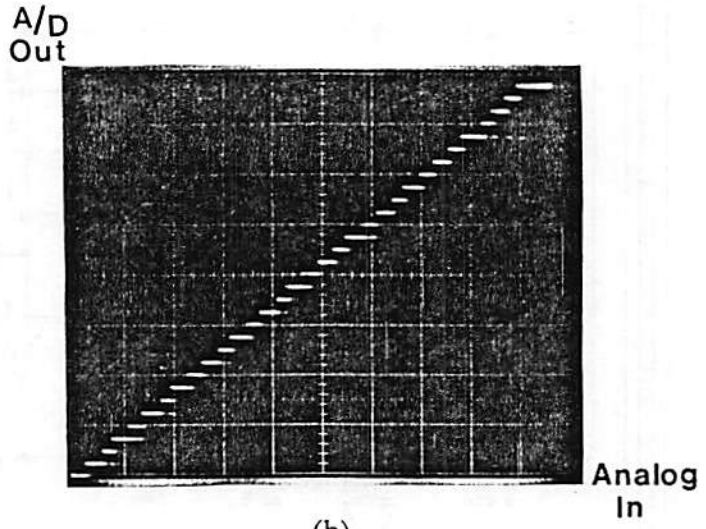


Fig. 2.8 A 5-bit modified neural-based A/D converter.
(a) Transfer characteristics of the Hopfield amplifier.
(b) Transfer characteristics of the correction amplifier.
(c) Complete circuit diagram.



(a)



(b)

Fig. 2.9 Measured transfer characteristics of 5-bit neural-based A/D converter.
(a) Original Hopfield A/D converter.
(b) Modified Hopfield A/D converter.

Table 2.2 Analog range after generalized correction

DIGITAL CODE				CORRECTION CURRENT			ANALOG INPUT RANGE			
D_4	D_3	D_2	D_1	I_2^C	I_3^C	I_4^C	V_1	V_2	V_3	V_4
0	0	0	0	+0.5	+1.5	+3.5	< 0.5	< 0.5	< 0.5	< 0.5
0	0	0	1	+0.5	+1.5	+3.5	> 0.5	< 1.5	< 1.5	< 1.5
0	0	1	0	-0.5	+1.5	+3.5	< 2.5	> 1.5	< 2.5	< 2.5
0	0	1	1	+0.5	+1.5	+3.5	> 2.5	> 2.5	< 3.5	< 3.5
0	1	0	0	-0.5	-1.5	+3.5	< 4.5	< 4.5	> 3.5	< 3.5
0	1	0	1	+0.5	-1.5	+3.5	> 4.5	< 5.5	> 4.5	< 5.5
0	1	1	0	-0.5	-1.5	+3.5	< 6.5	> 5.5	> 5.5	< 6.5
0	1	1	1	+0.5	-1.5	+3.5	> 6.5	> 6.5	> 6.5	< 7.5
1	0	0	0	-0.5	+1.5	-3.5	< 8.5	< 8.5	< 8.5	> 7.5
1	0	0	1	+0.5	+1.5	-3.5	> 8.5	< 9.5	< 9.5	> 8.5
1	0	1	0	-0.5	+1.5	-3.5	< 10.5	> 9.5	< 10.5	> 9.5
1	0	1	1	+0.5	+1.5	-3.5	> 10.5	> 10.5	< 11.5	> 10.5
1	1	0	0	-0.5	-1.5	-3.5	< 12.5	< 12.5	> 11.5	> 11.5
1	1	0	1	+0.5	-1.5	-3.5	> 12.5	< 13.5	> 12.5	> 12.5
1	1	1	0	-0.5	-1.5	-3.5	< 14.5	> 13.5	> 13.5	> 13.5
1	1	1	1	-0.5	-1.5	-3.5	> 14.5	> 14.5	> 14.5	> 14.5

correcting currents, the nonlinearities in the Hopfield A/D converter caused by the local minima are clearly removed.

2.4.2. The Correction Logic Approach

Two design problems exist in the correction method described in the section 2.4.1. First, the previous correction is obtained from violating the local minima existence condition that $GAP_k < 0$ of the digital codes in (2.13). Table 2.2 shows the analog input range after the previous correcting scheme is applied. Since the corrected input ranges for each neuron are the same, all correcting currents for a digital code should be exactly equal. In the VLSI implementation, this precision is difficult to achieve. Second, the offset voltage of the correction amplifier should be very small in order to keep the same network dynamics as the Hopfield neural network. In VLSI technologies, such a small and uniform offset voltage through all correction amplifiers is very difficult to realize.

The second approach to eliminate the local minima in Hopfield neural-based A/D converter avoids these two design problems. Since the overlapped input range for the neural-based A/D converter is determined by the adjacent digital codes as described in previous section, a detailed correction can be conducted between the specific codes [5]. The correction logic circuitry is used to generate the correction current at a specific digital code. Truth table of the correction logic is listed in Table 2.3. The output of the correction logic circuitry can take a discrete value of −1 V, 0 V, or 1 V in order to be compatible with the amplifier output voltage and the reference voltage. The circuit diagram using the correction logic is shown in Fig. 2.10.

Table 2.3 Logic table for the correction logic circuitry

AMPLIFIER OUTPUT				CORRECTION LOGIC OUTPUT		
D_4	D_3	D_2	D_1	C_4	C_3	C_2
0	0	0	0	0	0	0
0	0	0	1	0	0	+1
0	0	1	0	0	0	-1
0	0	1	1	0	+1	0
0	1	0	0	0	-1	0
0	1	0	1	0	0	+1
0	1	1	0	0	0	-1
0	1	1	1	+1	0	0
1	0	0	0	-1	0	0
1	0	0	1	0	0	+1
1	0	1	0	0	0	-1
1	0	1	1	0	+1	0
1	1	0	0	0	-1	0
1	1	0	1	0	0	+1
1	1	1	0	0	0	-1
1	1	1	1	0	0	0

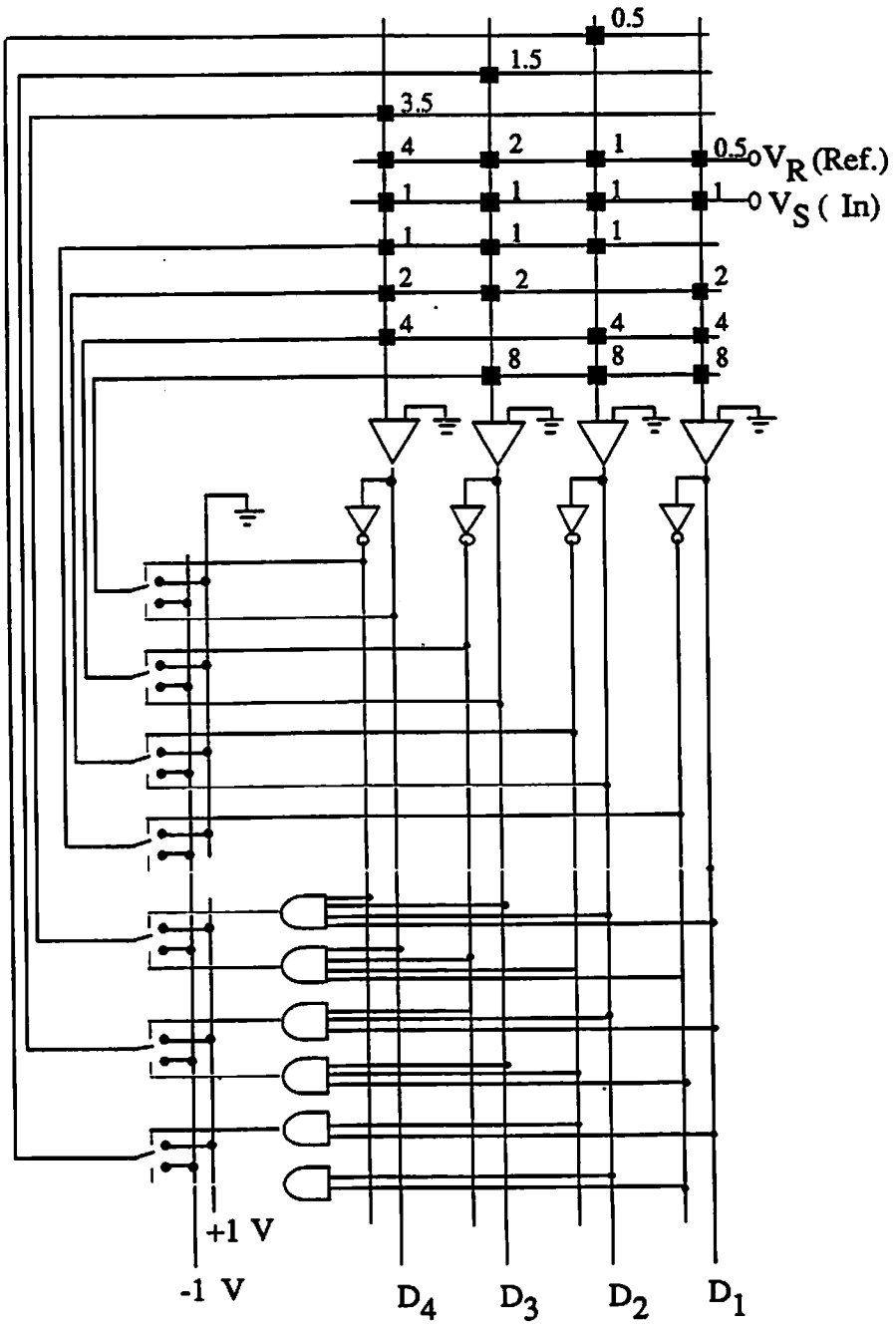


Fig. 2.10 Circuit schematic of a modified 4-bit Hopfield A/D converter with the correction logic gate.

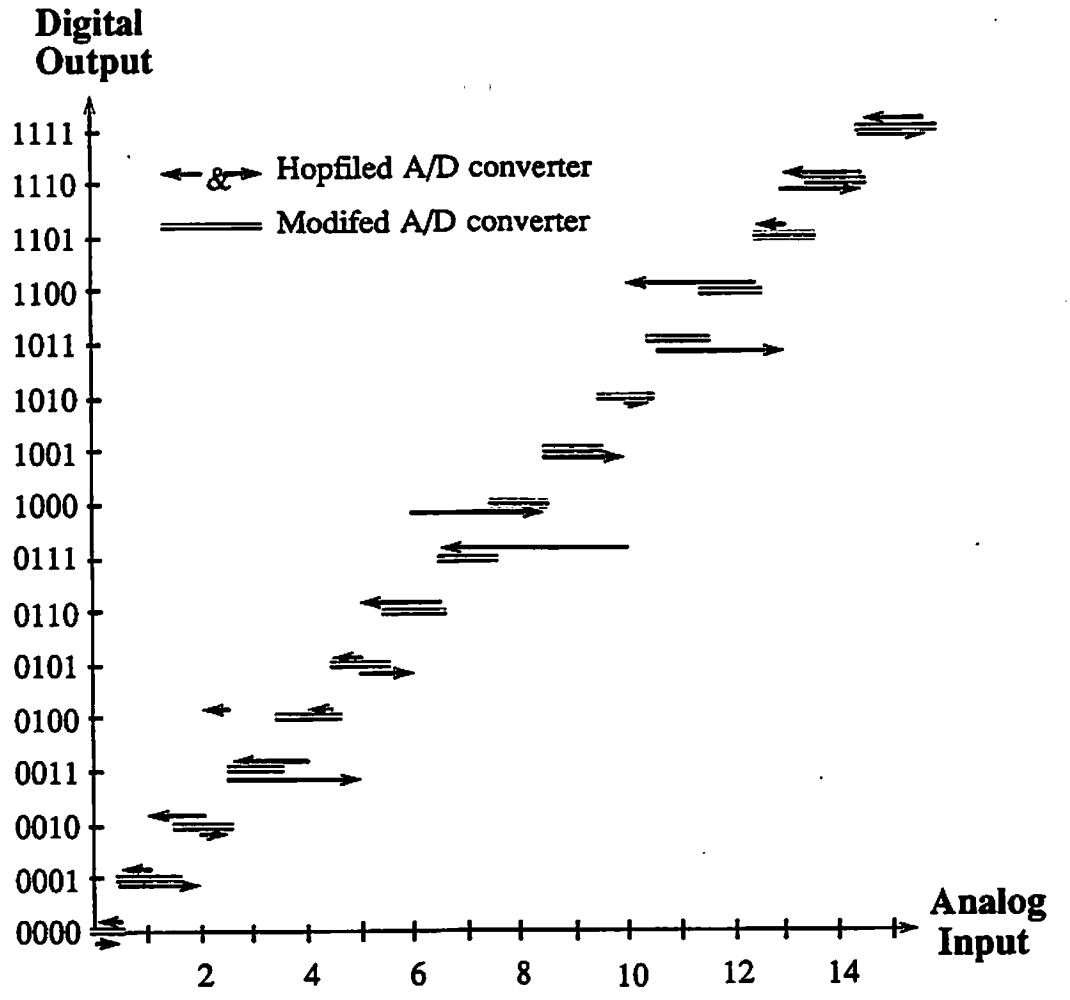


Fig. 2.11 SPICE simulation results on the transfer characteristics of the original Hopfield A/D converter and the modified A/D converter.

Table 2.4 Analog range for adjacent-code correction

DIGITAL CODE				CORRECTION CURRENT			ANALOG INPUT RANGE			
D_4	D_3	D_2	D_1	I_2^C	I_3^C	I_4^C	V_1	V_2	V_3	V_4
0	0	0	0				< 0.5	< 1.0	< 2.0	< 4.0
0	0	0	1	+0.5			> 0.5	< 1.5	< 3.0	< 5.0
0	0	1	0	-0.5			< 2.5	> 1.5	< 4.0	< 6.0
0	0	1	1		+1.5		> 2.5	> 2.0	< 3.5	< 7.0
0	1	0	0		-1.5		< 4.5	< 5.0	> 3.5	< 8.0
0	1	0	1	+0.5			> 4.5	< 5.5	> 3.0	< 9.0
0	1	1	0	-0.5			< 6.5	> 5.5	> 4.0	< 10.0
0	1	1	1			+3.5	> 6.5	> 6.0	> 5.0	< 7.5
1	0	0	0			-3.5	< 8.5	< 9.0	< 10.0	> 7.5
1	0	0	1	+0.5			> 8.5	< 9.5	< 11.0	> 5.0
1	0	1	0	-0.5			< 10.5	> 9.5	< 12.0	> 6.0
1	0	1	1		+1.5		> 10.5	> 10.0	< 11.5	> 7.0
1	1	0	0		-1.5		< 12.5	< 13.0	> 11.5	> 8.0
1	1	0	1	+0.5			> 12.5	< 13.5	> 11.0	> 9.0
1	1	1	0	-0.5			< 14.5	> 13.5	> 12.0	> 10.0
1	1	1	1				> 14.5	> 14.0	> 13.0	> 11.0

To perform SPICE circuit simulation [6], amplifiers were modeled as dependent voltage sources and the rest of the circuit was described at the transistor level. In our simulation, the normalized conductances were used. Figure 2.11 shows the simulated voltage transfer characteristics of the neural-based A/D converter. A monotonically increasing and decreasing analog input voltage was applied. The simulation results for the Hopfield A/D converter are plotted in solid lines, while those for the modified A/D converter are plotted in dotted lines. The SPICE results confirm the hysteresis and nonlinearity characteristics of the Hopfield A/D converter. Simulation results of the modified A/D converter show good output characteristics.

Figure 2.12 shows the die photo of the modified A/D converter. The chip size is 2300 μm x 3400 μm in MOSIS 3-μm CMOS process [7]. The electronic synapses are realized with p-well diffusion resistors and the electronic neurons are implemented with simple two-stage amplifiers. Since a simple CMOS amplifier has a large output impedance, it can not directly drive the feedback conductances. Big CMOS switches between the amplifiers and the resistors help to achieve good impedance matching. The CMOS switch size and conductance values are design parameters. In this chip, the unit resistor is chosen to be 100 kΩ and the on-resistance of the CMOS switch with ± 5 V power supplies is chosen to be 500 Ω. A significant portion of the chip area is occupied by the resistors and switches. If the resistor is replaced by the synthesized-resistors using MOS transistors, the chip size can be greatly reduced.

The measured voltage transfer curves for the original Hopfield A/D converter and the modified A/D converter are shown in Fig. 2.13. The analog input

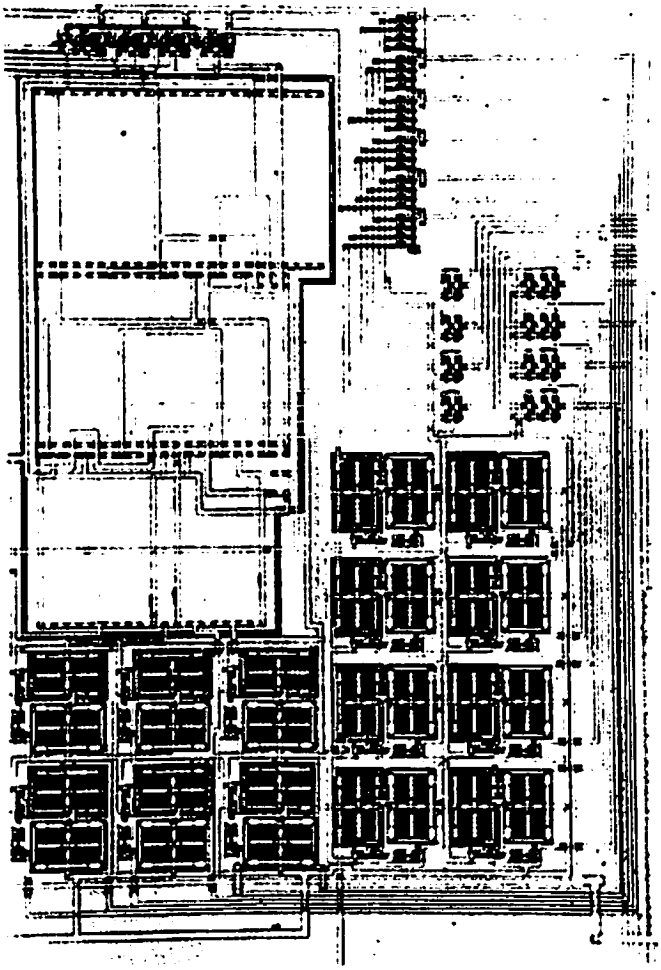
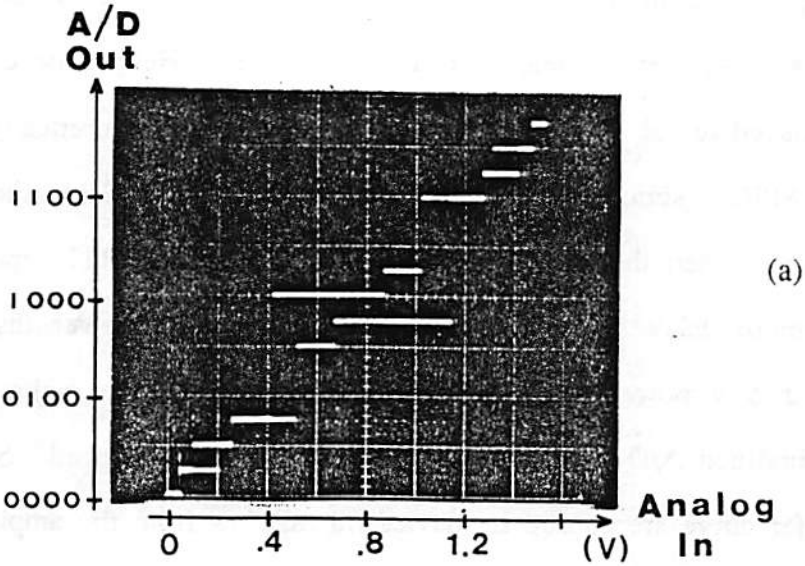
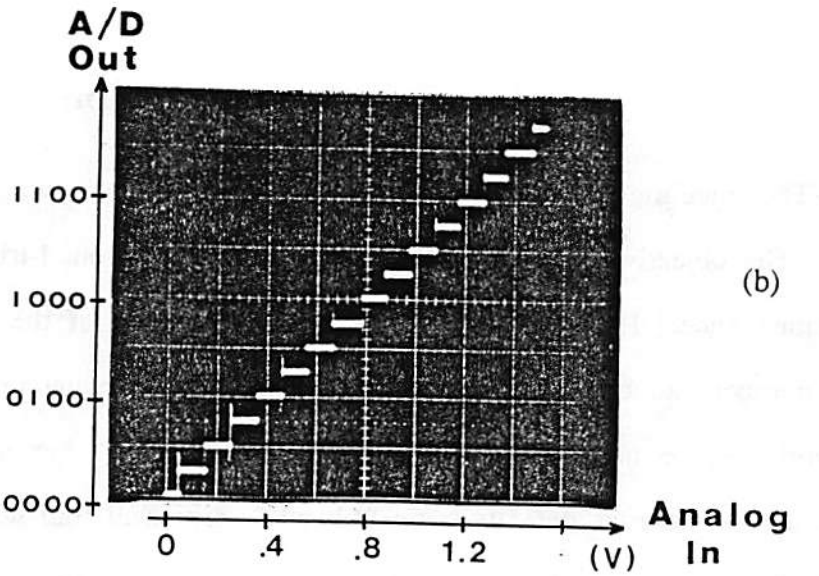


Fig. 2.12 Die photo of the modified A/D converter with correction logic gates.



(a)



(b)

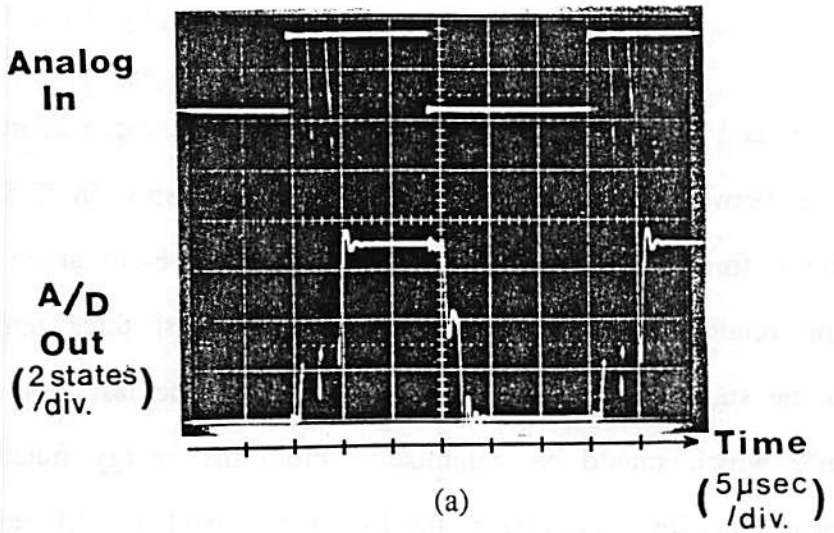
Fig. 2.13 Measured transfer characteristics of neural-based A/D converter.
(a) Original Hopfield network.
(b) Modified Hopfield network.

voltage range in this experiment was from 0 V to 1.5 V, because the conductances $\{T_{iS}\}$ were increased by a factor of ten. Hence, the conversion step size is reduced to 0.1 V. Experimental data agree with theoretically calculated results and SPICE simulation results very well. Figure 2.14 shows the converter response when the output changes from (0000) to (0111) and vice versa. The maximum delay time is about 5.7 μsec and total power dissipation is 6 mW with ± 5 V power supplies. Since the amplifier is the major time delay circuit, the modified A/D converter has a similar conversion speed. Small spikes on the transfer curve are caused by device mismatches from the amplifier offset voltage, amplifier gain, and resistance. The effect of mismatches can be eliminated by the learning process if programmable synapses are available.

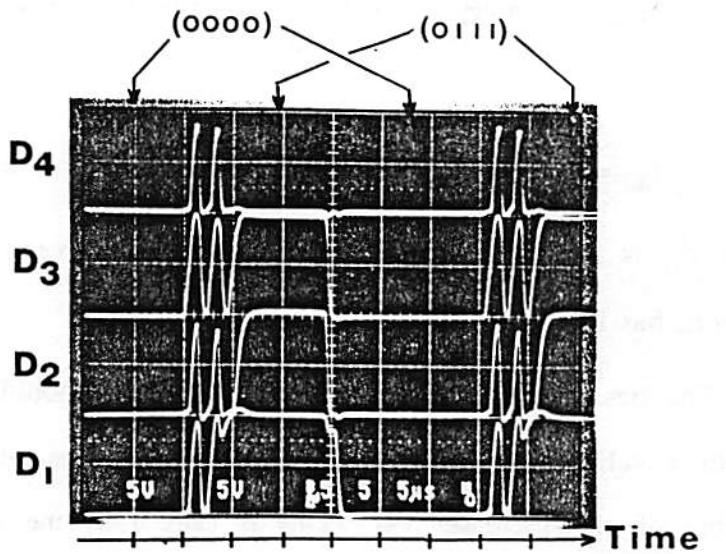
2.5. Traveling Salesman Problem

The traveling salesman problem (TSP) is a well-known np-complete problem. The objective of the TSP is to find the shortest round-trip distance visiting all cities once. However, only an acceptable solution of the TSP with a large city number can be obtained at a reasonable computational time. The Hopfield network can be used to efficiently solve the TSP [8]. For n-city problem, n^2 neurons (n neurons per city) are utilized to represent tour-sequence so that the final solution can be shown in the $n \times n$ square array format. The energy function is expressed as

$$E = \frac{A}{2} \sum_{X=1}^{n} \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} V_{Xi} V_{Xj} + \frac{B}{2} \sum_{i=1}^{n} \sum_{X=1}^{n} \sum_{Y \neq X, Y=1}^{n} V_{Xi} V_{Yi}$$



(a)



(b)

Fig. 2.14. Transient responses.
(a) A/D converter output.
(b) Individual amplifier outputs.

$$+\frac{C}{2}\left[\sum_{X=1}^{n}\sum_{i=1}^{n}V_{Xi}-n\right]^2+\frac{D}{2}\sum_{X=1}^{n}\sum_{Y\neq X,Y=1}^{n}d_{XY}V_{Xi}(V_{Y,i+1}+V_{Y,i-1})\ , \quad (2.52)$$

where X and Y represent cities, i and j represent tour-procedure, and d_{XY} is the distance between city X and Y. The first three terms in (2.52) are constraint functions from only one-visit-at-one-city, only one-city-at-one-time, and total visiting number, respectively. Notice that the first three terms become zero when the solution is valid. On the other hand, the last term is the total tour distance which should be minimized. From the energy function, the conductances and the input currents of the Hopfield network in (2.1) are given as

$$T_{Xi,Yj} = -A\,\delta_{XY}(1-\delta_{ij}) - B\,\delta_{ij}(1-\delta_{XY}) - C - Dd_{XY}(\delta_{j,i+1}+\delta_{j,i-1}) \quad (2.53)$$

and

$$I_{Xi} = Cn\ . \quad (2.54)$$

Here, δ_{ij} is 1 if $i=j$ and is 0 otherwise. Since every $T_{Xi,Yj} < 0$, the Hopfield network has local minima from (2.10).

The heuristic coefficients A, B, C, and D should be carefully selected to obtain a valid tour. The choice of coefficients becomes very difficult as the number of city increases [9]. This is caused by the mixture of the constraint functions and the objective function in the energy function. During the searching process to find a low energy well, the network can not distinguish the constraint function and the objective function. Thus, the final solution is usually invalid when the number of city is large. Several research results [10,11] have been reported to find a valid solution.

2.5.1. Competitive-Hopfield Network Approach

A new solution consists of the utilization of a Hopfield network and a competitive network has been developed [12]. The constraint functions are implemented in the competitive network, while the energy function with reduced constraint functions is implemented in the original Hopfield network as shown in Fig. 2.15. The competitive network monitors the outputs of the Hopfield network. Once the outputs are larger than a threshold voltage of neurons in the competitive network, the network starts to search for the highest output through the competing process with the 'winner-take-all' strategy. Here, signal delay in the competitive network is much smaller than that in the Hopfield network. Since the constraint functions of a combinatorial optimization problem are realized by the competitive network, the objective function in the energy function of Hopfield network can be maximized. Hence, solution searching process in the Hopfield network can be mainly determined by the objective function.

For a n-city TSP, the synapses $\{T^H_{Xi,Yj}\}$ and input currents $\{I_{Xi}\}$ for the Hopfield network are given as

$$T^H_{Xi,Yj} = -A\,\delta_{XY}(1-\delta_{ij}) - B\,\delta_{ij}(1-\delta_{XY}) - C - Dd_{XY}(\delta_{j,i+1}+\delta_{j,i-1}) \quad (2.55)$$

and

$$I_{Xi} = Cn + (A + B + C + D)V^C_{Xi}, \quad (2.56)$$

while the synapses $\{T^C_{Xi,Yj}\}$ for the competitive network are given as

$$T^C_{Xi,Yj} = \delta_{XY}(1 - \delta_{ij}) + \delta_{ij}(1 - \delta_{XY}) . \quad (2.57)$$

Here, V^C_{Xi} is the output of the competitive network and δ_{ij} is 1 if $i=j$ and is 0

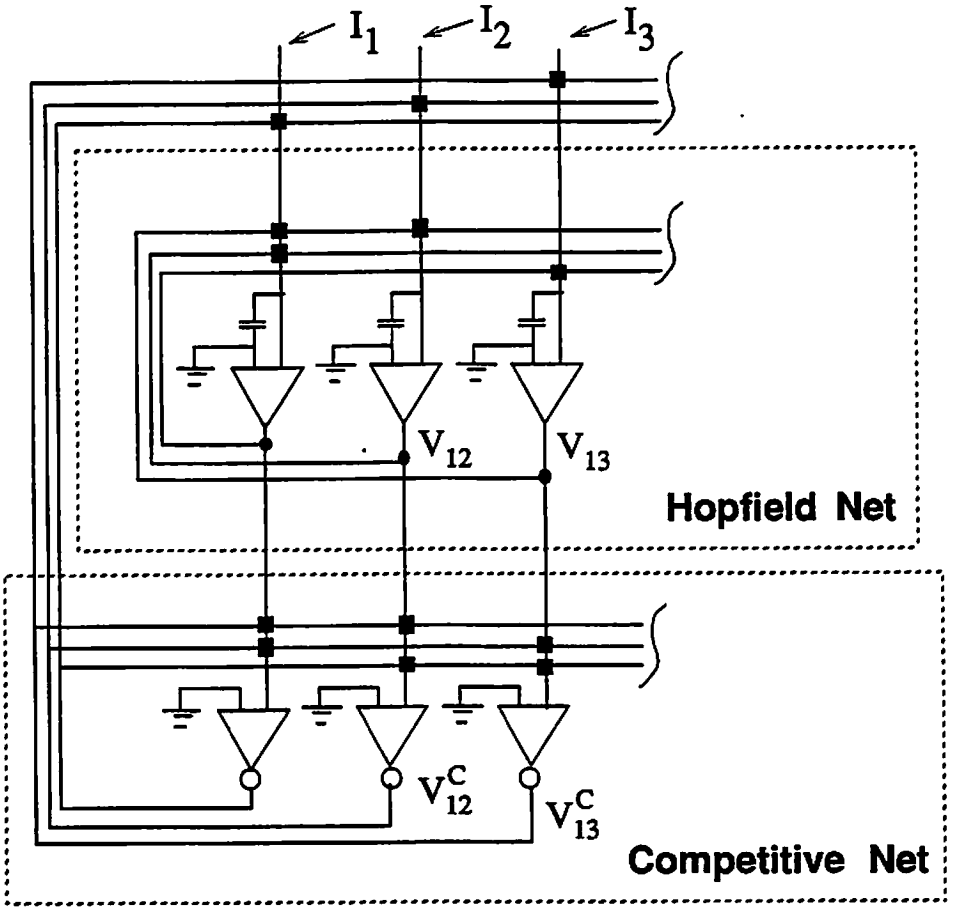


Fig. 2.15 A competitive-Hopfield neural network.

otherwise. Notice that the weighting factors A, B, C, and D are the same as in (2.53).

Dynamics of the Hopfield network can be described in the following differential equation,

$$C_{Xi}\frac{du_{Xi}}{dt} = \sum_{Y\neq X, Y=1}^{n}\sum_{j\neq i, j=1}^{n} T^{H}_{Xi,Yj}V_{Yj} - T_{Xi}u_{Xi} + I_{Xi} \quad , \tag{2.58}$$

where T_{Xi}, u_{Xi}, and C_{Xi} are the equivalent conductance, input voltage, and input capacitance at the i-th amplifier input node, respectively. To maintain the Hopfield network dynamics, the competitive network operates after each Hopfield network iteration in the software simulation.

2.5.2. Search for Optimal Solutions

The global minimum of the TSP can be obtained by adding the correction currents. Since the global minimum of a TSP is unknown in contrast to the neural-based A/D converter, the decision on the correction current is not obvious. However, the decision can be done by adjusting the amount of the correction currents as follows [13]. The lower bound of GAP_k for the codes in (2.13) can be given as

$$GAP_k = \sum_{j=1}^{k-1} T_{kj}(V_j^u - V_j^l) \geq -\sum_{j=1}^{k-1}|T^H_{kj}| \equiv \hat{T}_{kC} \ . \tag{2.59}$$

The correcting current I_{kC} can be determined with a weighting factor α,

$$I_{kC} = \alpha\hat{T}_{kC}\left[f^u(V_k) - f^l(V_k)\right] \ . \tag{2.60}$$

Here, $\alpha \leq 1.0$. In the under-correction case of having a small value of α, all

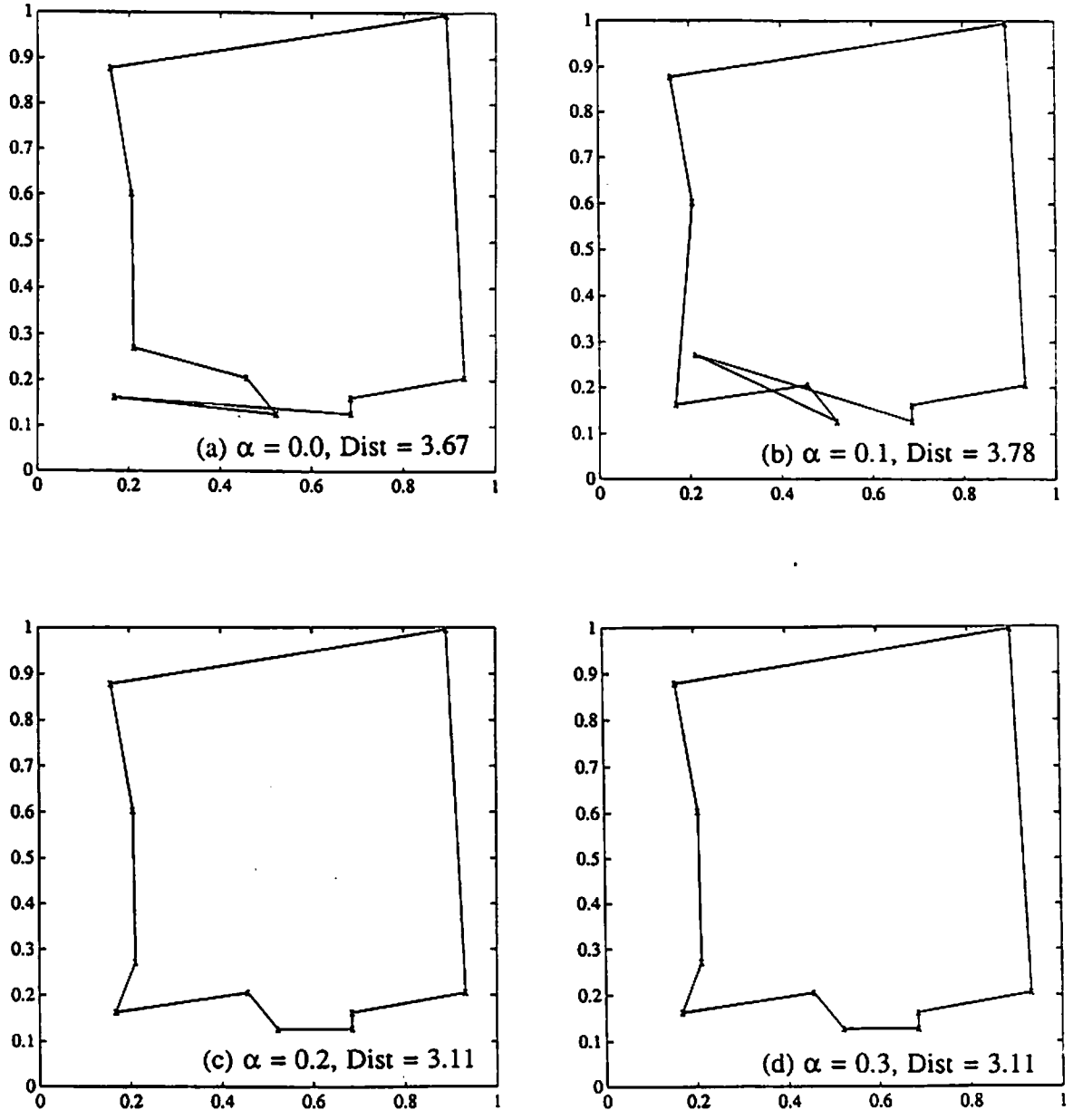


Fig. 2.16 One 10-city TSP.
With $\alpha = 0.2$ and 0.3, the global minimum is obtained.

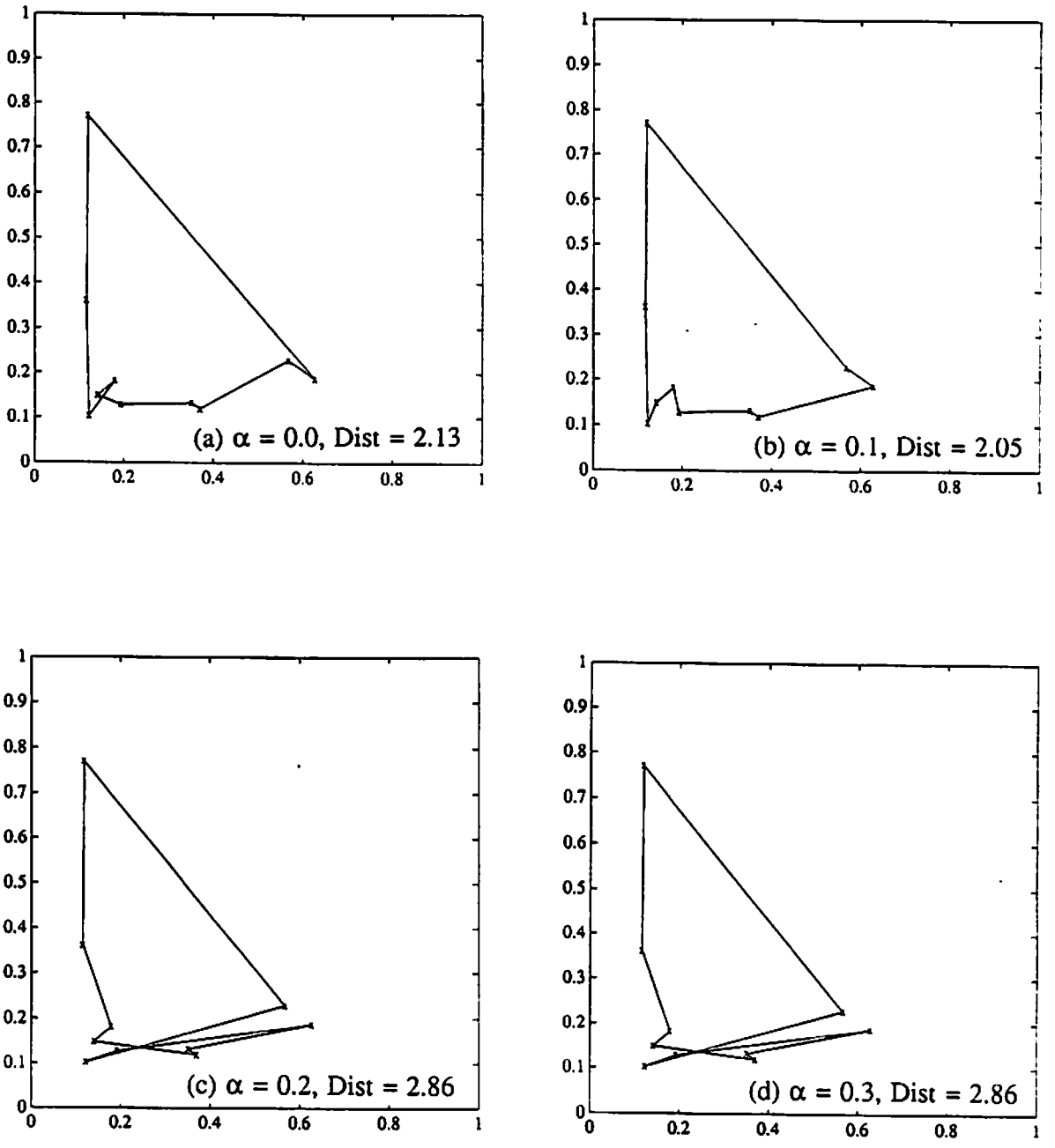


Fig. 2.17 Another 10-city TSP.
With $\alpha = 0.1$, the global minimum is obtained.

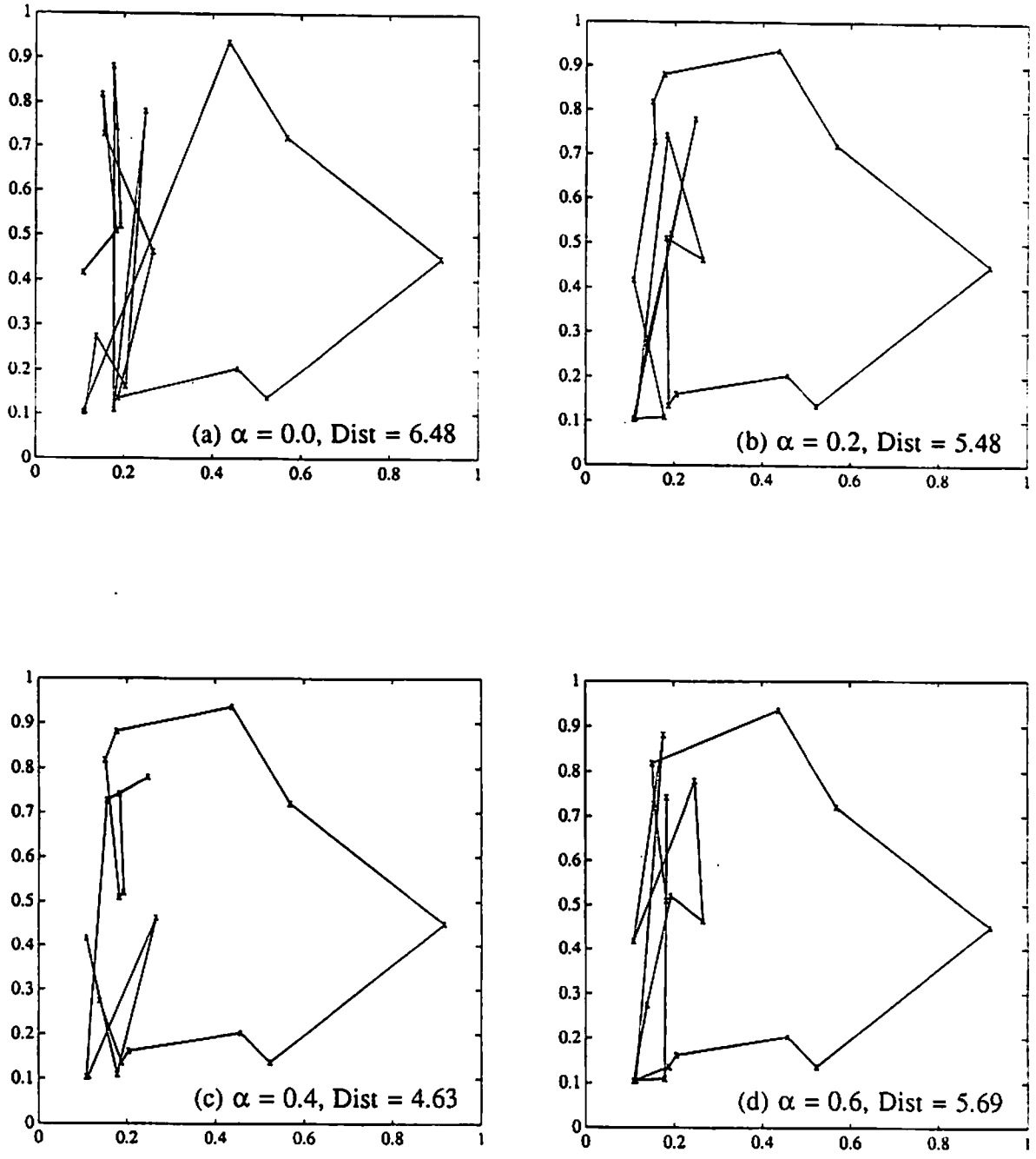
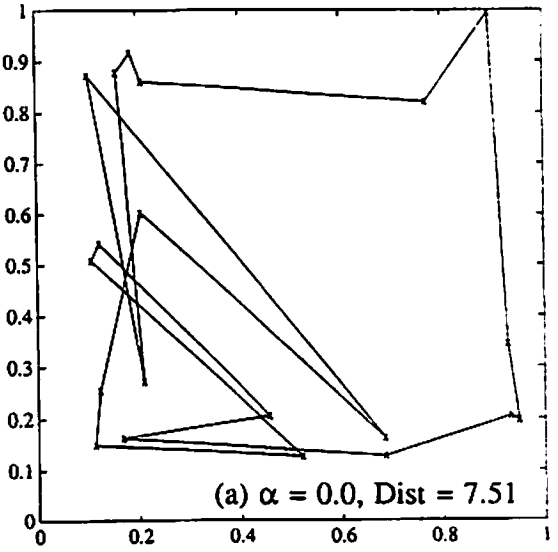
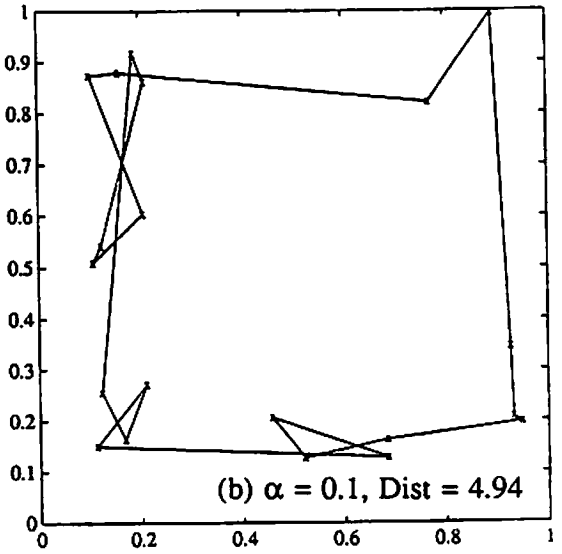


Fig. 2.18 A 20-city TSP. With $\alpha = 0.4$, a very good solution is obtained.





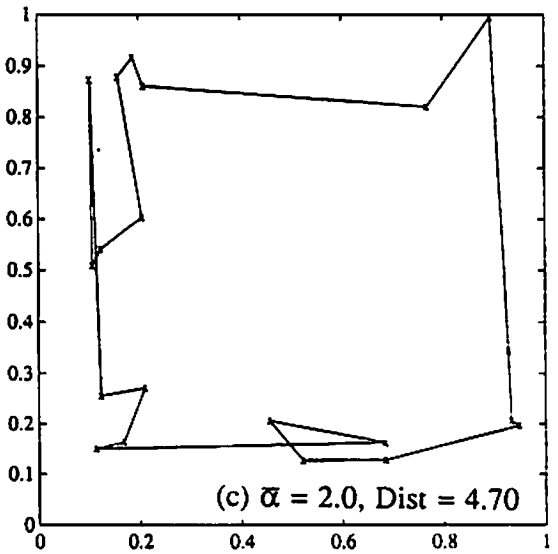


Fig. 2.19 One 20-city TSP with averaged correction scheme.

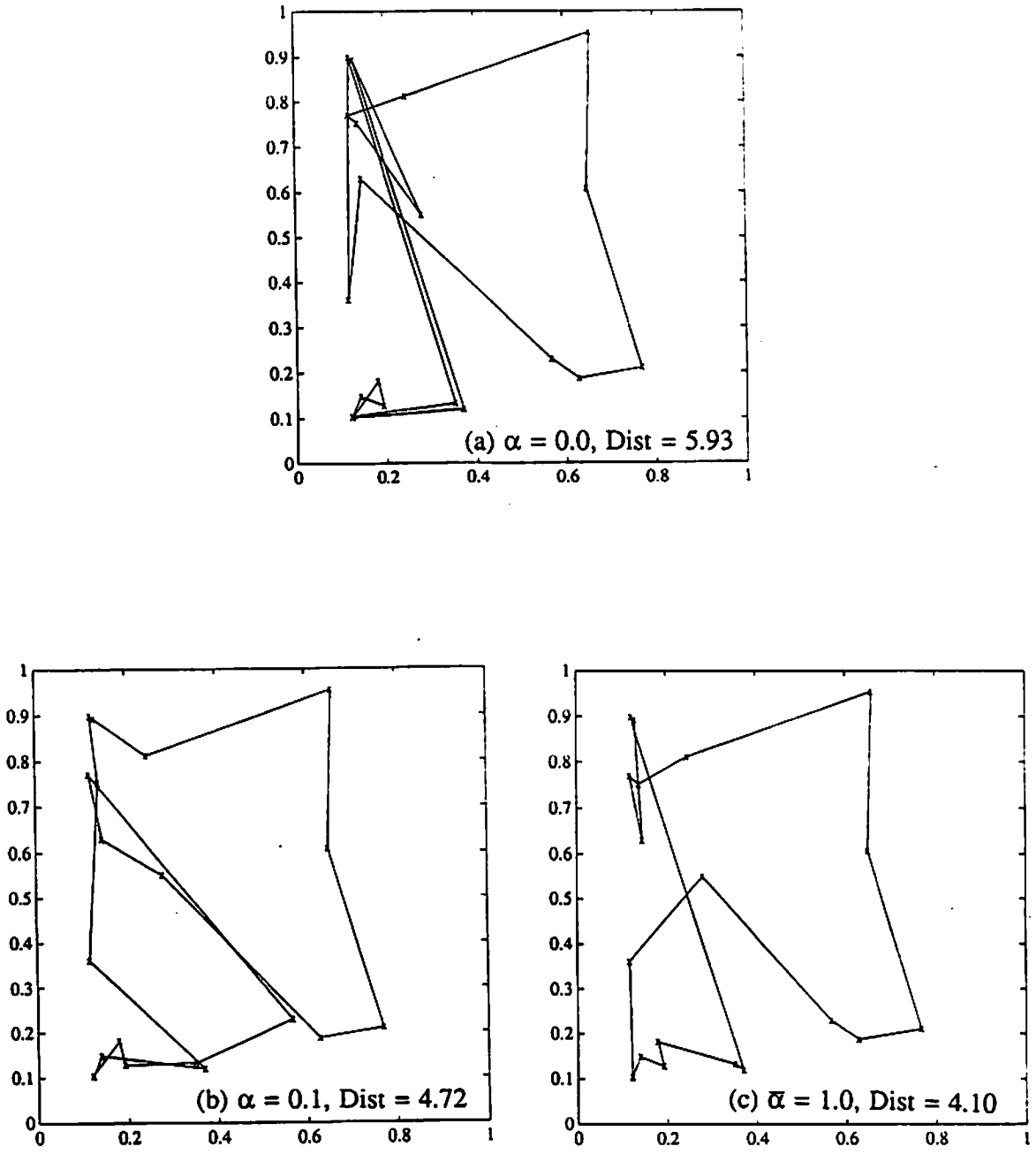


Fig. 2.20 Another 20-city TSP with averaged correction scheme.

local minima can not be eliminated. On the other hand, there is no stable state in the over-correction case of having a large value of α. The appropriate value for α can be determined by monitoring whether the energy function of corrected network is steadily decreasing, or not.

Experimental results for 10-city and 20-city are shown in Fig. 2.16 - 2.18. The first city on the tour was fixed and the competitive algorithm was applied [12]. For the 10-city TSP, global minimum can be obtained with a certain weighting factor for the correction currents. On the other hand, the global minimum of the 20-city TSP is not obtained but significant improvement in the tour distance was made with the correcting currents. When the weighting factor α is very large, the solution becomes poor because the over-correction can change the global minimum.

Since the lower bound of GAP_k in (2.59) increases with k, the correction of the neuron with high k is usually much larger than that of the neuron with low k. Therefore, the convergence speed to the final value decreases with the k value. A new statistically averaged value for the GAP calculation can be used to make the correction current as following,

$$I_{kC} = \bar{\alpha}\frac{\hat{T}_{kC}}{k}\left[f^u(V_k) - f^l(V_k)\right] . \tag{2.61}$$

Here, $\bar{\alpha}$ can be larger than 1.0. The simulation results using the averaged correction scheme are shown in Figs. 2.19 and 2.20. The minimum tour distance of the averaged scheme is usually smaller than that of the absolute scheme in (2.60).

References

[1] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad., Sci. U.S.A.*, vol. 81, pp. 3088 -3092, May 1984.

[2] R. P. Lippman, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April 1987.

[3] D. W. Tank and J. J. Hopfield," Simple 'neural' optimization networks: an A/D converter, signal decision circuit, and a linear programming circuit," *IEEE Trans. on Circuits and Systems*, vol. CAS-33, no. 5, pp. 533-541, May 1986.

[4] B. W. Lee and B. J. Sheu, "An investigation on local minima of Hopfield network for optimization circuits," *IEEE Inter. Conf. on Neural Networks*, vol. 1, pp. 45-51, July 1988.

[5] B. W. Lee and B. J. Sheu, "Design of a neural-based A/D converter using modified Hopfield network," *IEEE Jour. of Solid-State Circuits*, vol. 24, no. 4, pp. 1129-1135, Aug. 1989.

[6] T. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, SPICE3B1 User's Guide, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Jan. 1987.

[7] C. Tomovich, "MOSIS-A gateway to silicon," *IEEE Circuits and Devices Magazine*, vol. 4, no. 2, pp. 22-23, Mar. 1988.

[8] J. J. Hopfield and D. W. Tank, " 'Neural' computation of decisions in optimization problems," *Biol. Cybernetics*, vol. 52, pp. 141-152, 1985.

[9] G. V. Wilson and G. S. Pawley, "On the stability of the traveling salesman problem algorithm of Hopfield and Tank," *Biol. Cybernetics*, vol. 58, pp. 63-70, 1988.

[10] D. E. Van den Bout and T. K. Miller, "A traveling salesman objective function that works," *IEEE Inter. Conf. on Neural Networks*, vol. II, pp. 299-303, July 1988.

[11] Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso, "Combinatorial optimization with Gaussian Machines," *Inter. Joint Conf. on Neural Networks*, vol. I, pp. 533-540, June 1989.

[12] B. W. Lee and B. J. Sheu, "Combinatorial Optimization using competitive-Hopfield neural network," *Inter. Joint Conf. on Neural Networks*, vol. II, pp. 627-630, Jan. 1990.

[13] S. M. Gowda, B. W. Lee, and B. J. Sheu, "An improved neural network approach to the traveling salesman problem," *IEEE Tencon Proceedings*, pp. 552-555, BomBay, India, Nov. 22-24, 1989

[14] Professor I. Reed, University of Southern California, *Private communication*, 1987.

[15] R. A. Games, E. L. Key, and D. Moulin, "*An improved neural-network analog-to-digital converter*," Technical Report, The MITRA Coporation, Bedford, MA 01730, 1987.

Chapter 3

Hardware Annealing Technique for Artificial VLSI Neural Circuits

Engineering optimization is an important subject in signal and image processing. A conventional searching technique for finding the global minimum is to use gradient descent, which finds the direction for the next iteration from the gradient of the objective function. For complicated problems, the gradient descent technique often gets stuck at a local minimum where the objective function has surrounding barriers. In addition, the complexity of most combinational optimization problems increases dramatically with the problem size and makes it very difficult to obtain the global minimum within a reasonable amount of computational time. Several methods have been reported to assist the network output to escape from the local minima [1,2]. The simulated annealing method is a heuristic approach which can be widely applied to the combinational optimization problems [3,4]. The solutions by the simulated annealing technique are close to the global minimum within a polynomial upper bound for the computational time and are independent of initial conditions. The simulated annealing technique has been successfully applied in VLSI layout generation [5] and noise filtering in image processing [6].

3.1. Simulated Annealing in Software Computation

Simulated annealing in software computation of neural networks can be conducted in the following way.

Step 1. Start from a high temperature and a given reference state.

Step 2. Compare the energy value in a new state with that in the reference state.

Step 3. If energy value of the new state is higher than that of the reference state, then, weight the new state output by $e^{-\frac{\Delta E}{kT}}$.

Step 4. Replace the reference state with the new state.

Step 5. Check whether all states are frozen.

If yes, terminate.

Otherwise, decrease temperature and go to "Step 2".

Since it usually takes a lot of time at "Step 2" to compare all possible states, a specific perturbation rule is often used. The perturbation is usually called artificial noise in software computation.

The simulated annealing technique can help Hopfield neural networks to escape from local minima by replacing the characteristics of neurons from a sigmoid function [7] to the Boltzmann distribution function. In software computation, the Hopfield network operation is described at two consecutive time steps during each iteration cycle. At the first time step, input signals to the neurons are summed up; while at the second step, the neuron outputs are updated. The update rule for the original Hopfield networks is

$$V_i = g(u_i) \quad , \tag{3.1}$$

and that for the Boltzmann machine is

$$V_i = \frac{1}{1 + e^{-u_i q/kT}} \quad . \tag{3.2}$$

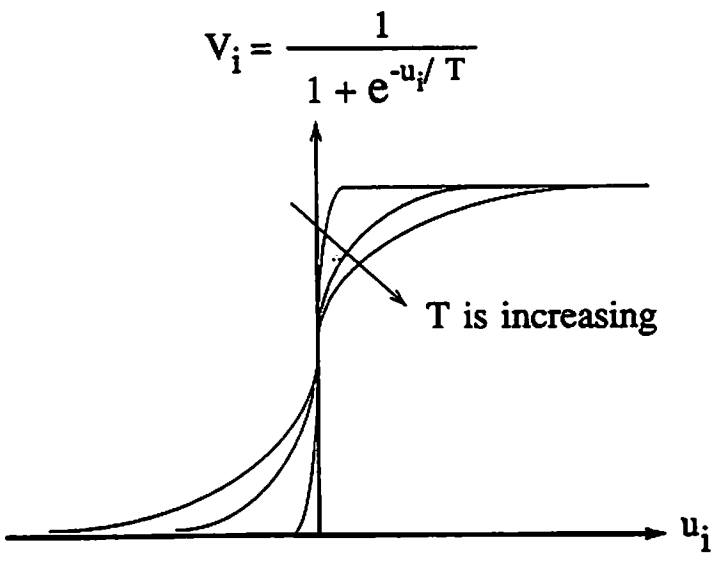


Fig. 3.1 Analogy between the annealing temperature of a Boltzmann machine and the amplifier gain of an electronic neuron.

Here, $g(.)$ is the amplifier input-output transfer function, q is electronic charge, k is Boltzmann constant, and T is the temperature of Boltzmann distribution as shown in Fig. 3.1. Notice that the searching process for all possible states in simulated annealing is a part of network operation at a temperature in the Boltzmann machine. At the steady state, relative probability of state $S1$ to state $S2$ in Boltzmann distribution is determined by the energy difference of these two states,

$$\frac{P_{S1}}{P_{S2}} = e^{-(E_{S1} - E_{S2})/kT} \quad . \tag{3.3}$$

Here, E_{S1} and E_{S2} are the corresponding energy levels. This update rule allows the network to escape from local minima in the energy well.

Simulated annealing is analogy to metallurgical annealing. To find the lowest energy level of a metal, the best way is known to melt the metal and to reduce temperature slowly in order to allow atoms to fit into the lattice. Similarly, the Boltzmann machine can find the global minimum by changing the temperature of the update rule gradually. A good strategy to apply the simulated annealing technique in software computation is to start from a high temperature which will make the network reach the steady state in a very short period of time. The ideal formulation for cooling schedule [3] is very difficult to implement, because a large number of iterations at each temperature and a very small temperature step are required to achieve the global minimum. Several approximated cooling schedules have been proposed [2]. Since the number of iterations at a given temperature and the change of temperature should be compromised to speed up the convergence process, a very large computational time is necessary

in software computation. On the other hand, recent advances in VLSI technologies make possible the design of compact electronic neural networks with built-in hardware annealing capability.

3.2. Electronic Annealing

The high-speed simulated annealing technique for a Boltzmann machine is most suitable for VLSI chip design [8]. To change the temperature of the probability function for a Boltzmann machine is equivalent to vary the amplifier gain. Thus, the cooling process in a Boltzmann machine is equivalent to the voltage gain increase process in an amplifier. The amplifier gain in electronic neural circuits can be adjusted continuously, while the annealing temperatures in software computation are always in the discrete fashion. Hence, the final solution of the electronic neural circuits after annealing is guaranteed to be global minimum, which is in sharp contrast to the approximated convergence [2] in software computation.

A systematic method to determine the initial and final temperatures for the amplifier "cooling" procedure is described below.

3.2.1. Starting Temperature of the Cooling Schedule

At a very high temperature, all metal atoms lose the solid phase so that they position themselves randomly according to statistical mechanics. An important quantity in annealing is the lowest temperature that still could provide enough energy to completely randomize the metal atoms; equivalently, the highest amplifier gain to make an electronic neural network escape from local

minima. Figure 3.2 shows a Hopfield neural network. Neurons are made of amplifiers while synapses are made of resistors. If the neurons consist of high-gain amplifiers, their outputs will saturate at 0 V and 1 V due to the positive feedback of the network. However, given a very low amplifier gain, the network could lose the well-defined state.

Assuming that all amplifiers operate linearly, the governing equation for the i-th amplifier in the Hopfield network is given as

$$C_i \frac{du_i(t)}{dt} + T_i u_i(t) = \sum_{j \neq i,\ j=1}^{n} T_{ij} v_j(t) + I_i(t) \ , \tag{3.4}$$

where T_{ij} is the conductance between the i-th and j-th neurons and T_i and C_i are the equivalent input conductance and input capacitance to the i-th amplifier. Here, $u_i(t)$ is the input voltage to the i-th amplifier and $v_j(t)$ is the output voltage of the j-th amplifier. Notice that $v_j(t)$ denotes the analog value and $V_j(t)$ denotes the saturated digital value. By taking Laplace transformation, (3.4) becomes

$$(sC_i + T_i)U_i(s) = \sum_{j \neq i,\ j=1}^{n} T_{ij} V_j(s) + I_i(s) + P_i, \tag{3.5}$$

where P_i is a constant and $U_i(s)$, $v_i(s)$, and $I_i(s)$ are transformed variables of $u_i(t)$, $V_i(t)$, and $I_i(t)$, respectively. If all amplifiers are assumed to operate in the linear region with the transfer function being $A(s)$ and to have bandwidth much larger than T_i/C_i, then

$$V_i(s) = A_i(s)U_i(s) \ . \tag{3.6}$$

The system equation can be expressed as $\mathbf{B}\,\underline{V} = \underline{F}$ with matrix $\mathbf{B}$ being a

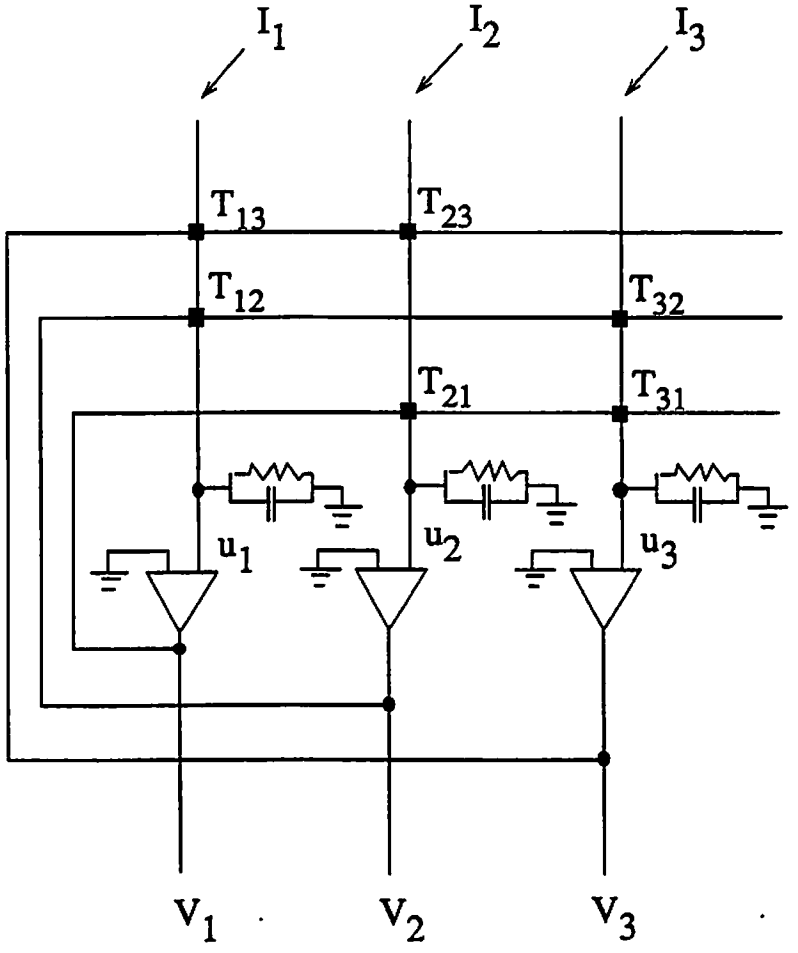


Fig. 3.2 A Hopfield neural network with non-inverting amplifiers as neurons.

$n \times n$ matrix and $\underline{V}$ and $\underline{F}$ being $1 \times n$ vectors. The matrix and vector expressions for **B**, $\underline{V}$, and $\underline{F}$ are

$$\mathbf{B} = \begin{bmatrix} -\dfrac{sC_1 + T_1}{A_1} & T_{12} & T_{13} & \cdots & T_{1n} \\ T_{21} & -\dfrac{sC_2 + T_2}{A_2} & T_{23} & \cdots & T_{2n} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ T_{n1} & T_{n2} & \cdots & & -\dfrac{sC_n + T_n}{A_n} \end{bmatrix} , \tag{3.7}$$

$$\underline{V} = \left[V_1, V_2, \ldots, V_n\right]^T , \tag{3.8}$$

and

$$\underline{F} = \left[-I_1 - P_1, \ldots, -I_n - P_n\right]^T . \tag{3.9}$$

The sum and product of eigenvalues of the system matrix B are

$$\sum_{i=1}^{n} \lambda_i = -\sum_{i=1}^{n} \frac{(sC_i + T_i)}{A_i} \tag{3.10}$$

and

$$\prod_{i=1}^{n} \lambda_i = \det(\mathbf{B}) \tag{3.11}$$

respectively, where det(**B**) is the determinant of matrix **B**. If the amplifier gain is sufficiently large, which is the condition used in Hopfield's analysis [9], (3.10) and (3.11) become

$$\sum_{i=1}^{n} \lambda_i \simeq 0 \tag{3.12}$$

and

$$\prod_{i=1}^{n} \lambda_i = \det(\mathbf{B}) \neq 0. \tag{3.13}$$

With the constraint that $T_{ij} = T_{ji}$, all eigenvalues will lie on the real axis of the s-plane. Thus, at least one positive real eigenvalue exists. It makes the amplifier outputs saturated at extreme values of 0 V or 1 V.

Figure 3.3 shows the radius of the eigenvalues determined from the Gerschgorin's Theorem [10],

$$\left| z + \frac{sC_i + T_i}{A_i} \right| \leq \sum_{j \neq i,\ j=1}^{n} |T_{ij}| \quad \text{for } \textit{all } i \ . \tag{3.14}$$

Notice that A_i and T_i are always positive. To assure that the network operates in the positive feedback mode, there should be at least one eigenvalue whose real part is positive. The lowest amplifier gain (A_n) which satisfies the above condition can be determined from

$$A_n = \max\left\{ \frac{T_i}{\sum_{j \neq i, j=1}^{n} |T_{ij}|} \quad \text{for } 1 \leq i \leq n \right\} \ . \tag{3.15}$$

Here, n denotes the number of amplifiers operating in the linear region. Notice that the above derivation is based on the condition that the eigenvalue lies on a circle. Since the validity of the condition depends on the maximum real value of the eigenvalues of matrix **B** determined by the resistive network $\{T_{ij}\}$, the

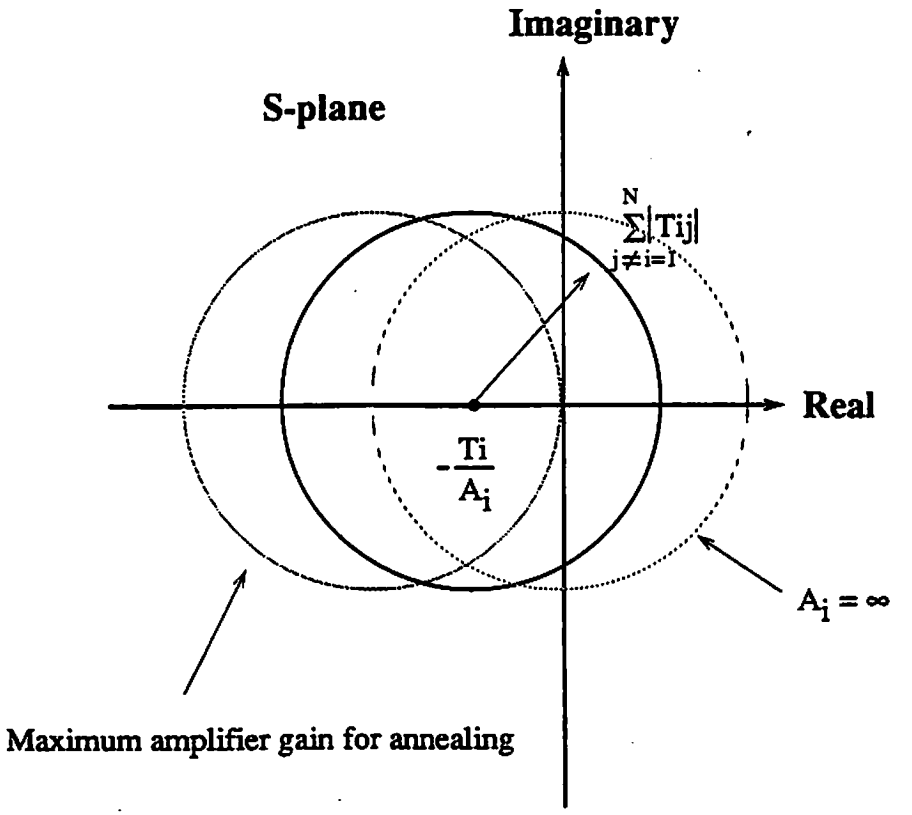


Fig. 3.3 Radius of eigenvalues with different amplifier gain.

above gain requirement is a sufficient condition that the amplifiers in Hopfield network stay in the positive feedback mode. With the amplifier gain less than A_n, all output states of the Hopfield network become legal for any input signal level. Thus, A_n is the maximum amplifier gain (equivalently the lowest annealing temperature) that can still randomize the neuron outputs.

3.2.2. Final Temperature of the Cooling Schedule

The amplifiers start to be biased in the saturation region for a given input voltage if amplifier gains are increased from A_n. Let's assume that only the k-th amplifier output is saturated at a digital value V_k. The governing equations for the new network condition are

$$C_i \frac{du_i(t)}{dt} + T_i u_i(t) = \sum_{j \neq i,k;j=1}^{n} T_{ij} v_j(t) + I_i(t) + T_{ik} V_k \quad \text{for any } i \neq k. \quad (3.16)$$

The corresponding system matrix can be formed with the k-th column and the k-th row being deleted from (3.7). Therefore, the lowest amplifier gain which makes the network stay in the positive feedback mode is determined by

$$A_{n-1} = \max\left\{ \frac{T_i}{\sum_{j \neq i,k;\ j=1}^{n} |T_{ij}|} \quad \text{for } 1 \leq i \leq n \right\}. \quad (3.17)$$

The same T_i is used in (3.15) and (3.17). Notice that the amplifier gain for the positive-feedback state increases as the number of linear-region amplifiers decreases. The worst case is when only two amplifiers operate in the linear region. Let's assume that the i-th and k-th amplifiers operate in the linear

region. The system matrix $\mathbf{B}_{ik}$ can be expressed as

$$\mathbf{B}_{ik} = \begin{bmatrix} -\dfrac{sC_i + T_i}{A_i} & T_{ik} \\ T_{ki} & -\dfrac{sC_k + T_k}{A_k} \end{bmatrix} . \tag{3.18}$$

Since the resistive network $\{ T_{ij} \}$ is symmetrical, $T_{ik} \times T_{ki}$ is always positive. The maximum amplifier gain for two linear-region amplifiers is

$$A_2 = \max \left\{ \sqrt{\frac{T_i T_k}{T_{ik} T_{ki}}} \right\} \quad \text{for every } i \text{ and } k \text{ with } i \neq k \ . \tag{3.19}$$

When the amplifier gain is increased beyond A_2, only one amplifier will operate in the linear region. Even though the amplifier output is an analog value, the digital bit can be easily decided using the middle level of amplifier output swing as a reference. The logical state of the amplifier biased in the linear region can then be determined.

The amplifier gain during the hardware annealing process should start from a value smaller than A_n and stop at a value larger than A_2. In a similar way, the annealing temperature range for the Boltzmann machine using the update rule of (3.2) is

$$\frac{1}{4 A_2} \leq T \leq \frac{1}{4 A_n} . \tag{3.20}$$

Since the updating process in the electronic network annealing can be done in a continuous fashion, its operation speed is much faster than that for the software computation.

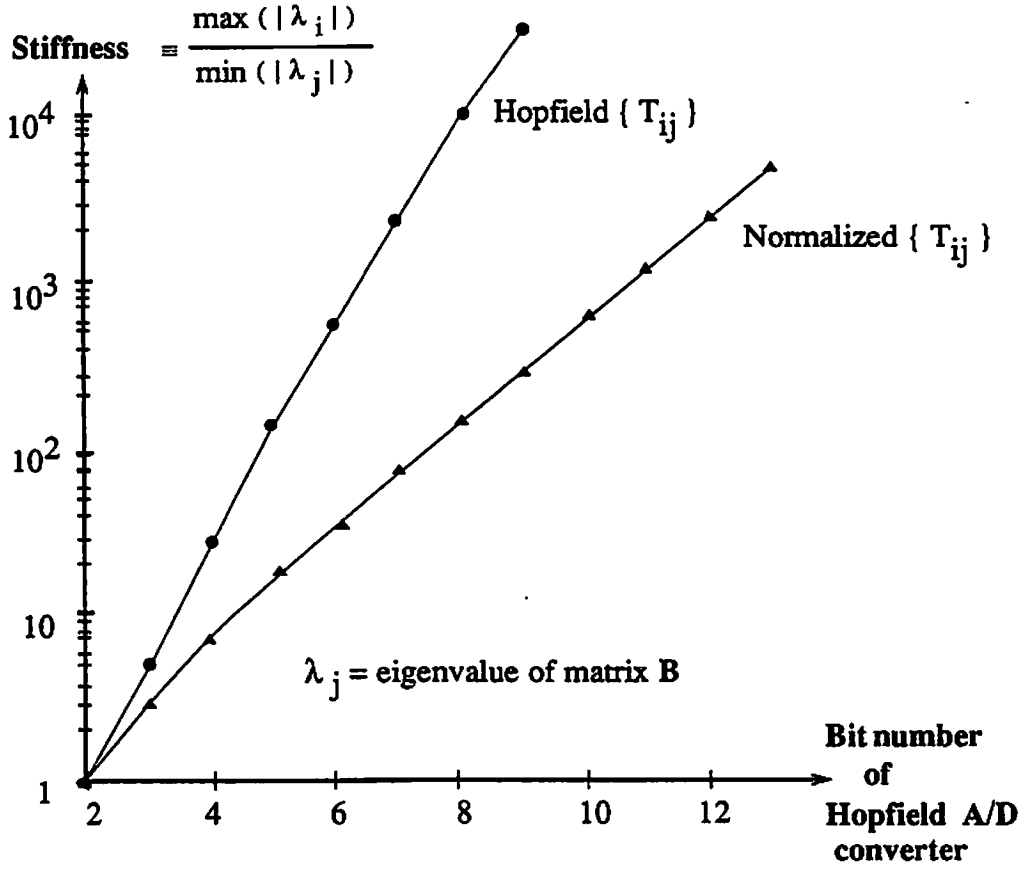


Fig. 3.4 Stiffness of the system function of a neural-based A/D converter.

3.3. Application to the Neural-Based A/D Converter

The neural-based A/D converter is a good example to examine the effectiveness of the hardware annealing because the global minimum at any analog input voltage is well-defined. In addition, the synapse matrix is very stiff as shown in Fig. 3.4. Due to the high stiffness, the convergence at a given temperature takes a very long computational time in the software computation. For simulated annealing, the temperature takes discrete values and changes by a small amount. Thus, the total software computation time to apply the annealing technique is extremely long. On the contrary, the updating of the hardware annealing process can be done continuously as inherent to the network operation. The computational time for hardware annealing is usually many orders of magnitude smaller than that for the software exercise.

3.3.1. Neuron Gain Requirement

The maximum neuron gain is determined when only two amplifiers operate in the high gain region and the others are stuck at logical values. With the maximum gain, one neuron output is saturated but the other neuron can still operate in high gain region. However, the neuron output can be classified into a logical value. Since the synapse weighting $\{T_{ij}\}$ of Hopfield neural-based A/D converter increases with i and j, the maximum gain can be obtained when the i and k are 1 and 2 in (3.19). When the number of bits n is very large, the amplifier gain A_2 is given as

$$A_2 = 2^{n-0.5} . \tag{3.21}$$

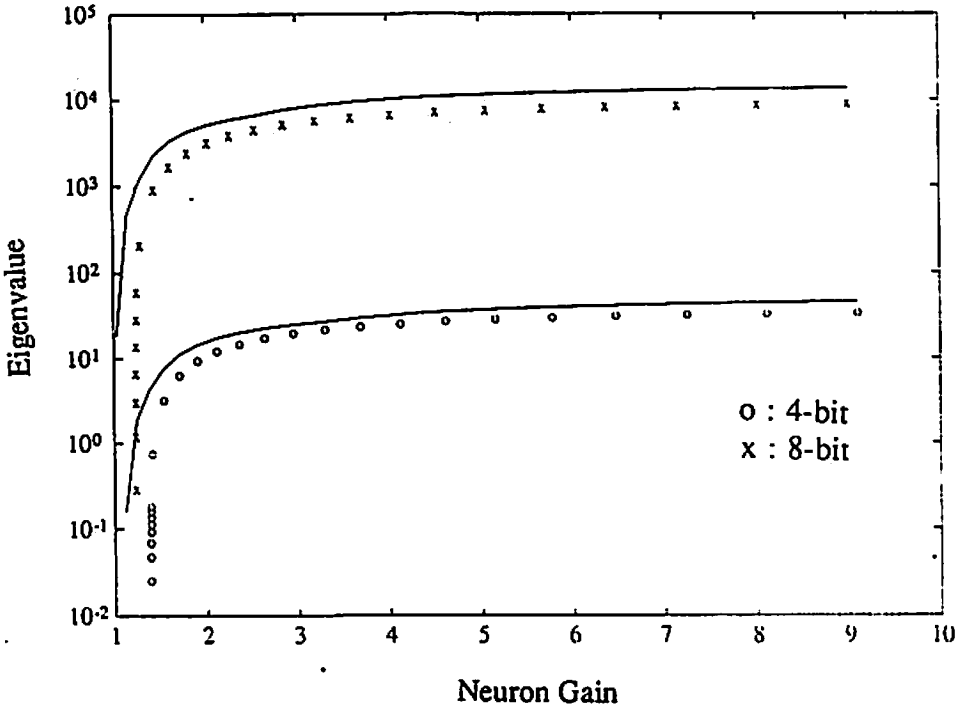


Fig. 3.5 Trajectory of the largest eigenvalue of Hopfield A/D converter against neuron gain.

For normalized synapse weightings in [9], the maximum neuron gain A_2^{nor} is given as

$$A_2^{nor} = 2^{n-1} . \tag{3.22}$$

Notice that the gain for both cases increases exponentially with n.

Since the formula of the minimum neuron gain to obtain the successful annealing results is derived from the assumption that the eigenvalue is located on the boundary Gerschgorin circle, the value in (3.15) is the lower limit. The actual value of the minimum gain is usually larger than 1.0. Figure 3.5 shows the trajectory of the maximum eigenvalue of Hopfield neural-based A/D converter [10]. When the amplifier gain decreases below 2, the eigenvalue quickly decreases. On the other hand, the maximum eigenvalue is saturated to a certain value when the amplifier gain is large. The simulation results show that Eq. (3.15) gives a good estimation to the lowest amplifier gain.

Figure 3.6 shows the neural-based A/D converter output as a function of the amplifier gain. In this SPICE circuit simulation, a neuron consists of a simple CMOS operational amplifier with its gain controlled by an MOS transistor at the amplifier input. A triangular voltage waveform was applied to the gate terminal of the controlling transistor to adjust the amplifier gain. Here, the analog input voltage is 3.0V. For the 8-bit case, the A/D converter output starts to change when the amplifier gain is about 46 dB. In comparison, the theoretically calculated amplifier gain from (3.19) is 45.13 dB. The amplifier gain A_2 increases exponentially with the number of the A/D converter bits.

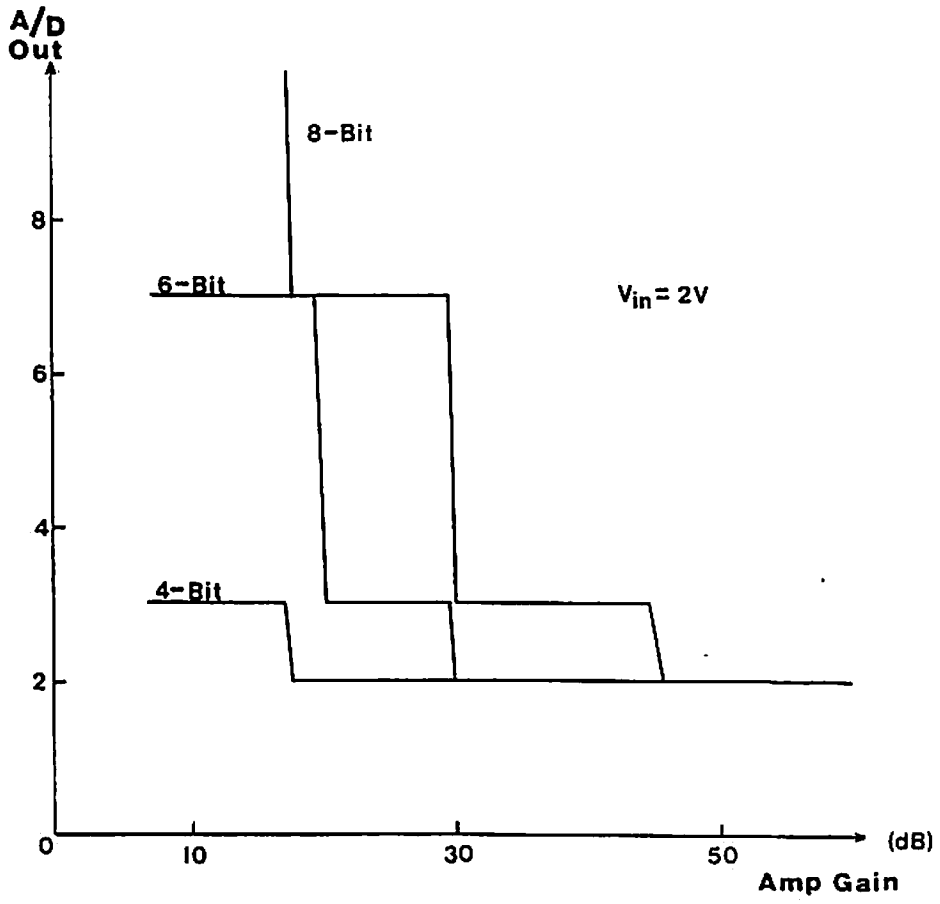


Fig. 3.6 Neural-based A/D converter outputs versus amplfier gain.

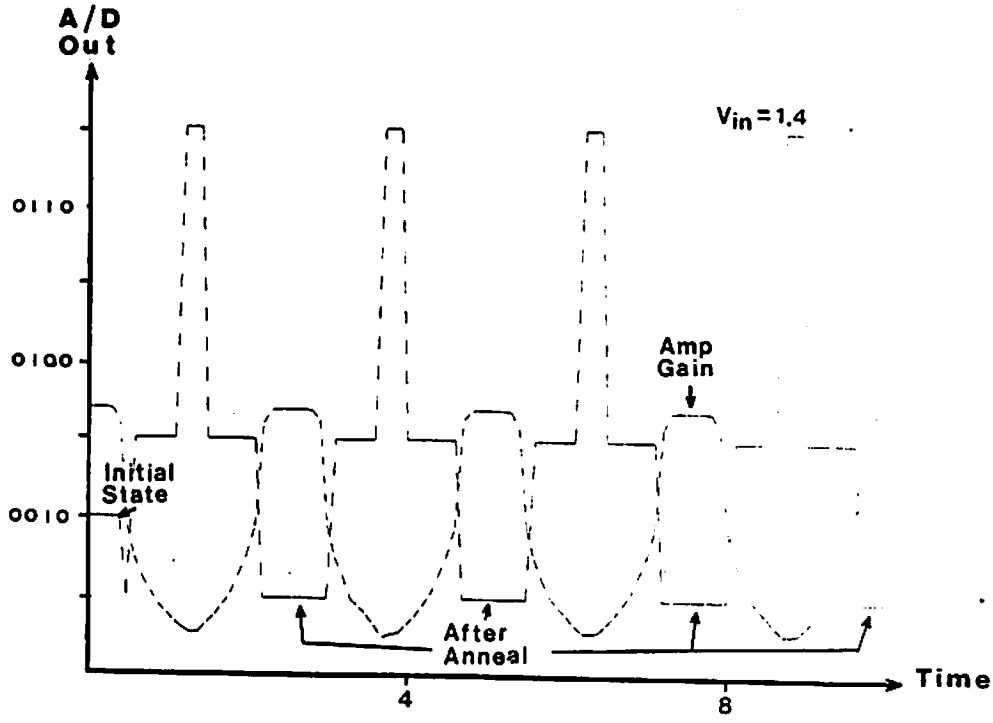
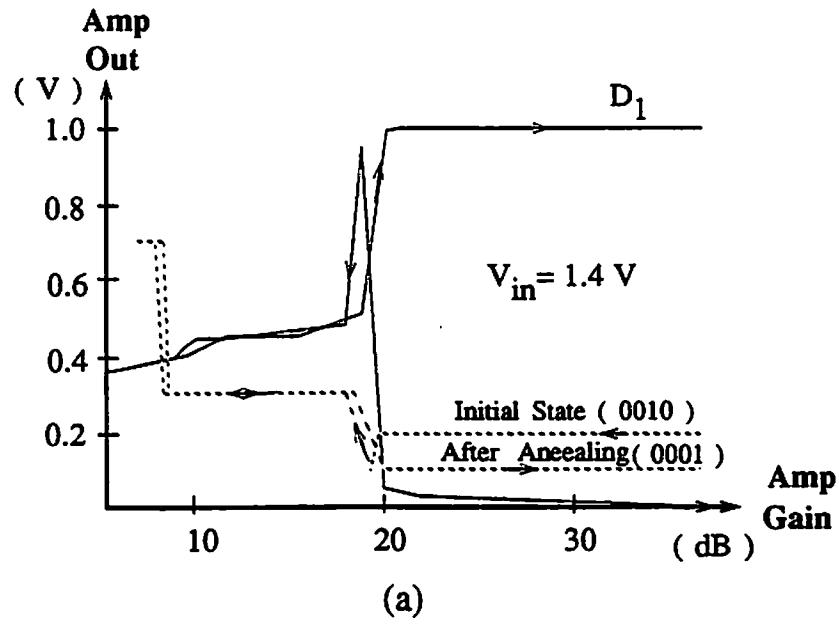
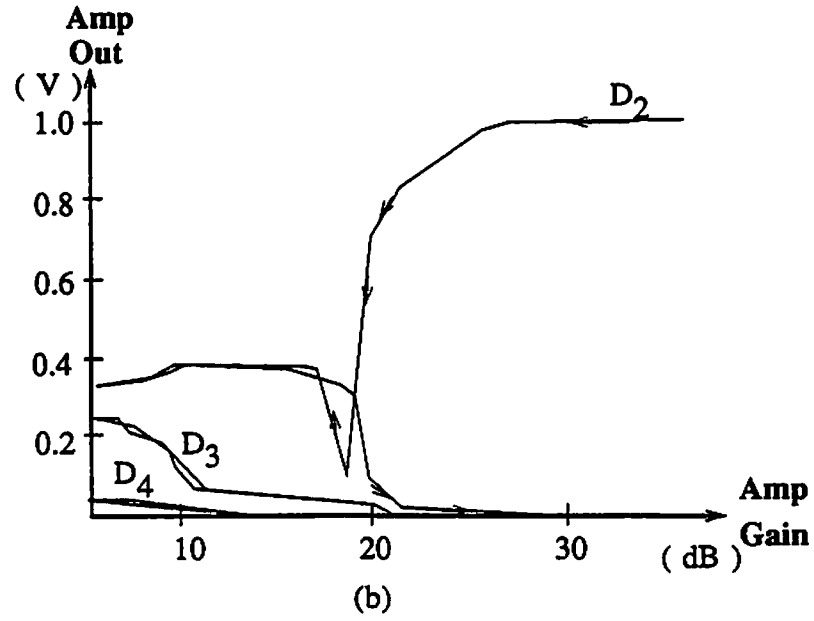


Fig. 3.7 SPICE simulation results of an 4-bit Hopfield neural-based A/D converter in time domain analysis with $V_{in} = 1.4V$.



(a)



(b)

Fig. 3.8 Trajectories of the A/D converter output versus the amplifier gain.

(a) Amplifier D_1.

(b) Amplifier D_2, D_3, and D_4.

A 4-bit Hopfield neural-based A/D converter was used to monitor each amplifier behavior. The A/D converter output is reconstructed with a D/A converter. The global minimum is always reached after one annealing cycle as shown in Fig. 3.7. Here, the initial network state is set to (0010) which is one of the local minima corresponding to $V_{in} = 1.4\ V$. Figure 3.8 shows the dynamics of the A/D converter with respect to the amplifier gain. As the amplifier gain decreases, the states of amplifier outputs V_1, and V_4 are flipped. The amplifier outputs remain at these states even when the high voltage gain is restored.

3.3.2. Relaxed Gain Requirements Using Modified Synapse Weightings

The large amplifier gain and synapse weighting ratio for a high-bit neural-based A/D converter can be dramatically reduced by using the weighted voltages instead of the weighted synapses. As shown in Fig. 3.9, the neuron output voltages and reference voltages have the following voltage ranges,

$$V_{Oi} = [\ 0V, -2^{i-1}V\ ] \tag{3.23}$$

and

$$V_{Ri} = -2^{i-2}\ . \tag{3.24}$$

Here, the voltage sources for the neuron outputs and reference voltages can be generated from a R-2R ladder network. In the direct-R implementation, the large ratio of condutances, which is extremely difficult to obtain in VLSI technologies, is used. On the other hand, only unity-ratioed conductance and binary weighted voltage sources are used in this modified structure. Notice that the

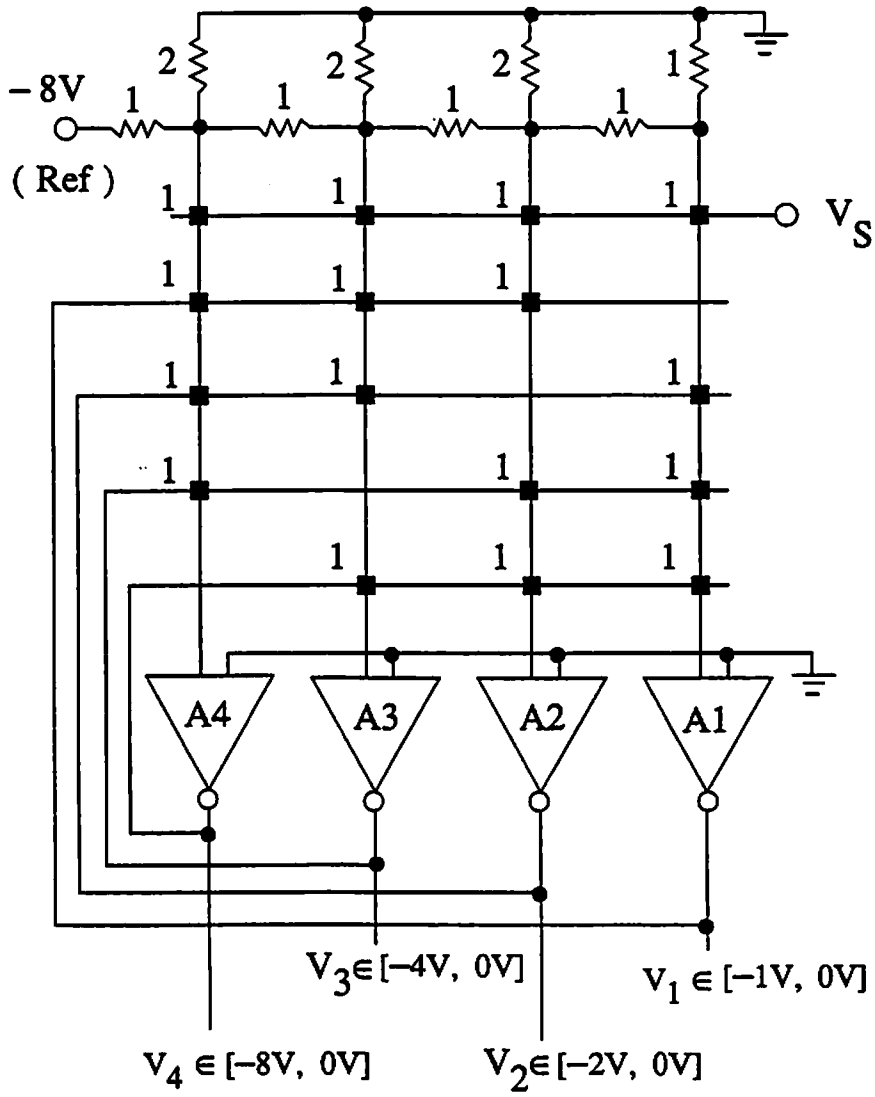


Fig. 3.9 A 4-bit neural-based A/D converter with unity synapse weighting.

large voltage ratio can be easily obtained in VLSI technologies.

Using this structure, neuron input voltage u_i is determined by

$$u_i = \frac{V_{Oi} + V_{Ri} + V_S}{n + 1} . \tag{3.25}$$

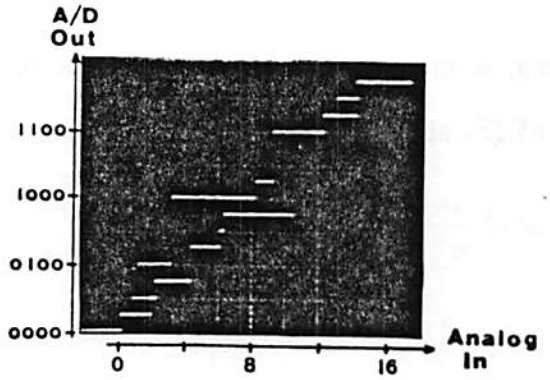
Since the synapse matrix is unity matrix except the diagonal terms, the amplifier gains A_n in (3.15) and A_2 in (3.19) become

$$\hat{A}_n = \frac{n + 1}{n} \tag{3.26}$$

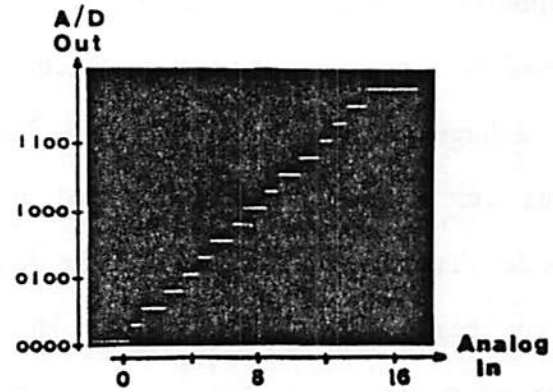
and

$$\hat{A}_2 = n + 1 . \tag{3.27}$$

In the hardware experiment, a 4-bit neural-based A/D converter was constructed using standard integrated-circuit parts. The amplifier gain is controlled with a junction-FET operating in the negative feedback loop of the amplifier. Since we used p-channel depletion-mode junction-FETs, a high control signal level makes a large amplifier gain. The 4-bit Hopfield neural-based A/D converter outputs were sampled and reconstructed by a D/A converter after every annealing cycle. Transfer characteristics of the A/D converter are shown in Fig. 3.10. The nonlinearity and hysteresis of the Hopfield A/D converter is caused by local minima in the energy function. By applying the electronic annealing technique, the global minimum is obtained. Figure 3.11 shows the time-domain response of the A/D converter with electronic annealing. Two analog input levels are applied. When the amplifier gain is high and $V_{in} = 3.3\ V$, the local minimum (1000) is reached. By applying one annealing cycle, the global



(a)



(b)

Fig. 3.10 Transfer characteristics of a 4-bit Hopfield A/D converter.
(a) Without hardware annealing.
(b) With hardware annealing.

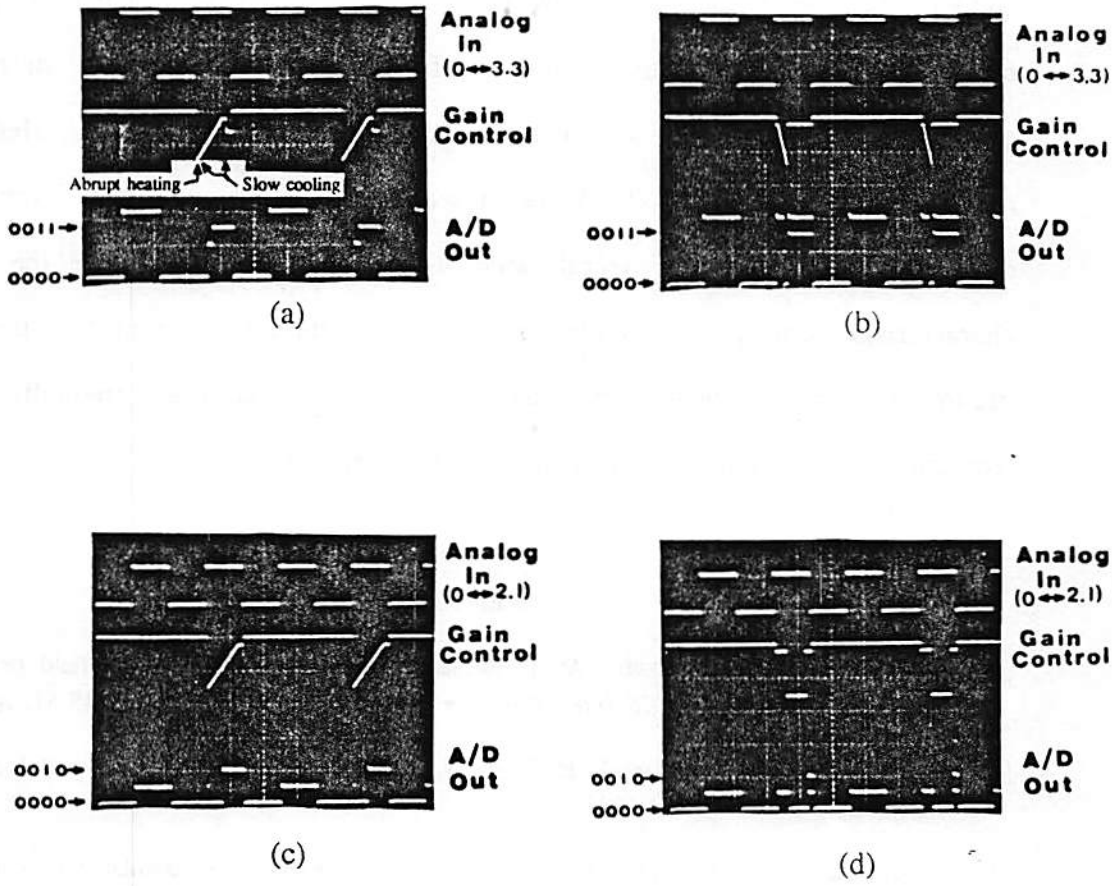


Fig. 3.11 Different annealing styles.

(a) Abrupt heating and slow cooling with $V_{in} = 0V$ and $3.3V$.

(b) Slow heating and abrupt cooling with $V_{in} = 0V$ and $3.3V$.

(c) Abrupt heating and slow cooling with $V_{in} = 0V$ and $2.1V$.

(e) Abrupt heating and abrupt cooling with $V_{in} = 0V$ and $2.1V$.

minimum (0011) is found as shown in Fig. 3.11(a). Other annealing styles have been investigated. With abruptly increasing amplifier gain, the outputs could be (0011) and (0100). This phenomenon is known as the Quenching Effect in condensed-matter physics [11]. When temperature of the heat bath is lowered instantaneously, particles in a solid are frozen into the metastable states. The other case shown in Fig. 3.11 (c) and (d) gives the same results. Here, local minimum (0001) at $V_{in} = 2.1\ V$ is caused by device mismatches. Even though device mismatches in the neural-based A/D converter circuit make the transfer characteristics change, the hardware annealing technique help to find the global minimum. This hardware experiment demonstrates that only gradually cooling temperature is effective to reach the global minimum.

References

[1] B. W. Lee and B. J. Sheu, "An investigation on local minima of Hopfield network for optimization circuits," *IEEE Inter. Conf. on Neural Networks*, vol. 1, pp. 45-51, July 1988.

[2] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., 1987.

[3] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi, "Optimization by simulated annealing," *Science* vol. 220, no. 4598, pp. 671-680, May 1983.

[4] E. H. L. Aarts and P. J. M. van Laarhoven, "A new polynomial-time cooling schedule," *Proc. IEEE Int. Conf. on Computer-Aided Design*, pp. 206-208, Nov. 1985.

[5] R. A. Rutenbar, "Simulated Annealing Algorithms: An Overview," *IEEE Circuits and Devices Magazine*, vol. 5, no. 1, pp. 19-26, Jan. 1989.

[6] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721-741, Nov. 1984.

[7] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, *Parallel Distributed Processing, vol. 1*, the MIT Press, pp. 282-317, 1986.

[8] Bang W. Lee and Bing J. Sheu, "Hardware simulated annealing in electronic neural networks," *IEEE Trans. on Circuits and Systems*, to appear in 1990.

[9] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Natl. Acad., Sci. U.S.A.*, vol. 81, pp. 3088 -3092, May 1984.

[10] G. D. Smith, *Numerical solution of partial differential equations: finite difference methods*, Oxford University Press, pp. 60-63, 1985.

[11] D. D. Pollock, *Physical properties of materials for engineers*, CRC Press, pp. 14-18, 1982.

Chapter 4

Programmable Synapses and Gain-Adjustable Neurons

A significant progress has been made in using fully analog circuit design techniques for early vision processing. The sensory neural networks differ from others in highly localized information processing [1]. On the other hand, artificial neural networks for digital signal and image processing usually require heavy interconnections between the neurons. Several experimental neural hardware design, which typically function as direct accelerators for software computation, have been reported [2]. These hardware accelerators consist of main digital processors, mathematical co-processors, and associated memories. In most cases, the rich properties of neural networks associated with massively parallel processing using analog neurons and synapses to process digital pictures have not been fully explored yet.

The endurability for long-term storage and the reprogrammability for adaptive functions of neural networks are necessary properties for VLSI synapses. The amorphous-Si approach from AT&T Bell Labs. has only one-time programming capability, which is similar to the use of read-only memories. On the other hand, the charge storage mechanisms used in DRAMs and EEPROMs can be utilized to retain the synapse weighting information. Electronic charge in the DRAM cell are stored at the dynamically-refreshed capacitor while the electric power is on. The electronic charge in EEPROM cell is stored permanently in the floating gate. Some initial investigation results on the design of programmable synapses using the two storage mechanisms have been reported [3,4].

To implement a stochastic neural architecture, a special randomizing circuitry is necessary for the VLSI neural networks. The special circuitry realized in a digital format [5] generates a random pulse signals, which is added to the synapse weighting. The random signal generator usually occupies large silicon area and decreases the neural information processing time. The hardware annealing technique, which needs gain-adjustable neurons, is a simple way to realize the stochastic process. To make the neural computing hardware more powerful, the design and implementation of gain-adjustable neurons and electrically programmable synapses represent a fundamentally important step.

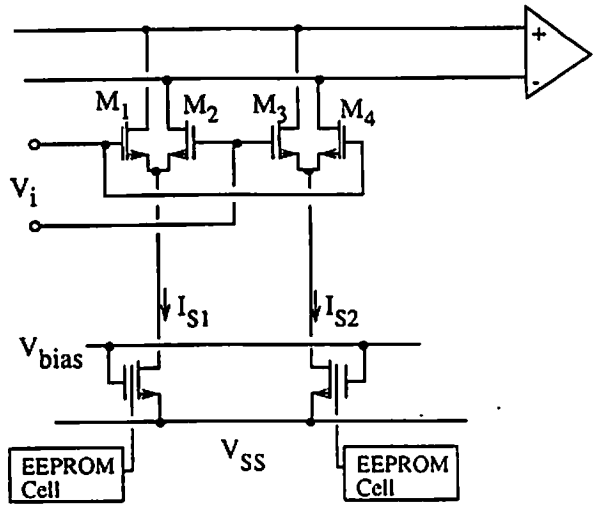
4.1. Programmable Neural Chips

The 64-neuron VLSI chip from Intel [4] includes a fully programmable synapse with EEPROM cells as shown in Fig. 4.1. The synapse weighting is coded with the bias currents of the analog multiplier. The input voltage V_i is applied to two differential pairs. Since the EEPROM-injected electronic charge at the floating gate can change threshold voltage of the transistor in the current mirror, bias currents of the analog multiplier are programmable. Figure 4.1(b) shows I-V characteristics of the synapse cell for various bias current values.

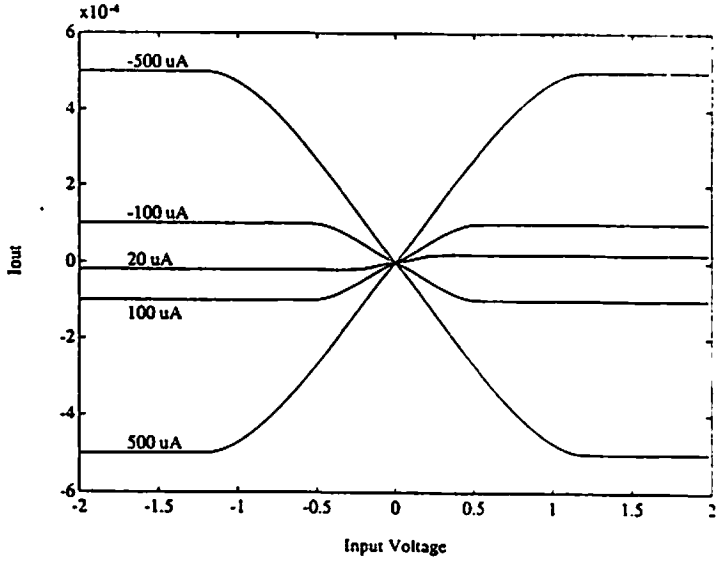
When V_i is very small, the equivalent conductance T_{eq} can be obtained as

$$T_{eq} \equiv \frac{I_{out}}{V_i} = \sqrt{\mu C_{ox} \frac{W}{L} I_{S1}} - \sqrt{\mu C_{ox} \frac{W}{L} I_{S2}} \,. \tag{4.1}$$

Here, I_{S1} and I_{S2} are the programmed currents. The conductance value is determined by the programmed differential current. However, linearity of the conductance is limited by



(a)



(b)

Fig. 4.1 A programmable synapse cell in [4].
(a) Circuit schematic.
(b) Simulated transfer characteristics.

$$V_{IR} = \sqrt{\frac{2 \cdot \min(I_{S1}, I_{S2})}{\mu C_{ox} W/L}} \tag{4.2}$$

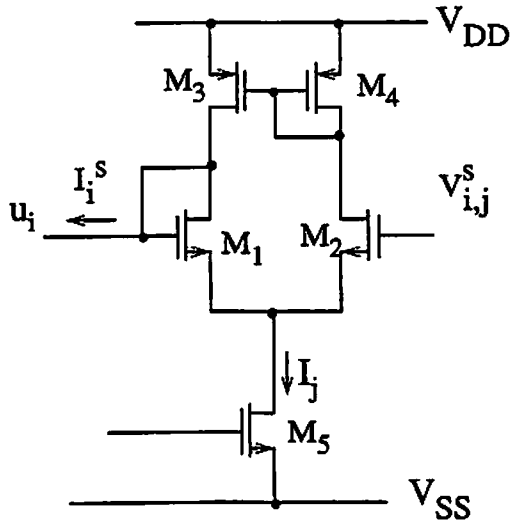
For a proper input voltage range, small device aspect ratio W/L and large programmed currents are required. In addition, the large dynamic range of the conductance value can not be obtained because the conductance is determined by the square-root of the programmed current.

A programmable synapse with very large dynamic range is shown in Fig. 4.2. The transconductance amplifier consisting of M_1 - M_4 is connected as a buffer. The information of a synapse weighting is stored at the capacitance associated with the controlling gate in the voltage format. The synapse output current I_{ij}^s of the transconductance amplifier is determined by a function of the differential input voltage and the bias current I_j. When the voltage difference between u_i and V_{ij}^s is small and the logic-1 input is applied, I_{ij}^s is given as

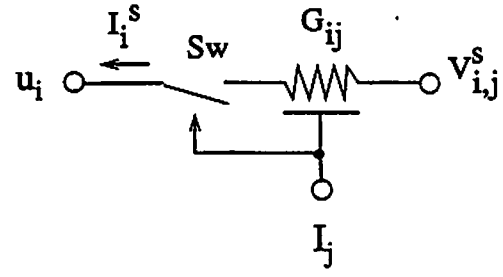
$$I_{ij}^s = \sqrt{\beta I_j}\ (V_{ij}^s - u_i)\ . \tag{4.3}$$

Here, β is the transconductance coefficient of M_1 and M_2, u_i is the input voltage of the i-th output neuron, and V_{ij}^s is the programmed voltage for the synapse between the j-th input neuron and i-th output neuron. Voltage V_{ij}^s is programmed at the capacitance when selected through the column decoder and the row decoder.

The programmable synapse operates as a three-terminal transconductor. The third terminal is used to change the effective conductance. For binary input signals, the equivalent model of the programmable synapse is shown in Fig. 4.2(b). Instead of programming actual conductance values, the equivalent current



(a)



(b)

Fig. 4.2 Transconductance amplifier as the core for the programmable synapse cell.
(a) Circuit Schematic.
(b) Equivalent model for binary inputs.

summation can be obtained by using the programming voltage V_{ij}^{s} with the fixed conductance G_{ij}. Here, G_{ij} is given as

$$G_{ij} = \sqrt{\beta I_j} \ . \tag{4.4}$$

Notice that current summing function of an artificial neuron can be realized by using either constant input range with programmed synapse weightings or programmed input range with constant synapse weighting.

Figure 4.3 shows the synapse characteristics simulated in SPICE3 and HSPICE [6,7] circuit simulators. The synapse output current is shifted horizontally by the synapse weighting voltage. Since the output neuron voltage is determined by u_i with the threshold at 0 V, the equivalent synapse weighting T_{ij}^{eq} is shown in Fig. 4.3(b). The dynamic range voltage V_{lmt} of T_{ij}^{eq} is limited by the bias current,

$$V_{lmt} = \sqrt{\frac{2I_j}{\beta}}. \tag{4.5}$$

When u_i is very small, T_{ij}^{eq} is given as

$$T_{ij}^{eq} = \begin{cases} \sqrt{\beta I_j} V_{ij}^{s} & \text{when } V_j = \text{'logical 1'} \\ 0 & \text{when } V_j = \text{'logical 0'} \ . \end{cases} \tag{4.6}$$

Notice that the relative value of T_{ij}^{eq} is important in the VLSI neural circuits, because u_i is determined by the ratio of synapse weightings. The maximum value of T_{ij}^{eq} is determined by V_{lmt}, while the minimum value of T_{ij}^{eq} is determined by the resolution of transconductance amplifiers. Thus, synapse dynamic range can be increased with large $I^{\max}$ and a small aspect ratio of transistors M_1 and M_2 in the transconductance amplifier.

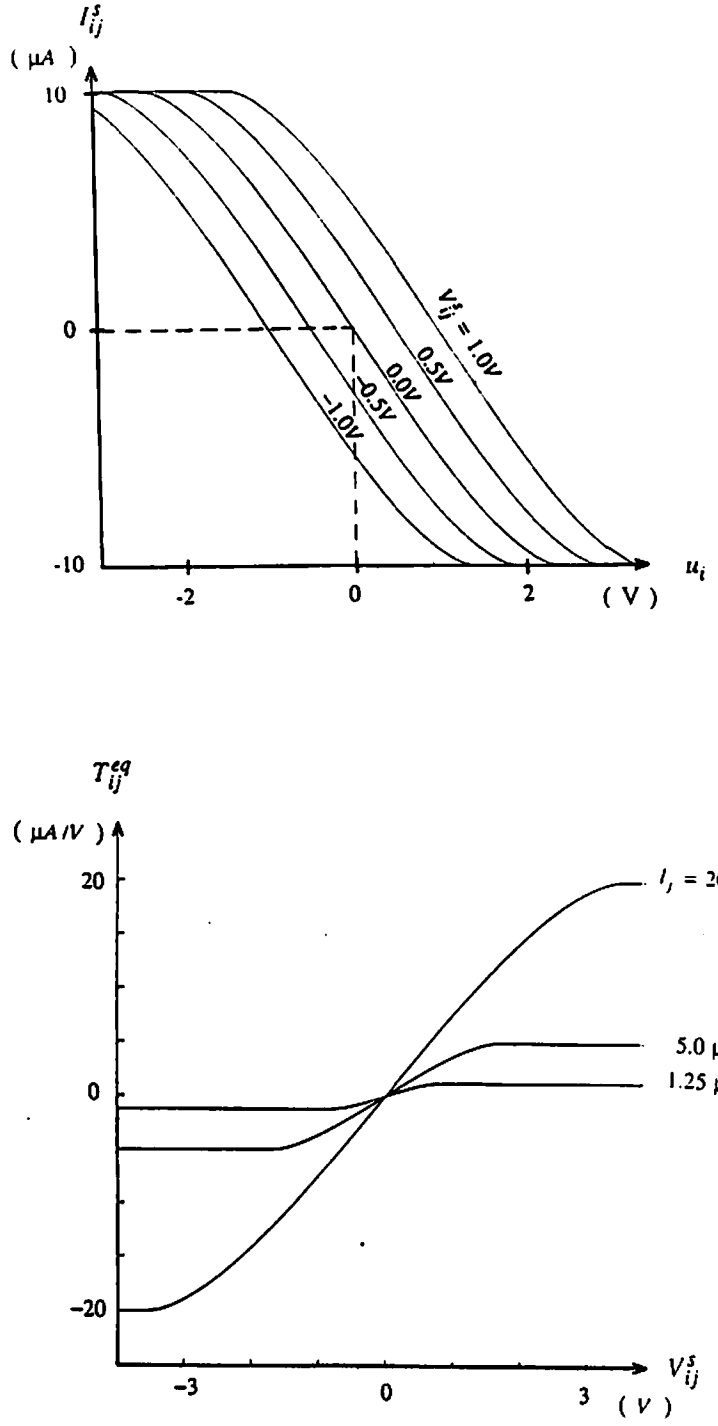


Fig. 4.3 Simulated synapse characteristics.
(a) Synapse output current versus neuron input voltage.
(b) Equivalent synapse conductance versus programmed synapse voltage.

The detailed circuit schematic including the programmable synapse, input neuron, and gain-adjustable output neuron is shown in Fig. 4.4. The input neuron consisting of M_8 - M_{10} steers the dynamic-range control current I^{max} into the current mirror and produces the bias voltage for multiple synapse cells. Each synapse output current is summed up and converted to the voltage format at the input terminal of the i-th output neuron. The output neuron, which is a basic 2-stage amplifier with externally adjustable gain, amplifies the converted voltage.

By assuming transconductance amplifiers in synapses $\{T_{ij};\ j = 1, 2, ..., n\}$ operating in the high gain region, the input voltage u_i is governed by the following equation,

$$\left[\sum_{j=1}^{n} G_{ij}\right] u_i = \sum_{j=1}^{n} I_{ij}^{s} \ . \tag{4.7}$$

Notice that this circuit realizes the synapse weighting in the current format, instead of the direct-R format [8,9]. The minimum amplifier gain for this programmable synapse is determined by the number of synapse. In the case of direct-R implementation, u_i is given as

$$u_i = \frac{\sum_{j=1}^{n} T_{ij} V_j}{\sum_{j=1}^{n} |T_{ij}|} \ . \tag{4.8}$$

Since u_i for the direct-R implementation is divided by the summation of synapse weightings, the minimum amplifier gain for a proper circuit operation is determined by the ratio of the synapse summation and minimum synapse weight [10].

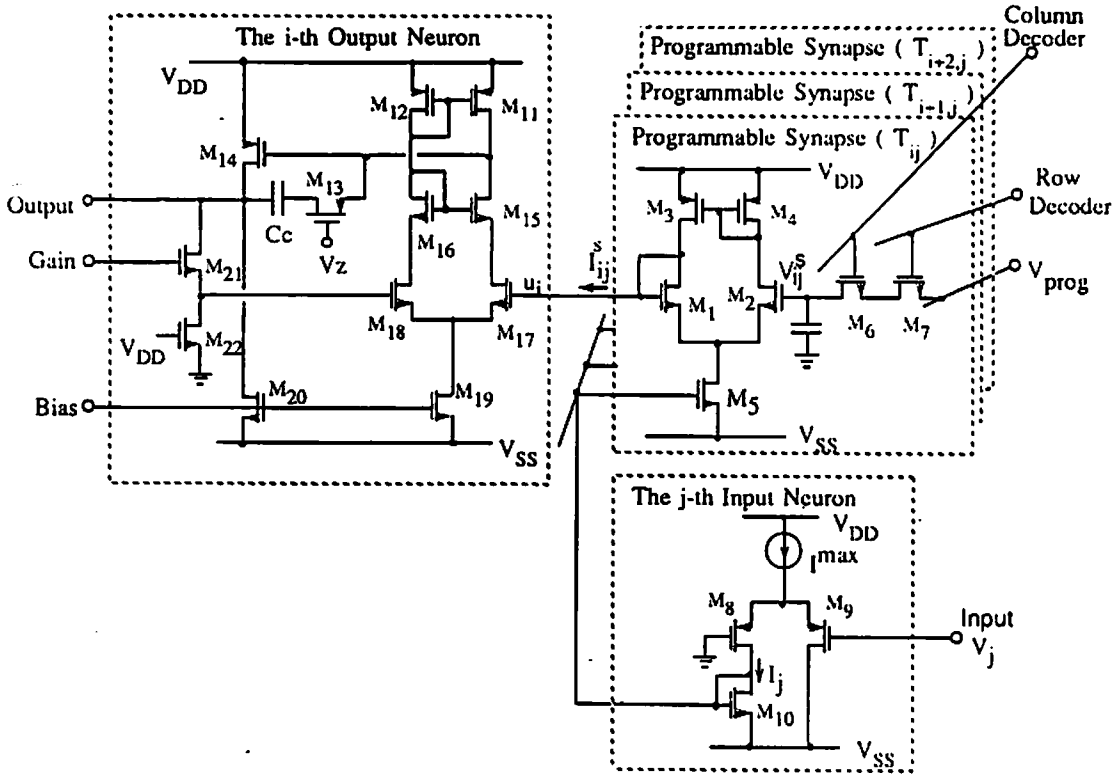


Fig. 4.4 Circuit schematic of the programmable synapse and related neurons.

Table 4.1. Device geometries of circuit in Fig. 4.4

(unit : μm)

Transistor	W / L	Transistor	W / L
M_1	3/39	M_{12}	6/6
M_2	3/39	M_{13}	3/18
M_3	4/4	M_{14}	6/6
M_4	4/4	M_{15}	6/3
M_5	5/3	M_{16}	6/3
M_6	4/2	M_{17}	18/2
M_7	4/2	M_{18}	18/2
M_8	5/3	M_{19}	6/4
M_9	5/3	M_{20}	6/4
M_{10}	5/3	M_{21}	54/3
M_{11}	6/6	M_{22}	2/80

Capacitor $C_c = 60 \times 60$

Notice that the input conductance of the i-th neuron which is usually negligible in a CMOS amplifier does not appear in (4.7).

The input neuron operates as a buffer which converts input neuron voltage V_j into current I_j. Notice that the actual output voltage range of neuron amplifier can be [V_{SS}, V_{DD}] while the output current of input neuron is [0, I^{max}]. For an analog input value of V_j, the input neuron adds a certain voltage gain. The transfer function of the input neuron can be effectively lumped to that of the output neuron in the previous layer of multi-layer neural networks. In the Hopfield neural network case, the transfer function of an input neuron is grouped into the output neuron due to the feedback connection. This current driving scheme to connect the input neuron with programmable synapses saves the silicon area tremendously as compared with the dedicated impedance-matching interface [9] between an input neuron and its associated synapses in a direct-R implementation.

The neuron transfer function in software computation is usually specified as an exact mathematical equation, which can decide the neural network properties in the retrieving and learning processes. In our design, transistors M_{15} and M_{16} form an improved cascode stage to increase the voltage gain and M_{13} operates as a resistor to improve amplifier frequency characteristics. The amplifier voltage gain can be changed by externally applied gain control voltage, which adjusts the drain-to-source resistance ratio of transistors M_{21} and M_{22}. If the gate voltage of M_{21} is below transistor threshold voltage, the amplifier operates in the open-loop high-gain configuration. When the gate voltage of M_{21} increases above the threshold voltage, the closed-loop gain of the amplifier

decreases. In the extreme case when a very large gate voltage is applied to M_{21}, the amplifier effectively operates in the unity-gain configuration. The gain-adjustable amplifiers can be used to exercise hardware annealing [11], which helps the network to efficiently find the global minimum. Figure 4.4 shows die photos of the input neuron and output neuron. The input neuron and the output neuron occupy area of 40 μm x 80 μm and 110 μm x 60 μm, respectively, in the Scalable 2-μm CMOS technology from the MOSIS Service of USC/Information Sciences Institute (Tel. 213-822-1511).

4.2. Medium-Term and Long-Term Storage of Synapse Weight

In a biological system, the neuron holds the processing data temporarily through the summing and thresholding steps while the synapse weight keeps a long-term history of the network behaviors. Thus, the biological neuron and synapse are called the short-term memory and long-term memory, respectively. A similar data storage function can be achieved in the artificial VLSI neural networks. The input capacitance and the finite time delay of a neuron amplifier correspond to the short-term memory and the synapse weight storage can be equivalent to the long-term memory.

The charge storage mechanisms used in DRAMs and EEPROMs can be utilized to retain the synapse weighting information. Electronic charge in DRAM are stored temporarily at the dynamic capacitor, while charges in the EEPROM is stored permanently in a floating-gate transistor.

4.2.1. DRAM-Style Weight Storage

The information of a synapse weighting is stored at the capacitance associated with the controlling gate in the voltage format. The synapse voltage is externally programmed through the address decoder as shown in Fig. 4.4. Since the capacitor charged by V_{ij}^s is connected to transistor M_6, V_{ij}^s is discharged through diffusion-to-substrate junction. The simulated results of synapse accuracy as a function of refresh cycle are shown in Fig. 4.5 by using a simple RC model. Since the network retrieving and synapse weighting refresh can be conducted concurrently, no dead time for refreshing the synapse voltage is required. Thus, the learning process can be directly realized. In addition, extra perturbations caused by the device mismatches are also easily compensated during the learning process.

The die photo of the synapse cell and charge retention characteristic of this synapse cell is shown in Fig. 4.6. The programmable synapse cell occupies an area of 46 μm x 64 μm in the Scalable 2-μm CMOS process from the MOSIS Service. In the charge retention experiment, 72 synapse cells are tied together to minimizing the effect of external stray capacitance. The RC time constant is about 50 sec. Here, the x-axis is 10 sec/div and the y-axis is 0.2 V/div. A refreshing cycle of 0.2 sec is adequate for the 8-bit synapse accuracy. The measured synapse characteristics are shown in Fig. 4.6(c). Since 10-to-1 resistor divider configuration is used in the test circuit, the output voltage is shown in 10x reduced format. In this experiment, the input neuron voltage of all synapses are set to logic-1 and the voltages (V_{in} and ground) are applied to the gate terminals of transistor M_2 in Fig. 4.2(a). The bias currents are 1.25 μA, 5 μA,

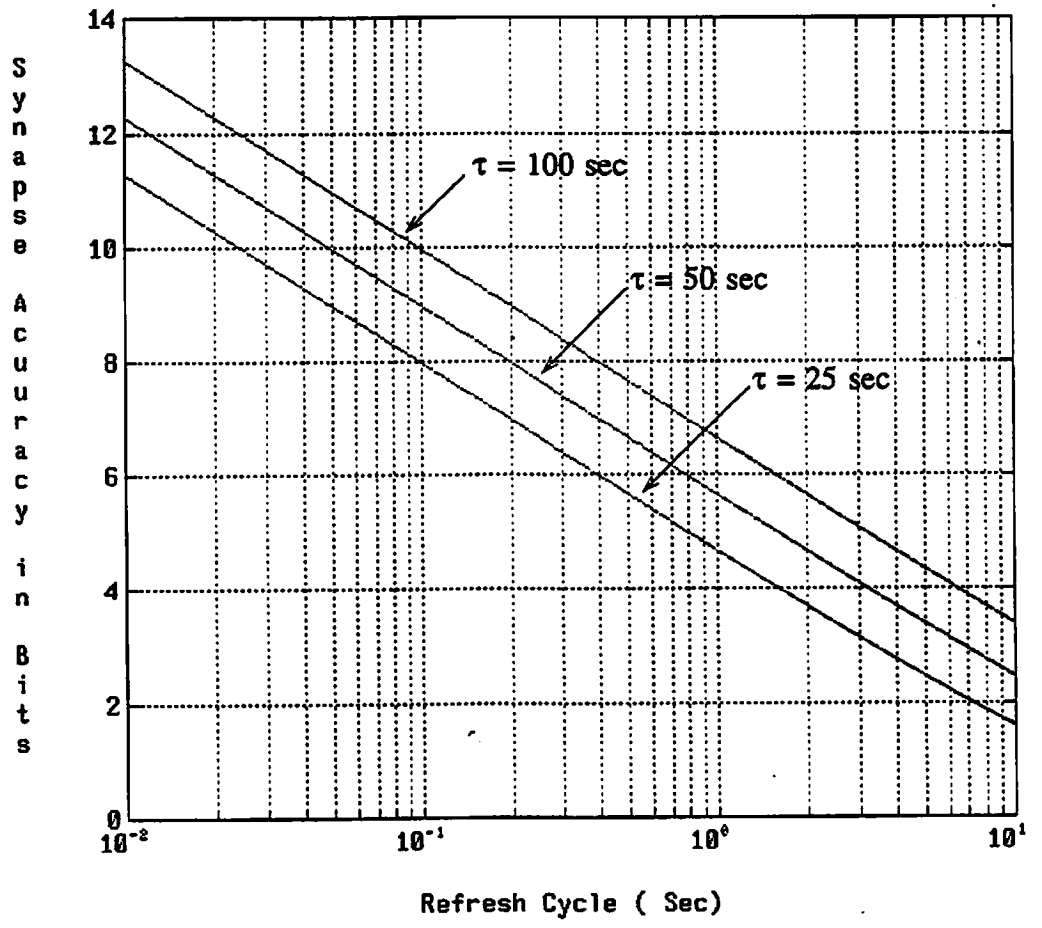
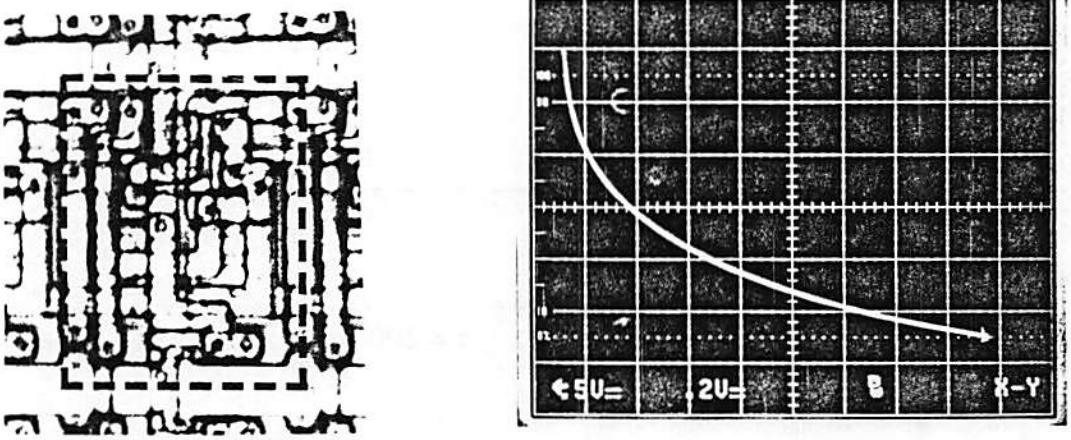
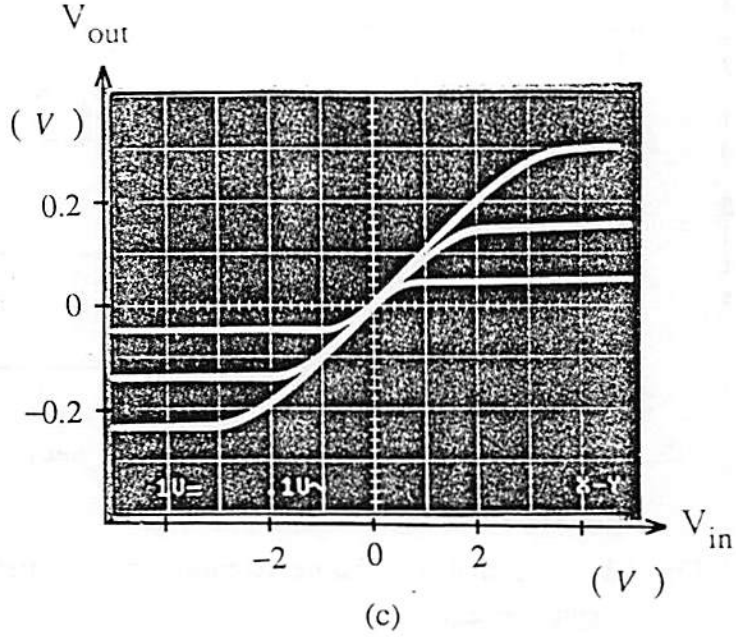


Fig. 4.5 Simulated charge retention characteristics in DRAM-style synapse cell.

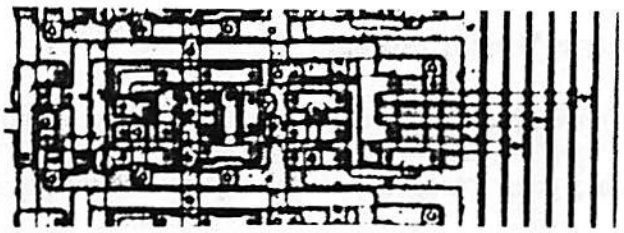


(a) (b)

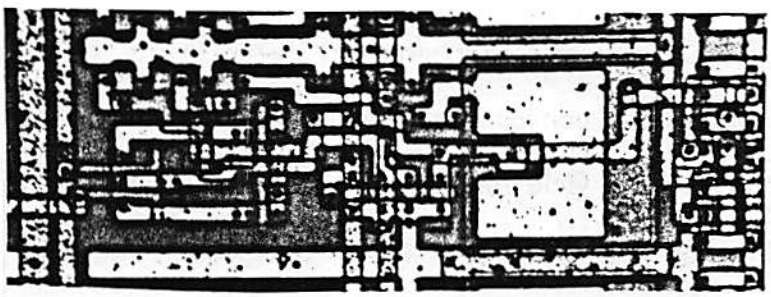


(c)

Fig. 4.6 DRAM-style synapse cell.
(a) Die photo.
(b) Measured result of charge retention test.
(c) Measured result of the synapse characteristics.



(a)



(b)

Fig. 4.7 Die photos.
(a) Input neuron.
(b) Output neuron.

and 20 μA, respectively. The input neuron and output neuron shown in Fig. 4.7 occupy an area of 90 μm x 46 μm and 140 μm x 64 μm.

4.2.2. EEPROM-Style Weight Storage

The extra refreshing circuitry using the DRAM-style storage can be saved by using the floating-gate storage approach. The usefulness of the DRAM-like storage is primarily limited by continuous charge leakage. A short refresh cycle is required to compensate the charge leakage, but the high speed clocking can induce switching noise. In the EEPROM-style storage, the synapse weight information is retained in very long-term so that the learning process of the network does not have to be repeated when the electric power of the system is lost.

The floating-gate transistor structure shown in Fig. 4.8 is used to store the synapse voltage. A conventional double-polysilicon CMOS process is used to achieve high product yield for logic ICs and good charge retention characteristics in the floating gate. The polysilicon-1 (POLY1) layer is floated and the polysilicon-2 (POLY2) electrode serves as the controlling gate. The oxide thickness between the two polysilicon layers is 55 nm. Since the bump-like area enhances the local electrical field by a factor of 4 to 5, the corresponding Fowler-Nordheim tunneling voltage [10] in the floating-gate transistor is similar to that of a conventional EEPROM device. In addition, the tunneling voltage decreases as the number of bumps increases [11]. The desired amount of electronic charge can be programmed to the floating layer with the external voltage applied to the POLY2 electrode.

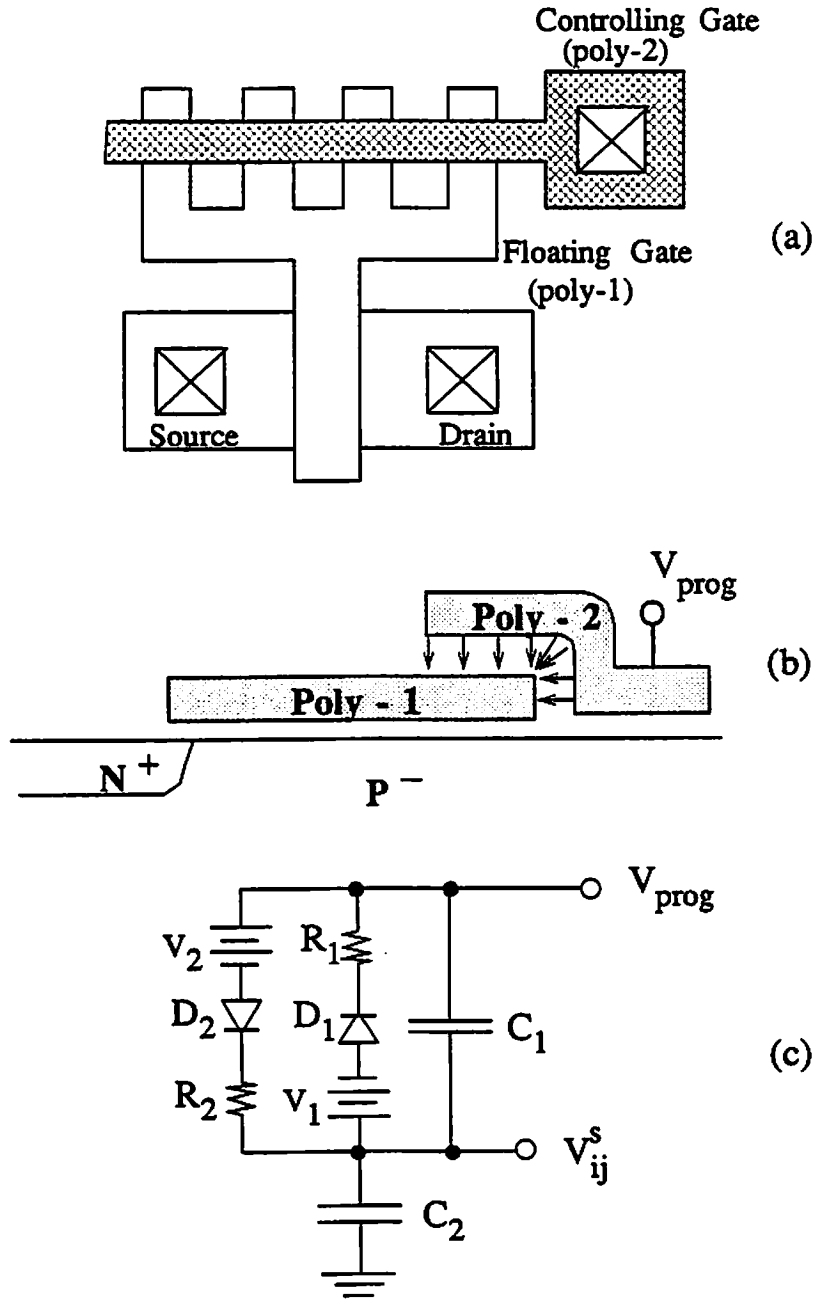


Fig. 4.8 Floating-gate transistor.

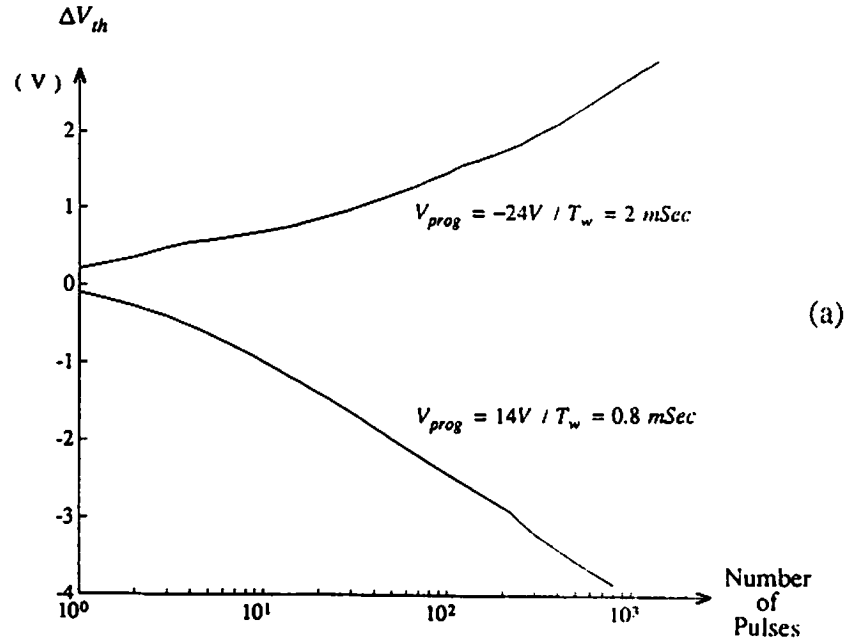
(a) Top veiw.

(b) Cross section.

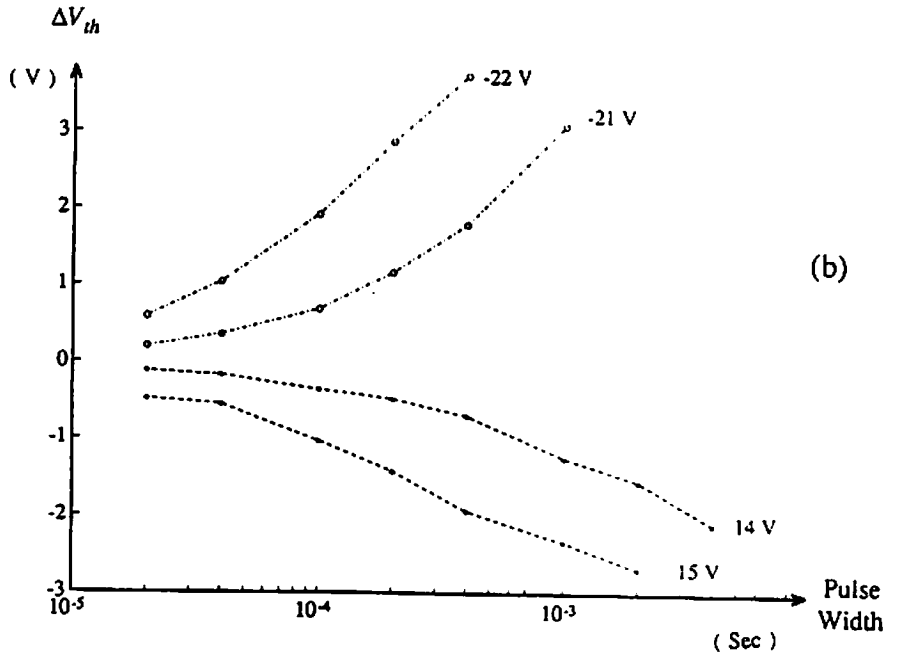
(c) Equivalent large-signal model.

When the programming voltage V_{prog} is applied to the POLY2 electrode, the initial value of V_{ij}^s is determined by the ratio of POLY2-to-POLY1 capacitance and POLY1-to-substrate capacitance. If the voltage drop across POLY2-to-POLY1 capacitance is big enough to initiate the tunneling mechanism, then V_{ij}^s is exponentially changed. The equivalent model for the double-ploy transistor is shown in Fig. 4.8(c). The stored charge can be controlled with the magnitude and pulse width of V_{prog}. For a wide range and a fine resolution of synapse weightings, a small and narrow pulse-width of V_{prog} is preferred. Notice that the network retrieving should be interrupted during synapse programming period, because the magnitude of V_{prog} is higher than normal voltage for network operation. To reduce the magnitude of the programming voltage, POLY2-to-POLY1 capacitance can be minimized with respect to POLY1-to-substrate capacitance. The allowable magnitude of V_{prog} is limited by the pn-junction breakdown voltage between the diffusion region and substrate of transistors $M_6 - M_7$ in Fig. 4.4. These transistors are part of the decoder circuitry. In the present MOSIS 2-μm CMOS technologies, the breakdown voltage is usually around 26 V [15].

In the laboratory experiment, one-bump floating-gate transistors with W = 16 μm and L = 2 μm were used. The measured oxide thickness between polysilicon layers is 55 nm and the gate oxide thickness under the POLY1 layer is 28 nm. Programming voltages with a fixed pulse width and magnitude were applied to the POLY2 electrode and the threshold voltage of the transistor was measured. Plots of threshold voltage changes in the fixed pulse width case and in the variable pulse width case are shown in Fig. 4.9(a) and (b), respectively.



(a)



(b)

Fig. 4.9 Measured results of several programming schemes.
(a) Threshold voltage change versus the number of fixed-width/magnitude pulses.
(b) Threshold voltage change versus a single pulse with variable width.

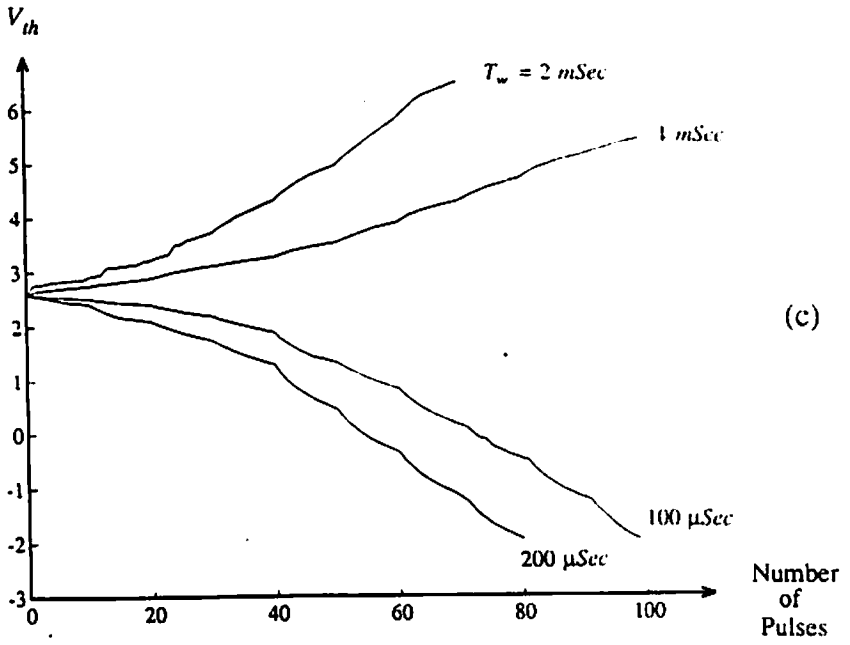


Fig. 4.9 (Continued)
(c) Threshold voltage versus the number of fixed-width pulses with programming voltage being increased from 12 V or decreased from -20 V by 0.5 V after every 10 pulses.

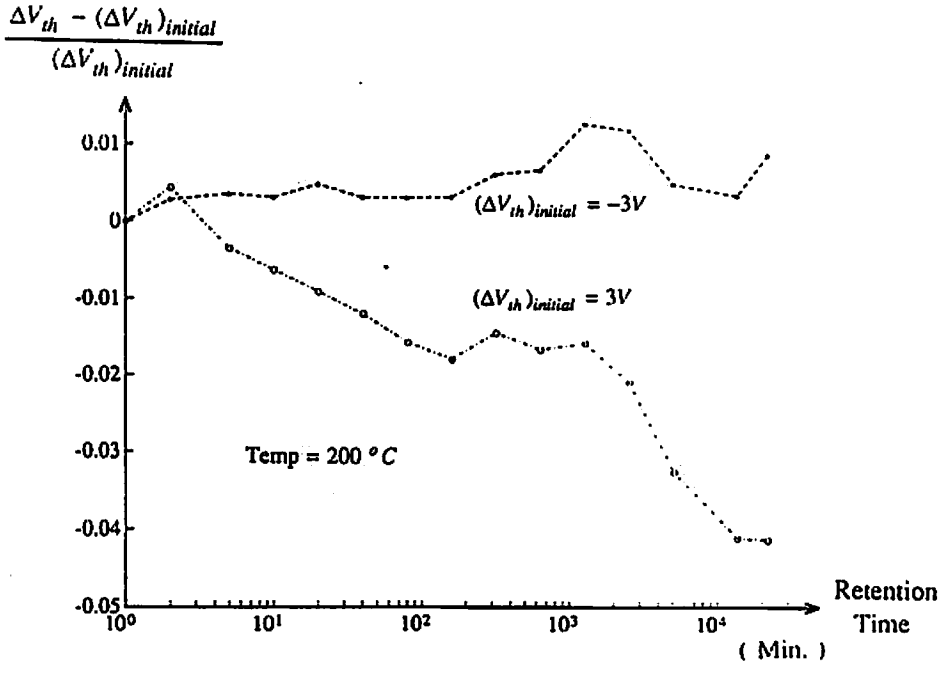
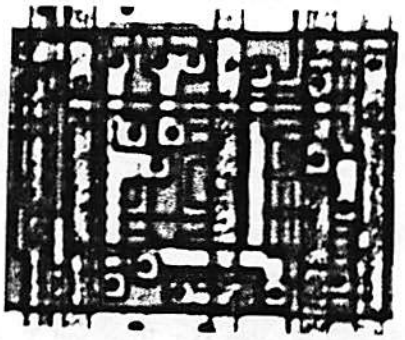


Fig. 4.10 Measured charge retention characteristics of floating-gate transistors at an elevated temperature.

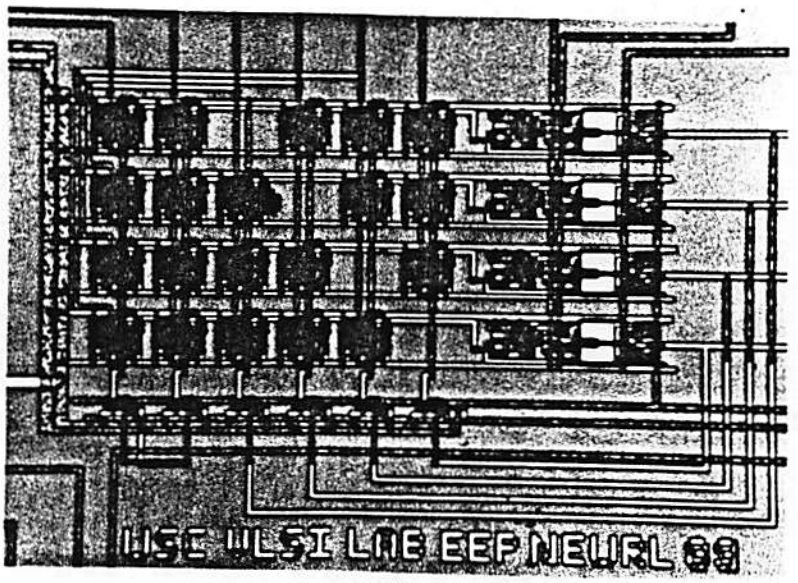
The transistor threshold voltage is changed exponentially with the programming time. Both programming time and voltage magnitude can be used as control parameters. Figure 4.9(c) shows the results of using the fixed-width pulses and increases by 0.5 V per every 10 pulses. The magnitude difference of V_{prog} for decreasing and increasing the programming voltage is caused by the MOS capacitor structure of POLY1-to-substrate capacitor. Experimental results on charge retention are shown in Fig. 4.10. The $(\delta V_{th})_{initial}$ is the amount of threshold voltage shift from the programming. Since the oxide thickness of this floating-gate transistor is much larger than that of a conventional EEPROM device, charge retention characteristics are also superior. By increasing the number of bumps and the ratio of POLY1-to-substrate capacitance to POLY2-to-POLY1 capacitance, the magnitude of V_{prog} can be reduced by a factor of more than 2 [14].

The 10 mV resolution can be achieved by single programming pulse of 100 μsec at $V_{prog} = 12\ V$ and $-20\ V$. In the case of $I^{max} = 20$ μA and ± 5 V-power supply, the maximum V_{limit} is 3.2 V and the dynamic range for synapse weighting is 50 dB. The synapse dynamic range will be increased, as the magnitude and pulse width of V_{prog} decreases. The die photos of a 4-bit neural-based A/D converter using the EEPROM-style storage cell are shown in Fig. 4.11. To increase the number of bumps, a finger-type POLY1 pattern is used. Each synapse cell including the programming circuitry occupies a compact area of 70 μm × 60 μm in 2-μm CMOS process, which is almost 10 times of a SRAM cell. Thus, a VLSI neural chip with 100k programmable synapses can be manufactured using the one mega-bit CMOS process.

(a)



(b)



(c)

Fig. 4.11 Die photos
(a) Programming cell.
(b) EEPROM-style synapse cell.
(c) 4-bit Hopfield A/D converter with EEPROM-style synapse cells.

References

[1] C. A. Mead, *Analog VLSI and Neural Systems*, Addison-Wesley Publishing Co. Inc, 1989.

[2] R. Hecht-Nielsen, "Neural-computing: picking the human brain," *IEEE Spectrum*, vol. 25, n0. 3, pp. 36-41, Mar. 1988.

[3] M. Holler, S. Tam, H. Castro, R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 'float gate' synapses," *Inter. Joint Conf. on Neural Networks*, vol. 2, pp. 191-196, June 1989.

[4] T. Morishita, Y. Tamura, and T. Otsuki, "A BiCMOS analog neural network with dynamically updated weights," *IEEE Inter. Solid-State Circuits Conf.* pp. 142-143, Feb. 1990.

[5] D. E. Van Den Bout and T. K. Miller III, "A Digital Architecture employing stochasticism for the simulation of Hopfield neural nets," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 5, pp. 732-738, May 1989.

[6] T. Quarles, SPICE3 Version 3C1 Users Guide, *Electron. Res. Lab. Memo UCB/ERL M89/46*, University of California, Berkeley, Apr. 1989.

[7] HSPICE User's Manual, Meta-Software Inc., Campbell, CA.,1989.

[8] R. E. Howard, D. B. Schwartz, J. S. Denker, R. W. Epworth, H. P. Graf, W. E. Hubbard, L. D. Jackel, B. L. Straughn, and D. M. Tennant, "An associative memory based on an electronic neural network architecture," *IEEE Trans. on Electron Devices*, vol. ED-34, no. 7, pp. 1553-1556, July 1987.

[9] B. W. Lee, B. J. Sheu, "Design of a neural-based A/D converter using modified Hopfield network," *IEEE Jour. of Solid-State Circuits*, vol. SC-24, no. 4, pp. 1129-1135, Aug. 1989.

[10] R. K. Ellis, "Fowler-Nordheim emission from non-planar surfaces," *IEEE Electron Device Letters*, vol. 3, no. 11, pp. 330-332, Nov. 1982.

[11] B. W. Lee and B. J. Sheu, "Electronic neural circuits with simulated annealing," *Inter. Joint Conf. on Neural Networks*, vol. II, June 1989.

[12] B. W. Lee and B. J. Sheu, "Hardware simulated annealing in electronic neural networks," *IEEE Trans. on Circuits and Systems*, June 1989.

[13] H. P. Graf, L. D. Jackel, W. E. Hubbard, "VLSI implementation of a neural network model," *IEEE Computer*, vol. 21, no. 3, pp. 41-49, Mar. 1988.

[14] H. A. R. Wegener, "Endurance model for textured-poly floating gate memories," *IEEE Inter. Electron Devices Meeting*, pp. 480-483, Dec. 1984.

[15] C. Tomovich, "MOSIS: A gateway to silicon," *IEEE Circuits and Devices Magazine*, vol. 4, no. 2, pp. 22-23, Mar. 1988.

Chapter 5

System Integration of VLSI Neural Computing Machines

VLSI neural circuits play an important role in real-time signal and image processing. Several neural hardware designs, which typically function as direct accelerators for software computation, have been reported [1]. These hardware accelerators including main processors, mathematical co-processors, and associated memories solve difference equations of a neural network instead of the original differential equations. The approximation of using difference equations requires a tremendously large computational time and also leads to a fatal failure caused by numerical errors [2]. The rich properties of neural networks inherent in the massively parallel processing scheme using analog neurons and synapses have not been fully explored yet. To make the neural computing hardware more powerful, compact and electrically programmable synapses are highly needed. Using the programmable neural chip with gain-adjustable neuron and adaptive synapses, reconfigurable neural systems with learning capability can be made. In addition, the hardware annealing technique can be easily applied to obtain the optimal solution.

5.1. System Module Using the Programmable Neural Chip

One good example of using the DRAM-style VLSI programmable neural network is shown in Fig. 5.1. The synapse weight data are stored in the buffer memory and are used to dynamically refresh the on-chip synapse voltage through a D/A converter. The host computer sets the synapse weight data into the

buffer memory through the interface circuitry. The memory address for the refresh operation is generated by the clock generator and counter. The learning process is simply done by writing synapse information into the buffer memory without stopping the retrieving process. Thus, the network retrieving and learning processes can be performed concurrently so that the back-propagation learning scheme can be easily implemented. The input and output signals for network retrieving are communicated with host computer according to the multiplexing signals. Notice that the analog multiplexors inside the VLSI chip are used for VLSI neural chips with high numbers of I/O's. The synapse accuracy can be easily enhanced by decreasing the refresh time and increasing the number of bits in external memory and the D/A converter.

The use of EEPROM-style programmable neural chips can save the external synapse weight memories and the refresh circuitry. Figure 5.2 shows the system-level block diagram for EEPROM-style design. During the learning process, the host computer sends out address signals and synapse weight data. At the high-voltage pulse generator, the synapse weight data is converted to the corresponding pulse signals with appropriate pulse width and magnitude. Since the synapse weight programming voltage is usually very high, the learning process should be separated from the network retrieving process. In supervised learning case, the learning process takes very long time because network outputs are monitored at each learning cycle. In addition, a certain waiting time after each programming cycle is also required due to the charge relaxation in the oxide layer. However, the excellent charge retention characteristics of the EEPROM-style neural chips can certainly save the large overhead for synapse

weight refresh.

The system module including the programmable neural chip is reconfigurable with external connections. Even though each module has a fixed number of neurons and synapses, any neural network architectures including Hopfield network, bidirectional associative memory, and multi-layer network can be built with multiple copies of such modules. For a complex application with very large number of neurons and synapses, the computing job can be partitioned into an appropriate size for each system module to handle. The module functions as a neural computing accelerator and the host computer stores and manages the intermediate data.

A multi-layer neural network using the system module approach is shown in Fig. 5.3. For an optimum system configuration, the refresh circuitry can be shared with several boards. The input/output signals for hidden layers can be directly connected between layers, while that for input and output layers are connected to host computer through the system interface logic.

5.2. Application Examples

Two VLSI neural chips which use the programmable synapse and gain-adjustable neurons are demonstrated. One chip with only analog circuitry operates asynchronously, while the other chip with mixed analog-digital circuitry operates synchronously. The computational time of the special neural hardware is also compared with that of the corresponding software computation on standard workstations.

5.2.1. Hopfield Neural-Based A/D Converter

The adaptive synapses and gain-adjustable neurons can make neural-based A/D converters more flexible in the output characteristics. Since many synapses have the same weighting, the voltage can be shared with many synapse cells as shown in Fig. 5.4. Figure 5.5(a) shows the die photo of an 8-bit neural-based A/D converter. The chip size is 800 μm x 550 μm in the MOSIS 2-μm Scalable CMOS technology. In this design, the values of feedback synapses are negative and the neuron outputs are in a non-inverted format. For synapses T_{iS} and T_{iR}, the analog input and reference voltage are directly applied to the terminals for synapse weight voltage storage, while the synapse bias currents (I_j) are always turned on. Thus, the synapse weight voltage of the reference synapses are also shared with that of the feedback synapses. Notice that the negative synapse weight can be obtained by applying a negative voltage to the weight-storage capacitor. When the synapse weight voltages are derived from the R-2R ladder circuit and external voltage V_{ext}, the conversion step size is $V_{ext}/2^n$. The rich local minima, which appear as the overlapped analog input range in the transfer function, can help to compensate the initial device mismatches. By applying the hardware annealing technique, the local minima can be eliminated so that the transfer curve becomes pretty linear.

The conversion speed of the original Hopfield A/D converter and the self-correction A/D converter are shown in Fig. 5.6. In this experiment, the 4-bit neural-based A/D converter is used. The response time can be easily grouped into 4 classes, which are dependent upon the number of convergence iterations. Notice that there are some missing codes in the Hopfield A/D converter because

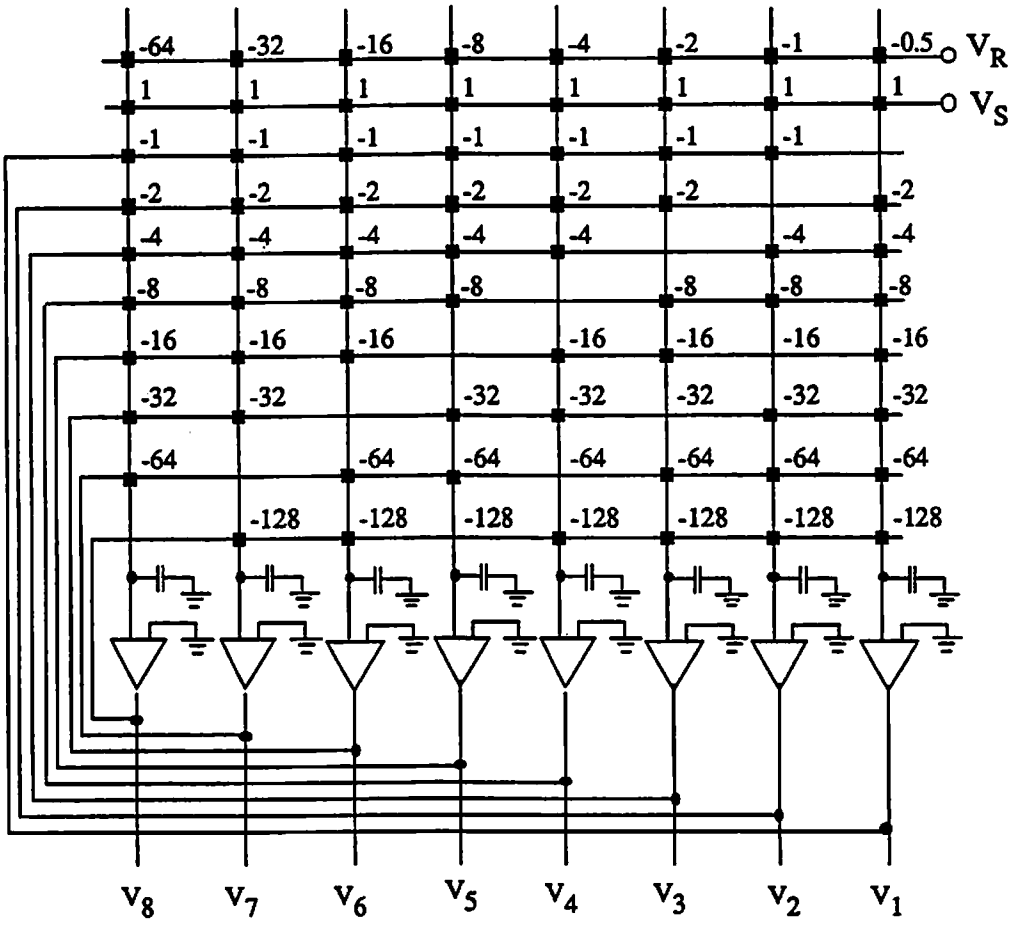


Fig. 5.4 Circuit schematic of an 8-bit neural-based A/D converter with the programmable synapses and gain-adjustable neurons.

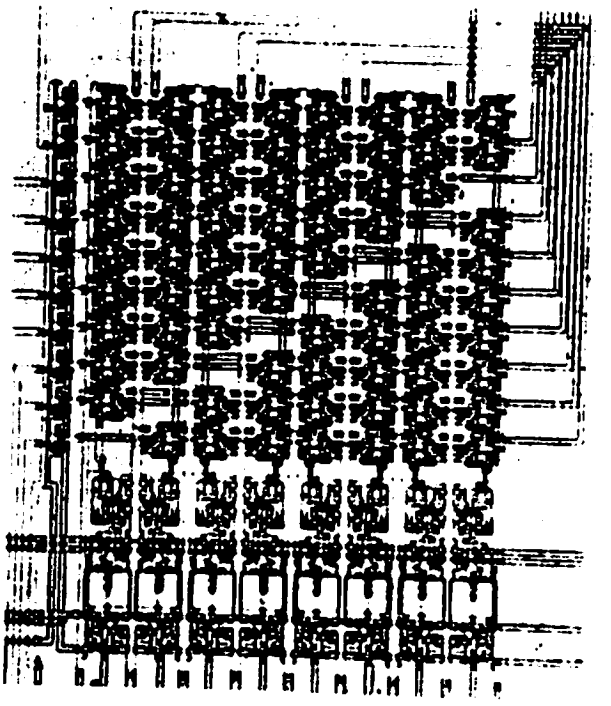


Fig. 5.5 Die photo of the 8-bit neural-based A/D converter in Fig. 5.4.

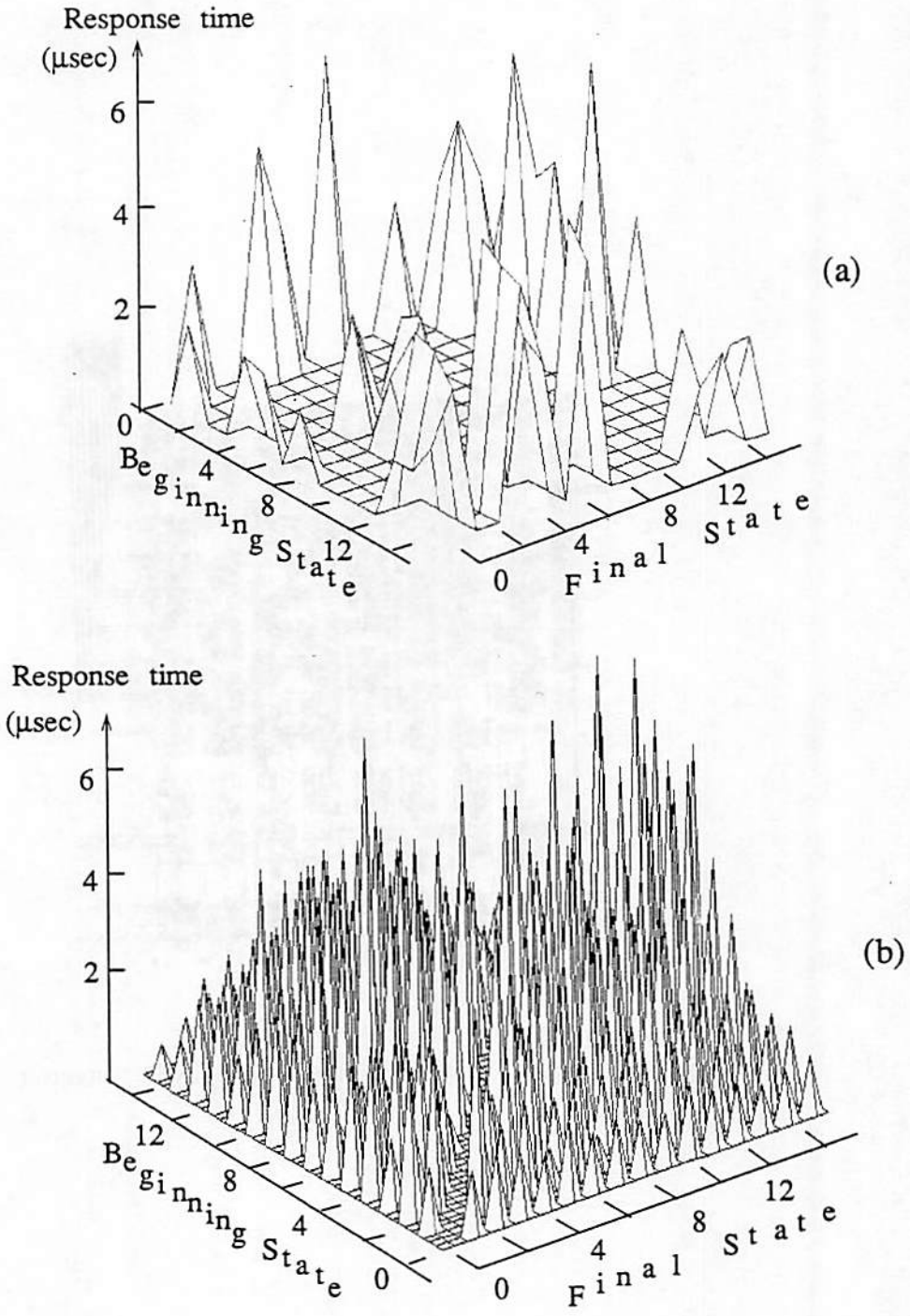
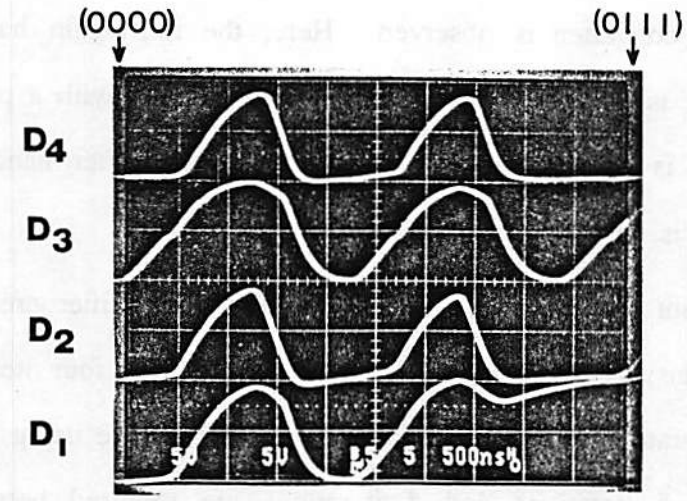


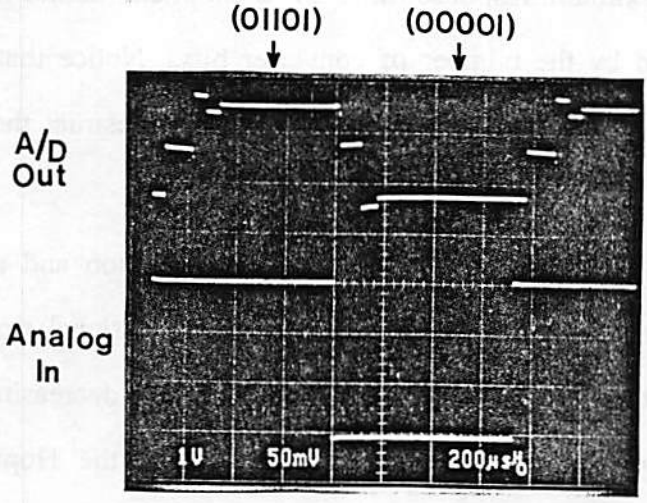
Fig. 5.6 Response time charcteristics of 4-bit neural-based A/D converters.

(a) Original Hopfield A/D converter.

(b) Self-correction A/D converter.



(a)



(b)

Fig. 5.7 Transient response.
(a) Each amplifier output of a Hopfield 4-bit A/D converter.
(b) 5-bit Hopfield A/D converter output.

of the local minima. No significant change of the response time in the self-correction A/D converter is observed. Here, the unity-gain bandwidth of the neuron amplifier is about 1 MHz and the response time with a pulsed input signal of 2 V_{p-p} is approximately 1.5 μsec. On the other hand, delay of the correction logic is approximately 10 nsec.

The transient characteristics of the individual amplifier are shown in Fig. 5.7. In the transition from (0000) to (0111), the four iterations to settle down the final state are clearly displayed. In another case using a 5-bit Hopfield A/D converter, 5 iterations and 4 iterations are required between transitions (00001)-to-(01101) and (01101)-to-(00001), respectively. This experimental result shows that the maximum response time of a Hopfield neural-based A/D converter is determined by the number of converter bits. Notice that the time delay of an off-the-shelf D/A converter, which is used to reconstruct the A/D converter output, is 1 μsec.

The speed comparison between the IC implementation and a software computation is listed in Table 5.1. Here, the total computational time is defined as the amount of time to obtain full-scaled increasing and decreasing analog inputs with 300 evaluation points. In software computation, the Hopfield differential equations are solved by Euler forward integration methods with fixed time steps and with adaptive time steps. For the adaptive time-step case, a local numerical error is estimated with a Richardson extrapolation method [2]. The program attached in Appendix-A was written in C-language and executed in SUN 4/60 workstations. For the speed calculation of the IC implementation, the worst-case conversion time (6 μsec) for one conversion was used. The initial condition

Table 5.1 Computational speed of 4-bit A/D conversion

Item \ Method	Hardware Computation using IC	Software Computation			
		Adaptive time step	Fixed-time step		
			$1.x\ 10^{-2}$	$1.x\ 10^{-3}$	1×10^{-4}
Computational Time (sec)	1.8×10^{-3}	314	182	1697	16147
CPU-time factor	1	$1.7x\ 10^{5}$	$1.0x\ 10^{5}$	$9.4x\ 10^{5}$	9.0×10^{6}
Quality of Solution	Good	Good	Bad	Poor	Good

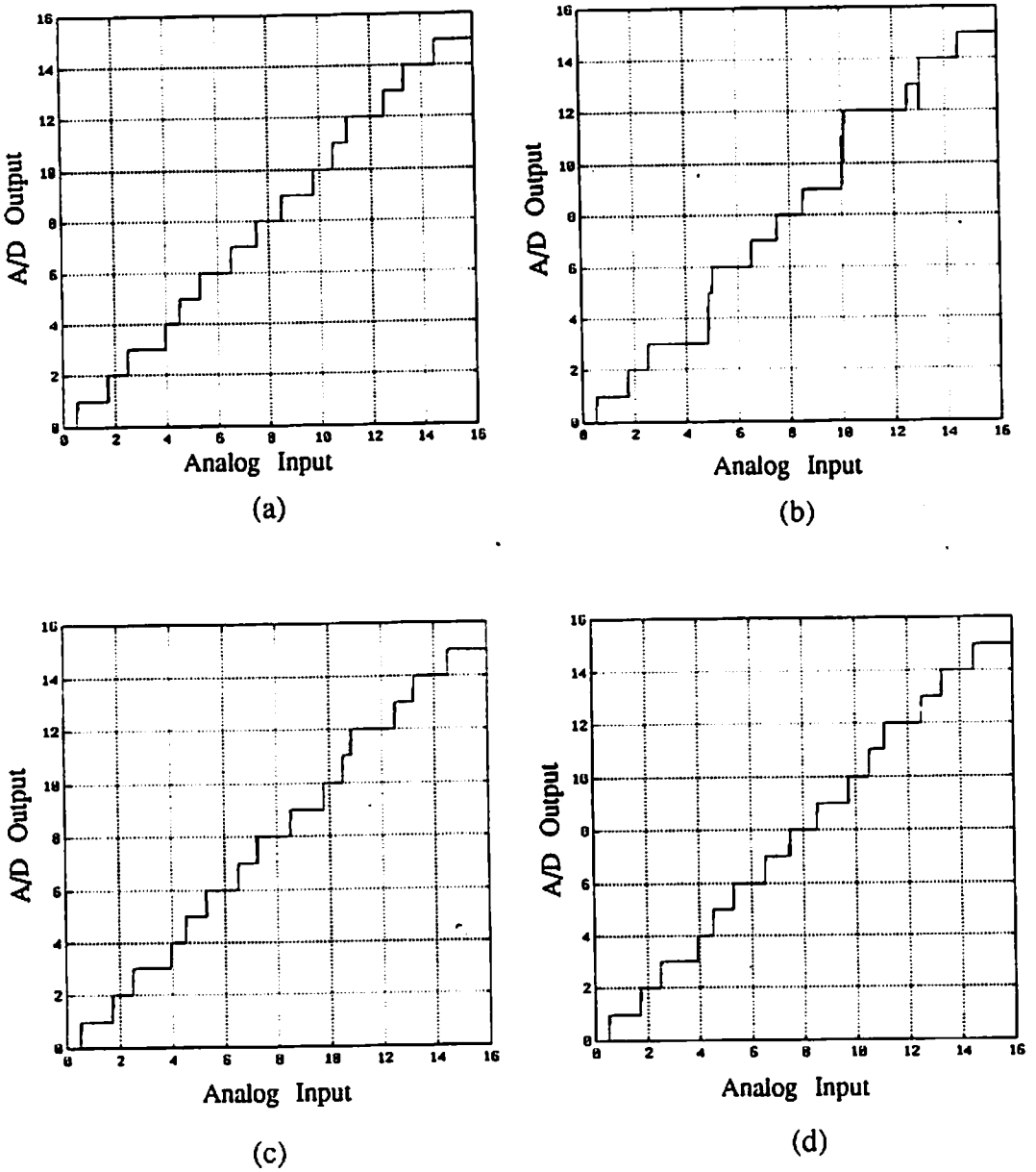


Fig. 5.8 Software simulation results.
(a) Adaptive time step.
(b) Fixed time step of 1.0×10^{-2} sec.
(c) Fixed time step of 1.0×10^{-3} sec.
(d) Fixed time step of 1.0×10^{-4} sec.

for each analog input voltage was reset in software computation. The simulated transfer curve for the time step of 1.0×10^{-4} sec is the same as that for the adaptive time step case as shown in Fig. 5.8. Notice that the adaptive time step scheme is equivalent to solve the original differential equations, while the fixed time scheme is to solve the approximated difference equations. As compared with the adaptive time step scheme, the hardware computation enhances the speed by the factor of 17,400.

In the VLSI implementation, amplifier offset voltage and synapse weight tolerance caused by the device mismatches are the major limitations. When the amplifier input offset voltage V_{ios} exists, it affects the decision voltage for the neuron output. The neuron input voltage is determined by

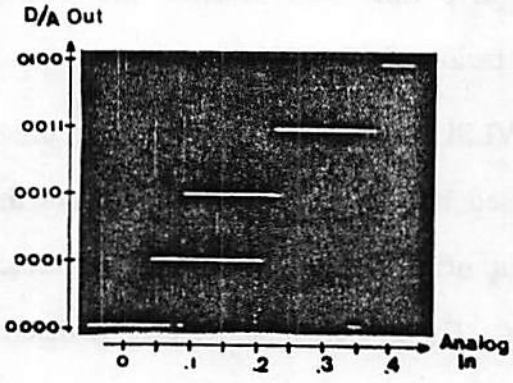
$$u_i = \frac{\sum_{j=1, j\neq i}^{n} T_{ij} V_j + T_{iS} V_S + T_{iR} V_{iR}}{T_i} , \tag{5.1}$$

where T_i is total input conductance to the i-th neuron. The shift of the analog input voltage is given by

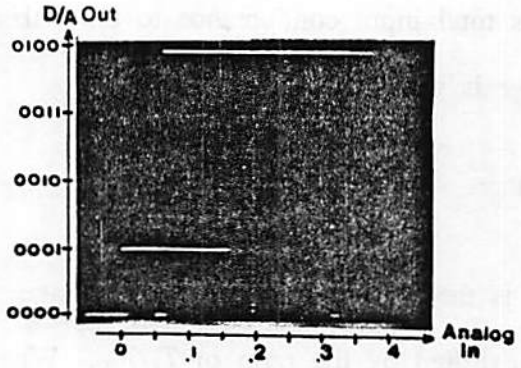
$$\Delta V_S = \frac{T_i}{T_{iS}} \times V_{ios} . \tag{5.2}$$

Here, V_{ios} is the amplifier input offset voltage. The shift of the analog voltage range is magnified by the ratio of T_i/T_{iS}. When the number of bits, n, is very large, the asymptotic value for magnification factor is

$$\frac{T_i}{T_{iS}} \simeq \begin{cases} n+1 & \text{with programmable synapse} \\ 2^n & \text{with direct-R synapse} . \end{cases} \tag{5.3}$$



(a)



(b)

Fig. 5.9 Finite gain and input offset voltage effects.
(a) Bias current for each amplifier is 2.5 μA.
(b) Bias current for each amplifier is 12.5 μA.

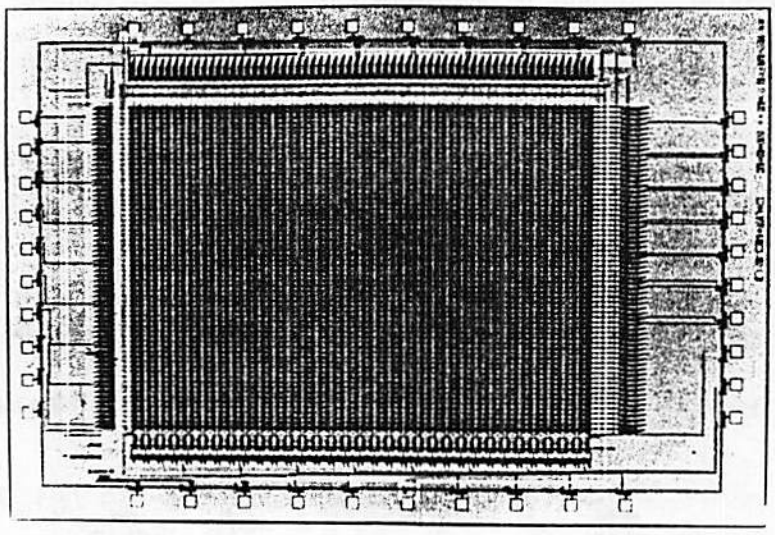
Notice that actual conductance value in the DRAM-style synapse is constant, while the synapse weight is determined by the external voltage. Thus, the input voltage shift of the programmable synapse is much smaller than that of the direct-resistance synapse.

The tolerance of the synapse weight also affects the analog input voltage range. When V_{ios} is very small and the amplifier gain is very large, the analog input voltage is determined by (2.39) and (2.40). Thus, the change of the analog input voltage range caused by the synapse weighting tolerance is given as

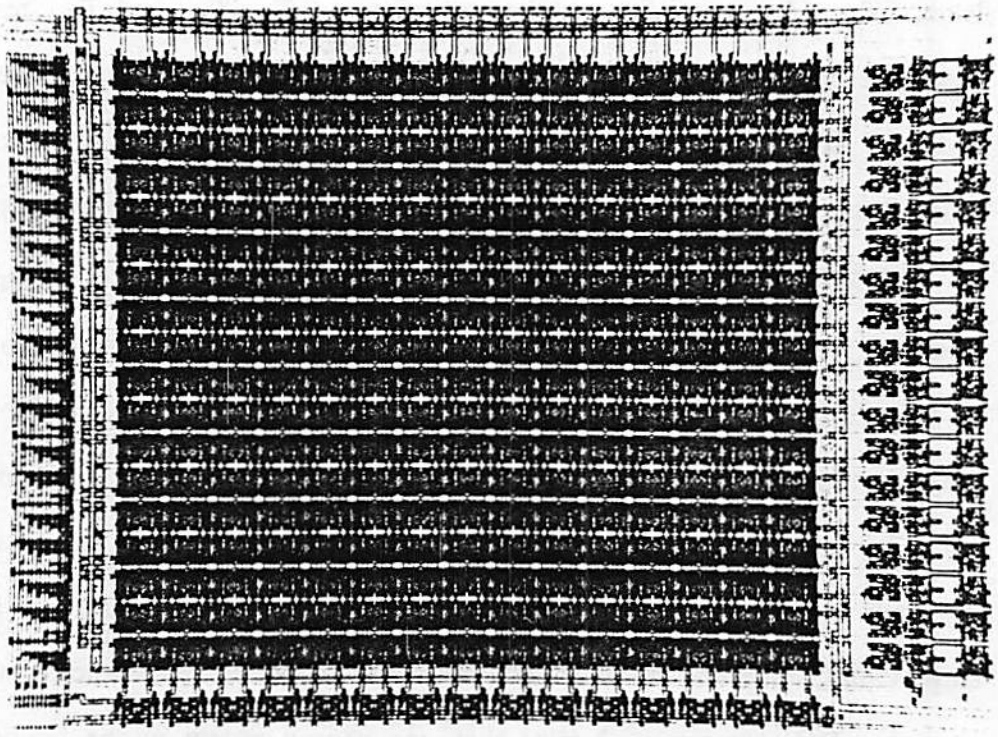
$$\delta V_S = -\frac{\sum_{j=1,j\neq i}^{n} V_j(\delta T_{ij}) - V_R(\delta T_{iR})}{T_{iS}} + \frac{\delta T_{iS}}{T_{iS}} \times \frac{\sum_{j=1,j\neq i}^{n} T_{ij}V_j + T_{iR}V_{iR}}{T_{iS}} . \quad (5.4)$$

Here, δT_{ij}, δT_{iS} and δT_{iR} are the tolerances of T_{ij}, T_{iS}, and T_{iR}, respectively. Since the absolute value of the tolerance usually increases with the synapse conductance value in the VLSI technology, the change in analog input range of a direct-R implementation is much larger than the DRAM-style implementation.

The measurement results of a Hopfield neural-based A/D converter whose synapses were realized with resistors using p-well regions are shown in Fig. 5.9. In this design, the T_{iS} is scaled up by 10 times and the analog input voltage range is reduced by 10 times. When the bias current for the amplifier increases from 2.5 μA to 12.5 μA, the analog input range is shifted by approximately 30 mV due to the increase of amplifier offset voltage. The neuron is a basic 2-stage CMOS amplifier and the A/D converter output is reconstructed with a conventional D/A converter. Notice that one neuron output at the boundary value of the analog input voltage range is usually not a discrete value due to the finite



(a)



(b)

Fig. 5.10 Die photos of general-purpose neural chips using DRAM-style storage.
(a) With multiplexing I/O pins.
(b) Without multiplexing I/O pins.

amplifier gain. Thus, a certain overlapped input range for digital codes (0000) and (0001) can exist due to the A/D output reconstructing process.

The die photos of two general-purpose chips using DRAM-style synapse cells are shown in Fig. 5.10. The chip sizes are 4400 μm x 6600 μm and 2500 μm x 2350 μm in MOSIS 2-μm CMOS technology, respectively. Each chip include a single layer network with fully connected synapses. The input/output pads of the chip in Fig. 5.10(a) are multiplexed in time in order to solve the limited number of package pins. Thus, the chip including 64 input neurons, 64 output neurons, and 4096 programmable synapse should operate in the synchronous mode. On the other hand, the other chip shown in Fig. 5.10(b) only including 25 input neurons, 12 output neurons, and 300 synapses can operate in the asynchronous mode.

5.2.2. Modified Hopfield Network for Image Restoration

Image restoration is an important task in digital image processing. The image quality is recovered from a degraded image caused by optical system aberration, motion, diffraction, and noise. The image restoration process, which find the improved image **X** by minimizing the error function E with respect to the degraded image **Y**, can be described as follows [4],

$$E = \frac{1}{2}||\mathbf{Y} - \mathbf{HX}|| + \frac{1}{2}\lambda||\mathbf{DX}|| \ , \tag{5.5}$$

where $||\ .\ ||$ is the L_2 norm, **H** is the blur matrix, λ is a constant, and **D** is the sharpness matrix. In the case that the image has L x L pixels with M-bit gray levels, the error function in (5.5) can be described as a Hopfield energy function.

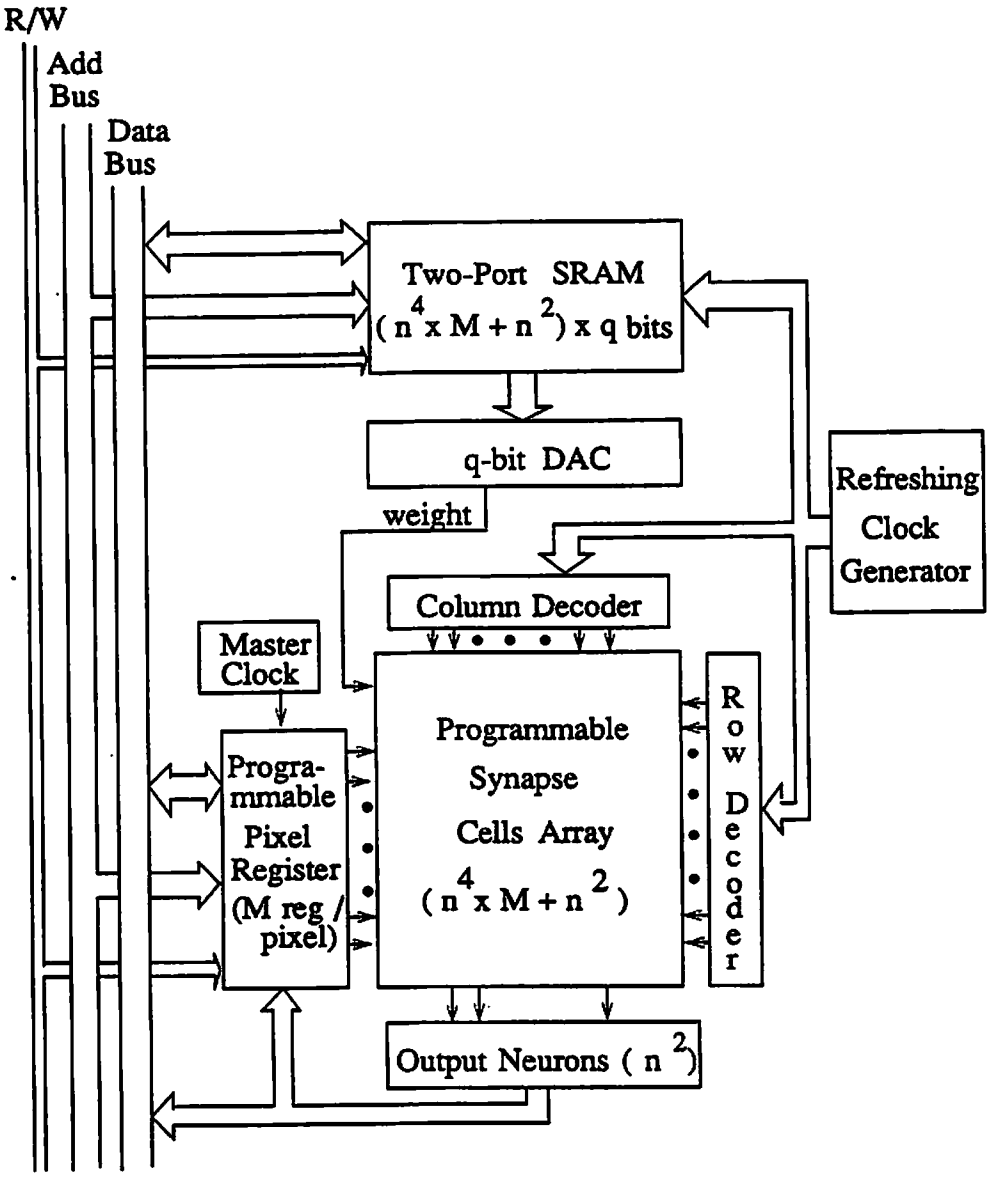


Fig. 5.11 Architecture of image processing neural chip.

The synapse weightings $\{ T_{i,k;j,l} \}$ and the input currents $\{ I_{i,k} \}$ of a Hopfield network are given as

$$T_{i,k;j,l} = -\sum_{p=1}^{L^2} h_{p,i} h_{p,j} - \lambda \sum_{p=1}^{L^2} d_{p,i} d_{p,j} \tag{5.6}$$

and

$$I_{i,k} = \sum_{p=1}^{L^2} y_p h_{p,i} \ . \tag{5.7}$$

Here, i, j, k, and l are integers in [1, L].

Since the self-feedback terms of synapse matrix $\{ T_{i,k;j,l} \}$ are not zero, the Hopfield network will not always move toward a lower energy level. In addition, L^2M neurons and $L^4M^2/2$ synapses are required for the Hopfield network. Detailed algorithms for image restoration using Hopfield networks can be found in [5]. In the modified Hopfield network, the number of neurons and synapses are reduced to L^2 and L^4, respectively. Each output neuron generates increment/decrement signal for the M gray-level register in a synchronous manner. During iterations, the gray level for each pixel is increased or decreased by the pixel neuron output. The initial states for each gray level register are from the blurred image.

The chip-level architecture for the real-time image restoration is shown in Fig. 5.11. For VLSI implementation, the whole image is partitioned into subimages of n x n pixels and the subimage is processed in one VLSI chip. The n^2 neuron outputs are sent to the pixel registers. Each pixel register consists of an M-bit counter which stores the pixel gray level. The initial content of pixel

registers is loaded from the blurred image and the final content of pixel registers is the restored image. Notice that the D/A conversion function for the pixel register is effectively realized in the binary-weighed synapses. Thus, one synapse value in (5.6) is coded into M synapses $T^r_{i,k;j,l}$.

$$T^r_{i,k;j,l} = T_{i,k;j,l} \times 2^r \ , \tag{5.8}$$

where r is [1, M]. The synapse weightings of the fully connected n^4 x M synapses are determined from parameters of the blur function and smoothing constraint, while that of the n^2 synapses are determined from the input currents. Although the synapse matrix is usually very sparse, the fully connected synapse matrix is implemented to examine various blur and smooth functions. Two-port SRAM with (n^4 x M + n^2) x q bits are used for the learning process and the retrieving process. The synapse weighting is described in q-bit accuracy. In typical applications, q = 8 is used. The refresh address generator provides the address signals to row/column decoder and the SRAM. The synapse weight data in the SRAM are sent to the synapse cells. Figure 5.12 shows the one-bit digital circuit cell for the pixel register, which includes the read/write circuitry and counter. The die photo of a modified Hopfield network for processing 5 x 5-pixel subimages is shown in Fig. 5.13. In this design, a 4-bit resettable counter per one pixel is used to restore an image with 16 gray levels. Thus, the chip includes 25 gain-adjustable neurons and 2525 programmable synapses in 6800 μm x 4800 μm chip size using the MOSIS 2-μm Scalable CMOS technology.

System-level simulation using a girl image and a house image of 256 x 256 pixels has been conducted to evaluate the performance of the image restoration

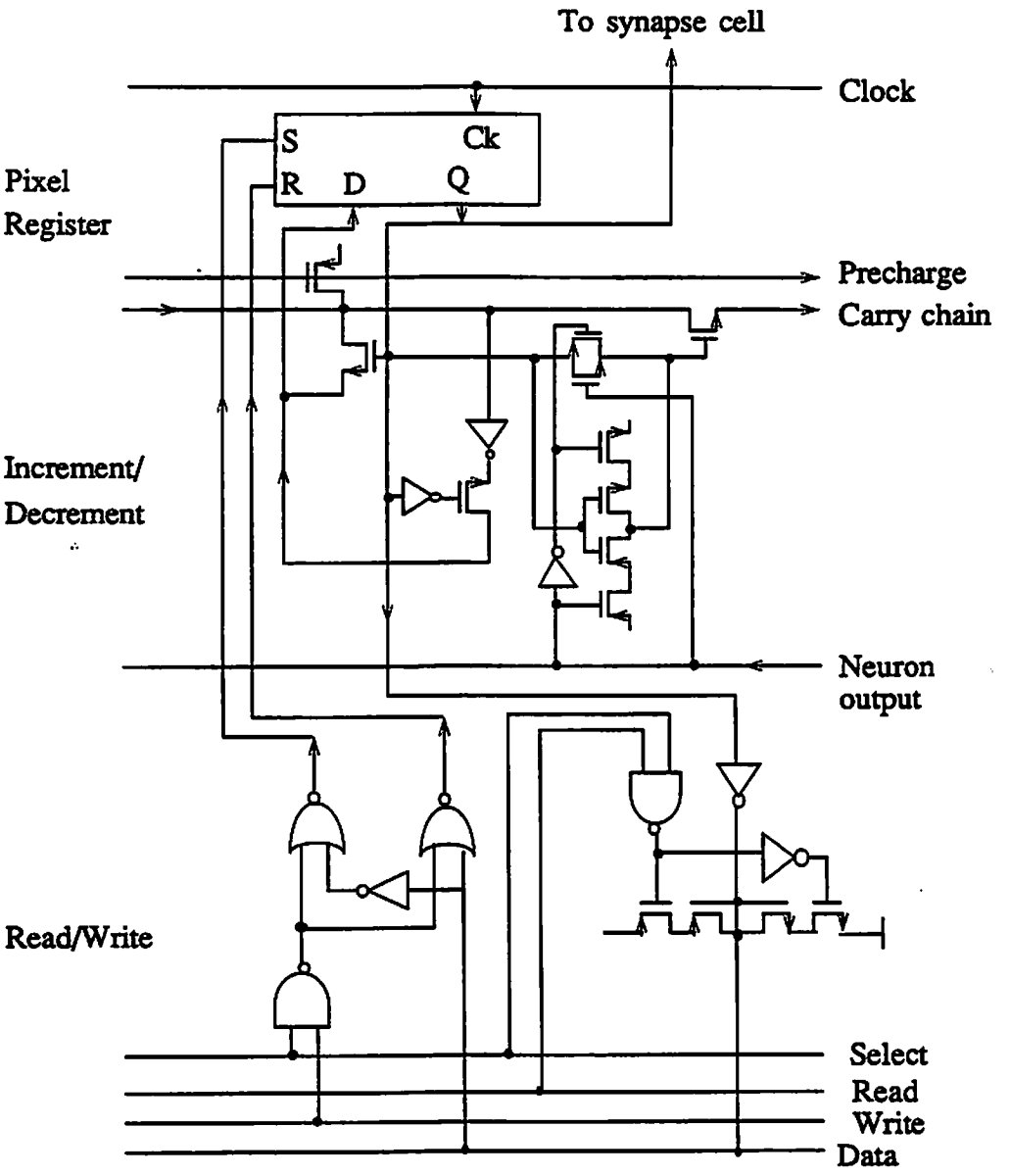


Fig. 5.12 Logic schematic of one pixel programmable register.

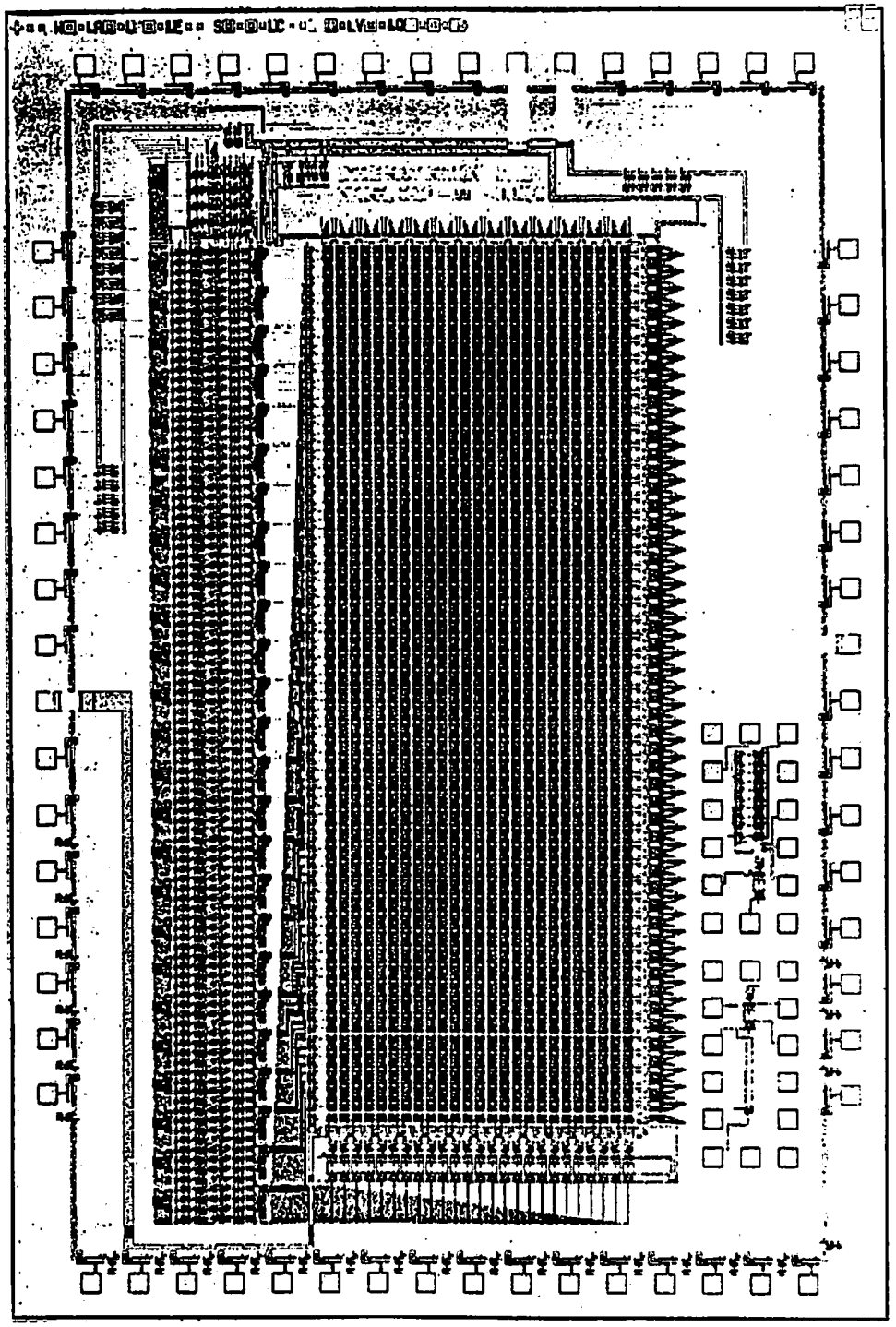
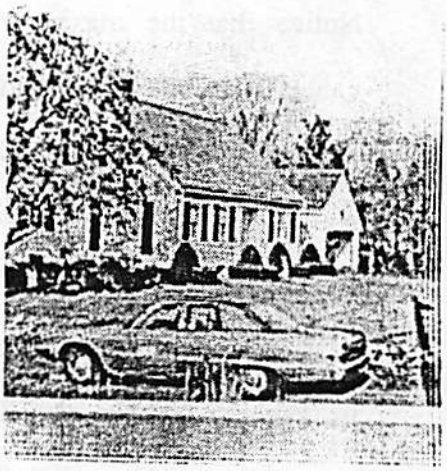


Fig. 5.13 Die photo of a synchronous Hopfield network for image restoration. The chip size is 4600 µm x 6800 µm in MOSIS 2-µm CMOS technology.

(a)

(b)

(c)



(d)

Fig. 5.14 System simulation results on the house image.
(a) Blurred picture of girl image.
(b) Restored picture of girl image.
(c) Blurred picture of house image.
(d) Restored picture of house image.
The image is partitioned into 5 x 5 subimages and processed using the synchronous Hopfield chip architecture.

chip. The images were blurred by a uniform and shift-invariant blur function of a 3x3 window size. The blurred function **H** used in this simulation is

$$\mathbf{H} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}. \tag{5.9}$$

Then, the synapse weighting in (5.8) is given as

$$T_{i,k;j,l} = \begin{cases} -9/81 & \text{if } i=j \cap k=l \\ -6/81 & \text{if } (|i-j|=1 \cap k=l) \cup (i=j \cap |k-l|=1) \\ -4/81 & \text{if } |i-j|=1 \cap |k-l|=1 \\ -3/81 & \text{if } (|i-j|=2 \cap k=l) \cup (i=j \cap |k-l|=2) \\ -2/81 & \text{if } (|i-j|=2 \cap |k-l|=1) \cup (|i-j|=1 \cap |k-l|=2) \\ -1/81 & \text{if } |i-j|=2 \cap |k-l|=2 \\ 0 & \text{otherwise .} \end{cases} \tag{5.10}$$

Notice that the maximum number of neurons whose outputs are not discrete values is 6 in this blurred function. The image is partitioned into 5 x 5 subimages and processed with the VLSI chip architecture of the modified Hopfield network. The image restoration process is performed for 20 iterations. The blurred and restored images are shown in Fig. 5.14. The performance comparison of adaptive neural chips with a Sun-3/60 workstation in image restoration is listed in Table 5.2. The speedup factor using a single chip of 25 neurons and 100 neurons are 341 and 618, respectively. Further improvement can be achieved with the use of an array of neural chips as shown in Fig. 5.15. Here, a 100-neuron chip can be fabricated in an industrial-level 1-μm CMOS technology with the 50 mm^2 chip size.

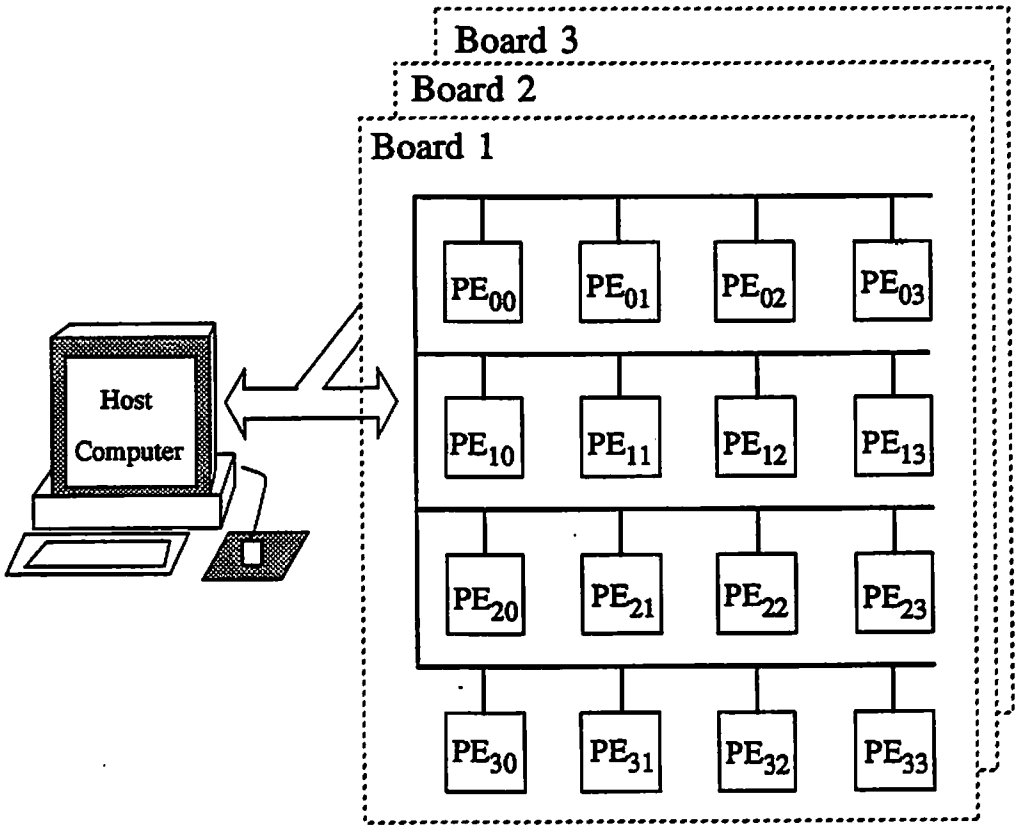


Fig. 5.15 Schematic diagram of a multi-chip image restoration system.

Table 5.2 Comparison of three image restoration systems

(An image with 256x256 pixels and 16 gray levels per pixel is used.)

System / Performance	25-neuron chip	100-neuron chip	Five copies of 100-neuron chip
Synapse Programming time	27 ms	441 ms	2205 ms
Pixel register read/write time	880 ms	880 ms	880 ms
Network execution time w/ 20 iter./subimage	2601 ms	625 ms	625 ms
Total processing time for whole image	3.51 sec	1.94 sec	3.7 sec
Speed-up factor to Sun-3/60	341	618	1620

References

[1] R. Hecht-Nielsen, "Neural-computing: picking the human brain," *IEEE Spectrum*, vol. 25, no. 3, pp. 36-41, Mar. 1988.

[2] B. W. Lee and B. J. Sheu, "Combinatorial optimization using competitive-Hopfield neural networks," *1990 Inter. Joint Conf. on Neural Networks*, vol 2, pp. 627-630, Jan. 1990.

[3] G. Dahlquist and A. Bjorck, *Numerical Methods*, pp. 330-350, Prentice-Hall Inc., 1974.

[4] H. C. Andrews and B. R. Hunt, *Digital Image Restoration*, Englewood Cliffs, NJ: Prentice-Hall, 1977.

[5] Y. Zhou, R. Chellappa, A. Vaid and B. Jenkins, "Image Restoration Using a Neural Network", *IEEE Trans. Acoustics, Speech & Signal Proc.*, vol. 36, pp. 141-1151, July 1988.

[6] B. W. Lee, J. Lee, and B. J. Sheu, "VLSI image processors using analog programmable synapses and neurons," *1990 Inter. Joint Conf. on Neural Networks*, San Diego, June 17-21, 1990.

Chapter 6

Conclusion and Suggested Future Studies

Due to the inherent parallel processing architecture, VLSI neural circuits are critical to the success of future high-performance computing systems. To design VLSI neural circuits, the artificial neural networks developed for software computation have to be modified. In this dissertation, theory and several design techniques for VLSI neural networks have been described:

1. Dynamics of Hopfield neural networks and techniques to eliminate local minima from the energy function;
2. Competitive-Hopfield networks to solve combinatorial optimization with multiple constraint functions;
3. Hardware annealing for efficient searching of optimal solution by changing amplifier voltage gain instead of annealing temperature;
4. Programming synapses and gain-adjustable neurons for multi-layered VLSI neural systems.

Table 6.1 shows the comparison of the self-correction technique and hardware annealing technique for Hopfield neural networks.

I propose and complete the design of the DRAM-style synapse cells for medium-term weight storage and the EEPROM-style synapse cells for long-term weight storage. Using these design technique, several prototyping neural chips operating in either asynchronous or synchronous fashion are developed. Two real-world applications including neural-based A/D conversion and the digital

Table 6.1 Comparison of correction and annealing schemes

	Self-Correction Technique	Hardware Annealing Technique
Computing Speed	Fast	Slow
Device Matching	Accurate	Not accurate
Applicable Network	Only Hopfield net	Any network
Limitation	Global minima should be well-known.	Effective for asynchronous net.

image restoration are presented. By using the compact synapse and neuron circuit cells, an electrically programmable neural chip containing about 500 neurons and 250k synapses can be fabricated in industrial standard 1-μm CMOS technologies.

More studies to improve the circuit performance and to reduce the area of synapse cells and neuron cells are necessary to integrate more complex neural networks in VLSI technologies. The dynamic range of programmable synapse weights with fixed standby power dissipation can be greatly enhanced by using the non-saturated input stage architecture described in Appendix B. Since the non-saturated differential pair always operates in the linear region, the power dissipation of reconfigurable VLSI neural circuits can be reduced dramatically.

Design automation can be utilized to build VLSI neural chips because of functional regularities in neural networks. In general, analog VLSI circuits are sensitive to device mismatches, and parasitic components, so that design automation for the analog circuits are still very primitive. However, components in neural networks do not have to be very precise or fast. In addition, the self-learning capability in neural networks can compensate not only initial device mismatches but also long term drift of the device characteristics. Several design methodologies for analog VLSI circuits which include analog array, standard cells, parameterized cells, and programmable cells can be used. Since neural networks have high degree of modularity consisting of a neuron and associated synapses, the analog standard cell approach will be suitable to design VLSI neural circuits.

Research toward system demonstration for large and practical problems can be done by using the programmable synapse and the gain-adjustable neuron. Research on on-chip learning is also very important to reduce total system learning time and to compensate nonideal characteristics of VLSI neural circuits. Major differences in VLSI neural circuits include device mismatches, limited synapse weights, and imprecise neuron transfer function. Thus, a special on-chip programming circuit, which monitors the neural circuit outputs and compensates the differences, should be developed for real-world applications.

Appendix A

Program for neural-based A/D conversion and traveling salesman problem

Many subrnoutines are commonly used for the two problmes. To compile, just type " cc Neural_ADC.c -lm " or "cc TSP.c -lm ".

```
/****************** filename = Neural_ADC.c *********************
************************ main program ************************
*****************************************************************/
#include "declarat.c"
main()
{
 int i, int_iter=0, initial_cond_flag, E_eq =0;
 double E_old = 1.0e30, E_new, temp,
     U_old[N_bit], U[N_bit], V_old[N_bit], V[N_bit],
     U2[N_bit], U1H[N_bit], V2[N_bit], V1H[N_bit],
     find_E(), est_local_error(),
     Time_final =1.0, factor_etol,
     local_error; /* local error tolerance */

 ask_initial_values(); /* get parameters */

 printf(" Analog Input voltage Vs =\n");
 scanf("%lf", &VS);
 printf("\n\n Vs = %f, Vref = %f\n",VS,VR);

 printf("Set initial condition at input/0 or at output/1\n");
```

```
                    /* set initial conditions */
while ( scanf("%d", &initial_cond_flag ) )
     if ( initial_cond_flag == 0 || initial_cond_flag == 1 )break;
if ( initial_cond_flag == 0 )
 {
  printf("\n Set input initial conditions \n");
    init_soln_o(U_old);
    for ( i = 0; i < N_bit ; ++i )
     {
      U_old[i] -= 0.5;
      U_old[i] /= 10.0;
      U[i] = U_old[i];
      }
   print_U(U);
   find_V(U,V_old);
 }
else
 {
  printf("\n Set output initial conditions \n");
  init_soln_o(V_old);
  for ( i=0; i < N_bit ; ++i )U_old [i] =0.0;
 }
for ( i = 0; i < N_bit; i++) V[i] = V_old[i];
print_V(V);

cre_I();
cre_T();
cre_Tc();
   while ( iterate == 'y')
 {
```

```
                    /* iteration start */
++ iter_count;
        /*** Runge-kutta 4-th method or Euler forward method
                        with adaptive step control */
euler_model(U_old,V_old,U2,V2,wu);  /* regular time step */
int_iter = 0;
local_error = 1.0e30;
factor_etol = 1.0;
temp =1.0;

while ( temp > 0.0 && fix_T != 'y' )
 {
 if ( ++ int_iter > 100 || factor_etol < 1.0e-10 )break;
 wu /= 2.0;
 euler_model(U_old,V_old,U1H,V1H,wu); /* 1st time with half
                                step size    */
 euler_model(U1H,V1H,U,V,wu);        /* 2nd time with half
                                step size */

 local_error = est_local_error(U2,U); /* estimate local error */
 E_new = find_E(0,V);
 temp =local_error - etol/(Time_final*1.0);
 if (batch ==0 )
  {
  printf("interal iter= %d, energy= %g,",++int_iter, E_new);
  printf(" local error= %e, wu= %g\n",local_error,wu);
  }

 for (i=0; i < N_bit; ++i)
  { U2[i] = U1H[i];
```

```
        V2[i] = V1H[i]; }
      }
    if ( fix_T == 'y' )
      for ( i=0; i < N_bit; ++i){U[i] = U2[i]; V[i] = V2[i]; }
    time += wu;
printf("iter= %d time= %e ", iter_count, time );

          /* terminating criteria by E updating */
    E_new = find_E(1,V);
    if ( fabs( E_new - E_old ) < 1.0e-6)
       ++E_eq;
     else
       E_eq = 0;
    if ( E_eq > 10 )terminate_flag =1;
    E_old = E_new;
    if ( batch == 0 || terminate_flag == 1 ) print_V(V);
    if ((iter_count / 100) * 100 - iter_count == 0) print_V(V);
    if ( iter_count > iteration_lmt ) terminate_flag = 1;
    for (i=0; i < N_bit; ++i )
     {
      U_old[i] = U[i];
      V_old[i] = V[i];
      }
    if ( fix_T != 'y')
     { if (local_error < 1.0e-10)  /* next step wu */
        local_error =1.0e-10;
       wu *= pow (etol/(local_error*Time_final), 1.0 );
       if ( wu > 1.0e-1 )wu=1.0e-1;
      }
    if ( terminate_flag == 1)break;
```

```
    }
  output(U,V);
} /* end of main program */

/********************** filename = tsp.c **********************
************************ main program ************************
*****************************************************************/
#include "declarations.c"

main()
{
 char ch_const, iterate, c, fix_T;
 int i, int_iter=0, initial_cond_flag, E_eq =0, itemp;
 double E_old = 1.0e30, E_new, temp,
     U_old[N_SQ], U[N_SQ], V_old[N_SQ], V[N_SQ],
     U2[N_SQ], U1H[N_SQ], V2[N_SQ], V1H[N_SQ],
     TD[N_SQ][N_SQ],
     find_E(), est_local_error(),
     Time_final =1.0, factor_etol,
     local_error; /* local error tolerance */

 printf("Type seed for distance marix !!");
 if ( N == 10 )printf("0 = fixed location");
 scanf("%d", &itemp);
 if ( itemp == 0 )
  init_dist();
  else
   {
    for ( i = 0; i < itemp; ++i) random_city_coord(0);
    random_city_coord(1);
```

```
  }
printf("\nseed = %d\n", itemp );

cre_delta();
set_up_dist();

ask_initial_values(); /* get parameters */

printf("Set initial condition at input/0 or at output/1\n");
                    /* set initial conditions */
while ( scanf("%d", &initial_cond_flag ) )
     if ( initial_cond_flag == 0 || initial_cond_flag == 1 )break;
if ( initial_cond_flag == 0 )
 {
  printf("\n Set input initial conditions \n");
    init_soln_o(U_old);
    for ( i = 0; i < N_SQ ; ++i )
     {
      U_old[i] -= 0.5;
      U_old[i] /= 10.0;
      U[i] = U_old[i];
     }
   print_U(U);
   find_V(U,V_old, initial_cond_flag);
 }
 else
 {
  printf("\n Set output initial conditions \n");
  init_soln_o(V_old);
  for ( i=0; i < N_SQ ; ++i )U_old [i] =0.0;
```

```
 }
for ( i = 0; i < N_SQ; i++) V[i] = V_old[i];
print_V(V);

cre_I();
cre_T(TD);
cre_Tc(TD);

   printf("\n\nVersion 1, May 10 1989\n");
   printf(" Euler w/ adative, w/o symmetric solutions\n");
   iterate = 'y';
   printf("\nBatch ? Yes/1 \n");
   scanf("%d", &batch);

   printf("\nNumber of iterations : ");
   scanf("%d", &iteration);

   printf("\nFixed Time Step ? y or n");
   while ( fix_T = getchar() )if ( fix_T == 'y' || fix_T == 'n')break;
   if ( fix_T == 'y')printf("\nfixed time step mode \n");
    else printf("\nadaptive time step mode \n");

   printf("Use competative network? y or n");
   while ( comp_net = getchar() )
      if ( comp_net == 'y' || comp_net == 'n')break;
   if ( comp_net == 'y')
{
    printf("\n Use Competative network\n");
    printf("Adaptived threshold voltage for competative net? y or n");
    while ( adapt_th = getchar() )
```

```
    if ( adapt_th == 'y' || adapt_th == 'n')break;
   if ( adapt_th == 'y')printf("\nAdaptive threshold mode \n");
  }
  else
   printf("\n Use Hopfield network Only \n");
  while ( iterate == 'y')
{
                        /* iteration start */
for (iter_count = 0; iter_count < iteration ; )
 {
     ++ iter_count;
          /*** Runge-kutta 4-th method or Euler forward method
                        with adaptive step control */
     euler_model(U_old,V_old,U2,V2,wu);  /* regular time step */
     int_iter = 0;
     local_error = 1.0e30;
     factor_etol = 1.0;
     temp =1.0;

     while ( temp > 0.0 && fix_T != 'y' )
       {
       if ( ++ int_iter > 100 || factor_etol < 1.0e-10 )break;
       wu /= 2.0;
       euler_model(U_old,V_old,U1H,V1H,wu); /* 1st time with half
                        step size    */
       euler_model(U1H,V1H,U,V,wu);       /* 2nd time with half
                        step size */
       local_error = est_local_error(U2,U); /* estimate local error */
       E_new = find_E(0,V);
       temp =local_error - etol/(Time_final*1.0);
```

```
        if (batch ==0 )
          { printf("interal iter= %d, energy= %g,",++int_iter, E_new);
          printf(" local error= %e, wu= %g\n",local_error,wu);            }
        for (i=0; i < N_SQ; ++i)
          { U2[i] = U1H[i];
            V2[i] = V1H[i]; }
      }
      if ( fix_T == 'y' )
        { for ( i=0; i < N_SQ; ++i){U[i] = U2[i]; V[i] = V2[i]; } }
      if ( comp_net == 'y')
       del_sym_inv(U,V,1);
       else
       del_sym_sol(U,V,1);
      time += wu;
printf("iter= %d time= %e ", iter_count, time );
      /*printf("After");  print_V(V); */
                        /* terminating criteria by  E updating */
      if ( fabs( E_new - E_old ) < 1.0e-6)
         ++E_eq;
       else
         E_eq = 0;
      if ( E_eq > 10 )terminate =1;
      E_old = find_E(1,V);
      if ( batch == 0 || terminate == 1 ) print_V(V);
      if ((iter_count / 100) * 100 - iter_count == 0) print_V(V);
      for (i=0; i < N_SQ; ++i )
        {U_old[i] = U[i];
         V_old[i] = V[i]; }
      if ( fix_T != 'y')
        { if (local_error < 1.0e-10)   /* next step wu */
```

```
            local_error =1.0e-10;
            wu *= pow (etol/(local_error*Time_final), 1.0 );
            if ( wu > 1.0e-3 )wu=1.0e-3;
          }
          if ( terminate == 1)break;
 }
 output(U,V);
 printf("\nWould you like more iterations ? : ");
       while ( iterate = getchar() )
        if ( iterate == 'y' || iterate == 'n')break;
       E_eq = 0;
}

} /* end of main program */
/***************filename = declarat.c ****************************
***************** declarations of global variables **************
***************** "include" files and #defines ******************
*************************************************************************/
#define N_bit  4        /* number of bits */
#include <math.h>   /* math declarations */
#include <stdio.h> /* standard io declarations */

int n,        /* weighting  constants */
   V_in[N_bit], city[N_bit],   /* vector inpt for distance
     calculation and posn of cities */
   batch=0, terminate_flag =0,  /* flags */
   iter_count,          /* iteration count */
   iteration_lmt;          /* maximum iteration limit */

char iterate = 'y', fix_T;
```

```
double  Uo,             /* gain */
    T[N_bit][N_bit], Tc[N_bit], /* hopfield and correction synapse
                    matrices , rows for city, col for posn. */
    TS[N_bit], TR[N_bit], /* input signal synapse, reference synapse */
    I[N_bit], Ic[N_bit],   /* input and correction currents */
    time =0.0,
    etol,           /* local error tolerance */
    VR = -1.0, VS,      /* reference voltage, input signal */
    wu, wc;

#include "initial.c"
#include "setup.c"
#include "update.c"
#include "output.c"
#include "rk.c"

/***************filename = initial.c *********************************/

/***** ASK_INITIAL_VALUES *****/
/***** prompt user for the constants *****/
ask_initial_values()
{
  printf("Amplifier gain, Uo : ");
  scanf("%le", &Uo);

  printf("Initial time step, wu : ");
  scanf("%le", &wu);

  printf("Correction weighting factor, wc : ");
```

```
scanf("%le", &wc);

printf("Error tolerance, etol : ");
scanf("%le", &etol);

printf("\nBatch ? Yes/1 \n");
scanf("%d", &batch);

printf("\nNumber of iterations : ");
scanf("%d", &iteration_lmt);

printf("\nFixed Time Step ? y or n");
while ( fix_T = getchar() )if ( fix_T == 'y' || fix_T == 'n')break;

/*** start to print title & information ***/
printf("\n ***************************** ");
printf("\n\n Version 1, Jan. 22 1990\n");
printf("\nAmplifier Gain (Uo) = %e",Uo);
if ( wc == 0.0 )
   printf(" No Correction Mode \n");
 else
printf(" Correction Coefficient (wc) = %e\n",wc);
printf("initial step size (wu) = %e",wu);
if ( fix_T == 'y')
   printf("\nfixed time step mode \n");
 else
{ printf("\nadaptive time step mode \n");
 printf("error tolerance (etol) = %e\n", etol); }
if ( batch == 1)
   printf("Batch mode\n");
```

```
else
 printf("Interactive mode\n");
 printf("Number of iterations (iteration_lmt) = %d\n",iteration_lmt);
}
/*****************************************************************/
/***** INIT_SOLN_O *****/
/***** initializes the output vector to prespecified values ****/
init_soln_o(V)
 double V[N_bit];
{  char c = 'A';
 int i, ii;

 printf("Choose the initial condition sets.\n");
 printf(" 0 --- reset all conditions \n");
 printf(" 1 --- randomly generated condition \n");
  while ( c = getchar() )
 if ( c == '0' || c == '1') break;
 switch ( c )
  {
   case '0':    /* reset all conditions */
       {
       for ( i=0; i < N_bit; ++i)V[i] = 0.0;  /* initialize */
       break;
       }
   case '1':    /* randomly generated output */
       {
       printf("Type the seed for random number generator");
       scanf("%d", &i);
       for ( ii = 0; ii < i; ++ii) random_gen(V);
 break;
```

```
        }
    default:
      break;
   }
}
/*************************************************************************/
random_gen(V)
double V[N_bit];
{  int i;

 for (i=0; i < N_bit; ++i)
  {
   V[i] = rand();
   while ( V[i] > 1.0 ) V[i] /= 10.0;
/*    printf(" V[%d] = %f;",i,V[i]);
   if ( (i+1) % 4 == 0 )printf("\n");
*/
  }
}
/************ filename = setup.c ********************************
************* Constant matrices such as T, delta ****************
************* and Tc are set up in this section  ****************
*************************************************************/

/***** CRE_T *****/
/***** creates matrix with connection values *****/
cre_T()
{
int i, j;
```

```
for ( i=0; i < N_bit; ++i)
for ( j=0; j < N_bit; ++j)
  {
   if ( i == j )
 T[i][j] = 0.0;
      else
        T[i][j] = - pow (2.0 , (double) (i + j ));
  }
}
/*****************************************************************/
/***** CRE_TC *****/
/***** connection weights for correction term *****/
cre_Tc()
{
 int i, j;
 char dummy, est_avg_w;
 double double_i, temp;

 if ( wc > 0.0 )
   {
 for (i = 0; i < N_bit; i++ )
    {
    Tc[i] = 0.0;
    for (j = 0; j < i ; j++ )
       Tc[i] = (0.5)*T[i][j];
    }
}
}
/*****************************************************************/
/***** CRE_I *****/
```

```
/***** create vector of input currents *****/
cre_I()
{  int i;

 for (i = 0; i < N_bit; i++ )
   { TS[i] = pow(2.0 , (double) i);
   TR[i] = pow (2.0 , (double) (2*i -1) );
   I[i] = TS[i] * VS + TR[i] * VR;
/*    printf("\nTS[%d]=%g, TR[%d]=%g, I[%d]=%g",i,TS[i],i,TR[i],i,I[i]); */
   }
}
/************ filename = update.c ********************************
************* update procedures for input and output ************
********** voltages, input current and correction current *******
*****************************************************************/

/***** FIND_U *****/
/***** update the input voltage vector *****/
find_U(U, V)
 double U[N_bit], V[N_bit];
{  int i, m1, n1;

 for (m1 = 0; m1 < N_bit ; m1++ )
   {     U[m1]=0.0;
    for (n1 = 0; n1 < N_bit ; n1++ )U[m1] += T[m1][n1] * V[n1];}
    U[m1] += I[m1];
                    /* adding correction current */
    if ( wc != 0.0 ) U[m1] += wc * Ic[m1];
   }
}
```

```
/*****************************************************************/
/***** FIND_V *****/
/***** updates the output voltage vector *****/
find_V(U,V)
 double U[N_bit], V[N_bit];
{  int n1;
 double term1;

 for (n1 = 0; n1 < N_bit; n1++)
    {term1 = U[n1]/Uo ;
     V[n1] = (1.0/2.0)*(1 + (tanh(term1)));      }
}
/*****************************************************************/
/***** FIND_IC *****/
/***** calculate correction current vector *****/
find_Ic(V)
 double V[N_bit];
{  int i, j;
 double gout;

 for (i = 0 ; i < N_bit; i++ )
   {     g(V[i], &gout);
    Ic[i] = Tc[i] * ( 2.0* gout - 1.0 );    }
}
/*******************/
g(x, out)  /* transfer function of correction logic */
    double x, *out;
{
 if (x > 1.0e-18 )
   *out =1.0;
```

```
 else
   *out = 1.0e18 * x;
}
/*************************************************/
double find_E(flag_prt,V)
 int flag_prt;
 double V[N_bit];
{  int x, y, i, j, index1, index2;
 double terma=0.0, termb=0.0, E=0.0;

  for ( i = 0 ; i < N_bit ; i++ ) /* first term of energy function */
     for (j = 0 ; j < N_bit ; j++ )
 terma += T[i][j] * V[i] * V[j];

  for ( i = 0 ; i < N_bit ; i++)  /* second term of energy function */
       termb += I[i] * V[i];

 E =- 0.5 * terma - termb;
 if ( flag_prt == 1) printf("  E = %e \n",E);
 return (E);
}
/************************* filename = output.c  *****************
****************** printing the results *************************
*****************************************************************/

/***** OUTPUT *****/
/***** summarizes the output *****/
output(U,V)
 double U[N_bit], V[N_bit];
{  char  pT, pI, pU, pV, dummy;
```

```
printf("\nPrint T matrix ? : ");
dummy = getchar();
pT = getchar();
if (pT == 'y')
  print_T();
printf("\nPrint I matrix ? : ");
dummy = getchar();
pI = getchar();
if (pI == 'y')
  print_I();
printf("\nPrint U vector ? : ");
dummy = getchar();
pU = getchar();
if (pU == 'y')
  print_U(U);
printf("\nPrint V vector ? : ");
dummy = getchar();
pV = getchar();
if (pV == 'y')
  print_V(V);
}
/*********************************************************************
************* printing the T vector (connections) **************
*********************************************************************/
print_T()
{ int i,j;

 printf("****************** T - matrix ******************\n\n\n");
 for (i = 0; i < N_bit; i++ )
```

```
  {     for (j = 0; j < N_bit; j++ )
    printf("%g ", T[i][j]);
    printf("\n\n");    }
 printf("*****************************************************\n\n");
}
/*****************************************************************/
print_I()
{ int i;

 printf("******************* I - matrix ******************\n\n\n");
 for (i = 0; i < N_bit; i++)   printf("%g ", I[i]);
 printf("\n\n*****************************************************\n\n");
}
/*****************************************************************/
print_U(U)
 double U[N_bit];
{ int i;
 printf("\nU - matrix : ");
 for (i = 0; i < N_bit; i++)
   {     if ( (i%N_bit) == 0 )
     printf("\n");
    printf("%.3g ", U[i]);    }
 printf("\n*****************************************************\n");
}
/*****************************************************************/
print_V(V)
 double V[N_bit];
{
 int i;
 printf("\nV - matrix :");
```

```
 for (i = 0; i < N_bit; i++)
  {  if ( (i%N_bit) == 0)    printf("\n");
    printf("%.3g ", V[i]);    }
 printf("\n*****************************************************\n"); }

/************************* filename = rk.c ***********************/
/* Runge-Kutta 4-th method */
rk4_model(Up,Vo,U,V,H)
 double Up[N_bit], Vo[N_bit], U[N_bit], V[N_bit], H;
 { int i;
  double U1[N_bit], U2[N_bit], U3[N_bit], U4[N_bit], UT[N_bit];

  /* first */
  find_Ic(Vo);
  find_U(UT,Vo);  /* calculate next input voltage */
              /* UT : output, Vo : input      */
  for (i=0; i < N_bit; ++i)
   { U1[i] = H * UT[i];
     UT[i] = Up[i] + 0.5 * U1[i];      }
  find_V(UT,V,1);   /* update outputs */
              /* UT : input, V : output       */
  /* second */
  find_Ic(V);
  find_U(UT,V);
  for ( i=0; i < N_bit; ++i)
   {  U2[i] = H * UT[i];
      UT[i] = Up[i] + 0.5 * U2[i];      }
  find_V(UT,V,1);
  /* third */
  find_Ic(V);
```

```
  find_U(UT,V);
  for ( i=0; i < N_bit; ++i)
   {
    U3[i] = H * UT[i];
    UT[i] = Up[i] + U3[i];
   }
  find_V(UT,V,1);
 /* fourth */
  find_Ic(V);
  find_U(UT,V);
  for ( i=0; i < N_bit; ++i) U4[i] = H * UT[i];

 /*** update ***/
  for ( i = 0; i < N_bit; ++i)
    {   U[i] = Up[i] + (U1[i] + 2.0*U2[i] + 2.0*U3[i] + U4[i])/6.0;    }
  find_V(U,V,1);
}
/*************************************************************************/
euler_model(Up,Vo,U,V,H)
 double Up[N_bit], Vo[N_bit], U[N_bit], V[N_bit], H;
 {
  int i;
  double UT[N_bit];

  if ( wc != 0.0 ) find_Ic(Vo);
  find_U(UT,Vo);  /* calculate next input voltage */
            /* UT : output, Vo : input     */
  for (i=0; i < N_bit; ++i)
   {
    U[i] = Up[i] + H * UT[i];
```

```
    }
  find_V(U,V);  /* update outputs */
            /* UT : input, V : output      */
}

/*************************************************************************/
/*  Estimate local error ( Richardson Extrapolation )  */
double est_local_error(V1, V2)
 double V1[N_bit], V2[N_bit];
 {
  int i;
  double subi, max = 1.0e-30;

  for ( i= 1; i < N_bit; ++i)
   {
    subi = fabs ( ( V1[i] - V2[i] ));
    if ( subi > max ) max= subi;
   }
  return (max);
 }
```

Appendix B

Non-Saturated Input Stage for Large Dynamic Range in The Synapse Circuit

The differential pair structure shown in Fig. A.1 has been widely used in the design of amplifier. The usefulness of the differential pair stems from wide common-mode input range (CMIR) and excellent common-mode rejection ratio (CMRR). However, the dynamic input range of the differential pair is usually very limited. Since one transistor of the differential pair is cut off when a large differential input signal is applied, the corresponding amplifier response is limited by the input stage behavior.

Figure A.2 shows an improved version of the MOS input structure. Since the second differential pair is moved away from the current mirror, the CMIR becomes very symmetrical. With all transistors in the input stage biased in the saturation region, the governing equations of the non-saturated input differential pair can be derived from Kirchoff's Voltage Law,

$$V_d \equiv V^+ - V^- = V_{GS1} - V_{GS6} + V_{GS4} + V_{GS2} \tag{A.1}$$

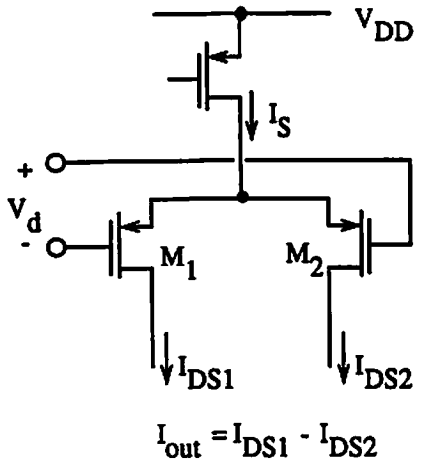
and

$$V_{GS3} - V_{GS5} = V_{GS4} - V_{GS6} \,. \tag{A.2}$$

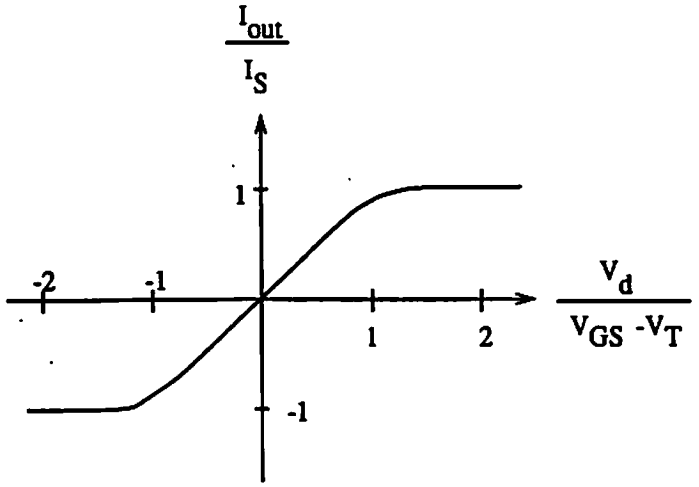
Here, V_d is the differential input voltage. If the threshold voltage mismatch and transconductance coefficient mismatch are neglected, V_d can be expressed as

$$V_d = \frac{\sqrt{I_{DS1}} - \sqrt{I_{DS2}} + \sqrt{I_{DS6}} - \sqrt{I_{DS4}}}{\sqrt{K/2}} \,, \tag{A.3}$$

Bang W. Lee and Bing J. Sheu, "A high-speed CMOS amplifier with dynamic frequency compensation," *1990 Custom Integrated Circuits Conference*, May, 1990.
Bang W. Lee and Bing J. Sheu, "A high slew-rate CMOS amplifier for analog signal processing," *IEEE Jour. of Solid-State Circuits*, to appear.



(a)



(b)

Fig. A.1 Conventional input differential pair in MOS amplifier.
(a) Circuit schematic.
(b) Transfer characteristics.

where $K = \mu C_{ox} W/L$. Here, device aspect ratios are chosen to achieve the same K value for transistors M_1 to M_6. By Kirchoff's Current Law, transistor drain currents are governed by the following relationships,

$$I_{DS1} = I_{DS3} + I_{DS6} , \tag{A.4}$$

$$I_{DS2} = I_{DS4} + I_{DS5} , \tag{A.5}$$

and

$$I_{DS5} = I_{DS6} = I_S . \tag{A.6}$$

Here, I_S is the bias current. A more compact relationship among I_{DS1}, I_{DS2}, and V_d can be obtained,

$$\frac{I_{DS2}}{I_S} = \left[2 - \sqrt{\frac{I_{DS1}}{I_S} - 1}\right]^2 + 1 \tag{A.7}$$

and

$$V_d\sqrt{\frac{K}{2I_S}} = -\sqrt{\frac{I_{DS1}}{I_S}} - \sqrt{\frac{I_{DS2}}{I_S}} - 1 + \sqrt{\frac{I_{DS1}}{I_S} - 1} . \tag{A.8}$$

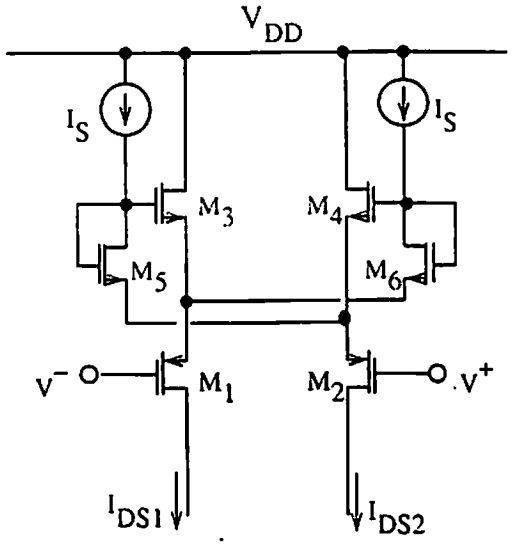
The ranges of I_{DS1} and I_{DS2} to make all input-stage transistors operate in the saturation region are

$$1 \le \frac{I_{DS1}}{I_S} \le 5 \quad \text{and} \quad 1 \le \frac{I_{DS2}}{I_S} \le 5 . \tag{A.9}$$

When the differential input voltage V_d is very large, transistor M_4 will be turned off and transistor M_2 will still remain in the saturation region. Currents I_{DS1} and I_{DS2} are decided by the following equations,

$$I_{DS2} = I_S \tag{A.10}$$

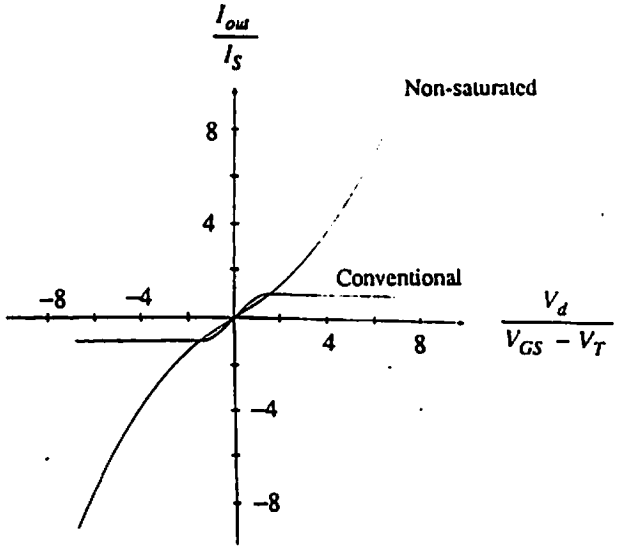
and



$I_{out} = I_{DS1} - I_{DS2}$

$V_d = V^+ - V^-$

(a)



(b)

Fig. A.2 Improved CMOS non-saturated input differential pair.
(a) Circuit Scematic.
(b) Transfer charcteristics.

$$V_d\sqrt{\frac{K}{2I_S}} = -2 + \sqrt{\frac{I_{DS1}}{I_S} - 1} + \sqrt{\frac{I_{DS1}}{I_S}} \quad . \tag{A.11}$$

Here, I_{DS1} is larger than $5I_S$. On the other hand, the currents of transistor M_1 and M_2 are decided by the following equations when a large negative V_d is applied,

$$I_{DS1} = I_S \tag{A.12}$$

and

$$V_d\sqrt{\frac{K}{2I_S}} = -2 + \sqrt{\frac{I_{DS2}}{I_S} - 1} + \sqrt{\frac{I_{DS2}}{I_S}} \quad . \tag{A.13}$$

While the drain current of either transistor M_1 or M_2 is limited to the bias current I_S at a large input signal, the drain current difference of the two transistors is unsaturated as shown in Fig. A.2(b). Hence, the non-saturated differential pair gives wider input dynamic range than the conventional differential pair. When the input voltage V_d is very small, the transconductance gain of the non-saturated input stage, G_m, can be determined from (A.8),

$$G_m \equiv \frac{\partial(I_{DS1} - I_{DS2})}{\partial V_d} = \frac{\sqrt{2}}{1+\sqrt{2}}\sqrt{2I_{DS1}K} \quad . \tag{A.14}$$

The transconductance gain of the non-saturated input stage is smaller than that of a conventional input stage by a factor 0.59. However, the wider input dynamic range of the non-saturated input stage can make transient response significantly faster.

By using a large bias current, the dynamic range of a programmable synapse using conventional differential-pair stage can be also increased. Since the number of synapses is square of that of neurons in a fully-connected neural network, total power dissipation is also squarely proportional to the synapse dynamic range. For a large dynamic range, the differential input transistors are usually very long channel. Notice that aspect ratio of the transistor used in Chapter 4 is 3/39. On the other hand, the dynamic range of a

synapse using the non-saturated input stage is independent of the bias current. In addition, the synapse cell size can be reduced because transistors with very long channel are not necessary in the non-saturated input stage.