# USC–SIPI REPORT #271

## An Efficient Digital VLSI Neural Processing Element Design for Image Processing

by

**Josephine Chia-Fen Chang**

**October 1994**

Signal and Image Processing Institute
**UNIVERSITY OF SOUTHERN CALIFORNIA**
Department of Electrical Engineering-Systems
3740 McClintock Avenue, Room 404
Los Angeles, CA 90089-2564 U.S.A.

To My Parents,

*Mau-Hsiung Chang*

and

*Fu-Hwa Lin*,

for their love and encouragement.

## Acknowledgements

I would like to express my deepest thanks to my research advisor Professor Bing J. Sheu for his guidance and support throughout the course of my Ph.D. research work. I wish to extend my sincere appreciation to Professor Murray Gershenzon, Chairman of Materials Science and Engineering Department, and Professor C.-C. Jay Kuo for serving as my dissertation committee members. I would also like to thank them along with Professor Irving Reed, Professor Richard Leahy and Professor Richard Nottenburg for serving on my qualifying examination committee.

I am very grateful to Professor Leonard Silverman, Dean of the Engineering School; Professor Hans H. Kuehl, Chairman of the Electrical Engineering - Electrophysics Department; Professor Melvin A. Breuer, Chairman of the Electrical Engineering - Systems Department; Professor Kai Hwang, Professor Alice C. Parker, Professor Sidney A. Wielin, Professor Alvin Despain, Ms. Ramona Gordon, Ms. Anna Fong, Ms. Gloria Halfacre and Ms. Susan Moore in the Electrical Engineering Program, for providing me the great research environment for my Ph.D. study at the University of Southern California (USC). This research work was conducted through connections with several research organizations at USC including Center for Neural Engineering (CNE), Signal and Image Processing Institute (SIPI), Center for Photonic Technology (CPT), and MOSIS Service of Information Science Institute (ISI). Very valuable suggesting opinions from Professor Jerry M. Mendel, Professor Michael A. Arbib, Professor Bart Kosko and Professor Ted Berger were highly appreciated.

Discussions with graduated doctoral colleagues from VLSI Signal Processing Laboratory were truly valuable, including Dr. Bang W. Lee and Dr. Joongho Choi on VLSI neural chips design, and Dr. Ji-Chien Lee on digital data bus design, Dr. Wen-Jay Hsu

and Dr. Sudhir Gowda on circuit simulation, and Dr. Wai-Chi Fang on image processing. Many thanks to Sa H. Bang and Oscal T.-C. Chen for helping to obtain some simulation results. I also thank Robert C.-H. Chang, Tony H.-Y. Wu, and Vincent W.-J. Wang for managing the computing facility. Interactions with Hiroto Okada and Barton Sano were also very useful.

Family support helped me to finish this dissertation smoothly. I would like to thank my parents, Mau-Hsiung Chang and Fu-Hwa Lin for their love and understanding, my brother Hung-Ming Chang for his encouragement. I am thankful to my cousins Professor Hung-Chun Chang of National Taiwan University for his inspiration of my study in USC; Chia-Lin Chang, Shih-Wen Chao in Los Angeles, and Boa-Chen Yu, Hung-Wen Chang in Berkeley for their continuous kind help since I came to U.S.A.

# Contents

**List Of Tables** vii

**List Of Figures** viii

**Abstract** xi

**1 Introduction** 1

1.1 Intelligent Machines . . . . . 2

1.2 Promises of VLSI . . . . . 3
1.2.1 Trends of VLSI Technologies . . . . . 5
1.2.1.1 Microprocessors . . . . . 5
1.2.1.2 Memories . . . . . 8
1.2.1.3 Power Consumption . . . . . 10
1.2.1.4 Packaging . . . . . 12

1.3 Artificial Neural Networks . . . . . 13

1.4 Hardware-Software Codesign . . . . . 14

1.5 Organization of This Dissertation . . . . . 15

**2 Neural Network Models** 16

2.1 Biological Neural Networks . . . . . 17

2.2 Artificial Neural Networks . . . . . 18
2.2.1 Data Representation . . . . . 22
2.2.1.1 Two-Level Neuron . . . . . 23
2.2.1.2 Multi-Level Neuron . . . . . 24
2.2.2 Major Network Architectures . . . . . 24
2.2.2.1 Iterative Networks . . . . . 25

2.2.2.2 Multi-Layer Perceptron Networks . . . . . 26
2.2.2.3 Kohonen Self-Organizing Feature Map . . . . . 28
2.2.2.4 Comparison . . . . . 30

**3 Multi-Level Neural Networks with Optimal Solutions** **31**

3.1 Simulated Annealing . . . . . 32

3.2 Paralleled Hardware Annealing . . . . . 33

3.3 Analysis Results . . . . . 39

**4 Digital Neural Processing Element** **45**

4.1 Hardware-Software Codesign Methodology . . . . . 45

4.2 System Architecture . . . . . 47

4.3 Algorithm Mapping . . . . . 50
4.3.1 Back-propagation Learning . . . . . 50
4.3.2 Self-organizing Feature Map . . . . . 53

4.4 Processing Element Design . . . . . 54
4.4.1 Chip Architecture . . . . . 56
4.4.2 VLSI Design Considerations . . . . . 59
4.4.2.1 Design Process . . . . . 59
4.4.2.2 Physical Design Style . . . . . 62
4.4.2.3 Logic Style . . . . . 63
4.4.2.4 Complementary Pass-Transistor Logic . . . . . 63
4.4.3 A Design Example . . . . . 65
4.4.3.1 Clock Distribution Scheme . . . . . 65
4.4.3.2 Computing Units . . . . . 66
4.4.3.3 Cache Memory . . . . . 72
4.4.3.4 Register File . . . . . 73
4.4.3.5 Input/Output Buffer and Microcodes . . . . . 74
4.4.3.6 Power Estimation . . . . . 74
4.4.3.7 Layout Design . . . . . 76

**5 Neural Network Applications** **81**

5.1 Printed Character Recognition . . . . . 82

5.2 Image Compression . . . . . 86

6 Impact of Dissertation Work and Future Directions 91

Reference List 93

Appendix A: Publications Out of the Dissertation Work 99

# List Of Tables


2.1 Comparison between a human brain and artificial neural networks. . . . . 21

2.2 Comparison of various neural network models. . . . . 30

3.1 Eigenvalues of a 6-bit A/D decision network. . . . . 38

3.2 Eigenvalues of an 8-bit A/D decision network. . . . . 38

3.3 Critical amplifier gains of a 6-bit A/D decision network. . . . . 40

3.4 Critical amplifier gains of an 8-bit A/D decision network. . . . . 40

4.1 Comparison of digital electronic design approaches. . . . . 55

4.2 Performance of selective neural network hardware examples. . . . . 57

4.3 Comparison of different design styles. . . . . 62

4.4 Characteristics of a PE. . . . . 78

5.1 A compact instruction set for artificial neural networks. . . . . 83

5.2 Descriptions of the 43 Mcodes. . . . . 84

5.3 Profile of the instruction set for a 3-layer neural network using back-propagation learning rule after 1,000 iterations. . . . . 86

5.4 Profile of the instruction set for a self-organizing network after 4,096 iterations. 89

# List Of Figures


1.1 An integrated intelligent information processing system with multi-media capability. . . . . 3

1.2 Trend of transistor feature size in a microprocessor. . . . . 6

1.3 Trend of microprocessor chip size. . . . . 7

1.4 Trend of number of transistors in a microprocessor. . . . . 7

1.5 Trend of clock rates of microprocessors. . . . . 8

1.6 Trend of performance of microprocessors. . . . . 9

1.7 Trend of transistor feature sizes of SRAM. . . . . 9

1.8 Trend of transistor feature sizes of DRAM. . . . . 10

1.9 Trend of SRAM cell size. . . . . 11

1.10 Trend of DRAM cell size. . . . . 11

1.11 Trend of power consumption of microprocessor and digital signal processors. 12

1.12 Trend of pin-count of a chip. . . . . 13

2.1 Biological Neuron. . . . . 18

2.2 Architecture of a typical three-layer fully-connected neural network. . . . . 23

2.3 Architecture of a Hopfield network with two-level neurons. . . . . 25

2.4 A basic structure of the self-organizing feature map. . . . . 29

3.1 A Hopfield 6-bit neural A/D decision network with 4-level neurons. . . . . 34

3.2 Transfer function of a 6-bit A/D decision network with. 4-level neurons. . . 36

3.3 Plots of the 4-level neuron functions with different controlling gains used in a 3-neuron A/D decision network. (a) Least-significant neuron. (b) Most-significant neuron. . . . . 37

3.4 Time domain responses of the voltages at the neuron input nodes in the A/D decision network when the analog input value $x_a$ is 7.9. (a) Neuron gain. (b) Neuron 0. (c) Neuron 1. (d) Neuron 2. . . . . 41

3.5 Time domain responses of the output values of neurons when the analog input value $x_a$ is 7.9. (a) Neuron 0. (b) Neuron 1. (c) Neuron 2. (d) Combined output. . . . . 42

3.6 The output state trajectory of the A/D decision network. . . . . 43

3.7 (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 100. . . . . 43

3.8 (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 10. . . . . 44

3.9 (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 2.5. . . . . 44

4.1 A general codesign strategy. . . . . 46

4.2 System architecture for systolic networks. (a) 1-D ring-connected array. (b) 2-D mesh-connected matrix. . . . . 49

4.3 Mapping diagram of a back-propagation neural network onto a mesh-connected PE matrix. (a) Feedforward propagation phase. (b) Backward error propagation phase. . . . . 52

4.4 Mapping diagram of a self-organizing neural network onto a mesh-connected PE matrix. . . . . 54

4.5 Building blocks and bus distribution of a processing element. . . . . 58

4.6 The external control circuitry for interfacing with the data cache (a) Between the cache memory and the I/O buffer. (b) Between the cache memory and the four bus segments REG1, REG2, CPU1 and CPU2. . . . . 60

4.7 Required control circuitry (a) For adder. (b) For register file. . . . . 61

4.8 CPL circuit modules. . . . . 64

4.9 Clock distribution scheme with selectively power down capability. . . . . 66

4.10 Block diagram of a 12-bit CLA adder. . . . . 68

4.11 Circuits in block A. . . . . 69

4.12 Circuits in block B. . . . . 70

4.13 Result of 12-bit adder after running irsim simulation. . . . . 71

4.14 Functional block diagram of an 8-bit Wallace multiplier. . . . . 71

4.15 8-bit Wallace multiplication. . . . . 72

4.16 Circuit diagram of a 1-bit CSA unit. . . . . 73

4.17 Result of 8-bit multiplier after running irsim simulation. . . . . 74

4.18 The control circuitry in the cache memory block. . . . . 75

4.19 Schematic diagram of selective circuits in the data cache. . . . . 76

4.20 Timing diagram of memory access in a memory block of 32 word × 2 bit. . 77

4.21 Floorplan of a PE. . . . . 78

4.22 Die photo of the single PE chip fabricated by a 0.8-$\mu$m CMOS technology. . 79

4.23 Performance requirements of several Grand Challenge problems and the relative performance of the proposed multi-PE chips. . . . . 80

5.1 Architecture of an integrated intelligent processing system. . . . . 81

5.2 Application of the back-propagation network to printed character recognition. 1,000 iterations were performed during training. . . . . 87

5.3 Image compression on the multi-PE chip using a self-organizing neural network. (a) The original image. (b) The reconstructed image. . . . . 90

## Abstract

With increasing computing power of a chip implemented by silicon technology, an intelligent machine which possesses basic skills of sensing, signal processing, and moving based on human brain models becomes feasible in the near future. In addition, artificial neural networks have the potential to solve many complex and time-consuming engineering and scientific problems with inherently massively parallel processing architectures. To obtain an optimized solution from a neural network, the paralleled hardware annealing method can be applied. The results on neural networks with multi-level nonlinearities are presented. Hardware using parallel architecture could greatly speedup neural network operations. By using the digital processor design approach, high-precision requirements for neural network algorithms can be easily achieved. A custom-designed digital VLSI processing element (PE) for general-purpose neurocomputing is presented. Detailed communication networks, instruction sets and circuit blocks are created for the one-dimensional ring-connected and two-dimensional mesh-connected systolic array. The reduced instruction set technique and microprogramming skills can be applied to optimize the software control of the processor array. A prototype PE has been designed and fabricated in a $6.19 \times 5.46\ mm^2$ microchip by using the 0.8-$\mu m$ CMOS technology from Hewlett-Packard Company through the MOSIS Service. By arranging the PE layout in a ring-connected array architecture, a 20-PE chip is estimated to occupy a silicon area of $2.09 \times 1.93\ cm^2$ by using a 0.5-$\mu m$ CMOS technology. A digital signal processor chip can be used to broadcast or pipeline microcodes to all processing elements in an array. This neural PE design is suitable for image processing. System-level simulation results of applications including printed character recognition and image compression based on neural network algorithms using the designed hardware are also presented.

# Chapter 1

# Introduction

Since the outburst of first-generation computers, machine computation capabilities based on Von Neumann sequential processing codes have progressed tremendously because of advances of very large scale integration (VLSI) technologies. Continuous reduction of feature sizes and acceleration of operating speeds pack more and more complicated and sophisticated functional units into a smaller silicon real-estate. Multiple copies of hardware targeting for the same function become possible to be grouped together in a system. The domain of a group can be multiple boards, multiple chips on a single board or a single substrate, or multiple functional units on a single chip.

The trend of moving from serial processing to parallel processing follows a natural rule. Brain cells of animals manipulate things by working cooperatively. One example is animal can run and catch, see and hear at the same time. Although how a brain really functions still remains a puzzle to the mankind, the knowledge that we have accumulated through the research on a brain do give good stimuli to the construction of a new-generation parallel processing computer.

The arithmetic calculating power of a computer so far has far prevailed over human's capability. But for complicated signal processing problems where large quantity of equations with massive amount of data need to be solved in real time, computing speed still need to be improved considerably. In addition, the capability of associating one thing to the other by a computer so far pale in comparison with that of a human today. To perform

associativity is essential for conducting intellectual activities. The parallel processing capabilities of artificial neural networks not only accelerate operating speed, but also have a great potential for associativity. Therefore, importance of research on how to take advantage of properties of artificial neural networks for developing an intelligent machine which has both large computing speed and profound associativity cannot be overemphasized.

## 1.1 Intelligent Machines

This dissertation is concerned with the design of components in an intelligent machine. The machine is an integrated information processing system which can communicate with the real world through audio and video channels, together with a large database. A configuration of such an intelligent information processing system is shown in Fig. 1.1. High-speed image processing, vision understanding, and smart graphics provide the systems with visual capabilities. Speech recognition and synthesis techniques provide the systems with audio processing capabilities. The system is also equipped with microsensor and controller units to accomplish physical actions. Such a powerful multi-media data-fusion machine can be used in many places, and will help people in their business, education, and daily lives.

An intelligent machine can be equipped with sensors, processors, and actuators [1]. Sensors interact with the environment to capture information relevant to task objectives; processors perform information processing; actuators execute the resultant outputs from processors. Intelligent machines must be able to operate successfully in unanticipated situations. Some decisions might have to be made with incomplete or uncertain information and competing constraints to solve problems and reach correct conclusions.

Predictably, in the coming years during the 90's, a multi-hundred-billion-dollar computer and information-processing industry is emerging, together with a generation of ubiquitous machine intelligence that works intimately with its human creators. For example, reliable person identification, using pattern-recognition techniques applied to visual and speech patterns, could replace locks and keys in many instances [2].

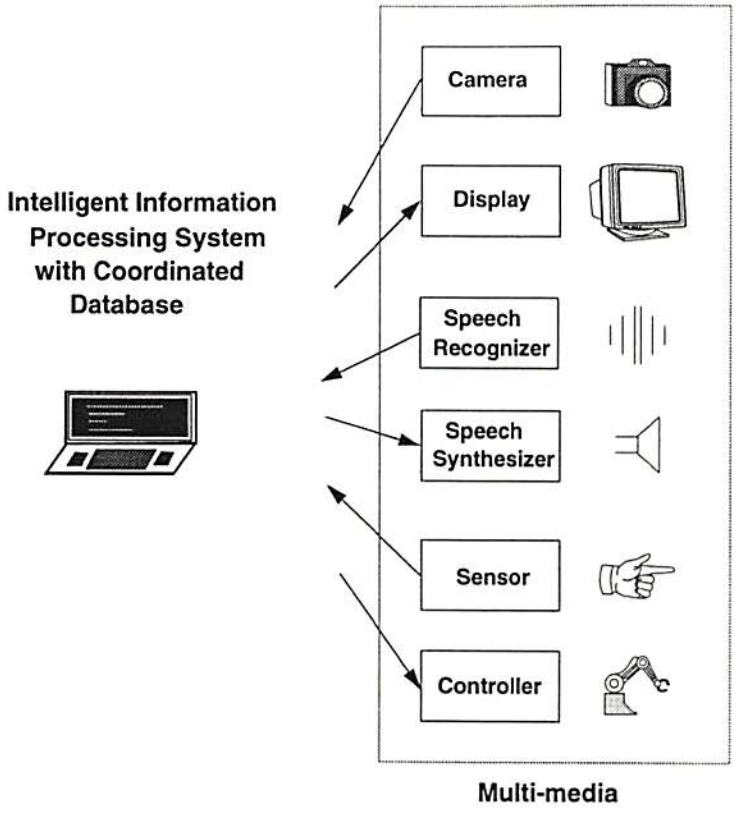



**Figure 1.1: An integrated intelligent information processing system with multi-media capability.**

## 1.2 Promises of VLSI

With the rapid progress of VLSI technologies during the past five years, to construct an intelligent information processing system which integrates several subsystems for processing different functions becomes highly possible. Currently, multi-million transistors can be integrated on a single chip; computer processors can operate at speed in the range of hundred million instructions per second (MIPS). In 1994, the feature size of 0.25 $\mu m$ has become widely used [3]. Engineers and scientists has already been looking into the next step of linewidth being 0.18 $\mu m$ and below that will be needed to build 1G-bit dynamic random access memories (DRAMs).

The range of power supply has decreased from more than 5 volts to less than 2 volts [4]. The original impetus to lower operating voltages was the need to improve the reliability of fine-geometry devices such as DRAMs. Consumers have two basic complaints about the portable machines: they are too heavy, and battery life often falls far short of the all-day operation that would make them practical. Running systems at a lower power supply can reduce power consumption, extend battery life, and even allow the designer to jettison some of the heavy battery cells. Devices available in low-voltage versions include DRAM, flash memory, gate arrays, microprocessors, and digital signal processors. In addition, lower voltages reduce noise and electromagnetic interference (EMI), factors becoming increasingly important as the computer devices move off the desktop into harsher environments. The low-voltage versions typically run slower, however, because the drain-source current is reduced as the square of the gate-source voltage. Internal capacitors require longer time to charge at the lower current, which then must run at a reduced frequency.

The submicron CMOS technology for arithmetic-oriented superchip implementation and monolithic wafer-scale integration also achieves a functional throughput rate of $10^{13}$ gate-Hz/$cm^2$ [5]. The use of VLSI circuits can greatly reduce the physical size and enhance the performance and reliability of microelectronic systems. In the microprocessor domain, continuous progresses on reduced instruction set computers (RISC) enables the introduction of the powerful Intel-i860 chip [6], the SPARC chip from Sun Microsystems Inc.[7], the 400-MIPS Alpha chip from Digital Equipment Corporation (DEC) [8], and the PowerPC 601 from IBM, Motorola, and Apple Computer over the past few years. In 1994, a 500-MHz 32-bit processor was announced by NEC Corporation [9]. The 3.1-million transistor Pentium chip from Intel Corporation [10] represents another design category called complex instruction set computers (CISC). Several powerful microprocessors have been announced out of various technologies such as a 300-MHz bipolar ECL microprocessor from DEC [11] and a 3.3-V 0.6-$\mu m$ BiCMOS superscaler microprocessor from Intel Corporation [12]. In the digital signal processor domain, the TMS320C40 chip from Texas Instruments

Inc.[13] includes 6 communication ports to facilitate various data communication schemes. A recent announced TMS320C80 chip includes 4 digital signal processors and employs programmable MIMD architecture to reach more than 2 billion operations per second [14].

### 1.2.1 Trends of VLSI Technologies

Intense competition between semiconductor manufacturers expedites the pace of technological development. The technology gap between memory LSIs and logic LSIs is becoming smaller, although the advance of process technology becomes difficult. Along with advances in silicon fabrication techniques, the ability to produce micromachines on a silicon substrate is also improved. In this section, progresses of VLSI technologies over the past decade will be reviewed. From the trend, future prospects of VLSI technologies can be drawn.

#### 1.2.1.1 Microprocessors

Microprocessors have been one of the most strongly contested areas. While one decade ago, NMOS technology was the dominant technology, currently, CMOS technology has taken the lead. The feature size of a transistor continue to shrink and submicron technology became available in designing microprocessor since 1991. Recently, 0.4-$\mu m$ technology has been used to implement a microprocessor. Figure 1.2 shows the trend of transistor feature size of a microprocessor. The plus signs in the figure are data mainly collected from the IEEE International Solid-State Circuits Conference. The solid line shown is the line best fits all data, that is, with minimum distances to all collected data. The dashed line is the line best fits data from smallest feature size of every year. The circle signs represent prediction of the next years from the solid line, and the x-mark signs represent prediction of the next years from the dashed line. Figures below will follow similar conventions. From prediction, in the year 2000, a feature size of 0.29 $\mu m$ can be widely used in the production of microprocessors.

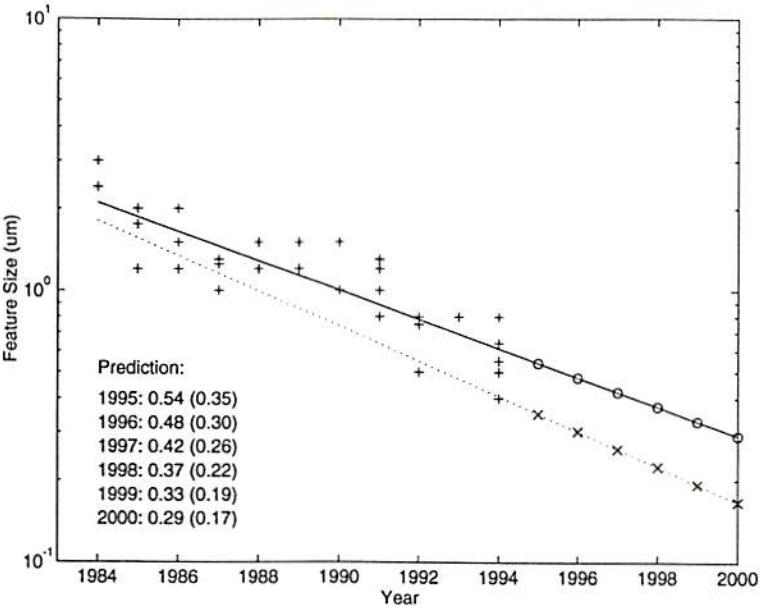



**Figure 1.2: Trend of transistor feature size in a microprocessor.**

On the other hand, the silicon area of a microprocessor chip available has become larger and larger as shown in Fig. 1.3. As a transistor feature size keep on shrinking, and a chip size keep on increasing, the intergration level of a chip becomes enormous. We started to see chips with more than 1 million transistors in 1990. Nowadays, it has become quite common to build million-transistor chips. Predictably, a chip with about 10 million transistors will be quite common in the year of 2000. Figure 1.4 shows the trend of integration level.

Clocking speeds of CMOS microprocessors continuously set tremendous records. Fig. 1.5 shows the increase of clock rates over the past decade, and the prediction of future years up to the year 2000. As Fig. 1.5 shows, the highest achievable clock rate has increased about 18 times over the past decade. From prediction, clock rates can possibly go up to more than 1 GHz by the year 2000. In 1994, low power design becomes a strong design consideration which should be an integral part of the whole design process to keep up with

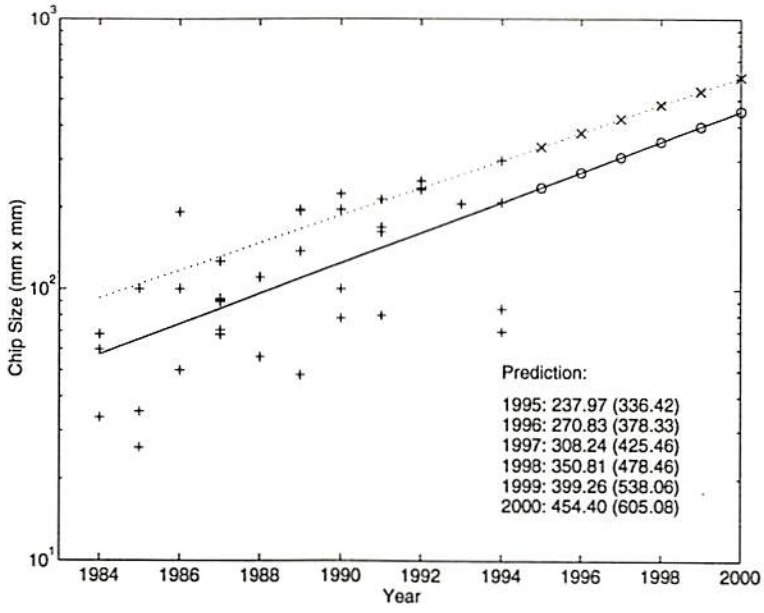



**Figure 1.3: Trend of microprocessor chip size.**

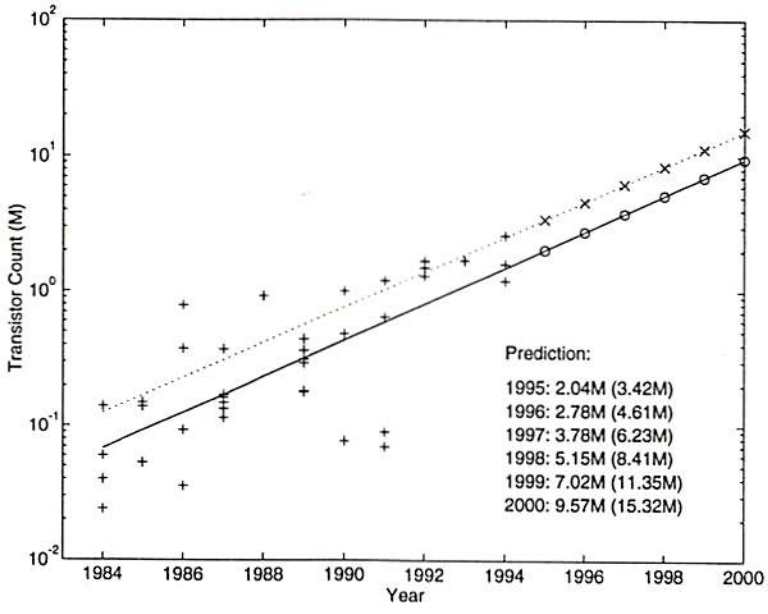



**Figure 1.4: Trend of number of transistors in a microprocessor.**

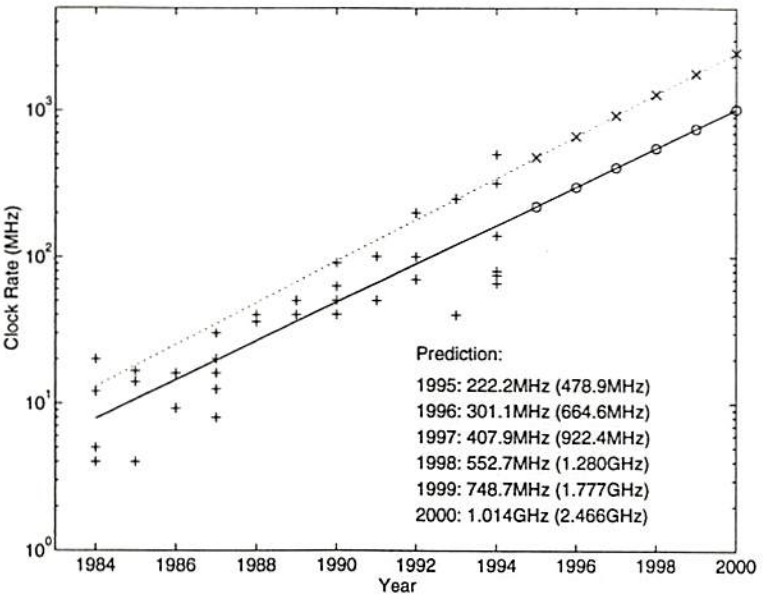



**Figure 1.5: Trend of clock rates of microprocessors.**

the competition. When considering low power design, clock rates drop as data from 1994 show.

With a higher clock rate, more operations can be performed for the same period of time. Figure 1.6 shows the trend of MIPS rates. It is expected by the year 1998, more than 1000 MIPS can be reached, and in the year 2000, about 2700 MIPS rate will be possible.

#### 1.2.1.2 Memories

Feature sizes of memories usually are smaller than those of microprocessors. Engineers continue to exploit smaller devices to construct denser static random access memory (SRAM) and dynamic random access memory (DRAM). Figures 1.7 and 1.8 show the trends and predictions of device feature sizes for SRAM and DRAM, respectively. By looking at the two optimistic lines in both figures, we can find both kinds of memories have possibilities of being constructed by devices under 0.1 $\mu m$ in the year 2000.

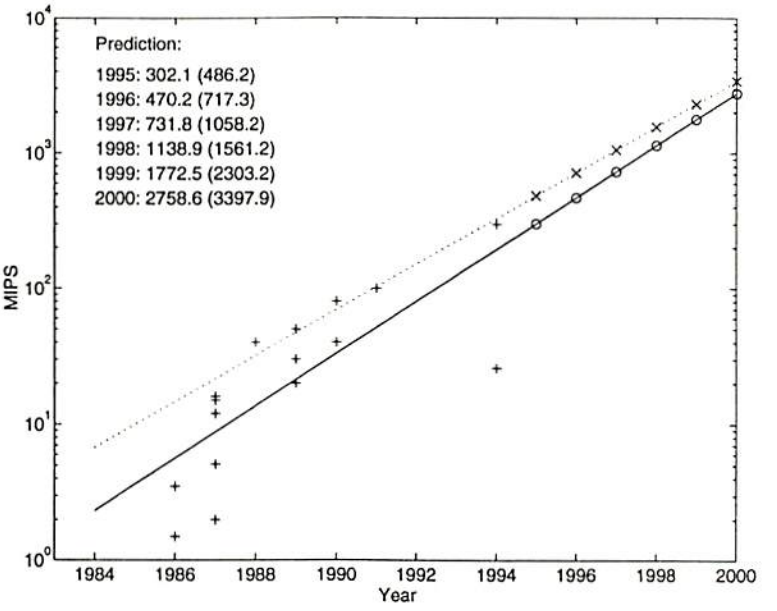



**Figure 1.6: Trend of performance of microprocessors.**

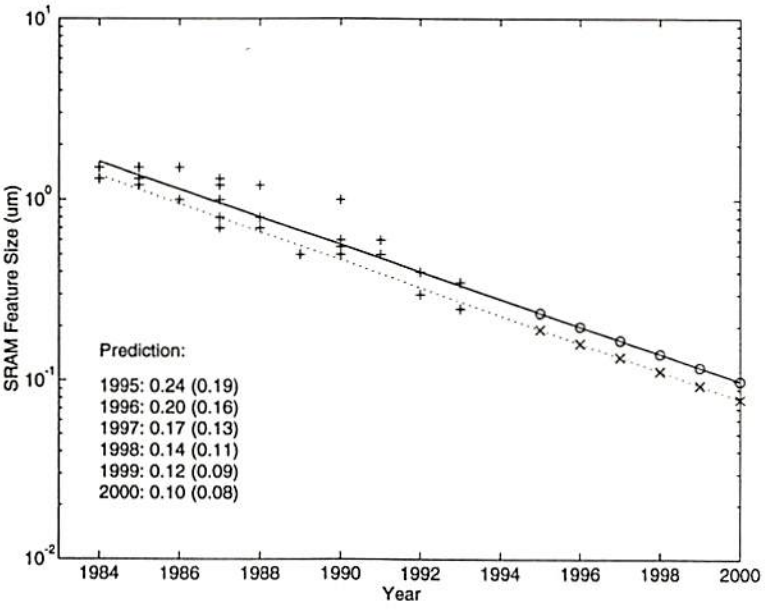



**Figure 1.7: Trend of transistor feature sizes of SRAM.**

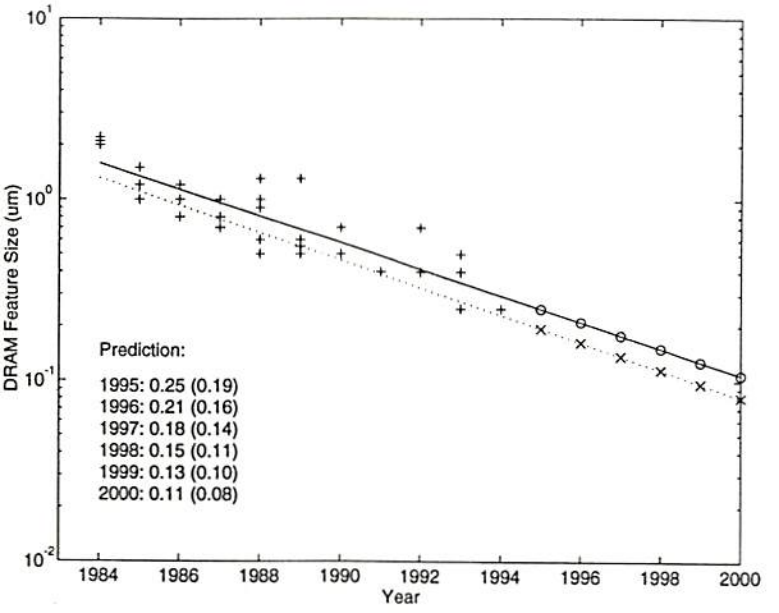



**Figure 1.8: Trend of transistor feature sizes of DRAM.**

From the trends of memory cell sizes as shown in Figs. 1.9 and 1.10, we can find that designers of DRAMs agree in the sizes in 1994. They all design memory cells of 0.72 $\mu m^2$ with 0.25 $\mu m$ technologies. By the year of 2000, a DRAM cell with area smaller than 0.1 $\mu m^2$ can be produced to construct 1G or even 4G-DRAM.

#### 1.2.1.3 Power Consumption

VLSI applications involve both high-end and low-end markets. The high-end markets include high-end workstations with an ultra high-speed CISC/RISC processor, that is, "hot chip". The low-end markets include personal digital consumer products using "cold chip" to realize battery-operated capability. Digital signal processors for digital cellular phones, whose power dissipation is minimized by optimized circuit design are the beginning of the "cold chip". Although the increase rate of clocking speeds will decrease because of supply-voltage scaling and power constraints. Development of "hot chips" will follow the trend discussed previously. On the other hand, the "cold chips" will show a steeper

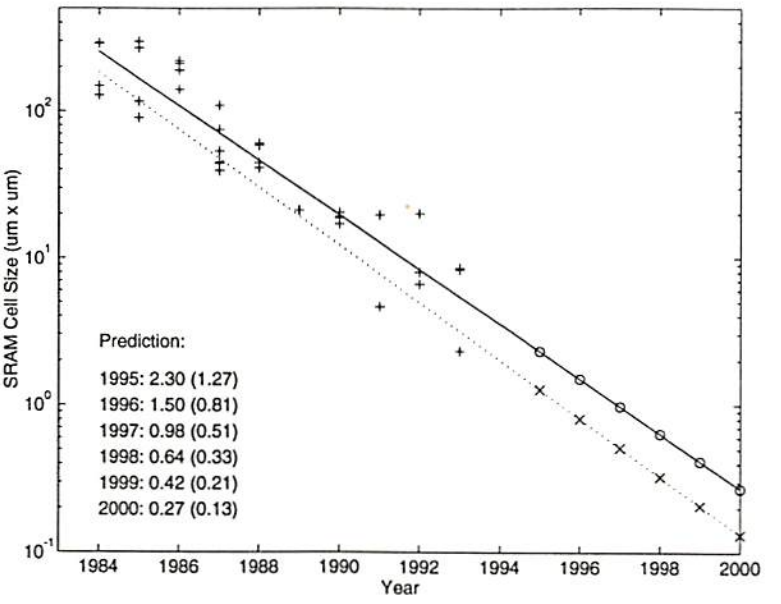



**Figure 1.9: Trend of SRAM cell size.**

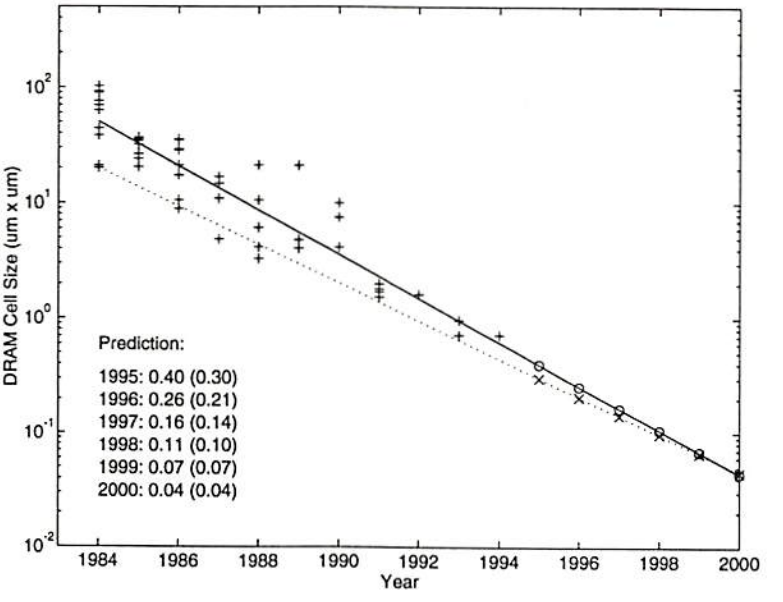



**Figure 1.10: Trend of DRAM cell size.**

slope in the power reduction as shown in Figure 1.11 [15] resulting from low-power circuit techniques, optimized architecture and improved system algorithm.

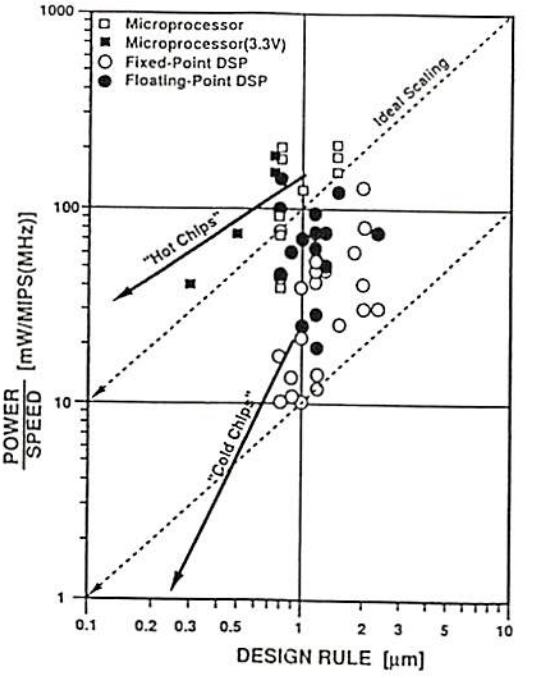



**Figure 1.11: Trend of power consumption of microprocessor and digital signal processors [15].**

#### 1.2.1.4 Packaging

The pin-count of a chip has increased rapidly as Fig. 1.12 shows. Multi-chip modules (MCMs) in a variety of configurations have emerged over the last few years as a possible method to raise system speed and reduce weight and volume [16]. The predominant focus has been in the digital arena where the ever-increasing chip I/O count has led to poor surface area of a board due to conventional packaging inefficiencies. In addition, as the clocking speeds for digital chips move beyond 50 MHz, packaging becomes a major factor limiting overall system speed. Several benefits result from interconnecting complex chips using a fine-pitch MCM technology. First, because MCM technologies allow chips to be placed next to each other, interconnect parasitics due to conventional packaging of single components and board interconnect length are virtually eliminated. As a result, such

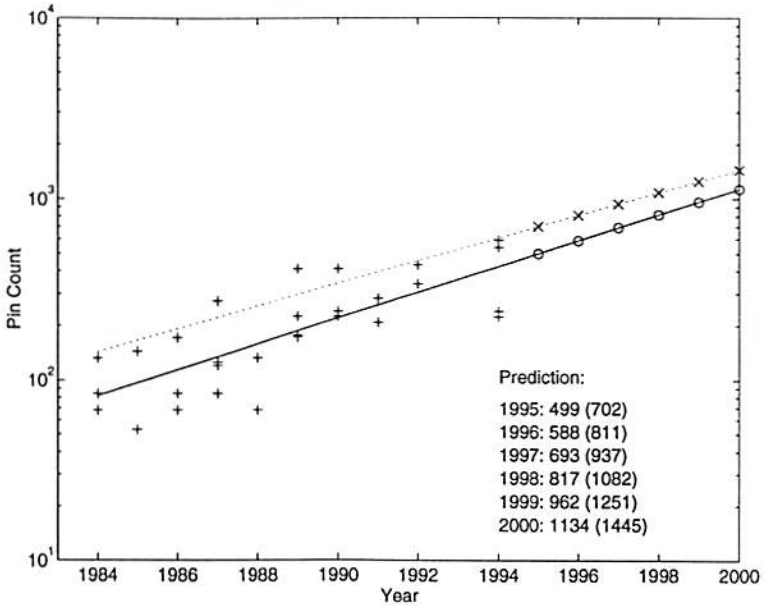



**Figure 1.12: Trend of pin-count of a chip.**

an implementation yields higher operating frequencies. Second, board area and weight are reduced. An example from General Electric Company is a quad digital-signal processor module implemented in the high density interconnect process. This design uses four TMS320C25 processors and contains 36 tile-packed chips in a 1.3 $\times$ 1.3 $in^2$ array. The 12-gram design executes 180 MIPS at clock rates approaching 100 MHz. Implementations of the same design using conventional surface-mount packaging techniques are rated for 40 MHz and are about 5 times larger [17]. Another example is a MCM-based CCITT H.261 codec with 50% reduction in the power dissipation and 98% reduction in the total system area from a board design [18].

## 1.3 Artificial Neural Networks

The capability of processing massive data parallelly by artificial neural networks (ANNs) gives the possibility of generating high-quality signals and images for real-time applications. Also, high performance computing and computer communication networks are becoming

then conducted to justify whether the requirements such as timing constraints are met. If some constraint is violated, some portion of the system must be turned into hardware. The design gradually moves software to hardware. In contrast to this process, a designer can start from a major-hardware system and gradually moves hardware to software according to performance evaluation.

## 1.5 Organization of This Dissertation

The rest of this dissertation is organized as follows.

Chapter 2 introduces the problems in designing a new-generation intelligent information processing machine. The methodologies of applying artificial neural networks to solve these problems will be given together with related work accomplished by other researchers.

Chapter 3 presents the method of applying hardware-based simulated annealing to artificial neural networks for achieving optimal solutions. Analog/Digital (A/D) decision networks constructed from Hopfield-type networks with two-level and multi-level neurons are used in demonstration of the annealing method.

Chapter 4 describes the methodology of using hardware-software codesign techniques in designing neural hardware. Architecture selections of a system and a processor are discussed. Several algorithms of neural networks will be mapped onto the neural system. Details of digital VLSI processing element design is presented.

Chapter 5 covers the software design portion of the processing element. Several real-world applications are demonstrated to show how to use a proposed multi-PE chip.

Chapter 6 summarizes the results of the thesis, points out the impact of the dissertation work, and gives pointers to future research that can be based on this exemplary work at the current status.

Appendix A lists journal and conference publications out of the dissertation work.

increasingly important to scientific advancement, economic competition, and national security. The Federal High Performance Computing and Communications (HPCC) Program has been driven by the recognition that unprecedented computational power and capability is needed to investigate and understand a wide range of scientific and engineering "grand challenge" problems [19]. These are fundamental problems whose solutions are critical to national needs. Examples of grand challenges addressed include: prediction of weather, climate, and global changes; improving research and education communications; and understanding the nature of new materials. The systems which will be discussed in this dissertation will be good candidates for dealing with the HPCC Programs with its fundamental properties of high computing power and high speed.

Different implementation methods have been used for constructing neurocomputers [20]-[22]. Optical and electronic implementations are two main streams. Mature CMOS VLSI technology and existing successfully implemented microcomputers make the electronic implementation immediately attractable. On the other hand, optical implementation has capabilities of achieving high bandwidth, 3-dimensional interconnect ideally. Whereas, the optical approach has several challenges. Efficient weight storage and computation in pure optics is still difficult to do, although improving. Electro-optics has significant conversion problems in that creating photons is hard to do electronically with reasonable cost-performance. Moreover, it is still very expensive to implement optically. Therefore, no volume to bring prices down like the silicon technology [22].

## 1.4 Hardware-Software Codesign

In hardware-software codesign, people consider tradeoffs between hardware and software so as to fulfill behavior and performance goals of a system. Some portions of a system are more suitable to be hardwired to gain higher speed while some portions are preferred to be software programmable. The codesign process can also be considered as a partitioning problem. For example, codesign can start from a system configuration which only consists of software modules on a standard core processor. Performance evaluation of the system is

# Chapter 2

# Neural Network Models

Artificial neural computing has enjoyed a rise in popularity due to advances in VLSI technology and the inability of previous generation computers to quickly and efficiently generalize and learn. They are networks of neurons connected together in layers, processing information asynchronously or synchronously.

Hardware implementation of neural networks utilize a parallel processing structure that has large number of processing elements (PEs) and many interconnections between them. A PE is also sometimes called a processor. A PE performs much simpler functions than a typical central processing unit (CPU) does. In a neural network, each PE is linked to many of its neighbors so that there are the number of interconnections is much larger than that of PEs. The power of a neural network lies in the tremendous number of interconnections.

What has triggered the most interest in neural networks is that models similar to biological nervous systems can actually be made to do useful computations. Furthermore, the capabilities of the resulting systems provide an effective approach to previously unsolved problems. However, neural networks can be difficult to train and unsuitable for some tasks. So why use them? Because they offer valuable characteristics unavailable together elsewhere. First, they can infer subtle, unknown relationships from data. Second, the networks can generalize, meaning they can respond correctly to patterns that are only broadly similar to the original training patterns. Generalization is useful because real-world data is noisy, distorted, and often incomplete. Third, they are nonlinear, that is, they can solve

some complex problems more accurately than linear techniques do. Nonlinear is common, but can be difficult to handle mathematically. Finally, they are highly parallel.

## 2.1 Biological Neural Networks

Neural network architectures are motivated by models of our own brains and nerve cells. A human brain consists of approximately $10^{11}$ computing elements called neurons [23, 24]. They communicate through a connection network of axons and synapses having a density of approximately $10^4$ synapses per neuron. A typical neuron has three major regions: the cell body, which is also called the soma, the axon, and the dendrites as shown in Fig. 2.1 [24]. The soma provides the support functions and structure of the cell. The axon is a branching fiber that carries signals away from the neuron, and the dendrites consist of more branching fibers that receive signals from other nerve cells. The connecting point between an axon and a dendrite is the synapse. The synapse is where the neuron introduces its signal which is a pulse to the neighboring neuron. The synapse is capable of changing a dendrite's local potential in a positive or negative direction, depending on the pulse it transmits. The interneuronal transmission is sometimes electrical but is usually effected by the release of chemical transmitters at the synapse [25]. Therefore, it occurs fairly slowly.

A neuron is able to respond to the total of its inputs aggregated within a short period of time. The neuron's response is generated if the total potential of its membrane reaches a certain level. Incoming impulses can be *excitatory* if they cause the firing of the response, or *inhibitory* if they hinder the firing of the response. If the excitation exceeds the inhibition by the *threshold* amount, which is typically 40 $mV$ [26], a firing occur. The characteristic feature of the biological neuron is that the signals generated do not differ significantly in magnitude; the signal in the nerve fiber is either absent or has the maximum value. In other words, information is transmitted between the nerve cells by means of binary signals.

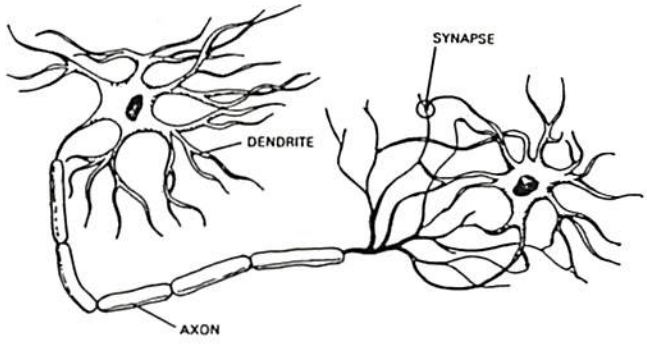



**Figure 2.1: Biological Neuron [24].**

By understanding of the basic concepts of a biological neural network (BNN), it is helpful for the design of a powerful artificial neural network which can solve problems that today's machine cannot accomplish.

## 2.2 Artificial Neural Networks

A traditional Von Neumann machine has a single CPU that performs all of its computations in sequence. A typical CPU is capable of a hundred or more basic commands, including adds, subtracts, loads, and shifts, among others. The commands are executed one at a time, at successive steps of a time clock. In contrast, a neural network processing unit may do only a few commands. Incremental changes are made to parameters associated with interconnections. This simple structure nevertheless provides a neural network with the capabilities to classify and recognize patterns, to perform pattern mapping, and to be useful as a computing tool.

Some of the operations that neural networks perform include [27, 28].

- Classification - an input pattern is passed to the network, and the network produces a representative class as the output.

- Pattern matching - an input pattern is passed to the network, and the network produces the corresponding output pattern.

- Pattern completion - an incomplete pattern is passed to the network, and the network produces an output pattern that has the missing portions of the input pattern filled in.

- Noise removal - a noise-corrupted input pattern is presented to the network, and the network removes some, or all, of the noise and produces a cleaner version of the input pattern as output.

- Optimization - an input pattern representing the initial values for a specific optimization problem is presented to the network, and the network produces a set of variables that represents a solution to the problem.

- Control - an input pattern represents the current state of a controller and the desired response for the controller, and the output is the proper command sequence that will create the desired response.

A comparison of biological and artificial neural networks can give us a clearer picture of what we have achieved and what need to be done in the future. Table 2.1 lists the comparison. In the table, artificial neural networks are considered in terms of digital implementation. The major difference between biological and artificial neural networks is in the complexity of the synapses. The biological synapses contain many components and active processes. The strength of biological synapses maybe affected by several factors such as size and number of synaptic vesicles; content of synaptic vesicles; influx and outflow of ions; release and uptake rates of neurotransmitters, etc [29].

In contrast, artificial neural networks have relatively simple interconnections. For example, during the feedforward phase in a standard multi-layer perceptron network, an operation can be expressed with a simple linear weighted sum combined with a nonlinear sigmoid function which performs thresholding. Other simple calculations may be done,

such as inhibition and competition. Sometimes, additional terms are added to the summation functions, such as a dependency on past changes, or the addition of second-order terms. However, these do not rival the biological synaptic processes in complexity.

BNNs send signals from one unit to another by means of impulse transmission. ANNs can transfer precise scalar values from unit to unit. Biological impulse transmission takes place at any time and its timing is determined by incoming signals. Artificial networks update their parameters periodically, in discrete time steps. Usually the whole network is updated during some specific period of time. Thus artificial networks can be considered to have synchronous updating, whereas in biological systems the updating is asynchronous. Biological neurons update whenever an impulse arrives, and may also have parameters that decay or change between arrivals.

One advantage of artificial networks is that the scalar values that are transferred from unit to unit can be implemented to be relatively precise. In biological neural networks, a single interconnection does not transfer a precise scalar value. If average firing rate is considered to be the value transferred, then it has limited precision, especially over a short time frame. Biological networks have a built-in temporal structure because impulses can occur at any time, and thus may form temporal patterns. The summation in the two systems must be done differently as a result of their different signaling characteristics.

Biological networks have predetermined wiring at the system level. For example, the major fibers and connections in the visual and auditory systems are the same for different individuals; at a more detailed level the circuitry appears to be different for different individuals. For example, animals with exactly the same genes do not have corresponding neurons with the same dendritic branching structure and topology.

ANNs are usually layered and fully interconnected, with all units in a given layer connected to all units in the layers above and below. Artificial networks can also be sparsely interconnected, or have connections removed selectively after training. Biological networks have layered structures. However, the layers are not simple rows of independent units, as in most artificial networks, and the neurons in each layer or cluster tend to be

Table 2.1: Comparison between a human brain and artificial neural networks.

| Element | BNN | ANN |
|---|---|---|
| Organization | network of neurons | network of PEs |
| Components | Dendrites and axons<br>Synapses<br>Summer<br>Threshold | Inputs and outputs<br>Weights<br>Summation function<br>Threshold function |
| Architecture | fixed gross wiring architecture, plus variation in detailed structure | usually fully interconnected |
| Neurons | 100 billion | usually up to 100 |
| Synapses | complex | simple |
| Interconnections per neuron | 10,000 | within 10 |
| Processing | analog | digital |
| Technology | biological | silicon<br>optical<br>molecular |
| Hardware | neuron | switching device |
| Switching speed | 1 msec | 10-100 nsec |
| Updating | continuous or asynchronous | synchronous |
| Learning | as fast as one pass | slow to converge |
| Signal | pulse transmission | activity value and connection strengths |
| Reliability | redundancy | redundancy |

densely interconnected to one another, again in contrast to most artificial networks. The layers in biological systems are not fully interconnected with layers above and below in the simplistic way that is found in many artificial systems, as biological connections may be sparse or may involve more than one synapse. Three-dimensional packing considerations discourage fully interconnected topologies for biological systems, and also pose a constraint in designing neurocomputing hardware.

Feature detectors occur in both biological and artificial systems. Distributed representation and processing is a characteristic of both biological and artificial networks. A pattern mapping, for example, uses the entire network, not just a single location in the network, to determine its output pattern given the input pattern. Redundancy, occurs in both biological and artificial networks, can increase the reliability of a system, allowing it to function even when some of the neural units are destroyed. In addition, redundancy may also counteract sources of noise.

Biological networks have the remarkable property of being able to learn in as little as one training presentation. In contrast, artificial networks usually require hundreds or thousands of training presentations in order for learning to take place, and are usually slow to converge.

### 2.2.1 Data Representation

Neural networks are made out of neuron-like nodes that are arranged in layers. Data as passed through weighted connections between nodes. The networks learn by changing the values of their weights. With suitable weights, a network can model any computable function. Figure 2.2 shows a typical three-layer fully-connected feedforward neural network. A *neuron* and a *synapse* are two fundamental components in an artificial neural network. Depending on the architecture and dynamics of a network, some additional components may be added. The synapse represents connection between two neurons with a weight value indicating the strength of the connection. The neuron does operations on input data and synapse weights.

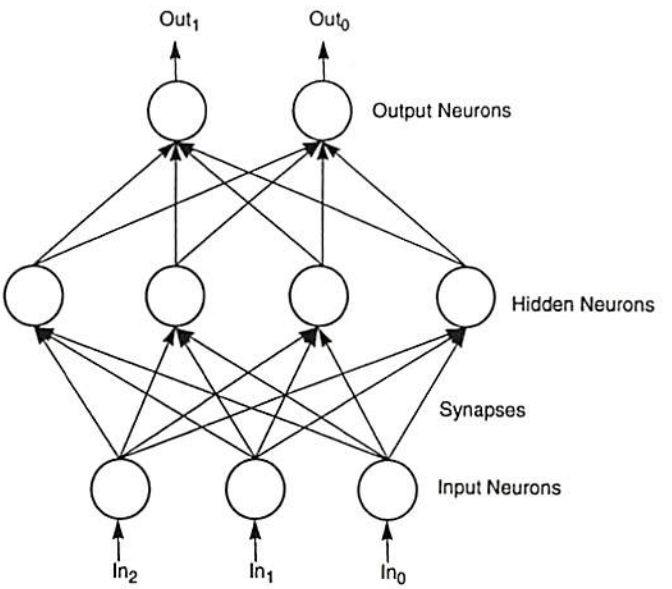



**Figure 2.2: Architecture of a typical three-layer fully-connected neural network.**

#### 2.2.1.1 Two-Level Neuron

A typical two-level neuron can be modeled by

$$S(x) = cf(x - \theta). \tag{2.1}$$

The transfer function $f$ transforms an input value $x$ into one of two output levels. Here, $\theta$ is the threshold value of the neuron, and the scaling factor $c$ is typically one. The transfer function $f$ is continuously and monotonically increasing [25, 30]. The sigmoid function has been widely used in the neural network research community. This function is a bounded differentiable real function that is defined for all real input values, and it has a positive derivative value everywhere. By using the sigmoid function as the transfer function, (2.1) becomes

$$S(x) = \frac{c}{1 + e^{-(x-\theta)}}. \tag{2.2}$$

Most neural networks in the literature use two-level neurons to classify input signals. A two-level neuron produces output with 1-bit accuracy. As the number of input signals

increases, the number of synapse weights increases accordingly. A large number of synapse weights will complicate the interconnection wiring between neuron layers, and significantly limit the size of a network to be implemented by a VLSI chip.

#### 2.2.1.2 Multi-Level Neuron

The output of each multi-level neuron represents multiple bits. Therefore, the total network size of a neural network composed of multi-level neurons can be significantly smaller than a conventional network with two-level neurons. The reduction in network size is a highly desirable feature in VLSI neuroprocessor implementation for large-scale applications. Recently, a neuron circuit performing multi-level transfer function has been proposed [31]. To represent a multi-level neuron with $m$ threshold values, $\theta_0$ to $\theta_{m-1}$, a neuron transfer function of the form [31]

$$M(x) = \sum_{j=0}^{m-1} c_j f_j(x - \theta_j) \tag{2.3}$$

can be used. The scaling factor for the $p$-th level is $c_0 + c_1 + \cdots + c_p$. Therefore, the output level has a step size of $\Delta k_p = c_p$. In addition, $\theta_p < \theta_{p+1}$ holds for $0 \leq p \leq m-1$. The accumulation function $M(x)$ is a monotonically increasing function. Again, by replacing $f_j$ with the sigmoid function, (2.3) becomes

$$M_s(x) = \sum_{j=0}^{m-1} \frac{c_j}{1 + e^{-(x-\theta_j)}}. \tag{2.4}$$

### 2.2.2 Major Network Architectures

Over the decades of the exploitation of biological neural networks, several different types of models have been highly discussed and developed. In this section, some major types of neural network architectures will be described. The way of realizing these networks by digital VLSI will be discussed in later chapters.

#### 2.2.2.1 Iterative Networks

In an iterative network, the output nodes of neurons feedback to the input nodes of neurons through interconnecting weights. Typically, iterative networks are started at some initial state, and then are converged to one of a finite number of stable states. There are three basic goals in the design of iterative networks. First, given any initial state, a network should always converge to some stable state. Second, the stable state to which the network converges should be the one closest to the initial state, as measured by some metric. Third, it should be possible to have as many stable states as desired. Some examples are the Hopfield network [32, 33], the Bidirectional Associative (BAM) network [34], the Brain State in a Box (BSB) network [35].

Figure 2.3 shows the architecture of a Hopfield network with two-level neurons. The output of the network changes until a stable state is reached. The initial state of the output has influence on the final output. Some initial states can cause nonconverging outputs. An external input $x_a$ can be added to the input nodes of neurons to direct the operation of the network.

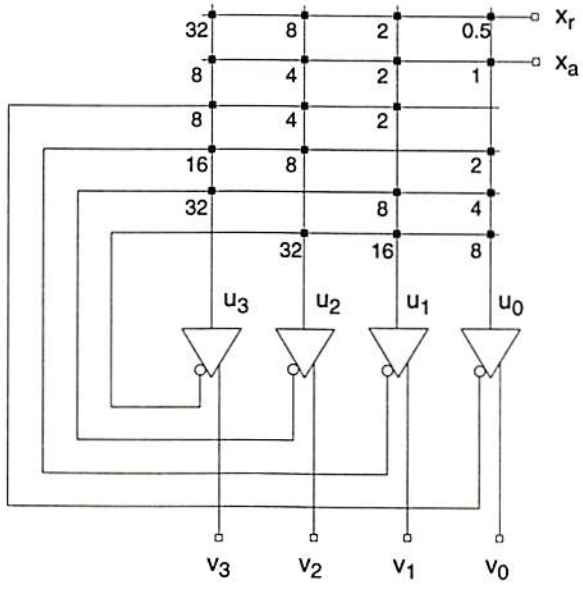



**Figure 2.3: Architecture of a Hopfield network with two-level neurons.**

The Hopfield network has two major limitations when used as a content addressable memory. First, the number of patterns that can be stored and accurately recalled is severely limited. If too many patterns are stored, the network may converge to a novel spurious pattern different from all exemplar patterns. Such a spurious pattern will produce a "no match" output when the network is used as a classifier. Hopfield [36] showed that this occurs infrequently when exemplar patterns are generated randomly and the number of classes is less than 0.15 times the number $N$ of input elements or nodes in the network. Therefore, the number of classes is typically kept well below $0.15N$. A second limitation of the Hopfield network is that an exemplar pattern will be unstable if it shares many bits in common with another exemplar pattern. This problem can be eliminated, and performance can be improved by a number of orthogonalization procedures [30].

#### 2.2.2.2 Multi-Layer Perceptron Networks

The network is usually structured as shown in Fig. 2.2. A neuron in such a network typically multiplies each input by its weight, sums the products, then passes the sum through a nonlinear transfer function to produce an output. The fundamental computation for a neural network is therefore the vector dot-product, and its computational speed depends on executing the underlying multiply-and-accumulate operations efficiently. Each neuron has an extra input called the threshold input, which acts as a reference level or bias for neurons. The intermediate hidden layer enhances the network's ability to model complex functions. The transfer function is usually a sigmoid function. The attenuation at the upper and lower limbs of the S-shape curve constrains the raw sums smoothly within fixed limits. The transfer function also introduces a nonlinearity that further enhances the network's ability to model complex functions [37].

The most popular training algorithm used in these networks is the back-propagation training. Synapse weights are adjusted by the back-propagation technique in the direction opposite to the instantaneous error gradient. Because it is possible to calculate the errors,

the training data must contain a series of input patterns labeled with their target output patterns.

During the training time, a network adjusts its synapse weight values through consecutive cycles of feedforward and back-propagation phases until predefined requirements are reached. During the running time, the trained network uses its weight values to obtain outputs through a feedforward phase.

Operations of a feedforward phase can be formulated as,

$$S_j(L) = \sum_{i=1}^{m_L} w_{ji}(L) \cdot a_i(L-1) \tag{2.5}$$

and

$$a_j(L) = f_j(S_j(L), \theta_j(L)), \tag{2.6}$$

where $L$ represents the layer number, $w_{ji}(L)$ is the weight between the neuron $j$ in the $L^{th}$ layer and the neuron $i$ in the $(L-1)^{th}$ layer, and $m_L$ is the total number of neurons in the $L^{th}$ layer.

During a back-propagation phase, a network passes derivatives of output errors back to a hidden layer, using the original weighted connections. Each hidden neuron then calculates the weighted sum of the back-propagated errors to find its indirect contribution to the known output errors. After each input and hidden neuron finds its error value, the neuron adjusts its weights to reduce its error. The weight updating rules can be formulated as follows:

$$\Delta w_{ji}(L+1) = \eta \delta_j(L+1) \cdot a_i(L) \tag{2.7}$$

$$w_{ji}^{new}(L+1) = w_{ji}^{old}(L+1) + \Delta w_{ji}(L+1). \tag{2.8}$$

The calculations for $\delta_j$'s are different in the output layer and in the hidden layers.
For the output layer:

$$\delta_j(L) = (t_j(L) - a_j(L)) \cdot f_j'(S_j(L)), \tag{2.9}$$

and for each hidden layer:

$$\delta_j(L) = e_j(L) \cdot f_j'(S_j(L)), \tag{2.10}$$

where

$$e_j(L) = \sum_{k=1}^{m_{L+1}} \delta_k(L+1) \cdot w_{kj}(L+1). \tag{2.11}$$

Here, $\delta_j$ is the error term in the present layer, $\delta_k$ is the error term in the next layer, $t_j$ is the corresponding desired output for the $j^{th}$ neuron, $\eta$ is the updating rate and can be either a local variable or a global constant. The equations that change the weights are designed to minimize the sum of the network's squared errors. This minimization has an intuitive geometric meaning. To see it, all possible sets of weights must be plotted against the corresponding sum-of-squares errors. The result is an error surface shaped like a bowl, whose bottom marks the set of weights with the smallest sum-of-squares error. Finding the bottom of the bowl–that is, the best set of weights–is the goal during training.

Back-propagation achieves the goal by calculating the instantaneous slope of the error surface with respect to the current weights. It then incrementally changes the weights in the direction of the locally steepest path toward the bottom of the bowl. This process resembles rolling a ball down a hill and is called *gradient descent.* Since gradient descent always follows the locally steepest path, the back-propagation algorithm can train a network into a local minimum that it cannot escape. This effect depends on the exact path down the gradient, which in turn depends on the initial values of the weights and other factors.

Back-propagation training is a kind of *supervised learning.* We call it supervised because the desired results are known in advance before learning, weight updating of a network can be directed by minimizing the difference between the actual output and the desired output. On the other hand, there are many cases that we don't know what are the correct responses of a network. *Unsupervised learning* is performed under these cases. The next network we will discuss is a kind of unsupervised learning.

#### 2.2.2.3 Kohonen Self-Organizing Feature Map

The feature mapping algorithm is supposed to convert patterns of arbitrary dimensionality into the responses of one- or two-dimensional arrays of neurons [38, 39, 40]. Learning within

self-organizing feature maps results in finding the best matching neuron cells which also activate their spatial neighbors to react to the same input. Such collective and cooperative learning tunes the network in an orderly fashion by defining some feature coordinates over the trained network. After learning, each input causes a localized response having a position on the neurons' array that reflects the dominant feature characteristics of the input. A basic structure of the self-organizing feature map is shown in Fig. 2.4.

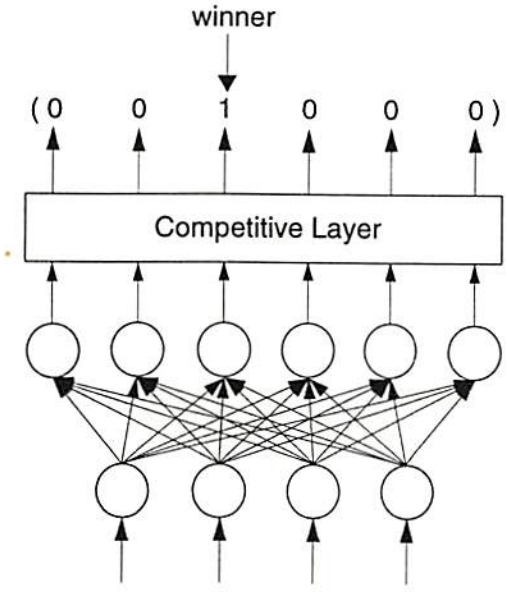



**Figure 2.4: A basic structure of the self-organizing feature map.**

The Kohonen self-organizing feature map has been shown to be quite effective for vector quantization (VQ) codebook design [41, 42, 43, 44]. For a full-search VQ, $N$ distortion measures between an input vector and $N$ code vectors are computed, and then the code vector, which is nearest to the input data, is chosen. Let $\vec{X} = x_j$ represents the k-dimensional input vector, and $\vec{C}_i = c_{ij}$ represents the $i$-th code vector in the codebook. The required computation is to find the minimum of the distance:

$$min \left\{ d(\vec{X}, \vec{C}_i),\ 1 \leq i \leq m \right\}, \tag{2.12}$$

where

$$d(\vec{X}, \vec{C}_i) = \sum_{j=1}^{k} (x_j - c_{ij})^2$$

is the distortion measure between $\vec{X}$ and $\vec{C}_i$.

#### 2.2.2.4 Comparison

In the following, the three types of neural architectures discussed previously along with perceptron, and Boltzmann machine are compared. The perceptron is primarily of historical interest, although it is still occasionally used. It receives inputs from several resources, and determines the output to be either logical-0 or logical-1 depending on the sum of the inputs being smaller than or greater than zero. The Boltzmann machine will be discussed further in the next chapter. Table 2.2 lists the strengths, limitations, and primary applications of the five types of artificial neural networks [45].

**Table 2.2: Comparison of various neural network models.**

| Neural models | Example applications | Strength | Limitation |
|---|---|---|---|
| Perceptron | linear filters | early neural network | cannot recognize complex patterns |
| Hopfield | retrieval of data/images from fragments | large-scale integration | weights not adaptable |
| Multi-Layered Perceptron | wide range: speech synthesis to loan-application scoring | most popular, work well, simple learning rules | supervised training with abundant examples |
| Self-organizing Map | pattern classification | better performance than many algorithmic techniques | extensive learning |
| Boltzmann Machine | pattern recognition for radar/sonar | simple network that uses noise function to jump out of local minimum | very long training time |

# Chapter 3

## Multi-Level Neural Networks with Optimal Solutions

Search for fast optimization in scientific and engineering applications has drawn researchers' interest in the field of engineering neural networks. A typical neural network for solving optimization problems multiplies input signals with synapse weights, and the summed results are processed by the output neurons.

The gradient descent method is a conventional approach for optimization. It finds the gradient of the cost function to determine the direction for the next processing step. This method suffers from the inherent problem of possibly converging at a local minimum in the cost function. Recently, Yuh and Newcomb [31] presented a method using a correction logic circuit to obtain the optimal solutions for multi-level Hopfield networks. Alspector et al. [46] have pioneered the analog electronic implementation of the Boltzmann machine by adding uncorrelated noise to find the globally optimal solution. Simulated annealing [47, 48] is another important method for searching for the optimal results on digital computers. It is a stochastic process modeled after metallurgical annealing. The metallurgical annealing consists of heating a raw material such as silicon to its melting point, then slowly cooling it based on a predetermined cooling schedule. If the raw silicon is cooled slowly, and a single-crystal seed is provided, atoms will fit into the lattice sites and reach a lower energy equilibrium. The slow decrease of the cooling temperature and the natural tendency towards a minimum-energy state are the key factors of metallurgical annealing and the Czochralski growth of purified semiconductors [49]. Due to a slow cooling schedule in software execution, the simulated annealing method on digital computers requires a lot of

computing time for a complex optimization problem. By constructing a hardware-based parallel annealing technique in analog electronics, the processing speed can be significantly improved.

A high-quality semiconductor crystal can be formed by providing a seed as a guidance in the process of metallurgical annealing. On the other hand, if no seed is provided, the melting semiconductor will be cooled down to an amorphous condition. In hardware annealing, which is an analog hardware version of the mean-field annealing [50, 51], an external bias voltage provides a similar effect as the seed in metallurgical annealing. During the whole annealing process, the constant external bias directs the outcome of a network. If no bias is applied, the output of the network undergoes influence from residual capacitances of the output nodes initially. These capacitances constitute the initial states of the network. Outputs of the network becomes unpredictable if the network goes through the annealing process without the external bias.

## 3.1 Simulated Annealing

The Hopfield network is a recurrent network which finds the stable solution through iterations. Due to the nonlinear property of the network, the output could be stable at one of the local minima. Simulated annealing helps to escape from local minima by using the Boltzmann distribution function as the transfer function of a neuron. Software implementation of simulated annealing uses the the following procedure:

```
Simulated_Annealing(init_state)
    begin
        stop := false;
        while (stop = false)
         begin
             while (equilibrium is not established)
              begin
                  Temp := some high temperature;
                  current_state := init_state;
                  new_state := selection_function(neighborhood of
                               current_state);
```

```
                ΔE := E(current_state) - E(new_state);
                if (ΔE <= 0)
                    current_state := new_state;
                else
                    if (e^(-ΔE/kTemp) > random([0, 1)))
                        current_state := new_state;
            end
            if (Temp < T_final)
                stop := true;
            else
                Temp := Temp - ΔT;
    end
end /* End of Simulated_Annealing */
```

## 3.2 Paralleled Hardware Annealing

In the Hopfield A/D network, a neuron with m-bit resolution is realized by $2^m$ amplifiers connected in parallel with their output currents summed together [31]. The summed current can be converted to the output voltage of the neuron by a current-to-voltage converter. The output voltages are connected back to the neuron inputs through interconnection conductances. An analog input value and a reference voltage are applied to all neurons. In hardware annealing, we change the voltage gain of the neurons in a continuous manner to achieve a similar effect as changing temperature in simulated annealing. Recently, some researchers have discussed methods of how to construct a gain-adjustable amplifier [52, 46, 53].

In the following, an analog-to-digital (A/D) decision network is used to illustrate the properties of using hardware-annealing in Hopfield networks with multi-level neurons because the optimal solution of an A/D decision network is always known [32, 33, 54].

A schematic diagram of a 6-bit A/D decision network by using three 4-level neurons are given in Fig. 3.1. A governing equation for the $i$-th neuron in an N-neuron network can be expressed as

$$C_i \frac{du_i}{dt} + G_i u_i(t) = \sum_{j \neq i, j=0}^{N-1} G_{ij} v_j(t) + I_i(t), \tag{3.1}$$

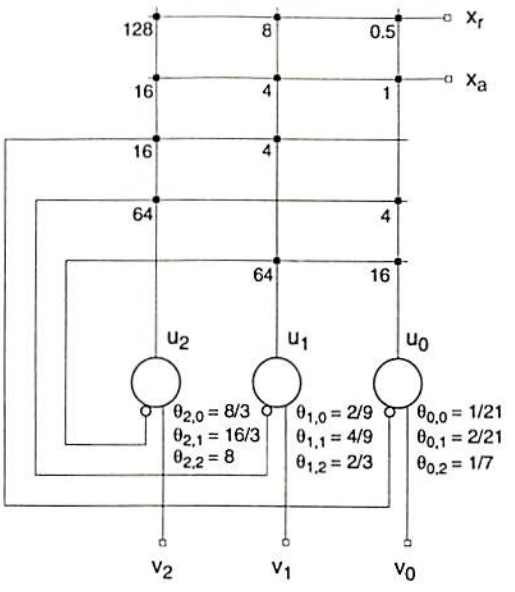



**Figure 3.1: A Hopfield 6-bit neural A/D decision network with 4-level neurons.**

and

$$I_i(t) = G_{ri}x_r(t) + G_{ai}x_a(t), \tag{3.2}$$

where $C_i$ and $G_i$ are the equivalent total capacitance and conductance at the input node of the $i$-th neuron, $G_{ij}$ is the conductance between the $i$-th and the $j$-th neurons, $x_r$ is the reference voltage, $x_a$ is the input analog voltage, $G_{ri}$ and $G_{ai}$ are the conductances connecting $x_r$ and $x_a$ to the $i$-th neuron. The voltage at the input node of the $i$-th neuron is $u_i(t)$, and the output voltage of the $j$-th neuron is $v_j(t)$.

The energy function for the network can be expressed as

$$E = -\frac{1}{2}\sum_{i=0}^{N-1}\sum_{j\neq i,j=0}^{N-1} G_{ij}v_iv_j - \sum_{i=0}^{N-1} I_iv_i + \sum_{i=0}^{N-1} G_i\int_0^{v_i} g_i^{-1}(v)dv, \tag{3.3}$$

where $g(\cdot)$ is the voltage transfer function of the amplifier. The time derivative of the energy function is

$$\frac{dE}{dt} = -\sum_{i=0}^{N-1}\frac{dv_i}{dt}\left[\sum_{j\neq i,j=0}^{N-1} G_{ij}v_j - G_iu_i + I_i\right]. \tag{3.4}$$

Substituting (3.1) into (3.4), we can obtain

$$\frac{dE}{dt} = -\sum_{i=0}^{N-1} \frac{dv_i}{dt} C_i \frac{du_i}{dt} = -\sum_{i=0}^{N-1} C_i g_i^{-1}(v_i) \left[\frac{dv_i}{dt}\right]^2. \tag{3.5}$$

Since $C_i$ is a positive value, $\frac{dE}{dt} \leq 0$ if $g_i(\cdot)$ is chosen to be a monotonically increasing function. The operation of the network evolves in the direction of decreasing the energy of the network until the equilibrium state is reached.

The function of an A/D decision network is to find a digital word which is a best representation of the analog input signal $x_a$. The synapse weight values can be determined by minimizing the squared value of the difference between the input analog value and the corresponding digital representation [31]

$$E_o = (x_a - \sum_{i=0}^{N-1} m^i v_i)^2. \tag{3.6}$$

After expanding (3.6) and dropping the constant term $x_a^2$, we obtain,

$$\hat{E}_o = -\frac{1}{2} \sum_{i=0}^{N-1} \sum_{j \neq i, j=0}^{N-1} (-m^{i+j}) v_i v_j - \sum_{i=0}^{N-1} m^i x_a v_i + \frac{1}{2} \sum_{i=0}^{N-1} m^{2i} v_i^2. \tag{3.7}$$

By assuming uniform spacings between the neighboring threshold values and the output levels in the decision network, i.e., $\Delta\theta$ and $\Delta k$ as defined in page 24 are constants, (3.3) can be reformulated as

$$E = -\frac{1}{2} \sum_{i=0}^{N-1} \sum_{j \neq i, j=0}^{N-1} G_{ij} v_i v_j - \sum_{i=0}^{N-1} (I_i - \frac{1}{2} G_i \Delta\theta_i \Delta k_i) v_i + \frac{1}{2} \sum_{i=0}^{N-1} G_i \Delta\theta_i \Delta k_i v_i^2. \tag{3.8}$$

By equating corresponding items in (3.7) and (3.8), we can obtain

$$G_{ij} = -m^{i+j}, \tag{3.9}$$

$$I_i = m^i x_a + \frac{1}{2} G_i \Delta\theta_i \Delta k_i = G_{ai} x_a + G_{ri} x_r,$$

and

$$G_i \Delta\theta_i \Delta k_i = m^{2i}.$$

Here, $\Delta\theta_i$ is the increment in the threshold value, and $\Delta k_i$ is the increment in the output-level value at the $i$-th neuron. The weights, inputs and thresholds for the neurons from

(3.9) are clearly labeled in Fig. 3.1. The negative conductances $G_{ij}$'s are implemented by taking results from the inverted outputs of the neurons. The $G_{ri}$'s are determined by setting $x_r$ to be one.

Let us carefully examine the transfer function of the A/D decision network. Assume that all neurons are biased in the linear region. In the steady state condition,

$$u_i = \frac{\sum_{j\neq i,j=0}^{N-1} G_{ij}v_j + G_{ri}x_r + G_{ai}x_a}{G_i} \tag{3.10}$$

and

$$\theta_{i,k} \leq u_i < \theta_{i,k+1} \tag{3.11}$$

for the $k$-th level at the $i$-th neuron. Substituting (3.10) into (3.11), we can obtain

$$\frac{-\sum_{j\neq i,j=0}^{N-1} G_{ij}v_j + G_i\theta_{i,k} - G_{ri}x_r}{G_{ai}} \leq x_a < \frac{-\sum_{j\neq i,j=0}^{N-1} G_{ij}v_j + G_i\theta_{i,k+1} - G_{ri}x_r}{G_{ai}}. \tag{3.12}$$

Figure 3.2 shows the plot of the transfer function of a three-neuron A/D decision network. All the possible states determined by (3.12) are included. The output value of each neuron

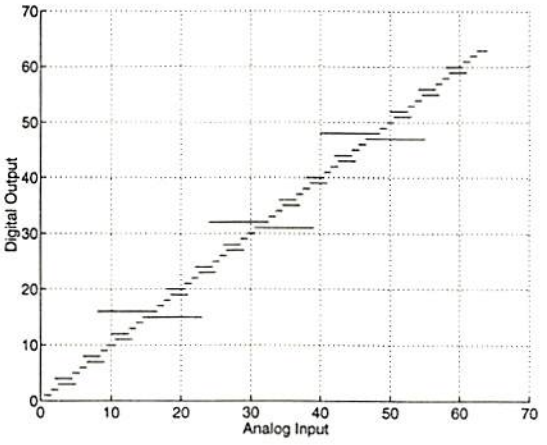



**Figure 3.2: Transfer function of a 6-bit A/D decision network with. 4-level neurons.**

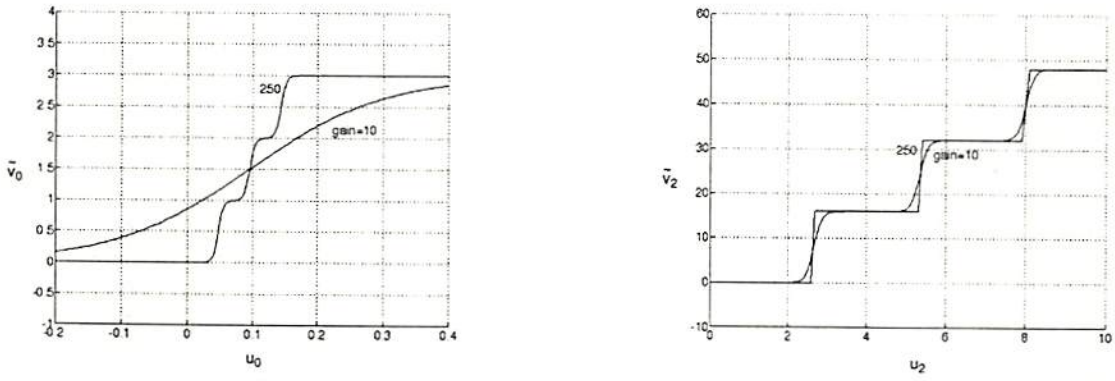
can be classified into one of four distinct levels. Please note that for some analog input values, the network can converge to different digital representations which correspond to

multiple local minima in the energy function. For example, when the input value is 19.8, the output value may be 15, 19 or 20 depending on the initial conditions of the network.

A multi-level Boltzmann neuron function [31] can be modified from (2.4):

$$M_b = \sum_{i=0}^{m-1} \frac{c_i}{1 + e^{-\eta(x-\theta_i)}} \tag{3.13}$$

where $\eta$ is a gain-controlling factor which controls the slope of a multi-level transfer function. With a large gain value, the slope of the transfer curve around the threshold value is quite steep. A smaller gain value gives a smoother transition. When the threshold values are close to each other as shown in Fig. 3.3(a), the transfer curve for a high voltage gain still shows distinct levels, while that for a low voltage gain shows no distinct levels. By assigning 1 to $\Delta k_i$, the three threshold values for the least-significant-bit neuron are 1/21, 2/21, and 1/7, respectively. The threshold values for the most-significant-bit neuron are 8/3, 16/3, and 8, respectively, as shown in Fig. 3.3(b).




(a) (b)

**Figure 3.3: Plots of the 4-level neuron functions with different controlling gains used in a 3-neuron A/D decision network. (a) Least-significant neuron. (b) Most-significant neuron.**

The sum and product of eigenvalues of the system matrix **M** of a Hopfield A/D decision network have the properties [55]

$$\sum_{i=1}^{N} \lambda_i \simeq 0 \tag{3.14}$$

and

$$\prod_{i=1}^{N} \lambda_i = det(\mathbf{M}) \neq 0 \tag{3.15}$$

when the voltage gain of neurons is sufficiently large. $\lambda_i$'s represent eigenvalues of the system matrix M. With the constraint that $G_{ij} = G_{ji}$, all eigenvalues will lie on the real axis of the $s$-plane. Thus at least one positive real eigenvalue exists, and the neuron outputs are saturated at digital values. Tables 3.1 and 3.2 list the eigenvalues for 6-bit and 8-bit A/D decision networks, respectively, at a very large neuron gain. The second

**Table 3.1: Eigenvalues of a 6-bit A/D decision network.**

| | I: 6-bit ($2^6$) | II: 6-bit ($4^3$) |
|---|---|---|
| $\lambda_1$ | 668.63 | 67.01 |
| $\lambda_2$ | -534.96 | -65.13 |
| $\lambda_3$ | -104.64 | -1.88 |
| $\lambda_4$ | -22.65 | |
| $\lambda_5$ | -5.19 | |
| $\lambda_6$ | -1.22 | |

I: Conventional Hopfield network

II: Multi-level Hopfield network

**Table 3.2: Eigenvalues of an 8-bit A/D decision network.**

| | I: 8-bit ($2^8$) | II: 8-bit ($4^4$) |
|---|---|---|
| $\lambda_1$ | 10,707.21 | 1,075.11 |
| $\lambda_2$ | -8,560.58 | -1,043.25 |
| $\lambda_3$ | -1,675.09 | -30.44 |
| $\lambda_4$ | -362.95 | -1.48 |
| $\lambda_5$ | -83.43 | |
| $\lambda_6$ | -19.78 | |
| $\lambda_7$ | -4.77 | |
| $\lambda_8$ | -1.16 | |

I: Conventional Hopfield network

II: Multi-level Hopfield network

columns in both tables list eigenvalues of networks consisting of two-level neurons, and the third columns list eigenvalues of networks consisting of multi-level neurons. The single positive value $\lambda_1$ in a conventional two-level neuron decision network is much larger than that in a multi-level neuron decision network. In the multi-level 6-bit case, $\lambda_1 + \lambda_2 + \lambda_3$ is equal to 0, and $\lambda_1 \cdot \lambda_2 \cdot \lambda_3$ is equal to 8,205. These results agree with (3.14) and (3.15).

In a conventional Hopfield network, the output of an A/D decision network is stablized at an equilibrium state which corresponds to one of the local minima of the energy function. When the voltage gain is reduced to a critically low value, all neurons operate in the linear condition. The output of the A/D decision network is not fully digital. When the voltage gain reaches

$$A_{N-k} = max\left\{\frac{G_i}{\sum_{j=0,j\in S}^{N-1}|G_{ij}|},\ for\ 0 \le i \le N-1,\ 0 \le k \le N-3\right\}, \tag{3.16}$$

where $S$ contains linear-region neurons, only $(N-k)$ neurons operate in the linear region. As the voltage gain is increased, fewer number of neurons operate in the linear region. When the voltage gain reaches a critically high value,

$$A_2 = max\left\{\sqrt{\frac{G_pG_q}{G_{pq}G_{qp}}},\ \forall p,q \in [0,N-1],\ p \ne q\right\}. \tag{3.17}$$

Only two neurons operate in the linear region [55]. When the voltage gain is slightly larger than $A_2$, only one neuron could be in the linear region. Although at this point the neuron output is still an analog value, the digital output of the amplifier can be easily determined. Tables 3.3 and 3.4 list the different critical voltage gains for 6-bit and 8-bit A/D decision networks, respectively. The $A_2$ values for the 4-level neural networks are smaller than those for the 2-level networks.

## 3.3 Analysis Results

In metallurgical annealing, a certain high temperature was first applied to highly mobilize the atoms. It is then slowly decreased to allow atoms to fit into lattice sites in a minimum-energy fashion. In this section, experiments of hardware annealing on Hopfield networks with multi-level neurons are discussed.

In the experiment, an A/D decision network which consists of three neurons was analyzed. Each neuron has four distinguishable levels which correspond to the 2-bit accuracy. The analog input has a nominal range of 0 to 64 which can be scaled to the range of 0 to 5 $V$ if electronic hardware implementation is considered. The network output is reconstructed by a D/A converter.

**Table 3.3: Critical amplifier gains of a 6-bit A/D decision network.**

| | I: 6-bit ($2^6$) | II: 6-bit ($4^3$) |
|---|---|---|
| $A_2$ | 44.72 | 10.37 |
| $A_3$ | 20.67 | 2.80 |
| $A_4$ | 8.57 | |
| $A_5$ | 3.73 | |
| $A_6$ | 1.55 | |

I: Conventional Hopfield Network
II: Multi-level Hopfield Network

**Table 3.4: Critical amplifier gains of an 8-bit A/D decision network.**

| | I: 8-bit ($2^8$) | II: 8-bit ($4^4$) |
|---|---|---|
| $A_2$ | 180.49 | 42.37 |
| $A_3$ | 84.67 | 15.60 |
| $A_4$ | 36.00 | 2.57 |
| $A_5$ | 16.53 | |
| $A_6$ | 7.74 | |
| $A_7$ | 3.56 | |
| $A_8$ | 1.51 | |

I: Conventional Hopfield Network
II: Multi-level Hopfield Network

The voltage gain of neurons is decreased first and then gradually increased. The analog input value $x_a$ was chosen to be 7.9 which has one local minimum at digital representation

(013) besides the global minimum at (020). When the voltage gain value was low, all neurons operated in the linear region. Then, the voltage gain was increased until all neuron outputs became saturated into digital representation. The input and output values of neurons are shown in Figs. 3.4 and 3.5, respectively. Initially, the output state of the

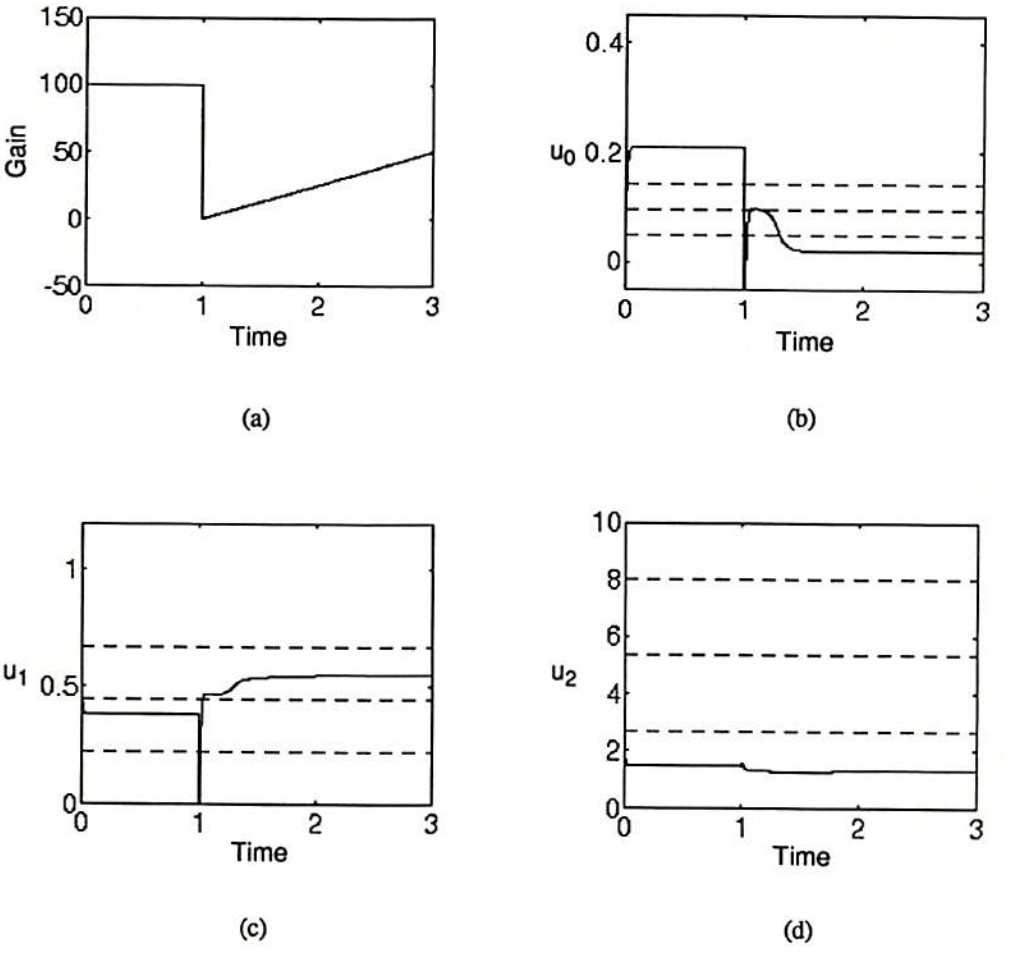



**Figure 3.4: Time domain responses of the voltages at the neuron input nodes in the A/D decision network when the analog input value $x_a$ is 7.9. (a) Neuron gain. (b) Neuron 0. (c) Neuron 1. (d) Neuron 2.**

A/D decision network was at (013) which corresponds to the local minimum. After the hardware annealing process, the output state changed to (020) which is the desired digital output. The trajectory of the output state is plotted in Fig. 3.6. Plots of energy surfaces

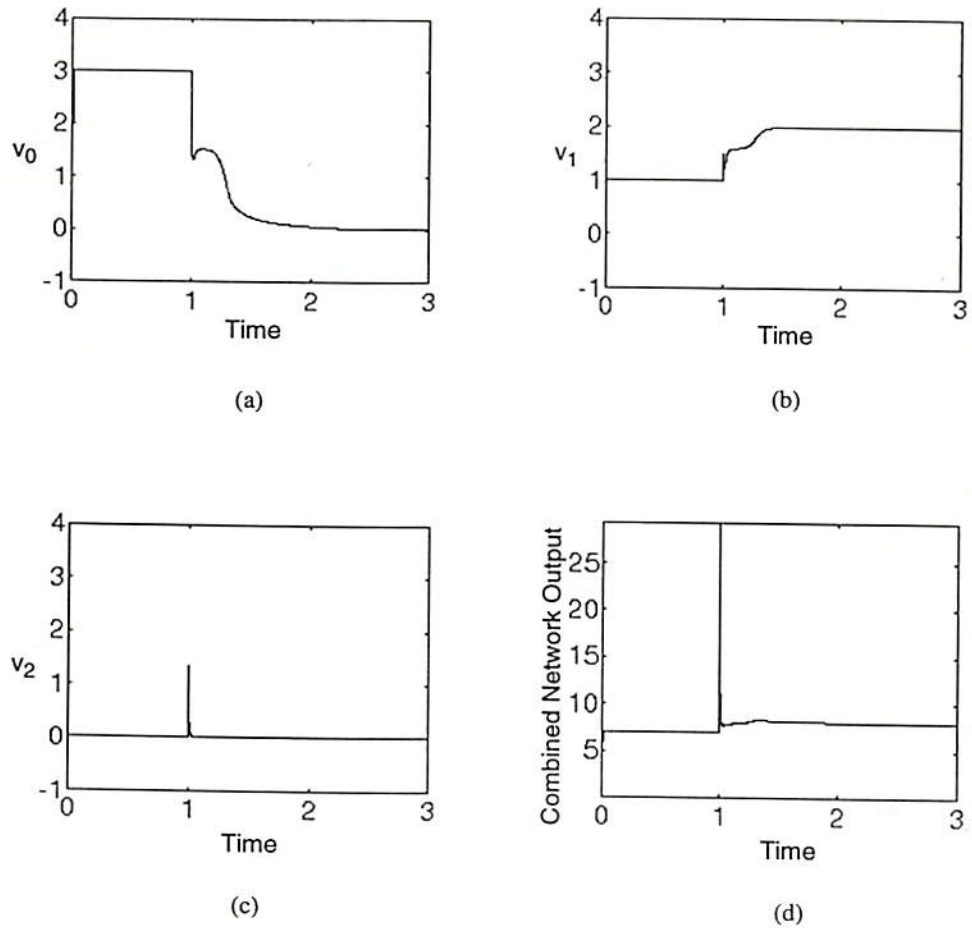



**Figure 3.5: Time domain responses of the output values of neurons when the analog input value $x_a$ is 7.9. (a) Neuron 0. (b) Neuron 1. (c) Neuron 2. (d) Combined output.**

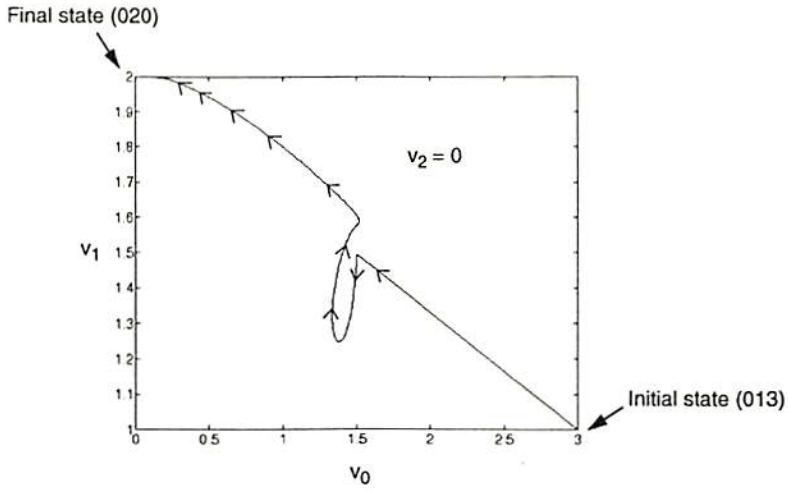
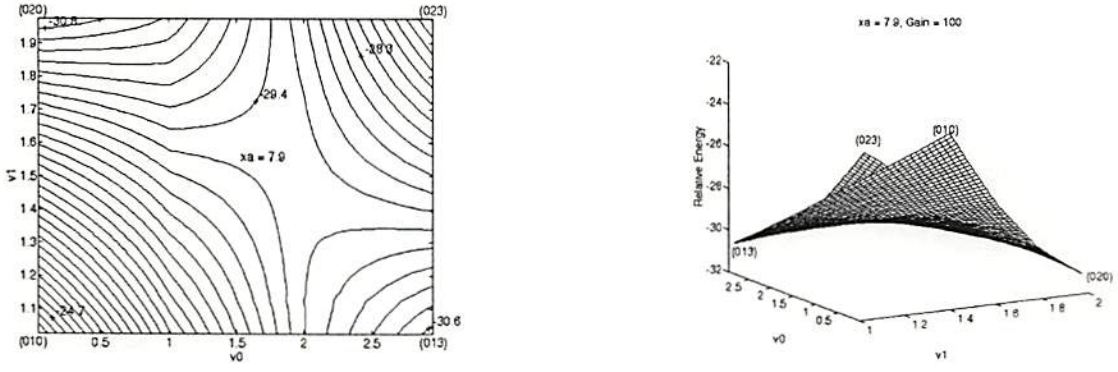


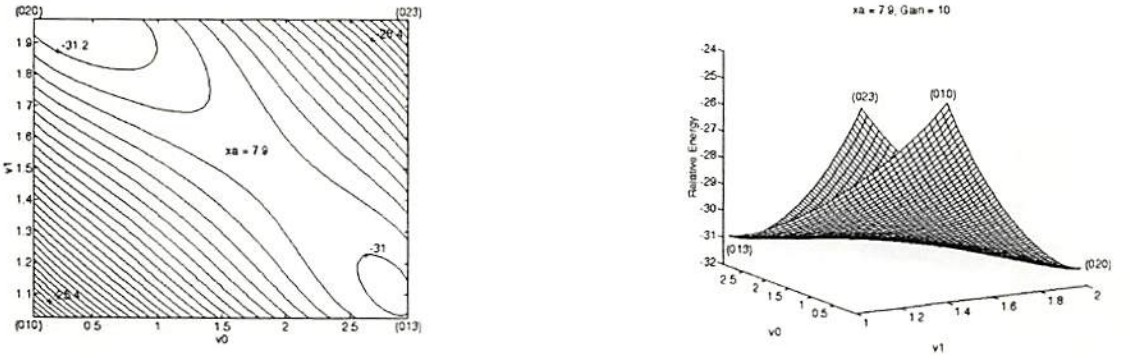

**Figure 3.6: The output state trajectory of the A/D decision network.**

and contours which correspond to different voltage gain values are shown in Figs. 3.7, 3.8, and 3.9. Two energy minima occur when the neuron is at a high voltage gain value of 100. When the voltage gain is reduced to 10, only one energy minimum left. At a low voltage gain value of 2.5, the minimum energy corresponds to a solution which is not a fully digital representation.



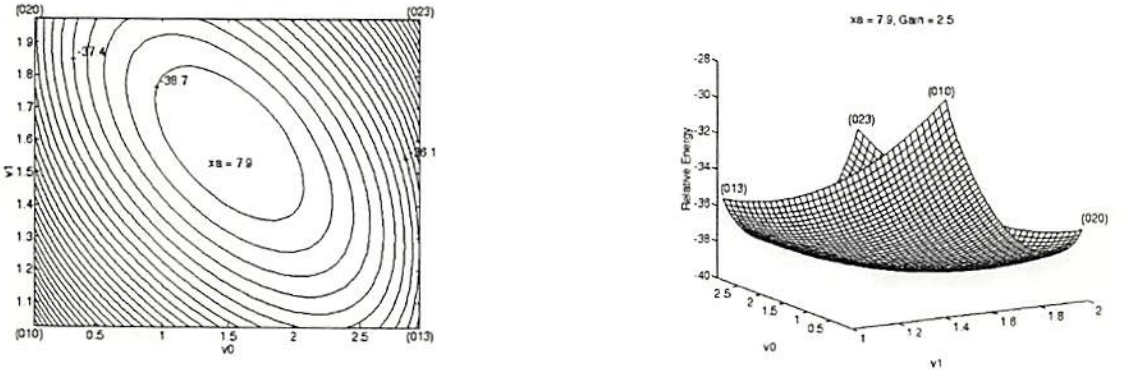

(a) (b)

**Figure 3.7: (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 100.**




(a) (b)

**Figure 3.8:** (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 10.




(a) (b)

**Figure 3.9:** (a) The energy contour. (b) The energy surface. The voltage at the input node of neuron $u_2$ is kept as 0. $x_a$ is 7.9 and amplifier gain is 2.5.

# Chapter 4

# Digital Neural Processing Element

In practical applications, the algorithms of artificial neural networks must be executed in real time. Therefore, a very fast hardware using parallel architecture is desirable to efficiently exploit the intrinsic parallel processing properties of ANNs. It can be conceived that a neural unit is made of a processing element. An ANN consists of a number of interconnected processing elements which perform weighted accumulation of inputs and level evaluation of outputs. The feature of the network is characterized by the network topology, interconnecting weights, and the transfer function. The transfer function determines the output activation levels.

## 4.1 Hardware-Software Codesign Methodology

The hardware-software codesign techniques were applied during this design process of neural coprocessors. The codesign task is to produce an optimal hardware-software design that meets the given specifications, within a set of design constraints such as real-time requirements, performance, speed, area, code size, memory requirements, power consumption, and programmability.

Figure 4.1 shows a generic codesign flow. Given a system specification, the designer develops an algorithm, using high-level functional simulations, without any assumptions about implementation such as available instruction set or register precision, circuit design style. The designer then partitions the algorithm into hardware and software, guided by

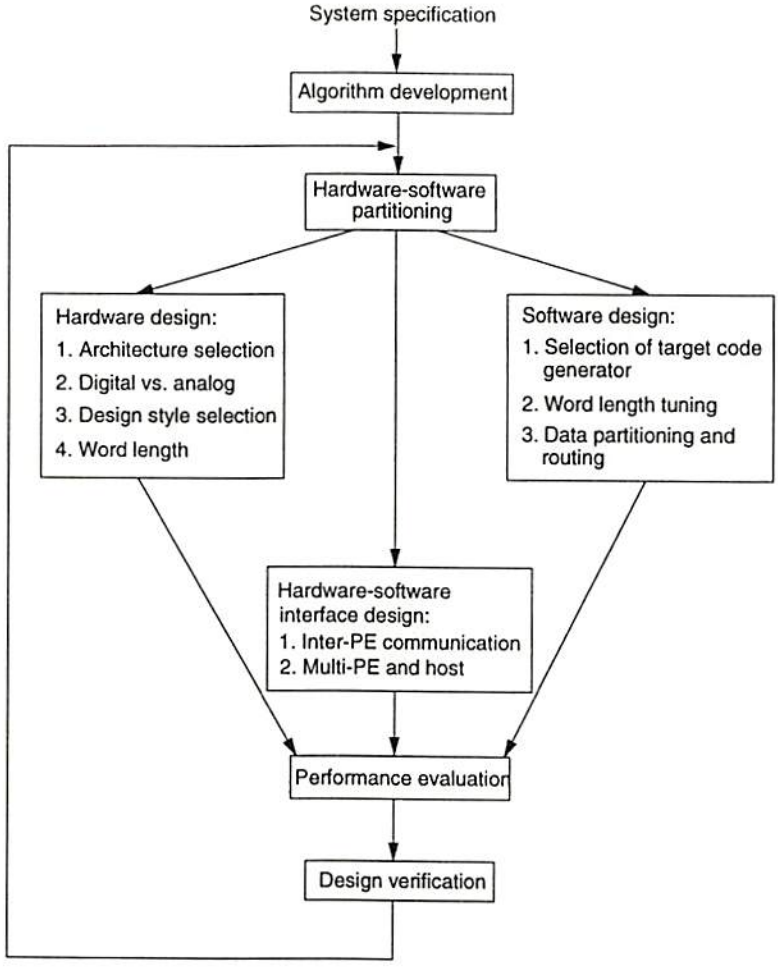



**Figure 4.1: A general codesign strategy.**

speed, complexity, and flexibility requirements. Components that need field programmability or that are inherently better accomplished in software are assigned to software implementations. Operations with critical execution speed are allocated to hardware. Of course, to explore the design space, the designer would iterate the partitioning process.

Partitioning is followed by hardware, software, and interface design. The three are closedly linked. Changing one has immediate effects on the others. Hardware design decisions include selection of a programmable processor which directly affects selection of the code generator. If the designer prefers to create a new processor by herself/himself, more issues including selection of design style, clocking strategy, etc are involved. In designing

custom data paths, the designer must choose the register word lengths. Some hardware structures (for example, filter realizations [56]) may meet performance requirements with smaller register widths than those estimated for other structures. Determination of the number of processors and their connectivity influences code partitioning and hardware-software interface design. On the software front, in the case of fixed-point processors, some algorithmic modifications might be necessary to minimize the effects of finite precision such as limit cycles and quantization errors. Software design involves partitioning and scheduling the code on multiple processors and creating the code for interprocessor communication. These decisions depend on the architecture selected. The designer partitions among different processors by optimizing cost functions such as communication cost, memory bandwidth, and local and global memory sizes.

Interface design involves adding latches, FIFOs, or address decoders in hardware and inserting code for I/O operations and semaphore synchronization in software. The typical way of solving this cyclic problem is to start with a design and work on it iteratively to explore different options.

Once the hardware and software components are designed, the next step is to run the resulted software on the resulted hardware. This involves interaction of a number of different simulators if various specification languages are used. The designer then uses the simulation results to verify whether the design meets the specifications. Having performed the hardware and software design, the designer can then estimate area, power, critical path, component and bus utilization, and other factors. After using these estimates to evaluate the design, the designer may repartition the system to try out different options. Thus, the entire process is iterative.

## 4.2 System Architecture

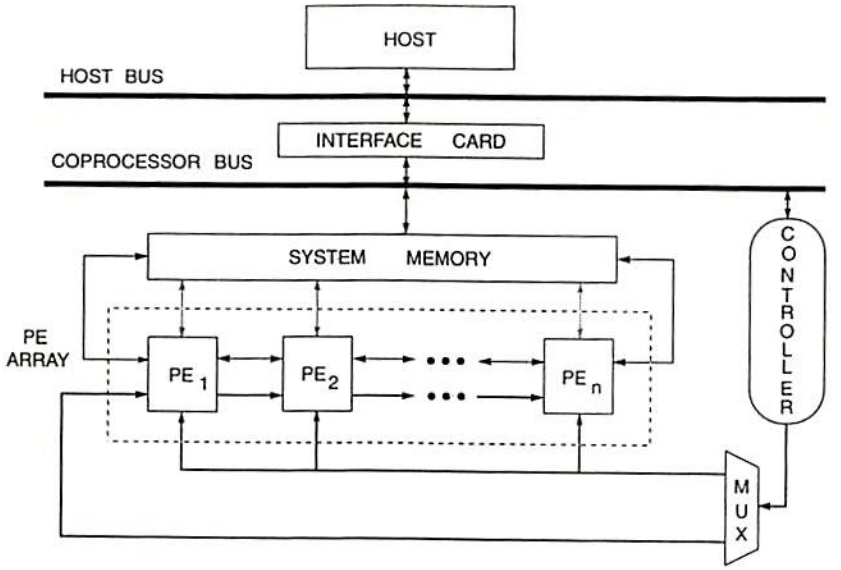
To fully utilize the substantial parallelism in an ANN, a multi-PE system should be considered. Since each processor must communicate data to each other in this system, the

architecture which explores efficient communication between processors will deliver most computational power per unit area.

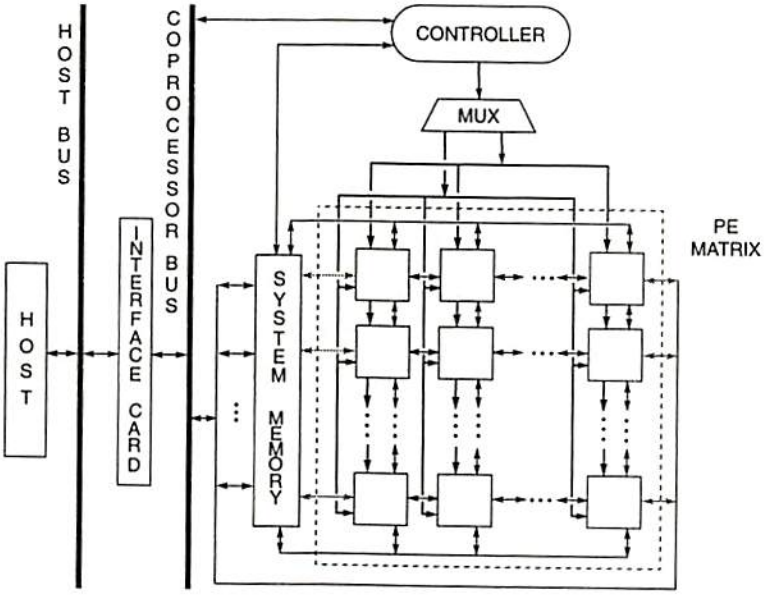
Systolic arrays typically combine intensive local communication and computation with decentralized parallelism in a compact package. They uses regular, modular, synchronous, and concurrent processing units to execute intensive and repetitive computation. Advances of VLSI technologies complement qualifications of the systolic array. Smaller and faster gates allow a higher rate of on-chip communication because data has a shorter distance to travel. Higher gate densities permit more complicated cells with higher individual and group performance. Granularity increases as word length increases, and concurrency increases with more cells. Economical design and fabrication processes produce less expensive systolic chips, even in small quantities. Better computer-aided design (CAD) tools allow arrays to be designed more efficiently.

A digital systolic implementation of connectionist networks greatly speeds up intensive computations in the networks in terms of data pipelining [57, 29, 58]. Several pipeline routes can coexist in a network with systolic architecture. The ring-connected PE array [59] and mesh-connected PE matrix [60] are two example architectures suitable for systolic implementation. In the realization of a multi-layer perceptron network, the same architecture is shared by both feedforward propagation and backward error propagation phases. This greatly suggests the feasibility of a digital design. Compared with analog neural circuits, the systolic architecture offers better flexibilities, higher precision, and full pipelinability.

The system architectures for a ring-connected systolic multi-PE system and a mesh-connected multi-PE system are shown in Fig. 4.2(a) and Fig. 4.2(b). All processing elements are microcode-controlled and a dedicated controller is used to provide a microcodes during each clock cycle. The microcodes and addresses for memory accesses can be received by the PEs through either a broadcast or a pipelined fashion which is selected by the multiplexer as shown in the figures. For the ring connection, I/O operations only occur on the right- and left-most PEs and the system memory is a two-port memory shared




(a)




(b)

**Figure 4.2: System architecture for systolic networks. (a) 1-D ring-connected array. (b) 2-D mesh-connected matrix.**

by the PE array. During initialization, every PE stores its local data from the system memory through local buses shown as dashed arrows. For the mesh connection, each PE communicates with its four nearest neighbors and I/O operations occur on the four boundaries. In addition, the system memory is a four-port memory shared by the PE matrix. Similarly, during initialization, local data of PEs come from local buses on the left and right boundaries of the PE matrix. A PE in the middle columns received its local data through pipelining. The ring-connected PE array can be considered as a special case of the mesh-connected PE matrix. Therefore, we can design a mesh-connected PE matrix which can also perform ring-connected systolic operations.

The system memory receives data from the host computer when a host call is received by the controller. The host computer serves as the interface between the user and the systolic array. It provides problem-specific information, such as input patterns, initial weights, convergence-controlling parameters, etc. The controller specifies and monitors executions of each PE and also governs data communication between PEs. The controller has its dedicated memory to store microcodes from the host computer.

## 4.3 Algorithm Mapping

Various algorithms of artificial neural network applications can be mapped onto the multi-PE architecture described in the previous section. In this section, mappings of two popular learning rules, a back-propagation learning in a multi-layer perceptron network [27, 28, 61] and a self-organizing learning network [38, 39, 40, 43, 44], onto the mesh-connected PE matrix are described. The mappings are based on an assumption that the dimensions of available PE matrices are larger than a learning network needed, i.e., a network does not have to be partitioned into several subnetworks to fit into PE matrices.

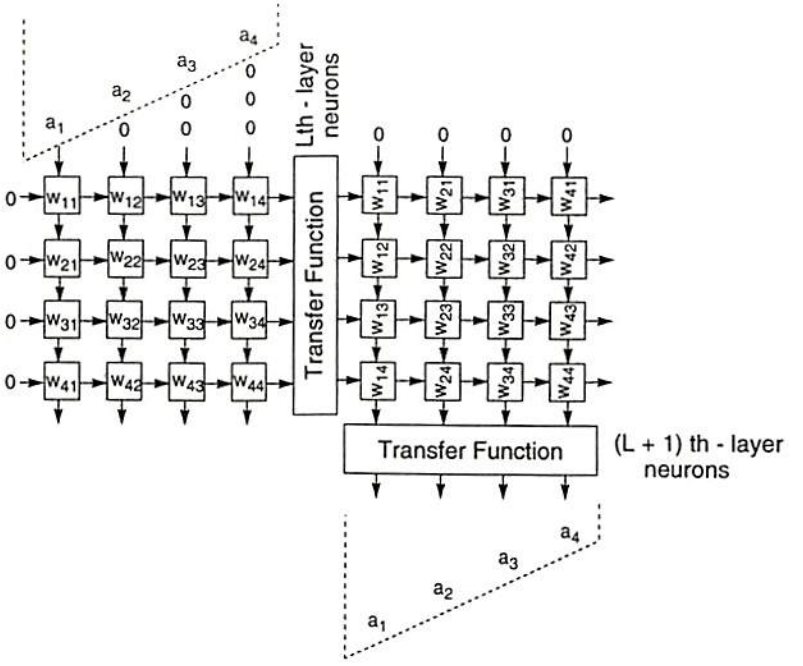
### 4.3.1 Back-propagation Learning

A multi-layer perceptron network can be constructed by using multiple PE matrices as shown in Fig. 4.3. When a neural system starts to operate, weight values are loaded

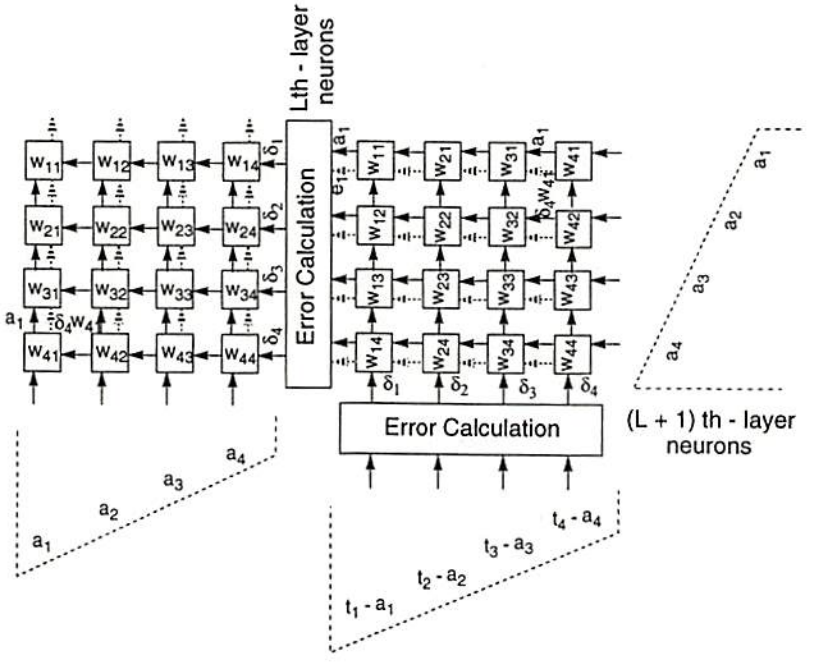
from the host computer to the system memory. The weight value $w_{ji}$'s are then stored into data caches of PEs through interprocessor pipelining. Equation (2.5) is executed by the whole PE matrix while equation (2.6) is computed in only one row (column) of PEs. Figure 4.3(a) shows the data flow during a feedforward phase when the size of the PE matrix is equal to or greater than the required number of neurons in a layer. The left block is for operations in the $L^{th}$ layer. Incoming data $a_i(L-1)$'s move downward and are multiplied by $w_{ji}(L)$'s. The products move rightward and are accumulated to produce $S_j(L)$'s. After the neuron transfer function operation, values of $a_i(L)$'s move rightward and perform multiplication with $w_{ji}(L+1)$'s. The products move downward to form $S_j(L+1)$'s. Then, the newly created $a_j(L+1)$'s proceed to the next layer. When the size of the PE matrix is smaller than the required number of neurons in a layer, the original neural net has to be partitioned to let a processing element represents several neurons to efficiently utilize the existing hardware. A processing element in this case represents several neurons. All of the information regarding processing element assignment, partition strategies and data routing are provided by the calling procedure from the host computer.

During the back-propagation phase, each processing element has a value of $\eta$ stored in the data cache. Equations (2.9) and (2.10) are executed by one column of PEs, while equations (2.7), (2.8), and (2.11) are executed by the whole matrix of PEs. Figure 4.3(b) shows the mapping of a back-propagation phase onto the PE matrix. The right block is for updating weights between the output layer and the hidden layer. The left block is for updating weights between the hidden layer and the input layer. For a multi-layer neural network with more than one hidden layer, the left block can be repeated to form multiple hidden layers.

Calculations of a back-propagation phase start from the right PE block. The error calculation box generates $f_j'(S_j(L))$'s. The bottom row of PEs calculate $\delta_j$ values and send them back to the system memory. Then $a_j$'s propagate leftward and $\delta_j$'s propagate upward. After receiving $a_i$ and $\delta_j$, a PE calculates $\Delta w_{ji}$ and updates $w_{ji}$ using (2.8). Notice that the $\delta_j$'s at this block are also the $\delta_k$'s at the left block. Therefore, before a $\delta_j$




(a)




(b)

**Figure 4.3: Mapping diagram of a back-propagation neural network onto a mesh-connected PE matrix. (a) Feedforward propagation phase. (b) Backward error propagation phase.**

value moves upward from a PE, the PE also calculates $\delta_k w_{kj}$ and accumulates it with the partial sum of the similar terms from the preceding PEs and moves it leftward. Finally, the $e_j$'s are obtained at the leftmost column of the PE matrix and is sent through the error calculation box for calculating $\delta_j$'s of the hidden layer.

At the left block, $\delta_j$'s move leftward and $a_i$'s move upward. If there is only one hidden layer, each PE only performs (2.7) and (2.8). On the other hand, if there are more than one hidden layer, each processing element has to perform (2.7), (2.8), and (2.11) , to calculate the $\delta_j$'s for the following hidden-layer weight adjustments.

### 4.3.2 Self-organizing Feature Map

Vector quantization has emerged in recent years as a powerful technique that can provide large reduction in bit rates while preserving the essential signal characteristics. The mapping of the self-organizing learning algorithm for vector quantization onto a PE matrix is shown in Fig. 4.4 and is described as follows:

1. Input vectors $x_j$, $j$ = 1, 2, ..., k, are sent to k rows of PEs.

2. Weight vectors, which are also called codevectors, $c_i$'s, moves downward to perform distortion calculation $E_i = d(x_j, c_{ij})$. A newly calculated minimum distortion $y_j$ moves to the right. An index pointer $p_k$ indicating the current winner also passes to the right PE. If the distortion calculated in a PE is the same as the original minimum distortion, the index pointer will not change its value.

3. After calculations in the first row is completed, $p_1$ indicates the unit in the first row with the smallest distortion. Label this unit as the winner and its weight vector as $c^*_{i1}$.

4. Adjust the selected weight vector

$$c^*_{i1}(r+1) = c^*_{i1}(r) + \varepsilon(r)[x_1(r) - c^*_{i1}(r)] \tag{4.1}$$

and write it back into the system memory, where r is the training time index.

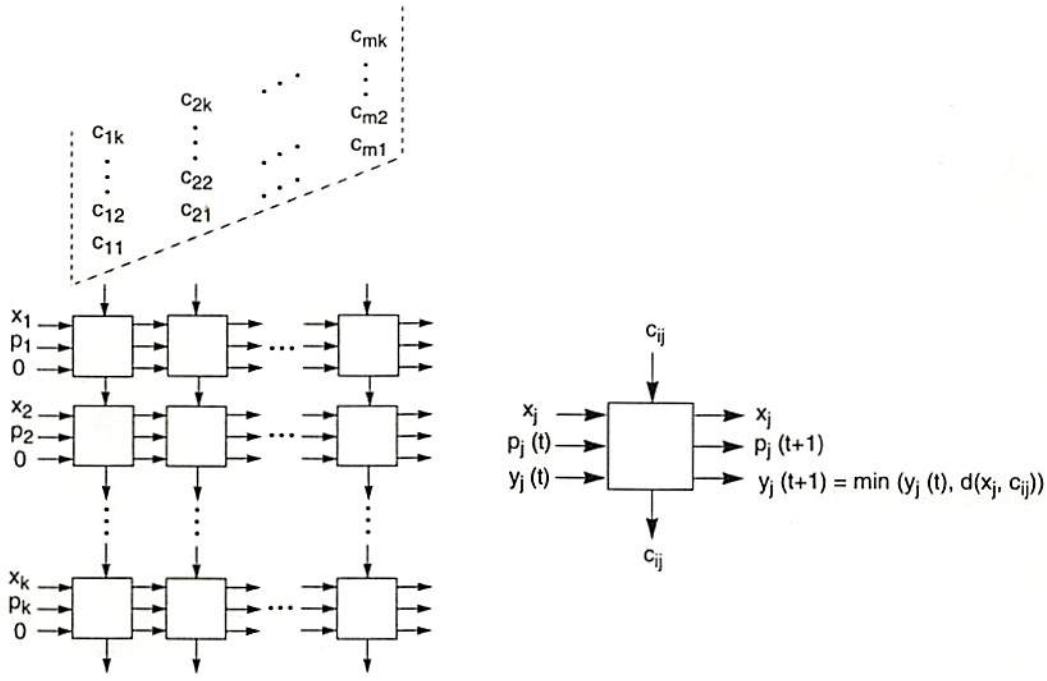



**Figure 4.4: Mapping diagram of a self-organizing neural network onto a mesh-connected PE matrix.**

5. Repeat steps (3) to (4) for all the other rows.

Because of pipelined propagation of $c_{ij}$'s, completion of the $j^{th}$ row will be two clocks later than that of the $(j-1)^{th}$ row. The chosen value for the learning rate parameter $\varepsilon(t)$ does not depend on the magnitude of the data. The learning rule moves the synapse weight toward the training vector by a fractional amount. Typically, $0 < \varepsilon(t) < 1$ and the $\varepsilon(t)$ value decreases as learning progresses.

## 4.4 Processing Element Design

Digital electronic implementation of neural networks has four major approaches: the microprocessor-based approach, the digital signal processor (DSP)-based approach, the custom-designed neuroprocessor approach, and the dedicated processor approach. The first two approaches use available processors from the market to build a system. The third approach uses a custom-designed hardware to perform general-purpose neural computing.

We can apply this hardware to various applications including digital signal processing, digital image processing, etc. The last approach is to build a dedicated hardware for some special neural network algorithms. Table 4.1 lists the comparison of these different approaches. To achieve a cost-effective design, the third approach is adopted. In this section, detailed design of a digital VLSI processing element will be discussed from architecture selection to circuit design, and then to physical layout design. Real-world application based on the designed circuits will be demonstrated in the next chapter.

**Table 4.1: Comparison of digital electronic design approaches.**

| Properties | General-purpose microprocessor-based | DSP-based | Custom-designed Neuroprocessor | Dedicated Processor |
|---|---|---|---|---|
| Parallelism | Lowest | Low | High | Highest |
| Flexibility | Highest | Low | High | Lowest |
| Precision | Highest | High | Low | Lowest |
| Programmability | Highest | Low | High | Lowest |
| Integration/Scalability | Low | High | Highest | Lowest |
| Processing Speed | Lowest | Low | High | Highest |
| Design Cycle | Lowest | Low | High | Highest |
| Manufacturing Cost | Highest | High | Low | Lowest |
| Applications | General Purpose | Digital Signal Processing | Various Areas | Very Limited |

The existing digital VLSI implementation examples show some common properties. For example, they all have on-chip learning, on- and off-chip weight memory, generally not more than 16-bit precision, and arrays of DSP-like processing elements (PEs). The CNAPS chip which consists of 64 PEs, from the Adaptive Solutions, Inc., is implemented in a 0.8-$\mu m$ CMOS technology with 11 million active transistors and operates at 25 MHz [62, 63]. The MA-16 chip and SYNAPSE-X system from Siemens Corporation have 4 PEs per chip and the maximum clock rate is 50 MHz [64]. An 8-PE digital processor chip from British Telecom Research Labs uses a 0.7-$\mu m$ CMOS technology and operates at 20 MHz [65]. The initial design of the SPERT chip from International Computer Science Institute also contains 8 PEs per chip [66]. A wafer-scale LSI neural network, achieving 2.3 giga

connection updates per second (GCUPS) in peak performance, has been developed by the Hitachi Central Research Labs [67]. The SNAP-64, which consists of 4 processor boards, from HNC, Inc. [68] can deliver 2.56 giga floating-point operations per second (FLOPS) computational performance. Each board includes 4 gate-array chips to implement 32-bit IEEE floating point operation. Intel announced in 1993 a 3.5 million transistor digital neural chip for classification based on radial basis functions. The classification rate is 400 patterns per second. A summary table of these existing examples is listed in Table 4.2 [53],[62]-[75].

Image processing applications have the desirable feature of moderately simple operations, yet they demand high performance and throughput. Furthermore, exploring the cost and performance trade-offs between different implementations is critical for consumer products and portable applications, where DSP is being widely used. We are focusing on the design of the hardware and software for such systems. The digital VLSI processing element described in this chapter can be efficiently applied to artificial neural networks with various interconnect configurations such as ring systolic arrays [59], mesh-connected matrices [60], and cellular neural networks [76, 77] in which the feedforward and learning operations can be expressed in terms of matrix and vector computations. The networks can be used for image processing applications. The design result can be extended to wafer-scale integration (WSI) or multi-chip module technology to achieve machines of small size and light weight.

### 4.4.1 Chip Architecture

Figure 4.5 shows the block diagram of the proposed PE. During initialization of the systolic array, initial parameters of training algorithms in an ANN are down-loaded into the system memory. Each PE consists of a data cache memory to facilitate data processing. A multiplier and an adder are main computing units of a PE. The register file provides temporary data storage. The design is based on the mesh-connected architecture as shown in Fig. 4.2(b). Each processing element uses the four I/O ports to communicate with its

Table 4.2: Performance of selective neural network hardware examples.

| | Institution | Signal representation | Technology | # of transistors | # of units | Chip size | Speed | Status |
|---|---|---|---|---|---|---|---|---|
| CNAPS chip & CNAPS system | Adaptive Solutions | Digital | 0.8-µm CMOS | 11M | 64 PEs | 26.2 mm x 27.5 mm | 25 MHz 1.6 BCUPS | Commercial see profit |
| 8-PE digital processor chip | British Telecom Research Lab. | Digital | 0.7-µm CMOS | 750K | 8 PEs | 9 mm x 11 mm | 20 MHz | Prototyped in 1991 |
| MA-16 chip & SYNAPSE-X system | Siemens AG | Digital | 1-µm CMOS | 580K | 4 PEs | 10 mm x 10 mm (*) | 50 MHz | Internal use |
| SPERT | Intl. Computer Science Inst., Berkeley | Digital | 1.2-µm CMOS | 280K (*) | 8 PEs | 7.4 mm x 9.2 mm | 50 MHz | Designed in 1992 |
| ETANN | Intel | Analog | 1-µm EEPROM | 150K (*) | 64 neurons 10240 synapses | 5 mm x 8 mm (*) | 2.5 BCPS | Diminishing |
| NET32K | AT&T | Mixed Analog & Digital | 0.9-µm CMOS | 412K | 256 neurons 4096 synapses | 4.5 mm x 7 mm | 300 BCPS | Prototyped in 1990 |
| ANNR | AT&T | Analog | 0.9-µm CMOS | 180K | 8 neurons 4096 synapses | 4.5 mm x 7 mm | 20 MHz 5 BCUPS | Internal use |
| Self-Organizing Processor | USC | Mixed Analog & Digital | 2-µm CMOS | 15K | 64 neurons 1600 synapses | 4.6 mm x 6.8 mm | 4 MHz 3 BCPS | In revision |
| Self-Learning neural chip with BNU Arch. | Mitsubishi | Analog | 1-µm CMOS | 1.6M (*) | 336 neurons 28K synapses | 14.5 mm x 14.5 mm | 1000 BCPS 28 BCUPS | Prototyped in 1991 |
| Ni1000 | Intel | Digital | 0.8-µm | 3.5M | 256K synapses | 15.8 mm x 13.7 mm | 40 MHz | Commercial |

(*) Estimated

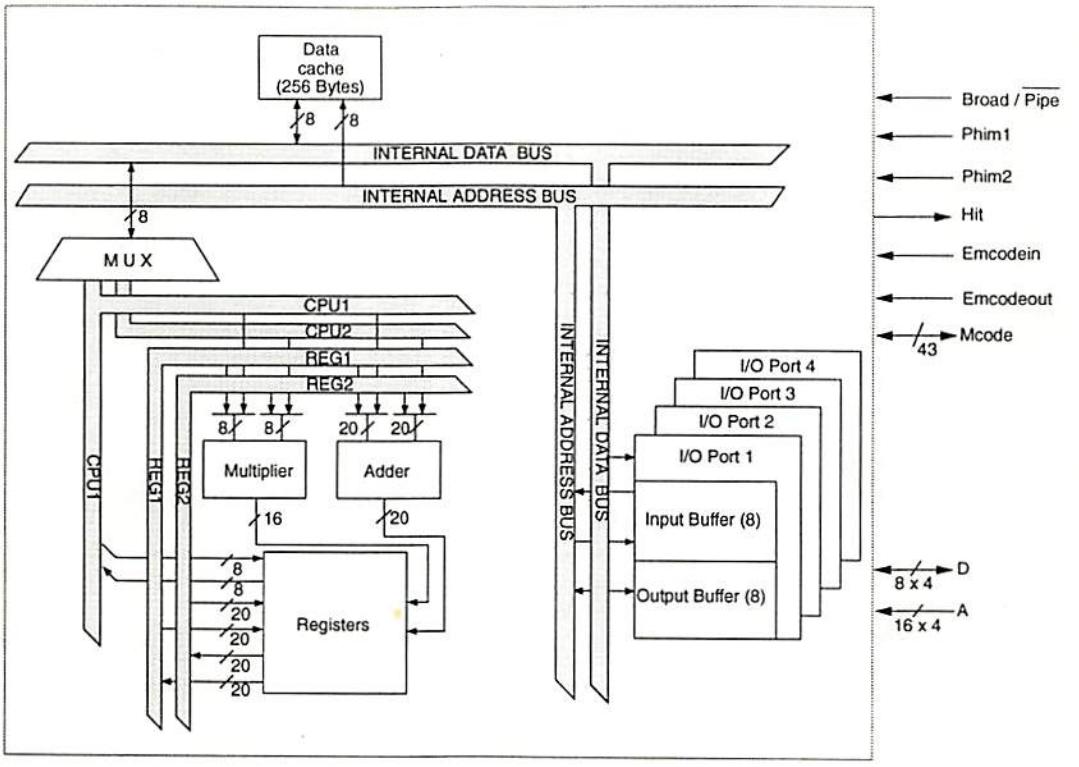



**Figure 4.5: Building blocks and bus distribution of a processing element.**

neighbors. A total of 145 I/O lines is required for a PE to perform one-direction ring-connected communication. A memory management unit determines memory allocation policies which include block replacement schemes, fetch and main memory update strategies. The memory management unit, residing in the controller, is shared by all PEs The system clock is synchronized for a PE array.

A table-look-up method is used to provide neuron transfer functions. Addresses from the external address bus are passed through the peripheral address bus to both the data cache memory and the next PE. The addresses can be either broadcast or pipelined depending on the user-specified instructions from the host computer. High level languages from the host computer are first compiled into macroinstructions which are sent to the controller. The controller decodes these macroinstructions into microinstructions and broadcast them to all PEs via the 43 microcode lines specified as "Mcode" in Fig. 4.5. The

reduced-instruction set approach has been used in designing the required instruction set. Our goal is to explore an optimal instruction set through microprogramming.

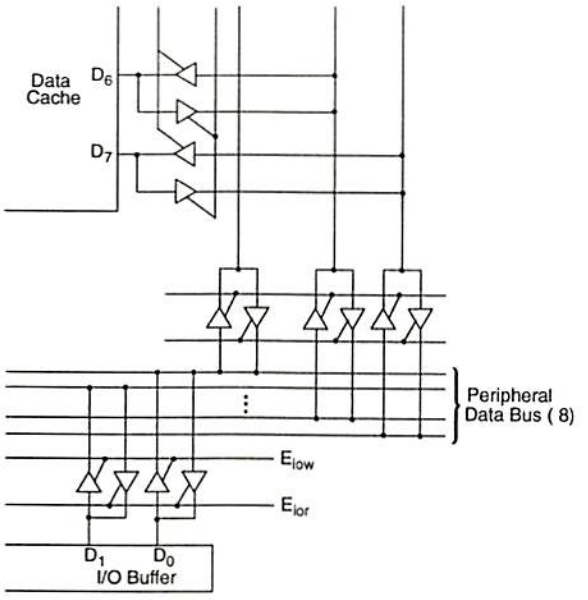
In order to increase the processing speed, the register bus has been separated into four bus segments: the REG1, REG2, CPU1, and CPU2. More steps can be overlapped because they do not occupy the same portion of the bus. A multiplexer is used to multiplex these four buses into the internal data bus. When data is sent between the data cache and the register file, the CPU1 bus is used. Figure 4.6(a) and 4.6(b) show the control circuitry for the data cache to communicate with other blocks through the buses.

For both the multiplier and the adder, one operand comes from either the CPU1 bus or the REG1 bus, the other operand comes from either the CPU2 bus or the REG2 bus depending on the addressing mode. The output of the multiplier is sent to the register file through the REG1 bus and the output of the adder is sent to the register file through the REG2 bus. Figure 4.7(a) and 4.7(b) show the bus control circuitry of the adder and registers, respectively.

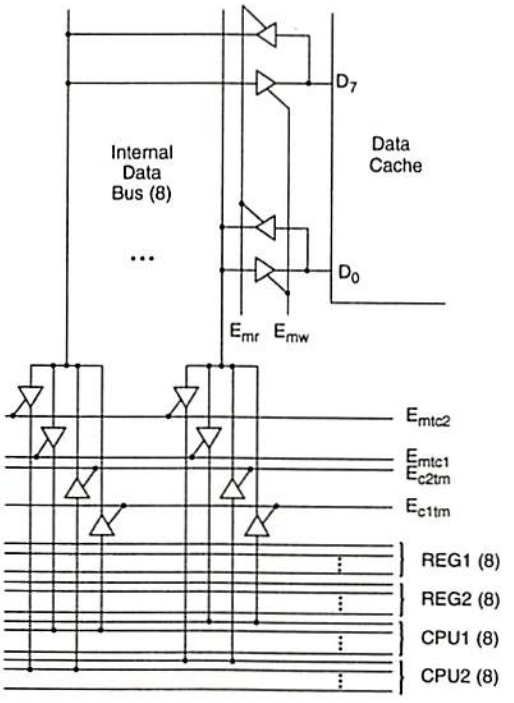
### 4.4.2 VLSI Design Considerations

#### 4.4.2.1 Design Process

An initial system-level design based on specification must be first verified by a simulator. Models of individual functional units created by language constructs are used to simulate the behavior of the design. After register-transfer language (RTL) level verification is finished, logic synthesis and logic optimization can be performed followed by gate-level verification. Timing analysis is an important step to determine the critical path and the performance of the design. To facilitate product measurement and accelerate design cycle, test vector generation and fault simulation is also performed in the design stage. Chip layout can be accomplished by different design styles. Post-layout verification including design rule checking (DRC) and electrical rule check (ERC) is necessary before a chip is sent for fabrication. Each step in the design process involves iterative modification to match design specifications.




(a)




(b)

**Figure 4.6: The external control circuitry for interfacing with the data cache (a) Between the cache memory and the I/O buffer. (b) Between the cache memory and the four bus segments REG1, REG2, CPU1 and CPU2.**

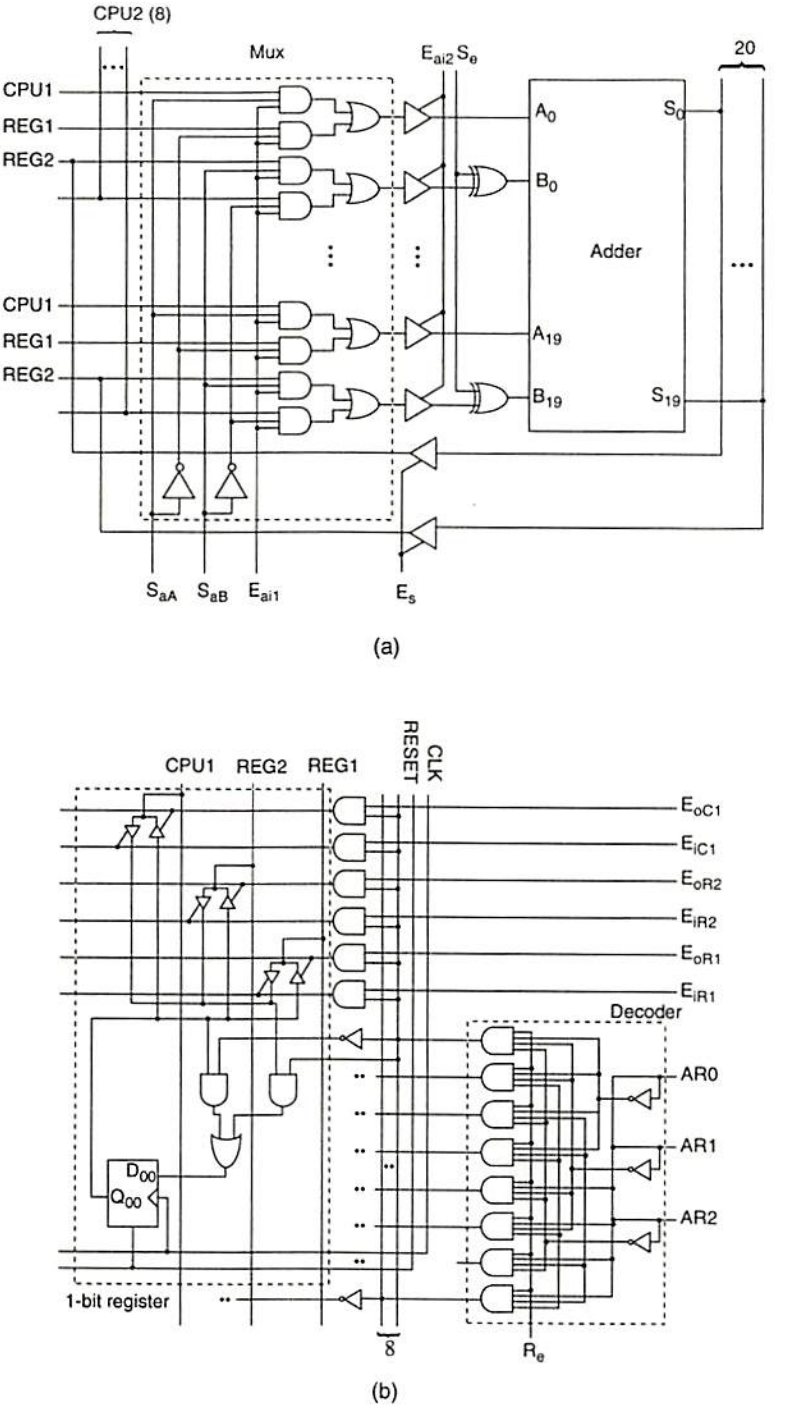



**Figure 4.7: Required control circuitry (a) For adder. (b) For register file.**

#### 4.4.2.2 Physical Design Style

Currently, the popular integrated circuit design styles are standard cells, gate arrays, and full-custom design. A comparison of these styles is shown in Table 4.3. Full-custom design

**Table 4.3: Comparison of different design styles.**

| | Full Custom | Gate Array | Standard Cell |
|---|---|---|---|
| Performance | 1 | 3 | 2 |
| Modularity | 3 | 2 | 1 |
| Area | 1 | 3 | 2 |
| Flexibility | 1 | 3 | 2 |
| Design Time | 3 | 1 | 2 |

1: Best; 2: Middle; 3: Worst

usually requires a handcrafted level of automation since the lack of constraints makes synthesis tools difficult to develop. Design rule checking and verification are mandatory because of inevitable errors. Full-custom design is very time-consuming. However, it is used when exploiting a new circuit design technique to achieve high performance goal.

The gate array design style combines the programmable array fabrication method and the cell-based design method. In this approach, a cell-based design is mapped onto a prefabricated, two-dimensional array of uncommitted transistors. Gate array foundries usually fabricate large numbers of wafers containing identical, unwired arrays. These arrays are then personalized for a particular circuit function by adding interconnection wires. Several popular floorplans for gate array design are rows, columns, islands, and sea-of-gates [78]. Row floorplans have routing channels running horizontally while column floorplans have routing channels running vertically. Island floorplans have routing channels running both horizontally and vertically. The cell sites form the islands. Sea-of-gates floorplans either have no routing channels or the channels are small compared to the gates sites, and wiring must be routed over the cells.

The advantage of gate array design is clear for prototyping and low-volumn production. Applying the interconnection layers is a low-risk, inexpensive operation; yields are high and fabrication time is short. However, for higher volume production, other design styles are usually more area-efficient and less costly to manufacture.

Circuit design with standard cells is similar to gate array design. Both use cell-based design methods where the cells are small- and medium-scale integration gates. However, a circuit designer has an extra degree of flexibility from a full mask set. For a given circuit, area is almost always smaller compared to gate arrays. Interconnection area is smaller since the routing and the cells can be positioned to accommodate the specific circuit being designed. Basic cells are smaller because they do not have to mapped onto a predefined array of transistors.

#### 4.4.2.3 Logic Style

The choice of using static or dynamic logic is dependent on many criteria. When low-power performance is desired, it appears that dynamic logic has some inherent advantages in a number of areas including reduced switching activity due to hazards, elimination of short-circuit dissipation, and reduced parasitic node capacitances. Static logic circuits have advantages on reduced charge sharing and elimination of precharge operations [79].

#### 4.4.2.4 Complementary Pass-Transistor Logic

A complementary pass-transistor logic (CPL) gate is constructed by using an nMOS pass-transistor network for logic function, and eliminating the pMOS latch [80]. It consists of complementary inputs and outputs, an nMOS pass transistor logic network, and CMOS output inverters. The pass-transistors function as pull-down and pull-up devices. Thus the pMOS latch can be eliminated, allowing the advantage of differential circuits to be fully utilized. One attractive feature of a CPL gate is that complementary outputs are produced by the simple four-transistor circuits. Because the logic-1 value level of the pass-transistor outputs is lower than the supply voltage $V_{DD}$ by the threshold voltage of

the pass-transistors, the signals have to be amplified by the output inverters. In addition, the CMOS output inverters shift the logic threshold voltage and drive the capacitive load. The logic threshold shift is necessary because that of the output inverter is lower than half of the supply voltage, due to the lowering of the logic-1 value.

Arbitrary Boolean functions can be constructed from the CPL network by combining the four basic modules: (1) AND/NAND module (2) OR/NOR module (3) XOR/XNOR module (4) wired-AND/NAND module. The four features are shown in Fig. 4.8. Because

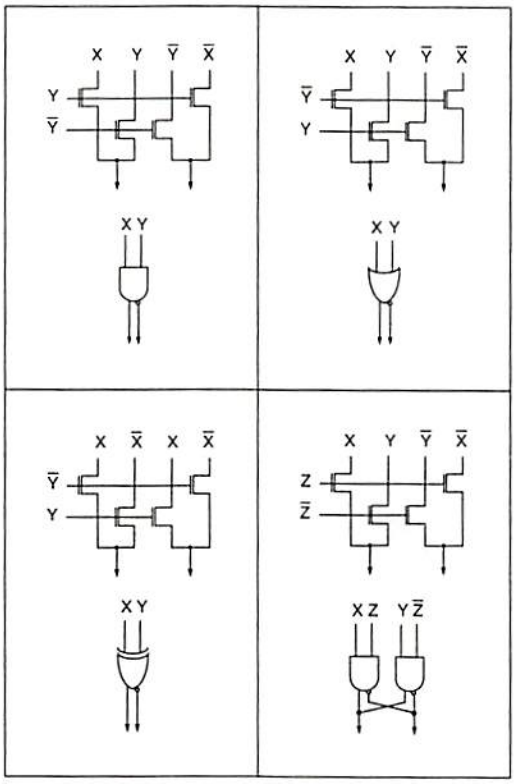



**Figure 4.8: CPL circuit modules.**

inverters are unnecessary in CPL circuits, the number of critical-path gate stages can be reduced. However, because the logic-1 input value at the regenerative inverters is not $V_{DD}$, the pMOS device in the inverter is not fully turned off, and hence direct-path static power dissipation could be significant. To solve this problem, reduction of the threshold voltage

has proven effective, although if taken too far it will incur a cost in dissipation due to subthreshold leakage and reduced noise margins [79].

### 4.4.3 A Design Example

A prototype PE has been fabricated in a 6.19 × 5.46 $mm^2$ microchip by using a 0.8-$\mu m$ three-metal CMOS technology from Hewlett-Packard Company through the MOSIS Service of USC/Information Science Institute. The minimum drawn size is 1.0-$\mu m$. A 132-pin PGA package is used. A PE chip runs at a clock rate of 50 MHz. In this neural processing element design, full-custom style has been applied to achieve a higher processing speed which is very important for neural network applications due to large amount of training cycles typically.

#### 4.4.3.1 Clock Distribution Scheme

From performance point of view, it is critical to keep skews small and edge rates sharp, so special attention has to be paid to clock generation and distribution. In the coprocessor chip design, the master clock is received from an off-chip controller. Therefore, the clock generation will not be discussed here. The clock distribution goals are to minimize clock skew, inductive ground bounce, and power consumption. These goals are achieved using three techniques. The first is maintaining a high degree of synchronous logic throughout the chip, so only one clock network is needed. The second is driving the network with one large clock buffer. To reduce effects of inductive ground bounce, the clock buffer is powered with isolated power and ground pads and isolated with a $p^+$-doped guard ring [81]. The third is reducing capacitive load on the main clock line. A single-phase flip-flop with a clock input load of a minimum-sized inverter is implemented.

A highly-regular metal three layout scheme is used to distribute power and clocks across the chip from the periphery, and clock buffers are placed within unused spaces in power pads. Both of these techniques minimize die area for power and clocking. To minimize power consumption, the clock distribution system can selectively power down sections of

the chip when they are not in use by using the clock enable signals [12]. Figure 4.9 shows the clock distribution scheme used in the design. The clock buffers and routing is balanced

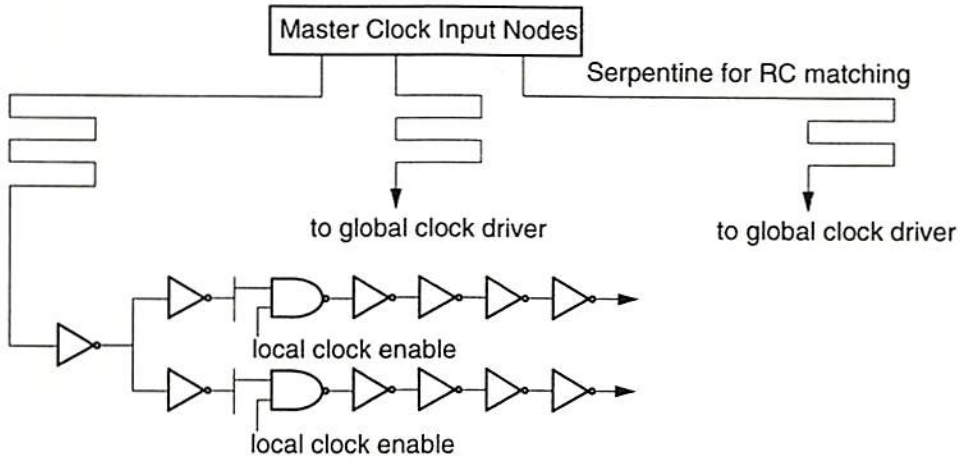



**Figure 4.9: Clock distribution scheme with selectively power down capability.**

by tuning the buffer size and dummy loads are added to the more lightly-loaded phases. An on-chip decoupling capacitor of 20 $nF$ is distributed throughout the buffer layout to minimize $V_{DD}/V_{SS}$ noise when the clocks switch rapidly. Internal clock skew is controlled to within 1 $ns$ via a combination of matched buffering and low impedance interconnect.

#### 4.4.3.2 Computing Units

The extensive development of high-performance signal processors in the past few years has naturally been accompanied by a corresponding development of the mathematical techniques required for the most efficient use of these processors. With expanding computer applications, the demand for high-speed processing has been increasing. Higher clock-rate operation is a key to enhancing system performance, especially in digital signal and image processing. To satisfy such a demand, many high-speed processing elements have been developed. A multiplier and an adder are two main computing units in a signal processor.

*A. Adder*

Integer addition is the simplest yet the most important operation in the computer arithmetic because a floating point operation eventually reduces to an integer operation. Therefore, to increase the speed of integer operation would also benefit floating point operations. Even for programs which don't do explicit arithmetic, addition also needs to be performed for address calculations. The design example presented here is a carry look-ahead (CLA) adder which has a speed of $O(logn)$.

An n-bit adder can be constructed by using a logic formula whose form is a sum of products and can be computed by a circuit with two levels of logic. The formula for the $i^{th}$ sum bit and carry bit are

$$s_i = a_i\bar{b_i}\bar{c_i} + \bar{a_i}b_i\bar{c_i} + \bar{a_i}\bar{b_i}c_i + a_ib_ic_i \tag{4.2}$$

$$c_{i+1} = a_ib_i + a_ic_i + b_ic_i = a_ib_i + (a_i + b_i)c_i = g_i + p_ic_i \tag{4.3}$$

If $g_i$ is true, $c_{i+1}$ is certainly true, and a carry is generated. Thus, $g$ is for generate. If $p_i$ is true, then if $c_i$ is true, it is propagated to $c_{i+1}$. $c_i$ can therefore be expressed as $g_{i-1} + p_{i-1}c_{i-1}$. Recursively expanding equation (4.3) gives the result

$$c_{i+1} = g_i + p_ig_{i-1} + p_ip_{i-1}g_{i-2} + \cdots + p_ip_{i-1}\cdots p_1g_0 + p_ip_{i-1}\cdots p_1p_0c_0 \tag{4.4}$$

From (4.4), a carry-lookahead adder requires a fan-in of $n + 1$ at the OR gate as well as the rightmost AND gate. In addition, the $p_{n-1}$ signal must drive $n$ AND gates.

Improvement can be accomplished by building up the $p$'s and $g$'s in steps and creating a simple, regular structure [82]. We have already seen that

$$c_1 = g_0 + c_0p_0 \tag{4.5}$$

This says that there is a carry from the $0^{th}$ position ($c_i$) when either there is a carry generated in the $0^{th}$ position, or there is a carry passed into the $0^{th}$ position and the carry propagates. Similarly,

$$c_2 = (g_1 + p_1g_0) + (p_1p_0)c_0 = G_{01} + P_{01}c_0 \tag{4.6}$$

The general equations for any $j$ with $i < j,\ j + 1 < k$ are as follows,

$$c_{k+1} = G_{ik} + P_{ik}c_i \tag{4.7}$$

$$G_{ik} = G_{j+1,k} + P_{j+1,k}G_{ij} \tag{4.8}$$

$$P_{ik} = P_{ij}P_{j+1,k} \tag{4.9}$$

These equations will also hold for $i \leq j < k$ if we set $G_{ii} = g_i$ and $P_{ii} = p_i$. A diagram of the data flow in a 12-bit adder is shown in Fig. 4.10, where the numbers to be added enter at the top, flow to the bottom to combine with $c_0$, and then flow back up to compute the sum bits.

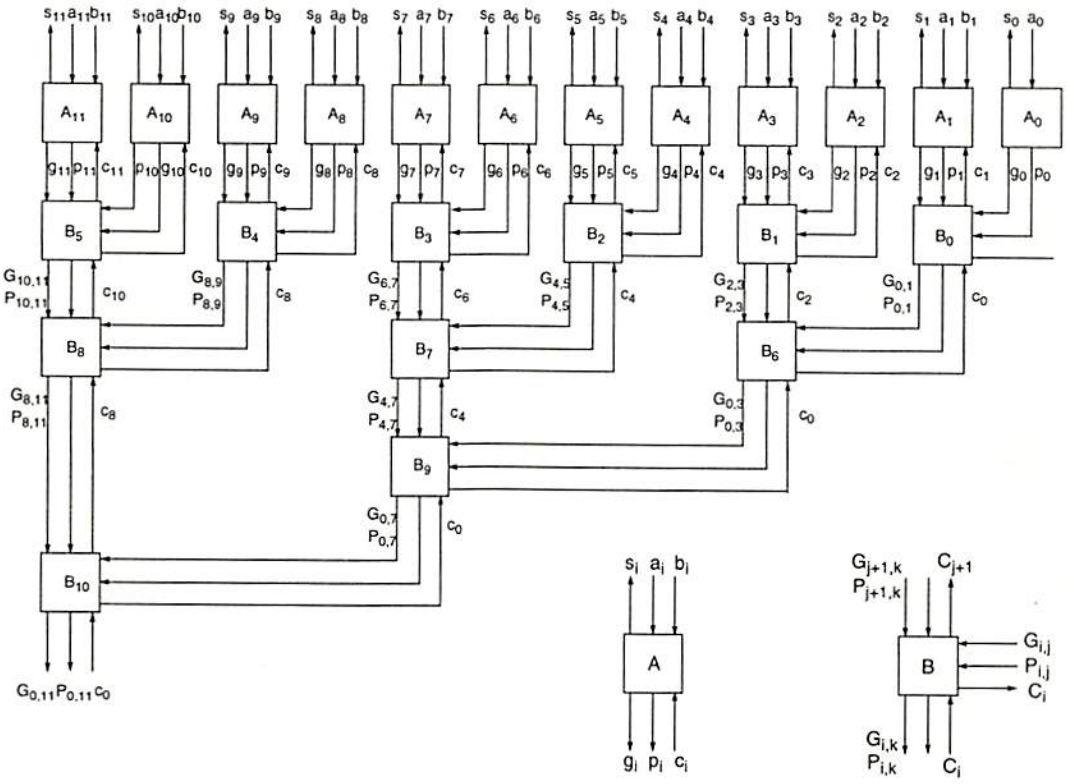



**Figure 4.10: Block diagram of a 12-bit CLA adder.**

The two blocks, A and B, shown in Fig. 4.10 consist of two groups of logic functions. Block A is used to perform (4.2) and (4.3). As shown in Fig. 4.11, three CPL circuit modules are used: $s_i$ bit, XOR/XNOR module; $p_i$ bit, OR/NOR module; $g_i$ bit, AND/NAND module. Figure 4.12 shows the circuits in block B, (4.7), (4.8) and (4.9) are performed in this block. Figure 4.13 shows the irsim (an event-driven logic-level simulator) simulation

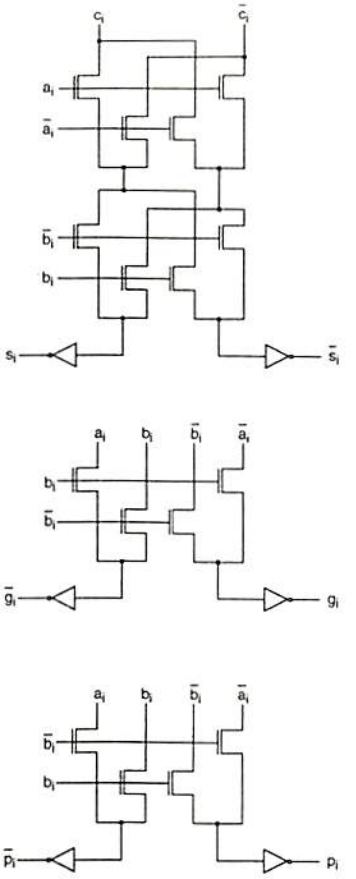


**Figure 4.11: Circuits in block A.**

results of a 12-bit CLA adder. This adder will be used as the carry propagate adder in a multiplier which will be described later. The 20-bit CLA adder which is one of the main computing units in a PE also adopts the CPL technique. The silicon area and speed of the 20-bit adder is 4,502 × 1,307 $\mu m^2$ and 5.4 $ns$, respectively.

*B. Multiplier*

The speed of the proposed PE is determined by the multiplier. In order to increase the operation speed, the Wallace multiplier is used. Because a Wallace structure can be separated into Wallace tree blocks, a pipeline procedure can be added and the speed of the PE can be very high. The circuit technique of complementary pass-transistor logic is employed to further increase the speed performance. Figure 4.14 is the functional block

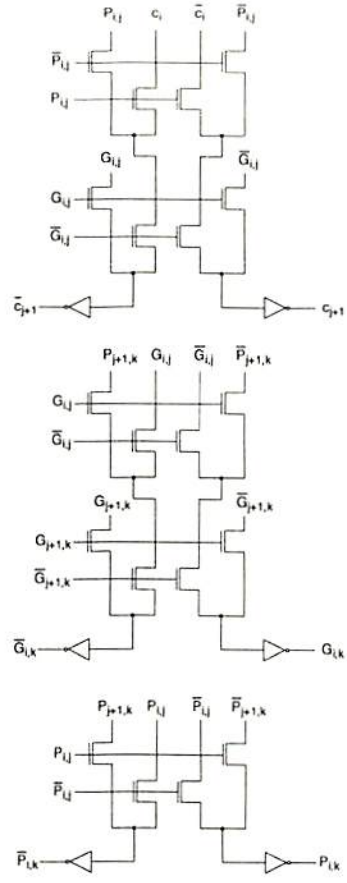


**Figure 4.12: Circuits in block B.**

diagram of an 8-bit Wallace multiplier. A signal and its complement both have to exist during computation in order to apply the CPL technique. The partial product generator produces 64 partial products and their complements from the two 8-bit inputs. The outputs from the partial product generator then are sent to the Wallace adder array for performing carry save additions. The final stage is a Carry Propagate Adder (CPA) whose output is the final product. A detailed description of 8-bit Wallace multiplication is shown in Fig. 4.15. Six carry save addition (CSA) are performed as indicated by alphabetical order A through F. The carry generated in each CSA are "saved" and later becomes the input of another CSA. A 12-bit CLA was chosen to be the CPA necessary in the final stage of a Wallace multiplier and the adder discussed in the previous section can be

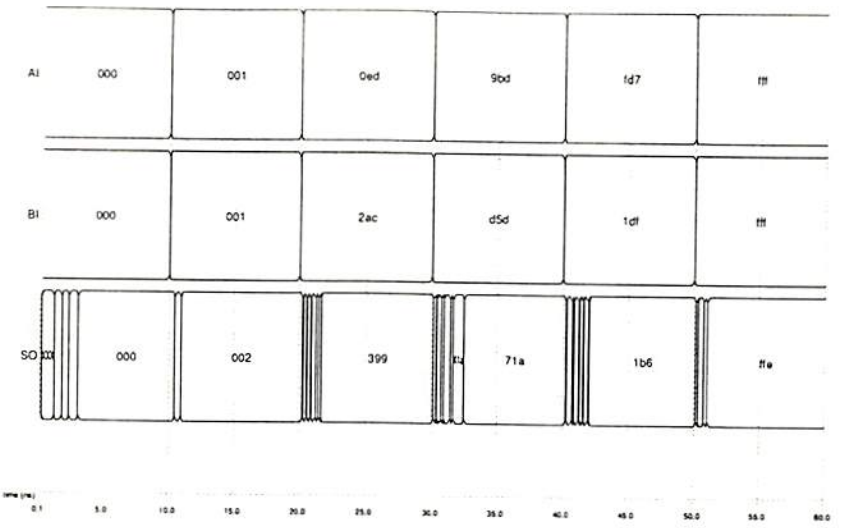



Figure 4.13: Result of 12-bit adder after running irsim simulation.

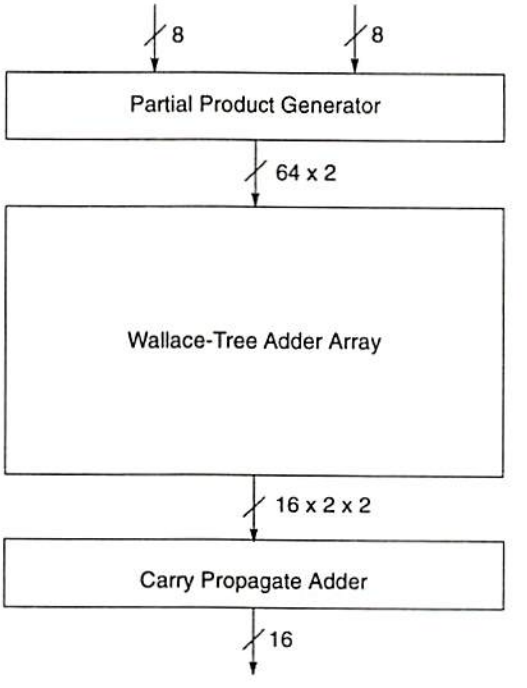



Figure 4.14: Functional block diagram of an 8-bit Wallace multiplier.

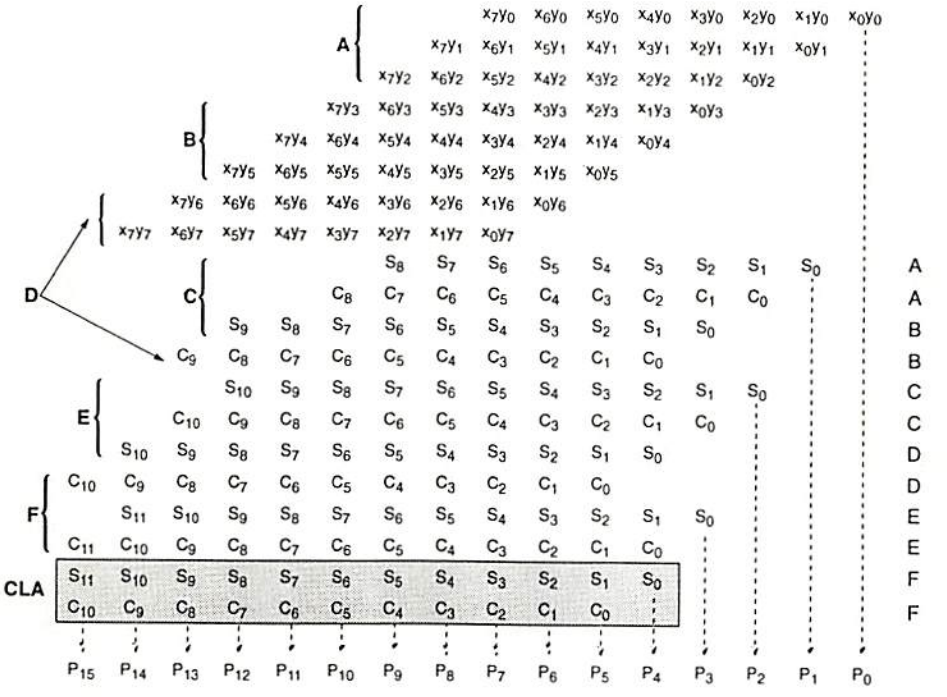



**Figure 4.15: 8-bit Wallace multiplication.**

used. The partial product generator can be constructed by using an AND/NAND module. Figure 4.16 shows the circuit diagram of a 1-bit CSA unit. The result of irsim simulation is shown in Fig. 4.17. The silicon area of an 8-bit multiplier is 4,206 × 3,066 $\mu m^2$, and the speed was measured to be 7.6 $ns$.

#### 4.4.3.3 Cache Memory

One on-chip cache memory with 256 words of 8-bit data is included in each PE to facilitate data processing. The on-chip data cache was made of SRAM. Data in the local memory are communicated to the 64K system memory through pipelining. Sixteen address lines can be either pipelined or broadcast so that data in the consecutive PEs are stored to the consecutive words or to the same words of different PEs. Eight memory sub-blocks constitute the data cache. Each sub-block has 32 word × 8 bit, and has an associative tag field. Figure 4.18 shows the schematic diagram of the control circuitry of the cache. Bit 5

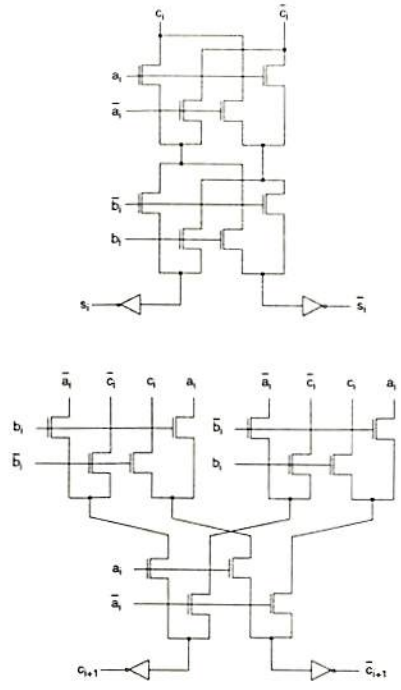


**Figure 4.16: Circuit diagram of a 1-bit CSA unit.**

to bit 7 of the address lines are used to determine which sub-block is used for data access. Bit 8 to bit 15 will be written to the tag when a data is to be stored in a memory location. Bit 0 to bit 4 are used to select a location in the cache memory. When a memory location is to be read, the address lines are compared with the content of the tag.

In case of cache miss, the "hit" signal will trigger the memory management unit to load the missed data. The circuit diagram of the SRAM with a high-speed current-mode sense amplifier is shown in Fig. 4.19. The 2.5-$ns$ memory access time and 5-ns cycle time for 32 words $\times$ 32 bits are achieved. Figure 4.20 shows the timing diagram of a 32 word $\times$ 2 bit memory block.

#### 4.4.3.4 Register File

The circuit for the register file is the same as that for the cache memory except that its address lines are part of the microcodes while the address lines of the cache memory need to be calculated. Eight 20-bit general-purpose registers are used in a PE.

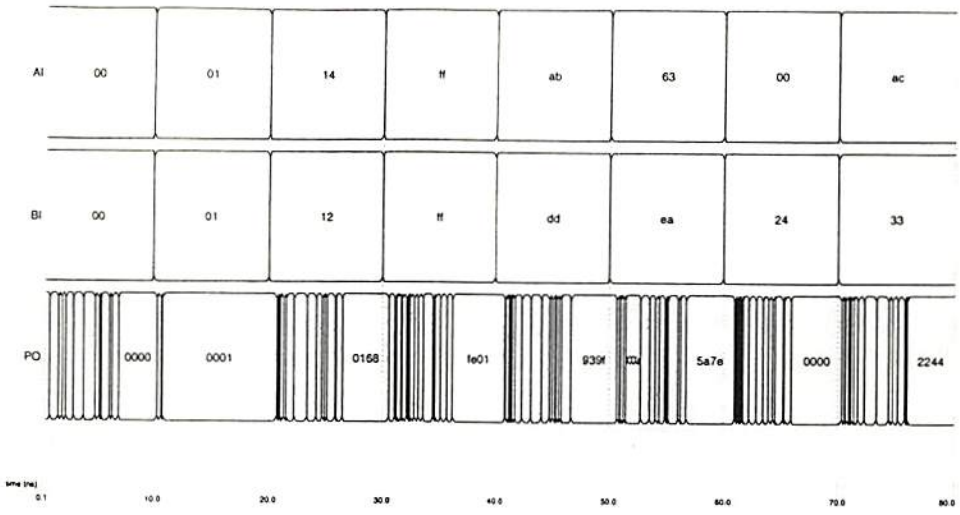



**Figure 4.17: Result of 8-bit multiplier after running irsim simulation.**

#### 4.4.3.5 Input/Output Buffer and Microcodes

Both the input and output buffers are 8-bit single-phase-clocked shift registers. Four input and four output buffers are employed to facilitate mesh-connected PE matrix architecture. When a clock arrives, data in an output buffer of one PE is shifted to an input buffer of the next PE. When bi-directional communication is preferred, additional control signals for the direction of data flow must be added to control the transfer of addresses, microcodes, and data.

#### 4.4.3.6 Power Estimation

Estimates of power are required for several reasons: avoiding metal migration in power and ground routing, and understanding the design trade-off between power versus area and power versus performance. The power and ground lines are never run in polysilicon since it has large resistance, carrying considerable voltage drops. The width of the metal lines is calculated as follows [83]: Calculate the maximum average current drawn by all the active devices connected to a supply line at the lowest level of the tree. Add these currents and then calculate the required width of the power and ground lines as follows.

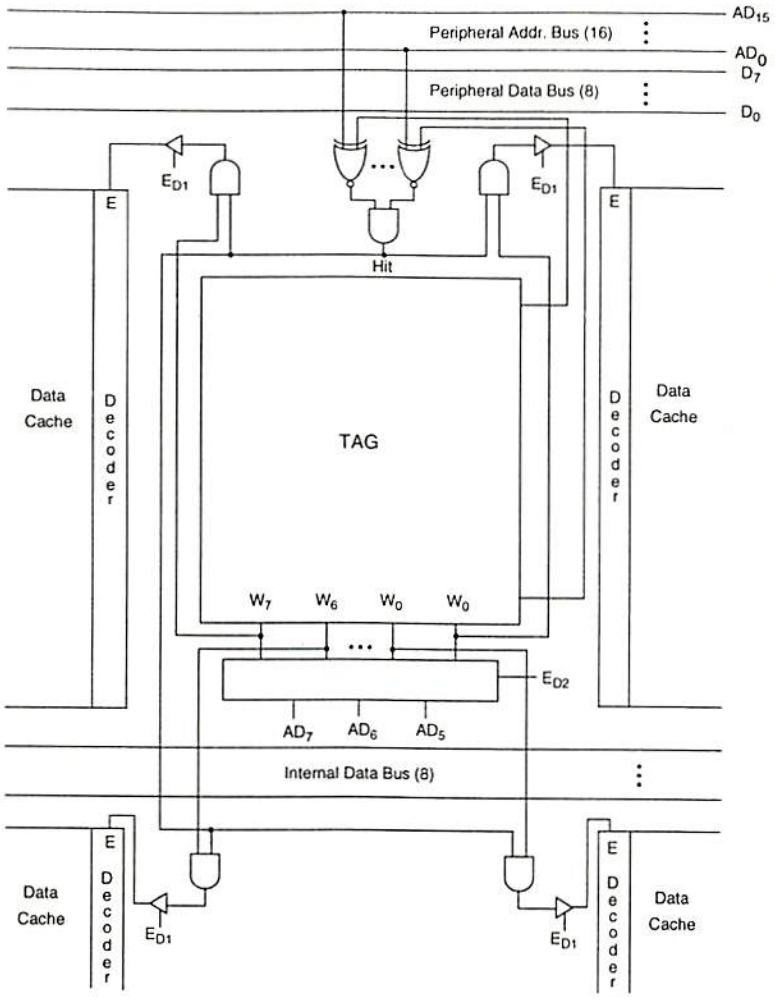



**Figure 4.18: The control circuitry in the cache memory block.**

The maximum current rating for a metal line is about 1 $mA/\mu m$. At the next highest level of the tree, the width must correspond to the sum of all currents flowing into all its lower level branches, and so on.

The critical objective in arranging the power supplies is to insure that power bus voltage noise spikes are small enough so that they have no significant effect on chip operation. In the frame of the chip, I/O switching is the principal source of noise. This noise was suppressed through the use of multiple power supply pads. Also, the output drives were

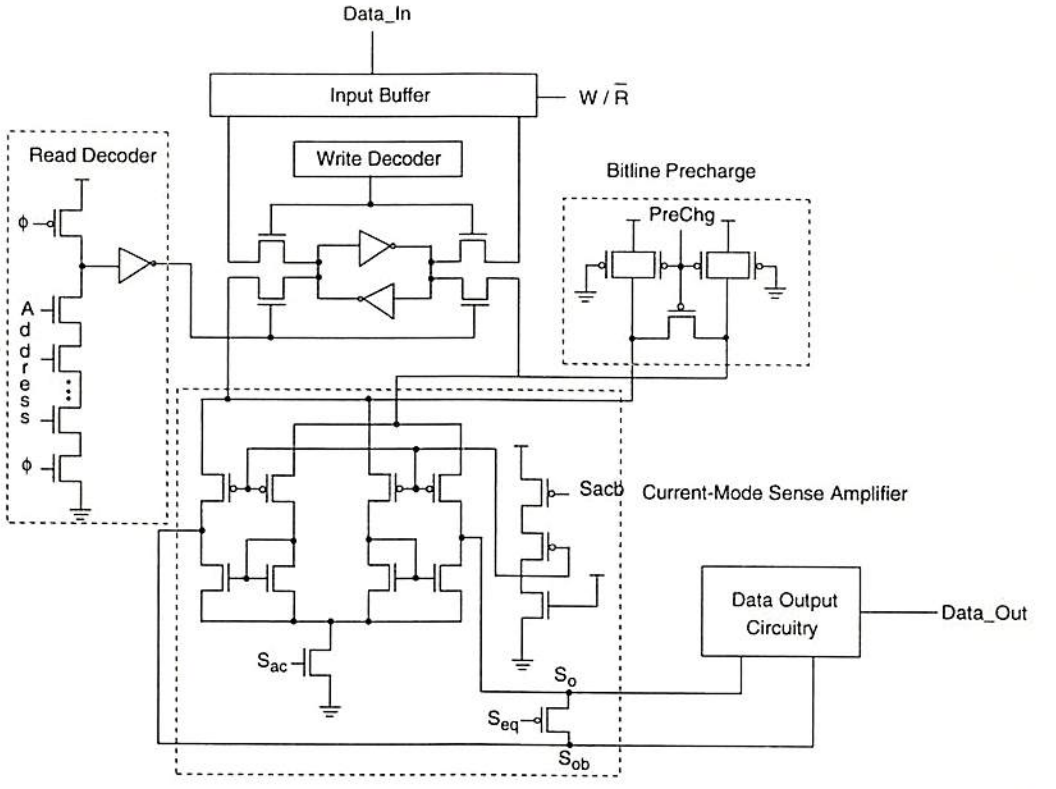



**Figure 4.19: Schematic diagram of selective circuits in the data cache.**

designed to have only enough drive capability to meet external requirements. Excess drive capability tends to make noise problems worse and must be avoided [84].

#### 4.4.3.7 Layout Design

The layout of the PE chip was generated by the Magic layout editor from University of California, Berkeley for maximum design flexibility. Additional capacitors were added between the power and the ground lines in any open space in order to decrease the power-on spark and switching glitches. In the prototype PE design, the data bus is 8 bits and the address bus is 16 bits. If the 16-bit accuracy is needed for some applications, two bytes of data will be sent consecutively. The floorplan of a PE is shown in Fig. 4.21. The die photo of the single PE chip is shown in Fig. 4.22. By arranging the PE layout in a ring-connected array architecture, a 20-PE chip is estimated to occupy a silicon area of

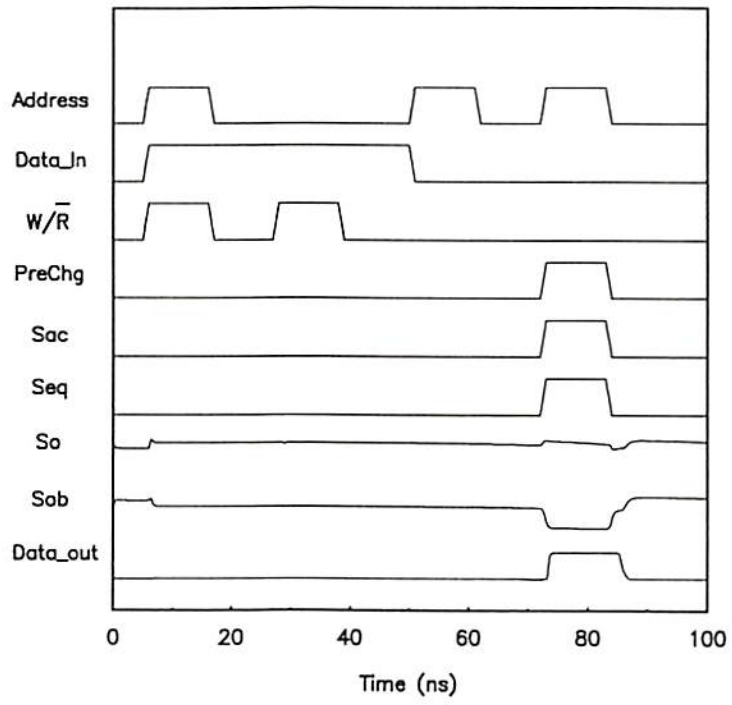



**Figure 4.20: Timing diagram of memory access in a memory block of 32 word × 2 bit.**

$2.09 \times 1.93\ cm^2$ by using a 0.5-$\mu m$ CMOS technology, and to achieve a speed of 2 billion calculations per second. A more compact layout can be designed by using advanced design automation tools, and a 35-PE array can be accommodated in the same area. The relative performance of the proposed multiprocessor chip and the performance requirements of the Grand Challenges on High-Performance Computing [19] are shown in Fig. 4.23. The speeds of the single-processor/superpipelining Alpha chip from DEC [8] and the TMS-320C40 DSP chip from TI [13] are also shown. Table 4.4 summaries the characteristics of a PE.

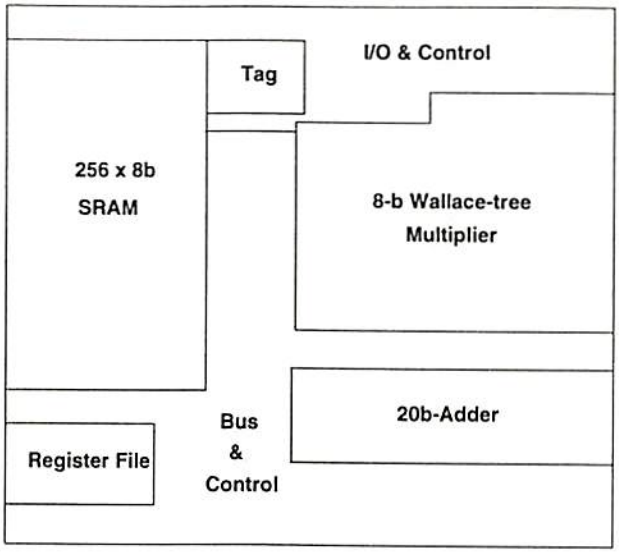



Figure 4.21: Floorplan of a PE.

Table 4.4: Characteristics of a PE.

| | |
|---|---|
| Design Rule | HP 0.8-μm CMOS, 3-metal |
| Chip Size | 6.19 mm x 5.46 mm |
| Operating Frequency | 50 MHz (Simulated) |
| Transistor Count | 42K |
| Supply Voltage | 5 V |
| Power Dissipation | 6.25 mW @ 5 V |
| Accuracy | 8 bits and 16 bits |
| No. of I/O | 145 |
| Design Style | full custom design |
| Signal Representation | digital |
| Data Cache | 256 word per PE |

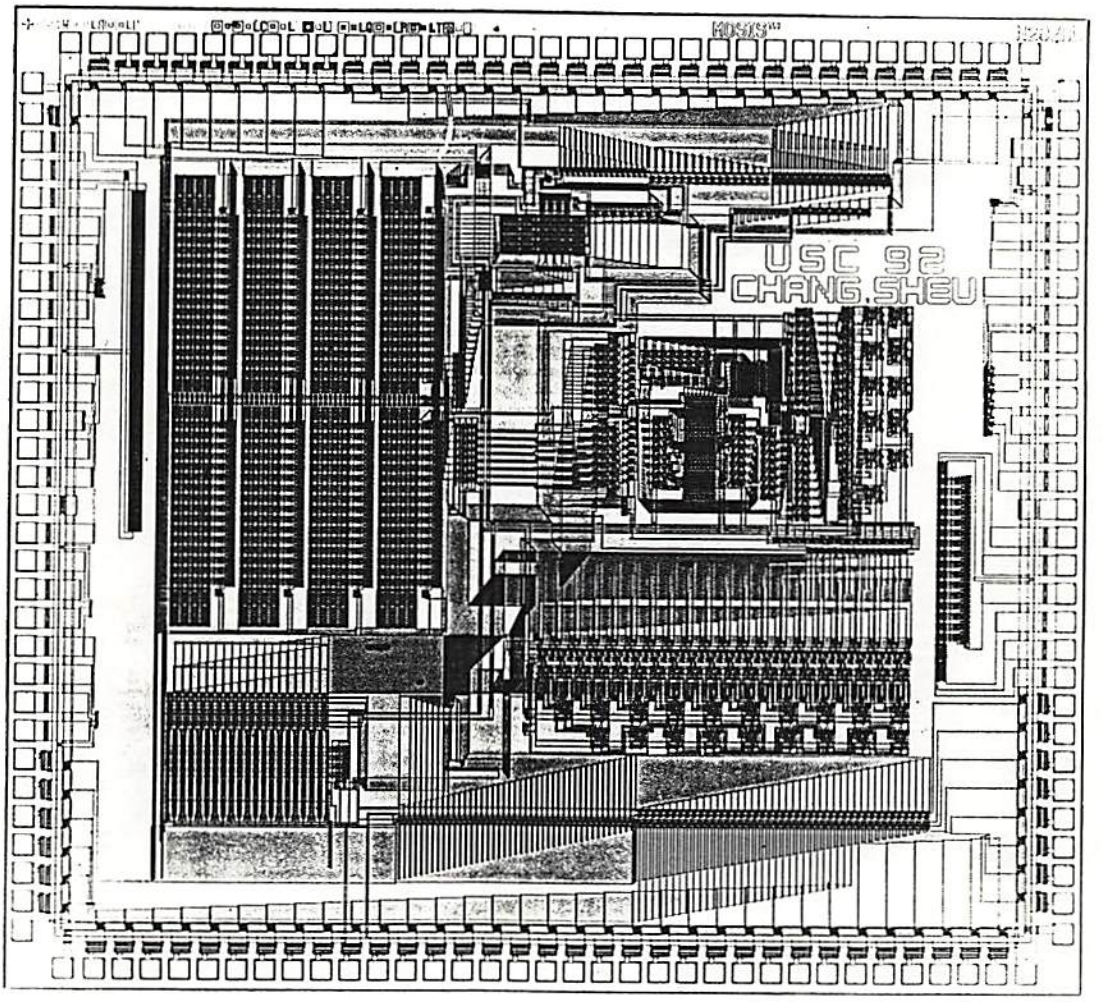



**Figure 4.22: Die photo of the single PE chip fabricated by a 0.8-$\mu$m CMOS technology.**

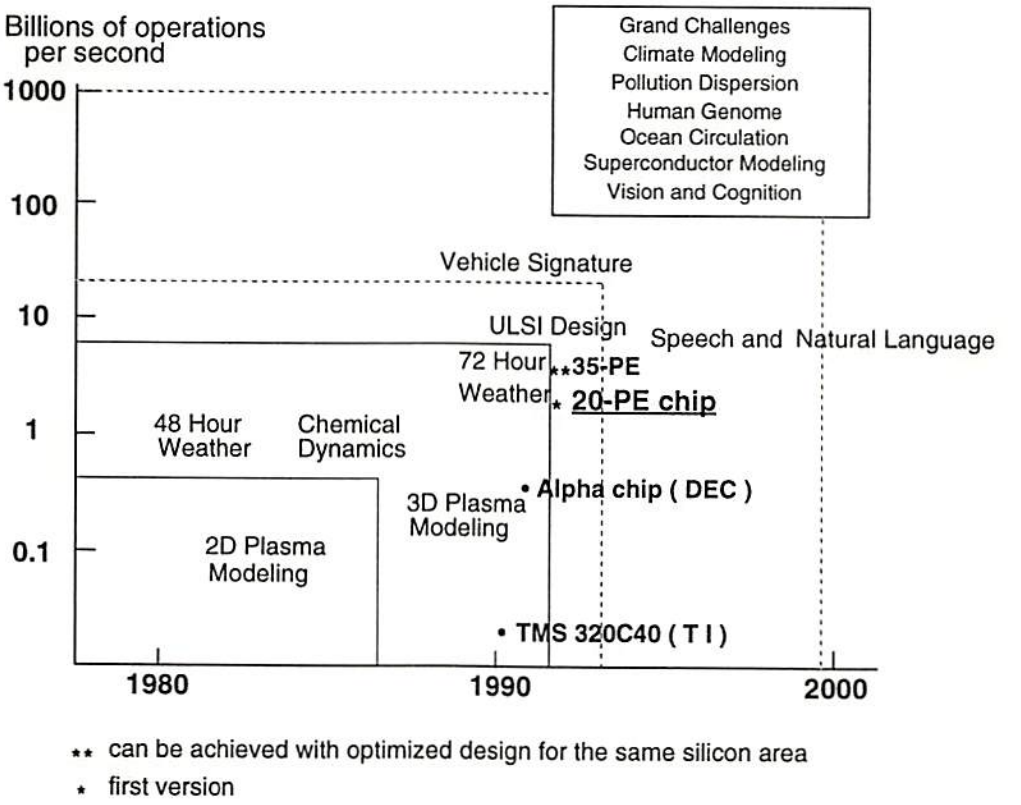



Figure 4.23: Performance requirements of several Grand Challenge problems and the relative performance of the proposed multi-PE chips.

# Chapter 5

# Neural Network Applications

The neural processing element design described in the previous chapter can be connected to form a multi-PE array to assist processing for a symbolic processor. A symbolic processor accesses data from different resources, such as digital data from its cache memory, the main memory and the disk memory. In addition, it can also receive processed data from an optical neural network processor, or an electronic neural network processor. The electronic neural network processor can consist of some digital multi-PE chips and analog front-end neural network chips. The results from the symbolic processor can be used as control signals. Figure 5.1 shows the schematic diagram of an integrated intelligent-processing system.

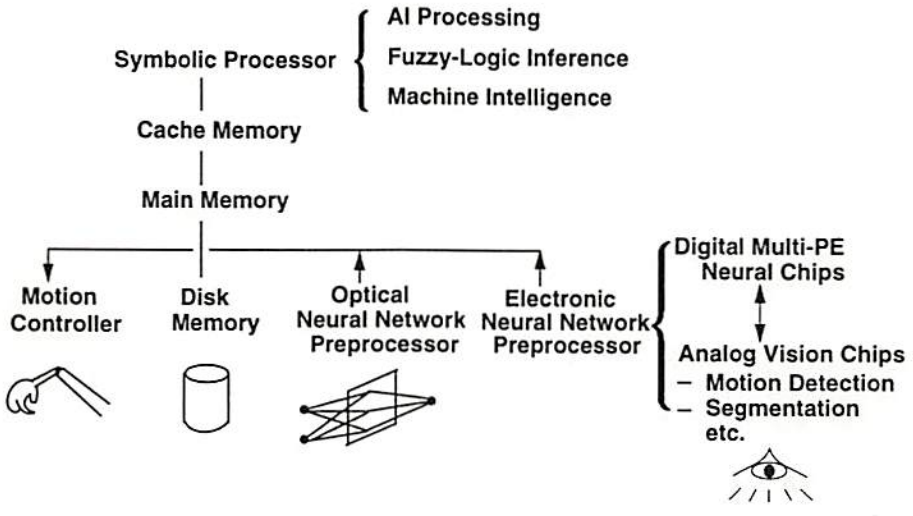



**Figure 5.1: Architecture of an integrated intelligent processing system.**

The artificial neural networks have made significant impacts in pattern recognition, classification, image analysis, system modeling, plant control, optimization, and robotics. This chapter describes the usage of a 20 ring-connected PE chip for both the back-propagation learning rule and the self-organizing feature map. The instruction set for neural network algorithms has been created and is listed in Table 5.1. The instructions are based on the 43 microcodes mentioned in chapter 4. Detailed functions of the microcodes are shown in Table 5.2.

There are four instructions IN, OUT, INOUT and PIPE for inter-PE communication. All the other instructions are for intra-PE data manipulation. The four instructions ADD, SUB, MUL and MOV can be used for different numbers of operands. The two, or three operands of these four instructions come from either the local cache memory or from the register. Every instruction can be set by the controller to be either pipelined or broadcast to a PE array.

## 5.1 Printed Character Recognition

Suppose there are $m$ layers in a neural network and the number of neurons in the $i^{th}$ layer is $n_i$. Let the maximum number of $n_i$'s be $n_{max}$, then $n_{max}$ is the number of ring-connected processing elements. If $n_{max}$ is too large, each processing element represents several neurons in the same layer [59].

During initialization, the synapse data and bias values are stored into the PEs by setting PIPE, INOUT and LDM instructions to be broadcast. The learning ratio is stored in every PE by setting PIPE, INOUT, and LDM instructions to be pipelined. Commands are pipelined during the feedforward phase of a neural network. The instructions for one neuron are shown as follows,

```
          CLR    r1
          ADD    M(addr_b), r1      /* add the bias value */
Loop-1:
          PIPE
          INOUT
```

Table 5.1: A compact instruction set for artificial neural networks.

| Instruction | Functions |
|---|---|
| IN | Sin → InBuf |
| OUT | OutBuf → Sout |
| INOUT | InBuf → OutBuf |
| PIPE | Sin → InBuf,<br>OutBuf → Sout |
| ADD m1, r2 | m1 + r2 → r2 |
| ADD r1, r2 | r1 + r2 → r2 |
| ADD m1, m2, r3 | m1 + m2 → r3 |
| SUB m1, r2 | m1 - r2 → r2 |
| SUB r1, r2 | r1 - r2 → r2 |
| SUB m1, m2, r3 | m1 - m2 → r3 |
| MUL m1, r2 | m1 × r2 → r2 |
| MUL r1, r2 | r1 × r2 → r2 |
| MUL m1, m2, r3 | m1 + m2 → r3 |
| MOV r1, r2 | r1 → r2 |
| MOV m1, r2 | m1 → r2 |
| MOV r1, m2 | r1 → m2 |
| CLR r1 | 0 → r1 |
| LDM m1 | InBuf → m1 |
| LDR r1 | InBuf → r1 |
| SRM m1 | m1 → OutBuf |
| SRR r1 | r1 → OutBuf |

Table 5.2: Descriptions of the 43 Mcodes.

| Name | Functions |
|---|---|
| $S_{mA}$ | the select signal for connection between a register bus and operand A of the multiplier |
| $S_{mB}$ | the select signal for connection between a register bus and operand B of the multiplier |
| $E_{mi1}$ | the signal to enable the multiplexer of the register buses for data transmission to the multiplier |
| $E_{mi2}$ | the signal to enable data to the operands of the multiplier |
| $E_{mw}$ | the signal to enable memory write from the internal data bus |
| $E_{mtc1}$ | the signal to enable connection from the internal data bus to the CPU1 bus |
| $E_{mtc2}$ | the signal to enable connection from the internal data bus to the CPU2 bus |
| $E_{c1tm}$ | the signal to enable connection from CPU1 bus to the internal data bus |
| $E_{c2tm}$ | the signal to enable connection from CPU2 bus to the internal data bus |
| $E_{iotm}$ | the signal to enable connection from the input buffer to the internal data bus |
| $E_{mtio}$ | the signal to enable connection from the internal data bus to the output buffer |
| $E_{ior}$ | the signal to enable connection from the input buffer to the peripheral data bus |
| $E_{iow}$ | the signal to enable connection from the peripheral data bus to the output buffer |
| $E_{p}$ | the signal to enable loading of the result of the multiplier to the REG1 bus |
| $S_{aA}$ | the select signal for connecton between the register buses and the operand A of the adder |
| $S_{aB}$ | the select signal for connection between the register buses and the operand B of the adder |
| $E_{ai1}$ | the signal to enable the multiplexer of the register buses for data transmission to the adder |
| $E_{ai2}$ | the signal to enable data to the operands of the adder |
| $E_{s}$ | the signal to enable loading of the result of the adder to the REG2 bus |
| $S_{e}$ | the signal to enable subtraction |
| AR0 | register address decoding line |
| AR1 | register address decoding line |
| AR2 | register address decoding line |
| $R_{e}$ | register decoder enable line |
| $E_{iR1}$ | the signal to enable data from the REG1 bus to a register |
| $E_{oR1}$ | the signal to enable data to the REG1 bus from a register |
| $E_{iR2}$ | the signal to enable data from the REG2 bus to a register |
| $E_{oR2}$ | the signal to enable data to the REG2 bus from a register |
| $E_{ic1}$ | the signal to enable data from the CPU1 bus to a register |
| $E_{oc1}$ | the signal to enable data fo the CPU1 bus from a register |
| $E_{D1}$ | enable the first level decoder |
| $E_{D2}$ | enable the second level decoder |
| $S_{eqr}$ | the precharge signal for reading data from the register |
| $E_{rw}$ | the write signal for the register |
| $S_{eqm}$ | the precharge signal for reading data from the memory |
| $E_{com}$ | the signal to enable tag comparison |
| $S_{eqt}$ | the precharge signal for reading data from the tag |
| $E_{tw}$ | the write signal for the tag |
| $E_{addin}$ | the signal for receving address |
| $E_{addout}$ | the signal for sending address |
| $\phi_{a1}$ | clock 1 for address transmission |
| $\phi_{a2}$ | clock 2 for address transmission |
| $\phi_{data}$ | clock for data transmission |

```
LDR    r2
MUL    r2, M(addr_s), r3 /* synapse multiplication */
ADD    r3, r1
JNZ    Loop-1, count1   /* loop handled by the local controller */
SRR    r1
```

According to the back-propagation learning rule, the instructions for the PEs are described as follows,

(1) Output Layer:

By setting instructions to be broadcast, the desired output data are piped into the PEs. The differences between the desired data and the actual output data are calculated. Synapse weight values are updated by applying the following procedure and setting all instructions to be pipelined:

```
PIPE                    /* pipe in the previous output data of
                           neurons in the (m - 1)^th layer */
INOUT
LDR    r1
MUL    r1, r2           /*input data multiply the difference value
                           between the desired output data and the
                           actual data */
MUL    M(addr_l), r2    /* multiplied by the learning ratio */
ADD    M(addr_s), r2    /* added by the old synapse data */
MOV    r2, M(addr_s)    /* update the synapse data */
```

(2) Hidden Layer:

By setting MUL and SRR instructions to be broadcast, the propagated error from each synapse connection is calculated. The summation from each propagated error is performed by setting PIPE, INOUT, LDR, ADD, and SRR instructions to be pipelined into PEs. The synapse updating procedure is the same as the one shown previously in the section of the output layer.

As an example, a neural network which contains 64 input neurons, 20 hidden neurons, and 5 output neurons is selected. $n_{max}$ is 20 in this network and therefore the network

can be implemented by 20 PEs in the form of linear array. Each PE contains 64 synapse values. After 1,000 iterations, 30% of instructions are for input/output operations. In the feedforward phase, both SUB and MOV instructions are not used. The profile of the instruction set for the 3-layer neural network using back-propagation learning is listed in Table 5.3. Figure 5.2 shows results of applying this network to recognize five characters.

**Table 5.3: Profile of the instruction set for a 3-layer neural network using back-propagation learning rule after 1,000 iterations.**

| Profile / Instruction | Feedforward | | Feedforward & backward training | |
|---|---|---|---|---|
| | Number | Percentage | Number | Percentage |
| PIPE | 125,406 | 24.64% | 825,406 | 30.44% |
| INOUT | 125,406 | 24.64% | 825,406 | 30.44% |
| LDM m1 | 87 | 0.02% | 1,087 | 0.04% |
| LDR r1 | 84,000 | 16.51% | 288,000 | 10.62% |
| CLR r1 | 2,000 | 0.39% | 22,000 | 0.81% |
| ADD ** | 86,000 | 16.90% | 270,000 | 9.95% |
| SUB ** | 0 | 0 | 1,000 | 0.04% |
| MUL ** | 84,000 | 16.51% | 272,000 | 10.03% |
| SRR r1 | 2,000 | 0.39% | 122,000 | 4.50% |
| MOV ** | 0 | 0 | 85,000 | 3.13% |
| Note : The neural network contains 64 input neurons, 20 hidden neurons, and 5 output neurons. | | | | |

** A collection of similar instructions

Each character is represented by a $8 \times 8$ frame which consists of 64 binary values. After training the 3-layer network for 1,000 times, each character can be recognized as one class.

## 5.2 Image Compression

Self-organizing feature map is a two-step procedure that couples the feedforward process with the learning process in a two-layer neural network. The network consists of an input layer and an output competitive layer. In the competition process, only the synapses

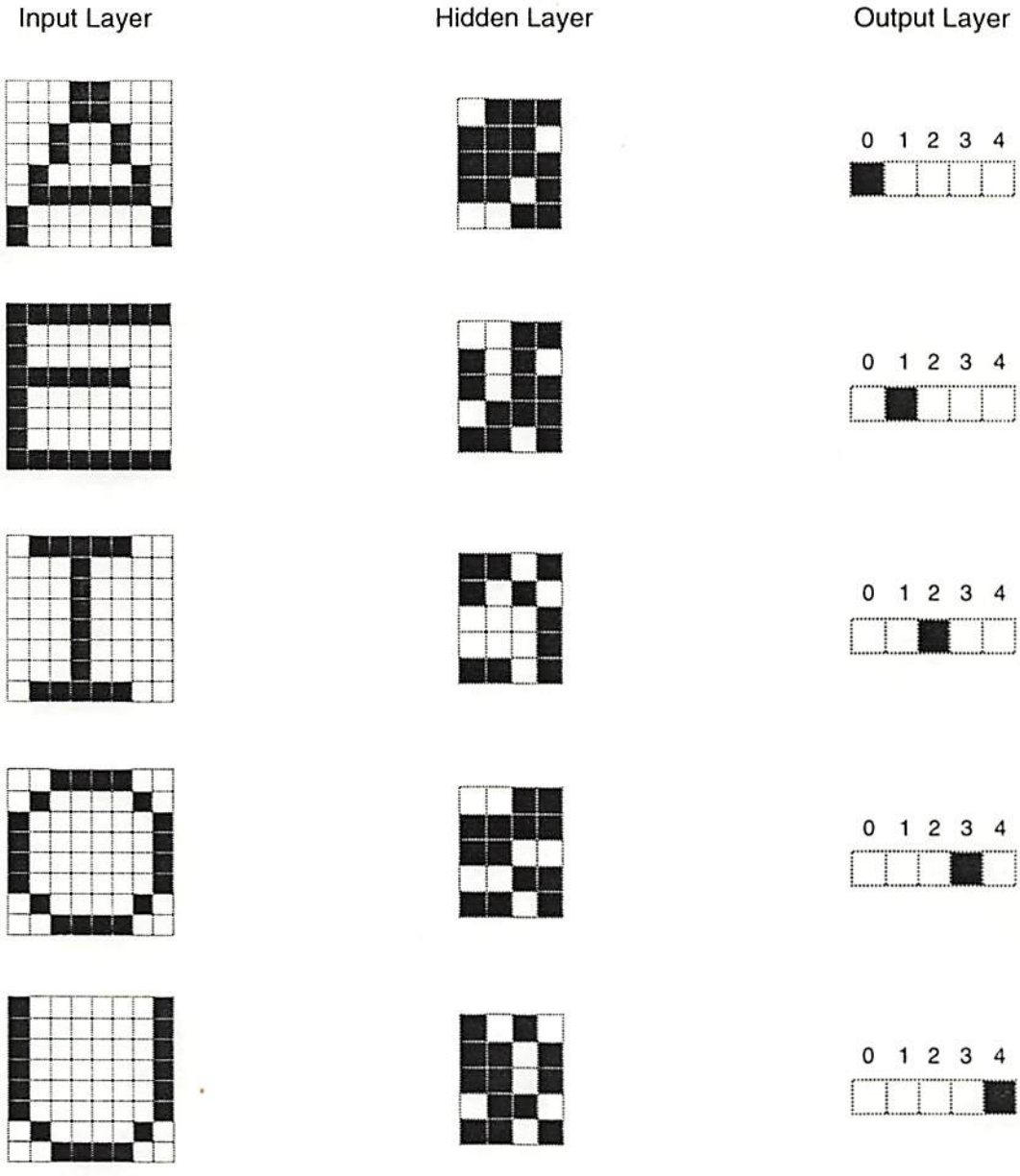



**Figure 5.2: Application of the back-propagation network to printed character recognition. 1,000 iterations were performed during training.**

connected to the winning neuron are to be modified by the training sample. The number of neurons in the competitive layer can be the number of the ring-connected processing elements. If the number of neurons in the competitive layer is too large, each processing element can be arranged to represent several neurons.

The synapse data are first stored in the PEs by setting PIPE, INOUT, and LDM instructions to be broadcast. The network is then feedforward propagated by setting all instructions to be pipelined. The procedure for one PE in the feedforward operation are shown as follows,

```
          CLR   r1
Loop-2:
          PIPE
          INOUT
          LDR   r2
          SUB   M(addr_s), r2      /* synapse data minus the input data */
          MUL   r2, r2             /* square the difference */
          ADD   r2, r1             /* accumulate the squared difference */
          JNZ   Loop-2, count2     /* controlled by the local controller */
          RR
```

By using broadcast instructions, the learning ratios are piped into the processor array. The learning ratio of the winning neuron is nonzero while all the other neurons are zero. The synapse values associated with this winner are updated. The following procedure updates the synapse value and is pipelined.

```
Loop-3:
          PIPE
          INOUT
          LDR   r1
          SUB   M(addr_s), r1      /* synapse data minus the input data */
          MUL   r2, r1             /* multiplied by the learning ratio */
          SUB   M(addr_s), r1      /* subtracted by the synapse data */
          MOV   r1, M(addr_s)      /* update the synapse data */
          JNZ   Loop-3, count3     /* controlled by the local controller */
```

Table 5.4 lists the profile of the instruction set for the neural network using self-organizing learning rule. The network topology consists of 16 input neurons, and 256

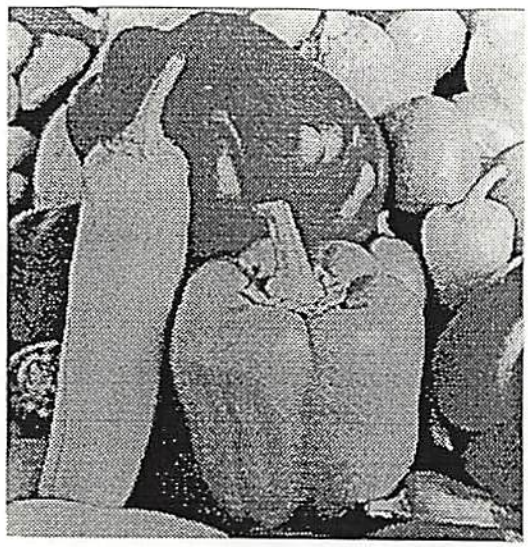
**Table 5.4: Profile of the instruction set for a self-organizing network after 4,096 iterations.**

| Profile / Instruction | Feedforward | | Feedforward & network training | |
|---|---|---|---|---|
| | Number | Percentage | Number | Percentage |
| PIPE | 1,921,024 | 26.11% | 2,068,480 | 25.91% |
| INOUT | 1,921,024 | 26.11% | 2,068,480 | 25.91% |
| LDM m1 | 208 | 0.003% | 208 | 0.003% |
| LDR r1 | 851,968 | 11.58% | 921,600 | 11.55% |
| CLR r1 | 53,248 | 0.73% | 53,248 | 0.67% |
| ADD ** | 851,968 | 11.58% | 851,968 | 10.67% |
| SUB ** | 851,968 | 11.58% | 983,040 | 12.31% |
| MUL ** | 851,968 | 11.58% | 917,504 | 11.49% |
| SRR r1 | 53,248 | 0.73% | 53,248 | 0.67% |
| MOV ** | 0 | 0 | 65,536 | 0.82% |
| Note : The neural network contains 16 input neurons, and 256 output neurons. | | | | |

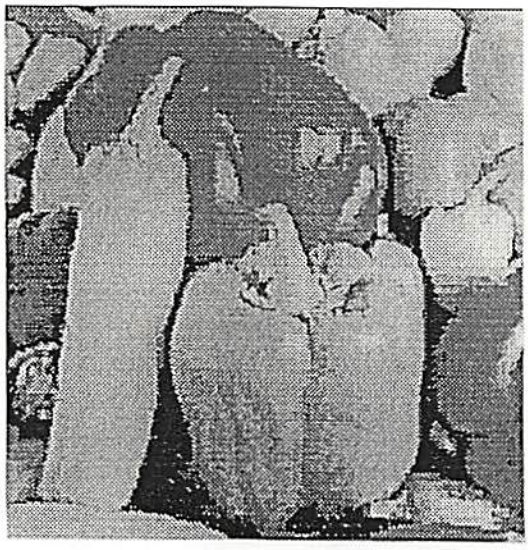
** A collection of similar instructions

output neurons. After 4,096 iterations, 25% of the instructions are for input/output operations. In the feedforward phase, the MOV instruction is not used since there is no synapse updating operation. When mapping this network to the 20-PE chip, 16 active PEs are for operation and 4 PEs are reserved for fault tolerance. Each PE represents 16 output neurons.

Figure 5.3 shows the result for applying the self-organizing network to vector quantization. Sixteen input neurons are used to represent the $4 \times 4$ window vector, 256 output neurons represent the codebook. The $256 \times 256$-pixel Pepper image needs 4,096 iterations for the codebook training to achieve a mean-squared error of 155.



(a)



(b)

**Figure 5.3: Image compression on the multi-PE chip using a self-organizing neural network. (a) The original image. (b) The reconstructed image.**

# Chapter 6

# Impact of Dissertation Work and Future Directions

Artificial neural networks provides solutions to certain complex problems. However, for time-critical applications such as real-time signal and image processing, the highly parallel neural network architectures must be implemented in parallel hardware in order to be real-time. The overall objective of this research was to develop and demonstrate a parallel neural network hardware with the ability of learning from examples.

Under the environment of accessing available technologies, a digital VLSI neural processing element has been designed. This processing element can be cascaded to achieve higher degree of parallelism. The design is based on a full-custom design approach to create high-speed modules by dint of hardware-software codesign. A prototype chip was fabricated through the MOSIS service of USC/Information Science Institute by using a 0.8-$\mu m$ CMOS technology. The chip allows us to demonstrate possible embeddable neuroprocessors which offer orders-of-magnitude speed enhancement in the required functionality. Furthermore, some of the hardware technology can be transferred to the industry.

At every step of the research work, significant results were documented and presented to the research community in the form of refereed publications and conference presentations. Also presented at the end is a list of publication which has resulted from the work carried out as a part of this research.

In future research, further investigation and understanding of brain functions can be conducted, and thereby creating better neuron models to solve problems of massively parallel processing. A flexible representation system of information from raw sensory data

such as visual images and auditory signals to high-level symbolic languages can also be investigated. Efficient neural network architectures and new hardware design techniques need to be explored further for large-scale real-world applications. Building resuable software and hardware modules can help the goal of creating one-million-unit neural networks to interact with external environment and to change structures adaptively In consideration of constructing an intelligent machine, besides neural networks, several hardware and software issues are involved including methods of massively parallel computing, logic and fuzzy programming, device and packaging technologies, and low-power design. Further, optical interconnection through optical fiber or optoelectronic components provides another scheme for intelligent inference machine and information processing system design.

## Reference List


[1] J. Herath, "Computer architectures for intelligent systems," *IEEE Computer Magazine*, vol. 25, no. 5, pp. 6–9, May 1992.

[2] R. Kurzweil, *The Age of Intelligent Machines.* MIT Press: Cambridge, MA, 1990.

[3] M. Asakura *et al.*, "A 34ns 256Mb DRAM with boosted sense-ground scheme," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 140–141, 324, San Francisco, CA, Feb. 1994.

[4] K. Self, "Solid state," *IEEE Spectrum Magazine*, vol. 30, no. 1, pp. 46–49, Jan. 1993.

[5] A. M. Miscione, R. V. Alemeida, H. J. Hennecke, and R. A. Mann, "TRW CPUAX superchip," in *IEEE Record on Hot Chips Symp.*, pp. 2.22–2.32, Palo Alto, CA, Aug. 1991.

[6] L. Kohn and N. Margulis, "Introducing the Intel i860 64-bit microprocessor," *IEEE Micro Magazine*, vol. 9, no. 4, pp. 15–30, Aug. 1989.

[7] Sun Microsystems, Inc., Mountain View, CA, *SPARC Architecture Manual*, 1987.

[8] D. Dobberpuhl *et al.*, "A 200MHz 64b dual-issue CMOS microprocessor," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 106–107, 256, San Francisco, CA, Feb. 1992.

[9] K. Suzuki *et al.*, "A 500MHz 32b 0.4$\mu$m CMOS RISC processor LSI," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 214–215, 341, San Francisco, CA, Feb. 1994.

[10] D. Alpert and D. Avnon, "Architecture of the Pentium microprocessor," *IEEE Micro Magazine*, vol. 13, no. 3, pp. 11–21, Jun. 1993.

[11] N. P. Jouppi *et al.*, "A 300MHz 115W 32b bipolar ECL microprocessor with on-chip caches," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 84–85, 266, San Francisco, CA, Feb. 1993.

[12] J. Schutz, "A 3.3V 0.6$\mu$m BiCMOS superscalar microprocessor," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 202–203, 338, San Francisco, CA, Feb. 1994.

[13] Texas Instruments, Inc., Dallas, TX, *TMS 320C4x User's Guide*, 1991.

[14] Texas Instruments, Inc., Dallas, TX, *Product Bulletin: MVP TMS320C80 DSP*, 1994.

[15] H. Komiya, "Future technological and economic prospects for VLSI," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 16–19, San Francisco, CA, Feb. 1993.

[16] M. S. Adler, E. J. Wildi, W. Daum, and C. A. Becker, "Multi-chip module for analog and microwave: dc to 18 GHz," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 170–171, 285, San Francisco, CA, Feb. 1993.

[17] M. Gdula *et al.*, "A 36-chip multiprocessor module made with General Electric high density interconnect technology," in *Proc. 41st Electronic Components and Technology Conf.*, p. 727, May 1991.

[18] C.-F. Chang and B. J. Sheu, "A multi-chip module design for portable video compression systems," in *Proc. IEEE Multi-Chip Module Conf.*, pp. 39–44, Santa Cruz, CA, Mar. 1993.

[19] U. G. Committee, *Grand Challenges: High Performance Computing and Communications.* National Science Foundation, 1991.

[20] B. Kosko, *Neural Networks for Signal Processing.* Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1992.

[21] N. Morgan, *Artificial Neural Networks Electronic Implementations.* IEEE Computer Society Press: Washington, DC, 1990.

[22] D. Hammerstrom, "Electronic neural network implementation," in *Tutorial Notes of IEEE/INNS Int'l Joint Conf. Neural Networks*, Baltimore, MA, Jun. 1992.

[23] G. M. Shepherd, *Neurobiology.* Oxford University Press: New York, NY, 2nd ed., 1988.

[24] DARPA, *DARPA Neural Network Study.* AFCEA International Press: Fairfax, VA, 1988.

[25] J. M. Zurada, *Introduction to Artificial Neural Systems.* West Publishing Company: St. Paul, MN, 1992.

[26] M. A. Arbib, *Brains, Machines and Mathematics.* Springer-Verlag: New York, NY, 2nd ed., 1987.

[27] P. K. Simpson, "Foundation of neural networks," in *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations* (E. Sánchez-Sinencio and C. Lau, eds.), pp. 29–37, IEEE Press: New York, NY, 1992.

[28] R. Hecht-Nielsen, *Neurocomputing.* Addison-Wesley Publishing Company, Inc.: Reading, MA, 1990.

[29] J. Dayhoff, *Neural Network Architectures: An Introduction.* Van Nostrand Reinhold: New York, NY, 1990.

[30] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE Acoust., Speech, Signal Processing Magazine*, vol. 4, pp. 4–22, Apr. 1987.

[31] J.-D. Yuh and R. W. Newcomb, "A multilevel neural network for A/D conversion," *IEEE Trans. Neural Networks*, vol. 4, no. 3, pp. 470–483, May 1993.

[32] J. J. Hopfield, "Neurons with graded response have collective computational properties those of two-state neurons," *Proc. National Academy of Science, U.S.A.*, vol. 81, pp. 3088–3092, May 1984.

[33] 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. Circuits and Systems*, vol. 33, no. 5, pp. 533–541, May 1986.

[34] B. Kosko, "Adaptive bidirectional associative memories," *Applied Optics*, vol. 26, pp. 4947–4960, Dec. 1987.

[35] J. A. Anderson, J. W. Silverstein, S. A. Ritz, and R. S. Jones, "Distinctive features, categorical perception, and probability learning: Some applications of a neural model," *Psychol. Rev.*, vol. 84, no. 5, no. 5, pp. 413–451, 1977.

[36] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. National Academy of Science, U.S.A*, vol. 79, pp. 2554–2558, Apr. 1982.

[37] D. Hammerstrom, "Working with neural networks," *IEEE Spectrum Magazine*, vol. 30, no. 7, pp. 46–53, Jul. 1993.

[38] C. von der Malsburg, "Self-organization of orientation sensitive cells in the striate cortex," *Kybernetik*, vol. 14, pp. 85–100, 1973.

[39] S. Grossberg, "Competitive learning: from interactive activation to adaptive resonance," *Cognitive Sci.*, vol. 11, pp. 23–67, 1987.

[40] T. Kohonen, *Self-Organization and Associative Memory*. Springer-Verlag: New York, NY, 2nd ed., 1988.

[41] R. Gray, "Vector quantization," *IEEE Acoust., Speech, Signal Processing Magazine*, vol. 1, pp. 4–29, Apr. 1984.

[42] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, no. 1, pp. 84–95, Jan. 1980.

[43] G. G. S. III, J. A. Reggia, and J. M. Maisog, "Supervised and reinforced competitive learning," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. I, pp. 563–567, San Diego, CA, Jun. 1990.

[44] A. K. Krishnamurthy, S. C. Ahalt, D. E. Melton, and P. Chen, "Neural networks for vector quantization of speech and images," *IEEE Jour. Selected Areas in Commun.*, vol. 8, no. 8, pp. 1449–1457, Oct. 1990.

[45] J. Lee, "Digital motion estimation and image restoration on VLSI," Tech. Rep. USC-SIPI Report No. 183, Signal and Image Processing Institute, University of Southern California, University Park/MC-2564, Los Angeles, CA 90089, 1991.

[46] J. Alspector, A. Jayakumar, and S. Luna, "Experimental evaluation of learning in a neural microsystems," *Proc. of Neural Information Processing Systems*, vol. 4, pp. 871–878, 1991.

[47] E. Rich and K. Knight, *Artificial Intelligence*. McGraw-Hill Publishing Company, Inc.: New York, NY, 2nd ed., 1991.

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

[49] S. M. Sze, *VLSI Technology*. McGraw-Hill Publishing Company, Inc.: New York, NY, 2nd ed., 1988.

[50] C. Peterson and J. R. Anderson, "A mean field learning algorithm for neural networks," *Complex Systems*, vol. 1, no. 5, no. 5, pp. 995–1019, 1987.

[51] C. Peterson, "Mean field theory neural networks for feature recognition, content addressable memory and optimization," *Connection Science*, vol. 3, pp. 3–33, 1991.

[52] B. J. Sheu, J. Choi, and C.-F. Chang, "An analog neural network processor for self-organizing mapping," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 136–137, 266, San Francisco, CA, Feb. 1992.

[53] M. Holler, S. Tam, H. Castro, and R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. II, pp. 191–196, Washington, DC, Jun. 1989.

[54] B. W. Lee and B. J. Sheu, "Parallel hardware annealing for optimal solutions on electronic neural networks," *IEEE Trans. Neural Networks*, vol. 4, no. 4, pp. 588–599, Jul. 1993.

[55] B. W. Lee and B. J. Sheu, *Hardware Annealing in Analog VLSI Neurocomputing*. Kluwer Academic Publishers: Boston, MA, 1991.

[56] A. Kalavade and E. A. Lee, "A hardware-software codesign methodology for DSP applications," *IEEE Design & Test of Computers Magazine*, vol. 10, no. 3, pp. 16–28, Sep. 1993.

[57] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing*. McGraw-Hill Publishing Company, Inc.: New York, NY, 1990.

[58] H. Kato, H. Yoshizawa, H. Iciki, and K. Asakawa, "A parallel neurocomputer architecture towards billion connection updates per second," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. II, pp. 47–50, San Diego, CA, Jun. 1990.

[59] J.-N. Hwang, J. A. Vlontzos, and S.-Y. Kung, "A systolic neural network architecture for Hidden Markov Models," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 12, pp. 1967–1979, Dec. 1989.

[60] K. W. Przytula, W.-M. Lin, and V. K. P. Kumar, "Partitioned implementation of neural networks on mesh connected array processors," *VLSI Signal Processing IV*, IEEE Press: New York, NY, pp. 106–115, Aug. 1991.

[61] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing* (D. E. Rumelhart and J. L. McClelland, eds.), vol. 1, pp. 318–362, MIT Press: Cambridge, MA, 1986.

[62] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. II, pp. 537–544, San Diego, CA, Jun. 1990.

[63] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, and B. Riley, "An 11-million transistor neural network execution engine," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 180–181, 313, San Francisco, CA, Feb. 1991.

[64] U. Ramacher, J. Beichter, and N. Bruls, "Architecture of a general-purpose neural signal processor," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. I, pp. 443–446, Seattle, WA, Jul. 1991.

[65] D. A. Orrey, D. J. Myers, and J. M. Vincent, "A high performance digital processor for implementing large artificial neural networks," in *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 16.3.1–16.3.4, San Diego, CA, May 1991.

[66] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer, "The ring array processor (RAP): A multiprocessing peripheral for connectionist applications," *Jour. Parallel and Distributed Computing*, vol. 14, pp. 248–259, 1992.

[67] M. Yasunaga *et al.*, "Design, fabrication and evaluation of 5-inch wafer scale neural network LSI composed of 576 digital neurons," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. II, pp. 527–535, San Diego, CA, Jun. 1990.

[68] Hecht-Nielsen Neurocomputers Company, San Diego, CA, *SNAP-SIMD Neurocomputer Array Processor Databook*, 1992.

[69] H. P. Graf and D. Henderson, "A reconfigurable CMOS neural network," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 144–145, 285, San Francisco, CA, Feb. 1990.

[70] B. E. Boser and E. Säckinger, "An analog neural network processor with programmable network topology," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 184–185, 314, San Francisco, CA, Feb. 1991.

[71] Y. Arima *et al.*, "A 336-neuron 28k-synapse self-learning neural network chip with branch-neuron-unit architecture," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 182–183, 313, San Francisco, CA, Feb. 1991.

[72] M. S. Melton, T. Phan, D. S. Reeves, and D. E. V. den Bout, "The TInMANN VLSI chip," *IEEE Trans. Neural Networks*, vol. 3, no. 3, pp. 375–384, May 1992.

[73] C.-F. Chang and B. J. Sheu, "Design of a digital VLSI neuroprocessor for signal and image processing," in *Proc. IEEE-SP Neural Networks for Signal Processing*, pp. 606–615, Princeton, NJ, Oct. 1991.

[74] C.-F. Chang, B. J. Sheu, and H. Okada, "Design of a multiprocessor dsp chip for flexible information processing," in *Proc. IEEE Int'l Conf. Acoust., Speech, Signal Processing*, vol. V, pp. 637–640, San Francisco, CA, Mar. 1992.

[75] C.-F. Chang and B. J. Sheu, "Digital VLSI multiprocessor design for neurocomputers," in *Proc. IEEE/INNS Int'l Joint Conf. Neural Networks*, vol. II, pp. 1–6, Baltimore, MA, Jun. 1992.

[76] L. O. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Trans. Circuits and Systems*, vol. 35, no. 10, pp. 1257–1272, Oct. 1988.

[77] J. M. Cruz and L. O. Chua, "A CNN chip for connected component detection," *IEEE Trans. Circuits and Systems*, vol. 38, no. 7, pp. 812–817, Jul. 1991.

[78] B. Preas and M. L. (eds.), *Physical Design Automation of VLSI Systems.* The Benjamin/Cummings Publishing Company: Menlo Park, CA, 1988.

[79] A. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE Jour. Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[80] K. Yano *et al.*, "A 3.8-ns CMOS 16x16-b multiplier using complementary pass-transistor logic," *IEEE Jour. Solid-State Circuits*, vol. 25, no. 2, pp. 388–395, Apr. 1990.

[81] M. Kuczynski *et al.*, "A 1Mb/s digital subscriber line transceiver signal processor," in *Tech. Digest IEEE Int'l Solid-State Circuits Conf.*, pp. 26–27, 256, San Francisco, CA, Feb. 1993.

[82] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc.: San Mateo, CA, 1990.

[83] A. Mukherjee, *Introduction to nMOS and CMOS VLSI Systems Design.* Prentice-Hall, Inc.: Englewood Cliffs, NJ, 1986.

[84] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design.* Addison-Wesley Publishing Company, Inc.: Reading, MA, 1993.

# Appendix A

# Publications Out of the Dissertation Work

### Journal Papers

- J. Choi, B. J. Sheu, and **J. C.-F. Chang**, "A Gaussian synapse circuit for analog VLSI neural networks," *IEEE Trans. on VLSI Systems*, vol. 2, no. 1, pp. 129-133, Mar. 1994.
- S. H. Bang, O. T.-C. Chen, **J. C.-F. Chang**, and B. J. Sheu, "Multi-level neural networks with optimal solutions," accepted by IEEE Trans. on Circuits and Systems, Part II, for publication in late 1994.
- J. Choi, B. J. Sheu, and **J. C.-F. Chang**, "Design and test of VLSI general-purpose neural network hardware," submitted to IEEE Trans. on Neural Networks.

### Conference Papers

- **J. C.-F. Chang**, B. J. Sheu, W.-C. Fang, and J. Choi, "A trainable analog neural chip for image compression," *IEEE Proc. of Custom Integrated Circuits Conf.*, pp. 16.1.1-16.1.4, San Diego, CA, May 1991.
- B. J. Sheu, **J. C.-F. Chang**, T.-H. Chen, and O. T.-C. Chen, "Neural-based trainable vector quantizer and digital systolic processors," *IEEE Proc. of Int. Symp. on Circuits and Systems*, pp. 1380-1383, Singapore, June 1991.
- J.-C. Lee, B. J. Sheu, and **J. C.-F. Chang**, and R. Chellappa, "Multiprocessor-based video motion detection using adaptive neural systems," *IEEE Proc. of Int. Symp. on VLSI Technology, Systems, and Applications*, pp. 74-78, Taiwan, May 1991.
- B. J. Sheu, B. W. Lee, and **J. C.-F. Chang**, "Hardware annealing for fast-retrieval of optimal solutions in Hopfield neural networks," *IEEE Proc. of Int. Joint Conf. on Neural Networks*, vol. II, pp. 327-332, Seattle, WA, July 1991.
- **J. C.-F. Chang** and B. J. Sheu, "Design of a digital VLSI neuroprocessor for signal and image processing," *Proc. of IEEE-SP Workshop on Neural Networks for Signal Processing*, pp. 606-615, Princeton, Oct. 1991.

- **J. C.-F. Chang**, B. J. Sheu, and H. Okada, "Design of a multiprocessor DSP chip for flexible information processing," *IEEE Proc. of Int. Conf. on Acous. Speech and Signal Processing*, vol. V, pp. 637-640, San Francisco, CA, Mar. 1992.
- B. J. Sheu, J. Choi, and **J. C.-F. Chang**, "An analog neural network processor for self-organizing mapping," *IEEE Int. Solid State Circuit Conf. Tech. Digest*, pp. 136-137, 266, San Francisco, CA, Feb. 1992.
- **J. C.-F. Chang** and B. J. Sheu, "Digital VLSI multiprocessor design for neurocomputers," *IEEE Proc. of Int. Joint Conf. on Neural Networks*, vol. II, pp. 1-6, Baltimore, MA, Jun. 1992.
- H. Okada, B. J. Sheu, and **J. C.-F. Chang**, "An analog VLSI edge detection chip and digital multiprocessor chip for neural-based vision processing," *IEEE Proc. of Int. Conf. on Systems Engineering*, pp. 361-364, Kobe, Japan, Sep. 1992.
- **J. C.-F. Chang**, B. J. Sheu, and J. Thomas "Multi-layer back-propagation neural networks for finance analysis," *Proc. of World Congress on Neural Networks*, vol. I, pp. 445-450, Portland, OR, Jul. 1993.
- **J. C.-F. Chang** and B. J. Sheu, "A multi-chip module design for portable video compression systems," *IEEE Proc. of Multi-Chip Module Conf.*, pp. 39-44, Santa Cruz, CA, Mar. 1993.
- S. H. Bang, B. J. Sheu, and **J. C.-F. Chang**, "Search of optimal solutions in multi-level neural networks," accepted by IEEE ISCAS94.
- J. Choi, B. J. Sheu, and **J. C.-F. Chang**, "A Gaussian synapse circuit for analog VLSI neural networks," accepted by IEEE ISCAS94.

**Book Chapter**

- **J. C.-F. Chang** and B. J. Sheu, *MOS Storage Circuits*, Section 12.5.4 in The Circuits and Filters Handbook, Wai-Kai Chen, editor-in-chief, CRC Press, Fall 1994.

**Technical Reports**

- B. J. Sheu and **J. C.-F. Chang**, Editors, *VLSI Neurocomputers*, USC-SIPI Report #180, May 1991.
- **J. C.-F. Chang** and B. J. Sheu, "Digital programmable signal processor for image processing," *Proc. of SIPI Annual Review, Signal and Image Institute*, University of Southern California, Apr. 1992.
- **J. C.-F. Chang** and B. J. Sheu, "Artificial neural networks and VLSI implementations", *Proc. of SIPI Annual Review, Signal and Image Institute*, University of Southern California, Feb. 1993.