USC-SIPI Report No. 183

Digital Motion Estimation and Image Restoration on VLSI

by

Ji-Chien Lee

SIGNAL and IMAGE PROCESSING INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
University Park/MC-2564
Los Angeles, CA 90089

This report is partially supported by the National Science Foundation under Grants MIP-8710825 and MIP-8904172, by TRW Inc. and by AT&T Bell Labs.

DIGITAL MOTION ESTIMATION AND IMAGE RESTORATION ON VLSI

by

Ji-Chien Lee

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillm

Requirements for

DOCTOR OF PHI

(Electrical Engi

DON'T COPY THIS PAGE

Linda

August 1991

Acknowledgments

I would like to express my deepest thanks to my research advisor Professor Bing Sheu for his guidance and support throughout the course of my Ph.D. research work. I wish to extend my sincere appreciation to Professor Rama Chellappa and Professor Irving Reed for serving on my dissertation committee. I would also like to thank them along with Professors B. Keith Jenkins and Jay Kuo for serving on my qualifying examination committee.

I am very grateful to Professor Hans Kuehl, Chairman of EE-Electrophysics Department; Professor Jerry Mendel, Chairman of EE-Systems Department; Professor Leonard Silverman, Dean of Engineering School; and Ms. Ramona Gordon, Senior Administrative Assistant in EE-Electrophysics Department, for providing me the great research environment for my Ph.D. study at USC. Encouragement from Dr. George Lewicki, the former Director of MOSIS Service, Mr. Cesar Pina, the present Director of MOSIS Service, and other members of the USC/ISI at Marina del Rey, CA is also highly appreciated.

Professor Mike Arbib, Director of Center for Neural Engineering (CNE) helps to promote the neural network researchs at USC. Professor Alexander Sawchuk, Director of Signal and Image Processing Institute (SIPI), helps to provide original images. Seminars and annual workshops from National Center for Integrated Photonic Technology (NCIPT) have been very helpful.

Valuable discussions with Dr. Yi-Tong Zhou on artificial neural networks, and Dr. Bang Won Lee on VLSI neural chip design were very useful. Interactions with Dr. Chung-Ping Wan, and Han Yang were also valuable. The help and friendship of my fellow graduate students in the VLSI Signal Processing Laboratory have contributed to this work. I thank Wai-Chi Fang for his contribution to the early development of the system architecture for motion estimation. I also thank Joongho Choi, and Chia-Fen Chang for helping to obtain some

measurement results. Many thanks are due to Wen-Jay Hsu and Sudhir Gowda for helping to solve some circuit simulation problems. Tzyh-Chiang Chen and Min Chen help to manage the computing facility and take some images.

I also wish to thank many people in Industrial Technology Research Institute, in Taiwan; especially Dr. Chui-Chou Lin, President, Dr. Chintay Shih, Vice President, Dr. David C. Hsing, ERSO General Director, and Dr. Steve Cheng, CCL General Director for their encouragement and fellowship support for the early few years in my graduate studies. I would like to thank Dr. Ben Valdez and other members of Hughes Aircraft Co. in Newport Beach, CA, to provide me an excellent and rewarding working experience from June to August 1989 to work on smart application-specific ICs and systems design.

Various meeting discussions with Professor/Director Chung-Yu Wu, Institute of Electronics in National Chiao Tung University, Hsinchu, Professor Hsin-Chia Fu, Chairman of Computer Science Department, National Chiao Tung University, Hsinchu, Professor Hsiao-Chun Wang, Chairman of EE Department, National Tsing Hua University, Hsinchu, and Professor/Dean Chin-Lien Yen, College of Engineering, National Taiwan University are very valuable. I would like to thank Professor Bing Sheu for sending me to attend and present papers in many IEEE conferences including International Conference on Computer Design in Cambridge, MA in 1989, 1990, International Joint Conference on Neural Networks, in San Diego, CA in 1990, Workshop on VLSI Signal Processing in San Diego, CA in 1990, International VLSI Symposium on VLSI Technologies, Systems, and Applications in Taiwan, 1991.

Finally I would like to thank my wife, Teresa, our daughter, Emily, and our parents for their understanding as well as the sacrifice put up by them through all these years. This research was partially supported by National Science foundation under Grants MIP-8710825 and MIP-8904172, by TRW Inc. and by AT&T Bell Labs.

Table of Contents

List of Tables

List of Figures

Abstract

The processing of video signals often requires a tremendous computational capability which can only be achieved by using highly parallel processing architectures. The inherent massive parallelism of artificial neural network architecture for flexible information processing provides a new paradigm of video signal processing. This dissertation describes the computational needs of supercomputing neurocomputers for flexible information processing. Two neural network architectures and the efficient VLSI implementation of video signal processors are presented. In the first VLSI architecture, the motion information from a sequence of image data can be estimated through a two-dimensional multiprocessor array in which each processing element consists of an analog neuroprocessor. Massively parallel neurocomputing is done by compact and efficient neuroprocessors. Local data transfer between the neuroprocessors are performed by using analog point-to-point interconnection scheme. Global data communication between the host computer and neuroprocessors is carried out in the digital common bus. A mixed-signal VLSI neural chip that includes multiple neuroprocessors for fast video motion estimation has been designed. Measured results of the programmable synapse, summing neuron, and associated winner-take-all circuitry are presented. Based on the measurement data, system-level analysis on a sequence of real images were conducted. The device mismatch effect of analog synapse cells has been included during system-level analysis. A $1.5 \times 2.8\text{-}cm^2$ chip in a 1.2-μm CMOS technology can accommodate 64 velocity-selective neuroprocessors and achieve 83.2 Giga connections per second. The speed-up factor over a Sun-4/75 SPARC-2 workstation is 24,242 for a system with 128 motion estimation chips. In the second architecture, an analog systolic multiprocessor for high-speed image restoration has been developed. For a two-dimensional image, parallel processing is performed in the row direction and pipelined processing is performed in the column direction. The mixed analog/digital design

approach is also used for the implementation of the neural-based image restoration system. Local data computation is executed by analog circuitry to achieve full parallelism and to conserve power dissipation. Inter-processor communication is carried out in the digital format to maintain adequate signal strength across the chips boundary and achieve direct scalability in neural network size. A compact and efficient VLSI neural chip which includes multiple neuroprocessors for real-time digital image restoration has been designed. To use output of neuron as an increment/decrement information to control the pixel register, each neuron can process multiple-bit image information. A $8.0 \times 6.0\text{-}mm^2$ chip from a 1.2-μm CMOS technology can accommodate 5 neuroprocessors and the speed-up factor over the Sun-4/75 SPARC-2 workstation is 475. This chip achieves 21.6 Giga connections per second. The future powerful and cheap supercomputing workstations will include flexible information processing capability for our daily lives and scientific applications.

Chapter 1

Introduction

With the advent of high-performance communication networking and computer techniques, we are reaching an era of universal communications, where everyone has easy and immediate access to widely distributed information sources in many media including text, image, and audio. People can communicate and share information without significant concern for time, location, or medium. An integrated information processing system which can communicate with the real world through audio and video channels will play a key role in this era. A configuration of such an integrated information processing system is shown in Fig. 1.1. Speech recognition and synthesis techniques provide the systems with audio capabilities. High-speed image processing, vision understanding, and smart graphics provide the systems with visual 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 personal lives [1-4].

To support such an advanced integrated information system, very high speed signal processing techniques are needed. For example, in modern video teleconferencing and high definition television (HDTV) applications, the required video signal processing speed has been above one billion pixel operations per second. According to the recent U.S. government report [5], intensive computation power is increasingly required for future information processing applications. In 1993,

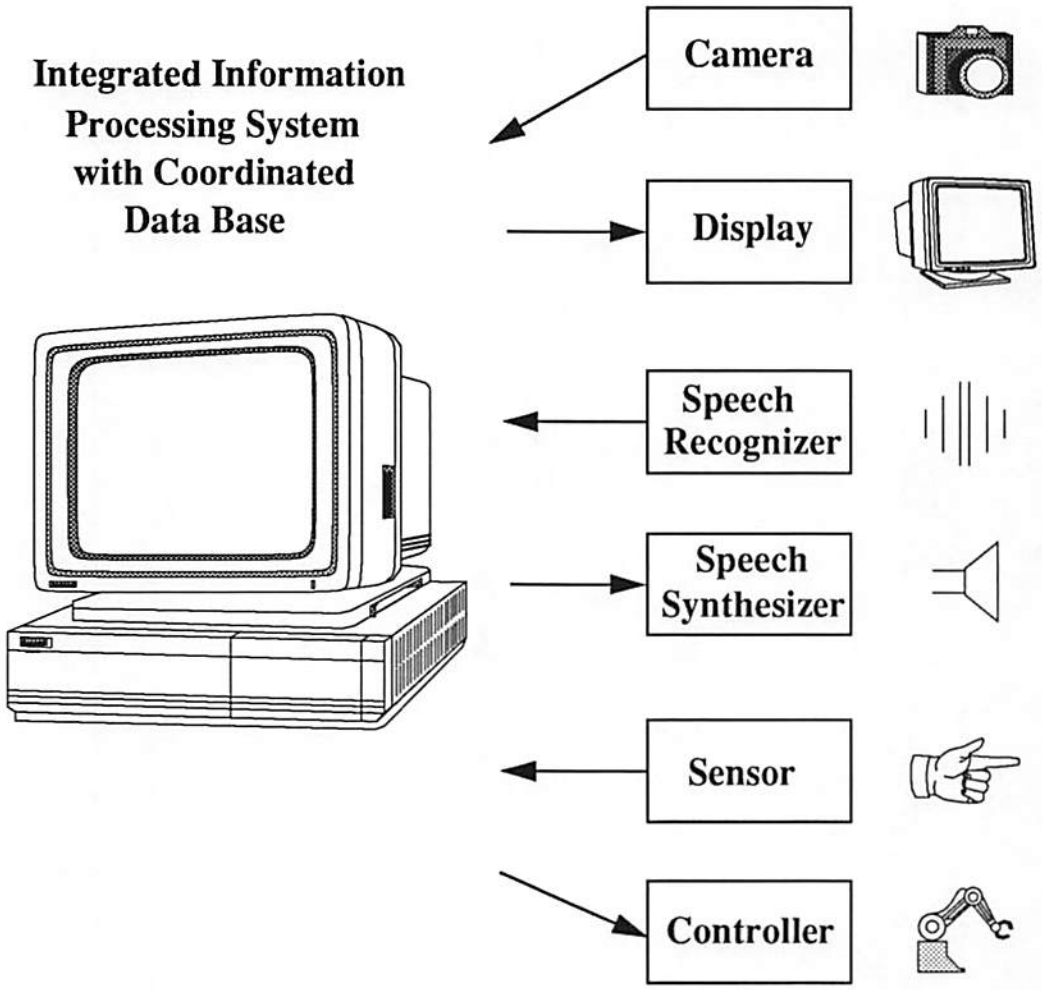


Fig. 1.1 A configuration of integrated information system.

100-giga operations-per-second systems for large-scale problems could emerge, and the deployment of tera operations-per-second systems is anticipated to be in 1996.

There are several approaches to enhance the computational capabilities in high-speed information processing. One of the approaches is to use the computational powers of computers and their development in various fields of computer science and engineering, such as signal processing, mathematical and scientific algorithms, and graph theory. The other approach is to mimic the computations performed in the human brain, i.e. the neural network approach. However, tremendous computational power is needed in both approaches. Very large scale integration (VLSI) technology can help to develop algorithm-specific multiprocessor chips to fully exploit the inherent computational powers of various information processing architectures.

Rapid advances in VLSI technologies have made possible the integration of multiple-million transistors on a single chip. The use of VLSI circuits can greatly reduce the machine size and enhance the performance and reliability of microelectronic systems. Currently, VLSI microelectronics technology has inspired many innovative designs in data processing architectures. In the microprocessor domain, continuous progress on reduced instruction set computers (RISC) enables the introduction of the Intel-i860 chip [6], and the SPARC chip from SUN Microsystems Inc. [7]. In the digital signal processing (DSP) domain, the TMS-320C40 chip from Texas Instruments Inc. [8] includes 6 communication ports to facilitate various data communication schemes. In the dedicated neural computing domain, the N64000 chip from Inova Microelectronics Inc. and

Adaptive Solutions Inc. [9] includes 64 digital processors for general-purpose neural network execution.

The artificial neural network architecture provides a new paradigm of parallel processing [10,11]. Neuroscientists have revealed that the massive parallel processing power in the human brain lies in the global and dense interconnections among a large number of identical logic elements or neurons. These neurons are connected to each other with variable strengths by a network of programmable synapses. By using analog operational amplifiers and resistors, a fully connected Hopfield neural network has been constructed for solving many engineering optimization problems [12].

From recent studies, early vision processing has been shown to be "ill-posed" problems. Examples of early vision processes are image restoration, edge detection, computation of optical flow, shape from shading, structure from motion, and etc [13]. To solve these problems, an extensive computational power is required and thus limit the performance and the speed of machine vision systems. The inherent massive parallelism of the artificial neural networks can provide an excellent means to solve these problems.

In this dissertation, two neural systems and their associated VLSI architectures and circuit-level designs for the motion estimation and the image restoration are presented. The neural system for motion estimation fully explores the local connectivity in most early vision algorithms. Since each pixel is affected by the nearest neighbors, each neuroprocessor has to communicate data to the neighboring neuroprocessors during the network operation. A new scheme to use analog point-to-point interconnection and digital common bus is the key to

implement three-dimensional interconnection in VLSI chips. In the image restoration neural system, each neuron is used to represent multiple bit information. The key is to use the neuron result as an increment/decrement information to control the pixel register. The pixel register content is combined in the synapse array with a distributed digital-to-analog conversion operation. This architecture reduces the number of neurons by a factor of 2^M in the M-bit gray-level image processing.

In Chapter 2, some related research topics are reviewed. The detailed system design for motion estimation is discussed in Chapter 3. The detailed system design for image restoration is discussed in Chapter 4. Chapter 5 concludes this dissertation and suggests the future work along this research direction.

References

[1] J. Clark, A. Yuille, *Data Fusion for Sensory Information Processing Systems*, Kluwer Academic Publishers: Boston, MA, 1990.

[2] F. Tomita, S. Tsuji, *Computer Analysis of Visual Textures*, Kluwer Academic Publishers: Boston, MA, 1990.

[3] C. Thorpe, T. Kanade, *Vision and Navigation: The Carnegie Mellon Navlab*, Kluwer Academic Publishers: Boston, MA, 1990.

[4] K. Goodman, S. Nirenburg, *The KBMT Project: A Case Study in Knowledge-Based Machine Translation*, Morgan Kaufmann Publishers: San Mateo, CA, 1991.

[5] "Grand challenges: High performance computing and communications, The FY 1992 U.S. Research and Development Program," *National Science Foundation*, Washington D.C., 1991.

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

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

[8] *Second-Generation TMS 320 User's Guide*, Texas Instruments, Inc., Dallas, TX, 1990.

[9] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, B. Riley, "An 11-million transistor neural network execution engine," *IEEE Proc. of Int. Solid-State Circuits Conf.*, pp. 180-181, San Francisco, CA, 1991.

[10] R. Lippman, "An introduction to computing with neural nets," *IEEE Acoustic, Speech, and Signal Processing Magazine*, pp. 4-22, April 1987.

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

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

[13] T. Poggio, V. Torre, C. Koch, "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314-319, Sept. 1985.

Chapter 2

High Performance Computing Requirements

To achieve required high performance computational capability for modern video signal processing applications, four related research topics are briefly reviewed and discussed in this chapter. The first is VLSI technologies, the second is multiprocessor architectures, the third is artificial neural networks, the fourth is early vision processing.

2.1 VLSI Technologies

VLSI technology has produced dramatic advances over the past 20 years, enabling the manufacture of high density integrated circuits (IC) with millions of devices on a single die. At present, the device feature size for the memory chips is around 0.5 μm, and for the processor chips is around 0.8μm. The target for the next-generation technology will be 0.25 μm [1]. In the chip complexity, a 64 megabit dynamic RAM has started to appear in the market [2]. An 11-million-transistor digital neural network execution engine has been developed [3]. In addition, speed performance of the memory chips has been pushed to lower than 10 *n*sec [4], and the processor which reaches 100 MHz has also been announced [5].

Multiple-layer interconnection techniques are crucial in realizing high-performance VLSI chips, especially for emerging neurocomputing chips in which the main computational power will come from their highly connected

architecture. Currently, VLSI technologies provide different multi-layer interconnections. One well-established process is the double-metal double-polysilicon process. Its capability of supporting various device structures such as floating-gate device and charge-couple device (CCD) makes it popular in today neural chip implementations [6,7]. The triple-metal single-polysilicon process is also available for very high speed integrated circuits (VHSIC).

The chip size of current million-transistor VLSI chips is approximately 1 to 2 cm^2. To increase the integration level, two possible methods are proposed. The first method is the wafer-scale integration [8]. The second method is the three-dimensional integration. Since the main part of neural networks shows a regular and modular architecture, they are predetermined for the wafer-scale integration technique. The inherent high fault tolerance of the neural networks is also beneficial for the wafer-scale integration. Three-dimensional integration offers interesting aspects for solving the problem of connectivity. One of the most important 3D techniques is the silicon-on-insulator technique (SOI technology). The concept of 3D integration is very fascinate for the integration of neural networks. By integrating CCD arrays with neurocomputing arrays, a compact VLSI neural system with both sensing and computing abilities can be achieved.

A summary of VLSI technologies at present and at five years later is listed in the Table 2.1.

Table 2.1 Summary of VLSI Design Technologies

	Present (1991)	Five Years Later (1996)
Memory Chip	64 Mega Bit	256 Mega Bit
Microprocessor Chip	i860	
DSP Chip	6-Processor	64-Processor
Product	Advanced TV	High Definition TV
Silicon	0.5 um	0.25 um
Compound	Optical Communication	Optical Computing

2.2 Multiprocessor Architectures for Image Processing

With recent advances in VLSI design technologies, many multiprocessor architectures have been proposed to provide massively parallel processing for computer vision applications [9,10]. The Single Instruction Multiple Data (SIMD) architecture is that in which all the parallel processors are synchronized and they all respond to a single instruction from a single controller. There is also Multiple Instruction Multiple Data (MIMD) architecture in which the number of processors is normally a few orders of magnitude less than that in SIMD machines, however, each processor is a powerful general purpose processor with its own program and data memory. Normally, MIMD machines fall into two categories: shared memory and distributed memory machines, though many architectures exhibit both paradigms. The systolic array architecture is also proposed for vision applications [11,12].

In the following discussion, several multiprocessor architectures are described. Their architecture topologies, salient features and limitations with respect to solving vision problems are discussed. The main classification of these architectures are distinguished according to the interconnection topology between processors or processor-memories.

2.2.1 Mesh Connected Multiprocessors

Mesh connected multiprocessors have been one of the first multiprocessors proposed for computer vision and image applications. For image processing applications, meshes seem to be an obvious choice because the images map quite naturally onto its structure. Figure 2.1 shows the topology of a mesh-connected

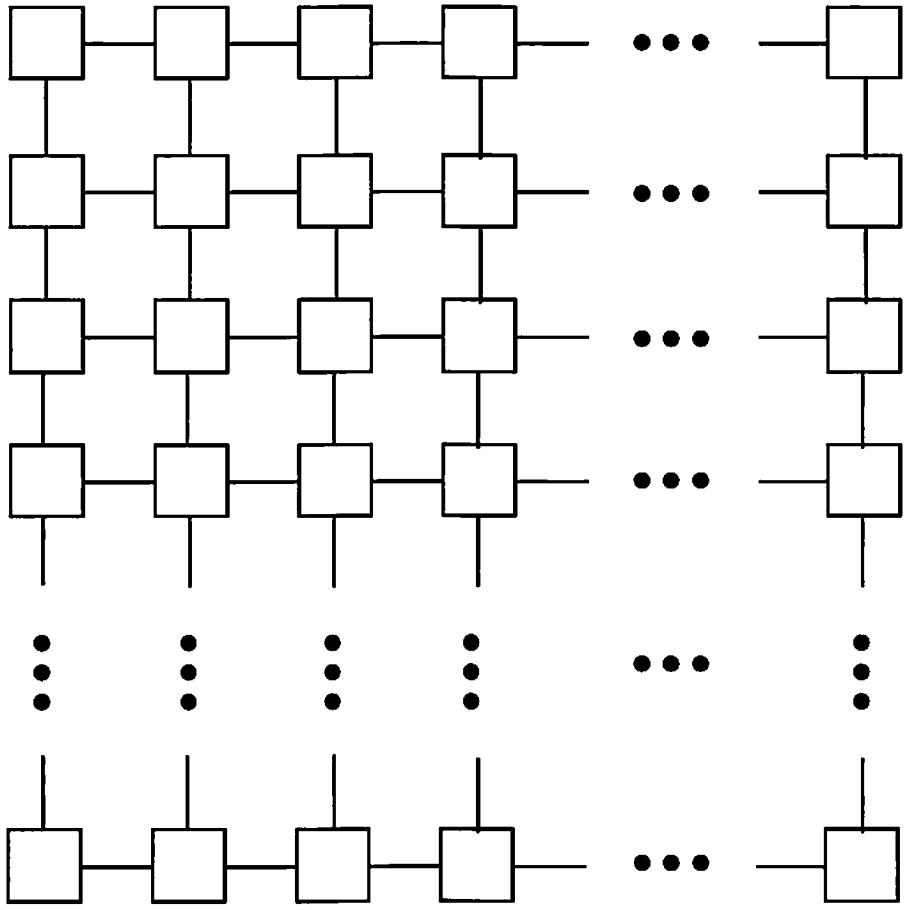


Fig. 2.1 The topology of a mesh-connected multiprocessor.

multiprocessor. In the mesh-connected configuration, the processing elements are arranged in a two-dimensional array. The widely used configuration is the 4-connected mesh in which each processor is bi-directionally connected to its four nearest neighbors. The 6- and 8- connected meshes have also been used. Most mesh-connected machines are based on the SIMD architecture. Each processing element has its own local memory and receives the instructions broadcast by an array controller.

The advantage of this architecture is that images map quite naturally onto its structure. When the image size matches the size of the multiprocessor (e.g., $N \times N$ mesh for $N \times N$ image), maximum parallelism can be obtained for those operations that require computations on individual pixels or a very small neighborhood of pixels. The major drawback of this architecture is that data communication across large distances is expensive and inefficient. Therefore, unless the computation is regular and local, meshes do not perform well. Furthermore, meshes have been proposed only as SIMD machines, and that means lack of MIMD processing capability that is necessary to support high level vision. In order to cost-effectively build a multiprocessor with thousands of processors, individual processor must be small and simple. To most efficiently use a mesh, it is required that the data size exactly match the processor size.

Several mesh-connected multiprocessors have been built. Examples of mesh-connected computers include CLIP-4 [13], GRID [14], GAPP [15], and the MPP [16]. To alleviate the global communication problems in the mesh-connected architecture, several enhancements have been proposed. Wrapped around connections of the boundary processing elements is one way in which top

row processing elements are connected to the bottom row processing elements and the first column processing elements are connected to the last column processing elements. This arrangement is called Torus. It reduces the long distance communication time. Other enhancements include connecting processing elements in rows and columns by busses to broadcast the common data.

2.2.2 Pyramid Multiprocessors

The concept of pyramid multiprocessors is essentially an extension of meshes in the third dimension. This structure has been proposed in various forms, but the main idea is that an image sized mesh-connected array is augmented by layers of successively lower resolution mesh-connected arrays as shown in Figure 2.2. Each array in a pyramid is typically one fourth as large as the array below it. Except for the bottom array, each processing element in a pyramid is connected to four processors in the level below it, in addition to the neighbors connections in the same level. All the processing elements operate in SIMD mode under the directions of a single controller. Pyramid multiprocessor architecture provides straightforward implementation of the divide-and-conquer based approach. It provides the capability for quickly changing the resolution of an image, which can significantly improve the execution speed of some low level algorithms, especially for those that depend upon communication between cells that are spatially distant in an image.

However, pyramid processors are more difficult to build than meshes because of the more complex arrangement for communication links. Hence, no pyramid multiprocessor has been built commercially.

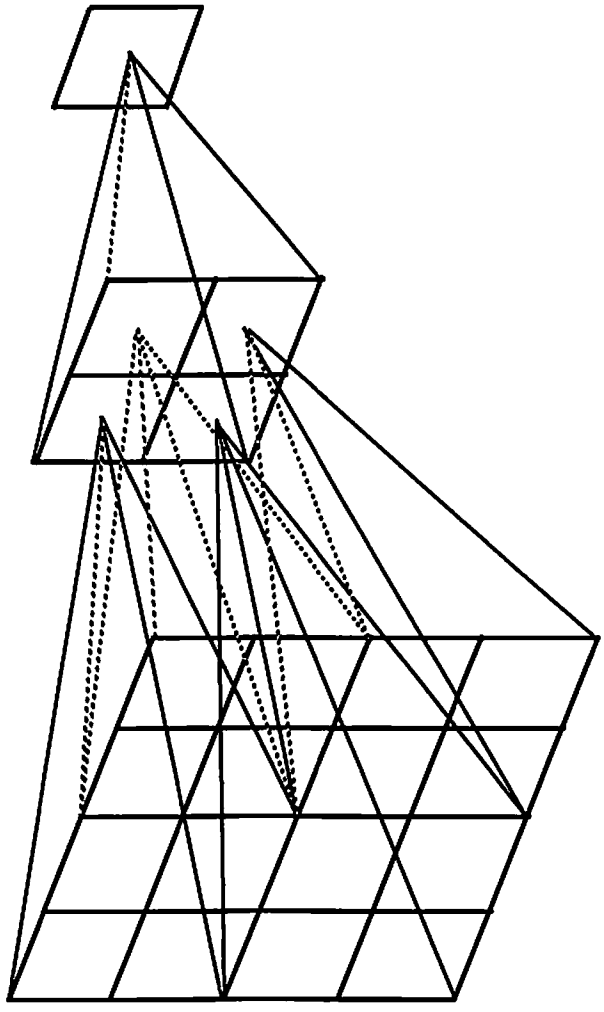


Fig. 2.2 The topology of a pyramid multiprocessor.

2.2.3 Hypercube Multiprocessors

Hypercube multiprocessors provide more efficient long distance communication that is absent in meshes or pyramids. Machines in this class consists of processors connected by communication links whose arrangement is topologically equivalent to a n-dimensional cube. A hypercube consists of $N = 2^n$ processing elements for a n dimensional cube. Each processing element is connected to n other processing elements such that their binary representations differ in exactly one bit position. Therefore, any processing element can communicate with any other processing element using at most n communication links. Figure 2.3 illustrates the organization of a hypercube multiprocessor.

Several commercially available machines have been built that use the hypercube topology. Both SIMD and MIMD types of machines have been built. The Connection machine is a SIMD hypercube multiprocessor [17]. In a connection machine, two communication networks are provided. Each processing element is connected to its four NEWS neighbors through a NEWS network, and groups of processors are connected in a hypercube fashion that provides efficient long distant communication. Such a machine can be used for most low level vision algorithms and some intermediate vision algorithms. MIMD hypercube multiprocessors are also commercially available. In fact, several companies have built MIMD hypercubes of large sizes (up to 1024 processors). Examples include Intel Hypercube [18], and Cosmic Hypercube [19].

A typical processor node in a machine consists of a general purpose microprocessor, local memory and routing hardware. Each multiprocessor is controlled by a host processor. The advantage of the hypercubes is that they provide

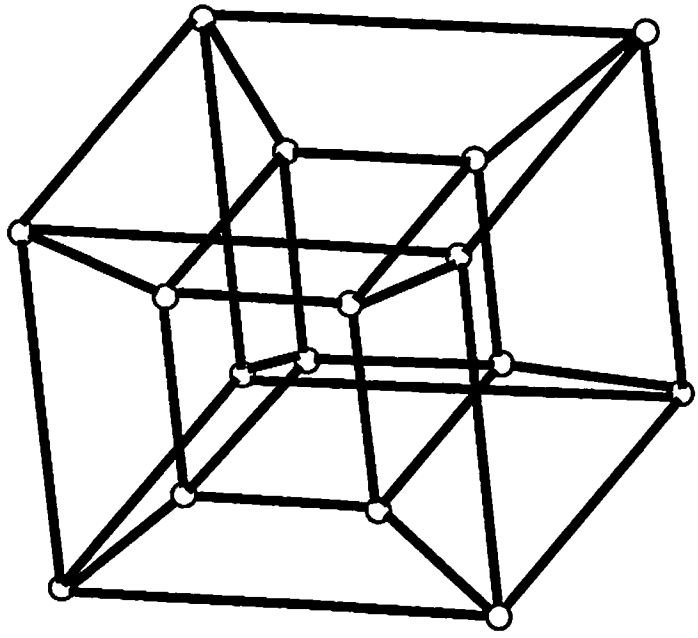


Fig. 2.3 The organization of a hypercube multiprocessor.

efficient long distance communication between processors. One drawback with hypercubes is that if a processor needs to communicate with a processor that is not one of its nearest neighbors, the data must be routed via intervening processors; this can slow down the overall processing rates if it occurs frequently.

2.2.4. Shared Memory Multiprocessors

Shared memory multiprocessors proposed and built are normally MIMD machines. Each processing element is a general purpose processor with a small local memory. Each processing element has access to a large global memory through an interconnection structure that connects the processing elements and global memory. The design of an interconnection network itself has been a huge area of research. Almost all the machines built today have variations of two common interconnection networks: bus-based and multistage interconnection networks. Bus-based system have a limitation on the number of processors, due to the bus access bottlenecks, and therefore, are not easily scalable. However, design is relatively simple and cost-effective. Another class of shared memory multiprocessors use multistage interconnection networks for processor-processor or processor-memory interconnections. Some bottlenecks of bus-based systems are alleviated in such a system; however, the interconnection networks are complex to build. Scalability in such architectures is much better than that in bus-based systems.

The main advantage of shared memory architecture is the ease of programming and uniform view of the system. In other words, control of information and synchronization is much easier compared to that in distributed memory systems. This class of machine is best suited for high level vision tasks. However,

since communication between processor and all the interactions between cooperative tasks are done through the global memory. The bottlenecks and hot-spots will occur. Furthermore, accessing global memory is at least an order of magnitude higher than accessing local memories, and therefore, communication speed is very slow compared with computation speed. Hence, such machines are efficient for only large grain parallelism tasks which have little interactions and exhibit regular memory access patterns.

2.2.5 Systolic Arrays

A systolic array multiprocessor consists of processors connected in a pipelined fashion. On one machine cycle each processor takes values from its input ports, performs the specified computation, and passes the results to its output ports. A systolic array can be perceived as a one-way pipeline of processing stations. Once the pipe is filled with data, all of the processing stations function at the same speed. Systolic array elements can be general purpose programmable function units or special purpose fixed function units. The primary advantage provided by the systolic array is high performance for relatively low cost. The main disadvantage of the systolic array is that any evaluation of processing results must wait until all the data has passed through the array.

CMU Warp systolic processor is an example of a programmable systolic array designed and built for scientific and image processing applications [20]. The Warp machine is capable of performing 10 MFLOPS. Figure 2.4 shows the organization of the Warp computer. A typical Warp array includes 10 cells. Data flow through the array on two communication channels, X and Y. The addresses for cells' local memories and control signals are generated by the

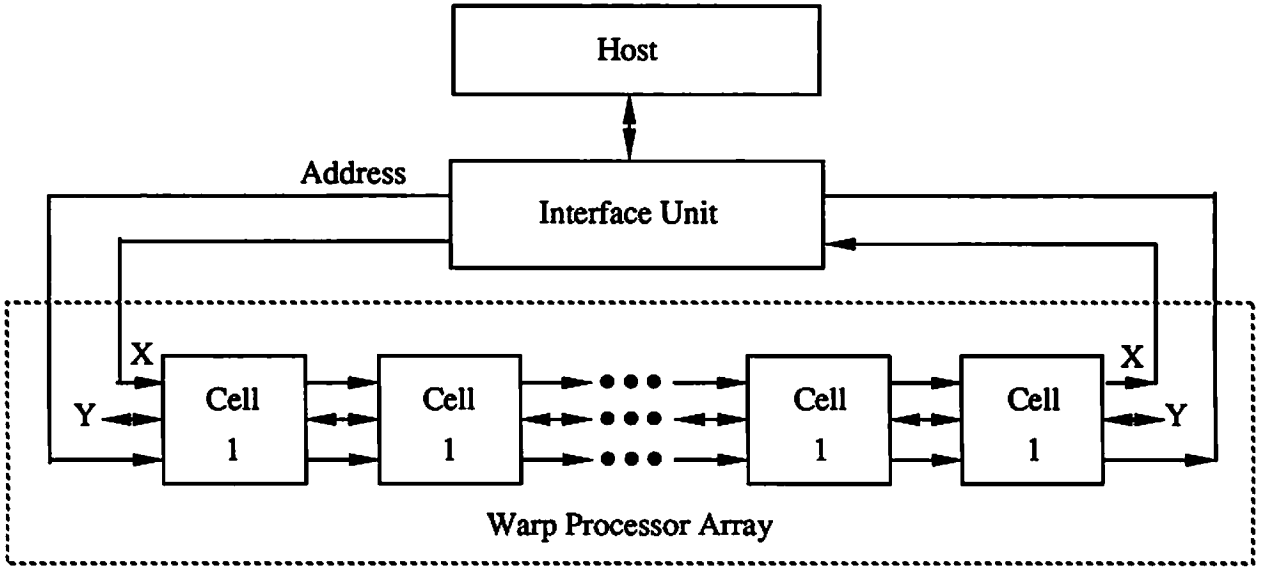


Fig. 2.4 The organization of a Warp computer.

Interface Unit and propagate down the address channel. The direction of Y is reconfigurable. The Warp machine can perform in many modes. It is mostly suitable for low and intermediate level vision processings.

2.3 Artificial Neural Network

Artificial neural networks consists of a large collection of simple processing elements which are highly interconnected. Inspired by the physiology of the human brain, these processing elements perform mathematical algorithms to carry out information processing through their state responses to stimuli. Artificial neural networks have demonstrated the ability to deliver simple, powerful solutions in areas that for many years have challenged conventional computing approaches [21]. Many different classes of artificial neural networks exist [22]. Table 2.2 presents some of the best known neural network models, together with their properties. The neural networks can be characterized by a few key properties, such as network topology, retrieving procedure, training/learning procedure, and input values.

The network topology gives the most distinguished feature. The grouped neurons which are arranged into a disjointed structure will form a layer. In the Hopfield model [23], for instance, a single layer of processing elements is used. The output from each processing element feeds back to all of its neighbors. In a Boltzmann machine [24] or Back Propagating network [25], the network consists of one or more layers between the input and output processing elements. Figure 2.5 shows several neural network topologies which include single-layer network, single-layer network with feed-back, and multiple-layer feedforward

Table 2.2 Major Neural network Models and Properties

Neural model	Primary applications	Strengths	Limitations
Perceptron	Typed-character recognition	Oldest neural network	Cannot recognize complex patterns
Hopfield	Retrieval of data/images from fragments	Large-scale integration	Does not learn, weights must be set
Multilayer Perceptron/ Delta Rule	Pattern recognition	Simple network, more general than the perceptron	cannot recognize complex patterns
Back Propagation	Wide range: speech synthesis to loan - application scoring	Most popular, work well, and is simple to learn	Supervised training with abundant examples
Boltzmann Machine	Pattern recognition for radar/sonar	Simple network that uses noise function to reach global minimum	Long training time
Self-Organizing Map	Mapping one geometrical region onto another	Better performance than many algorithmic techniques	Extensive learning

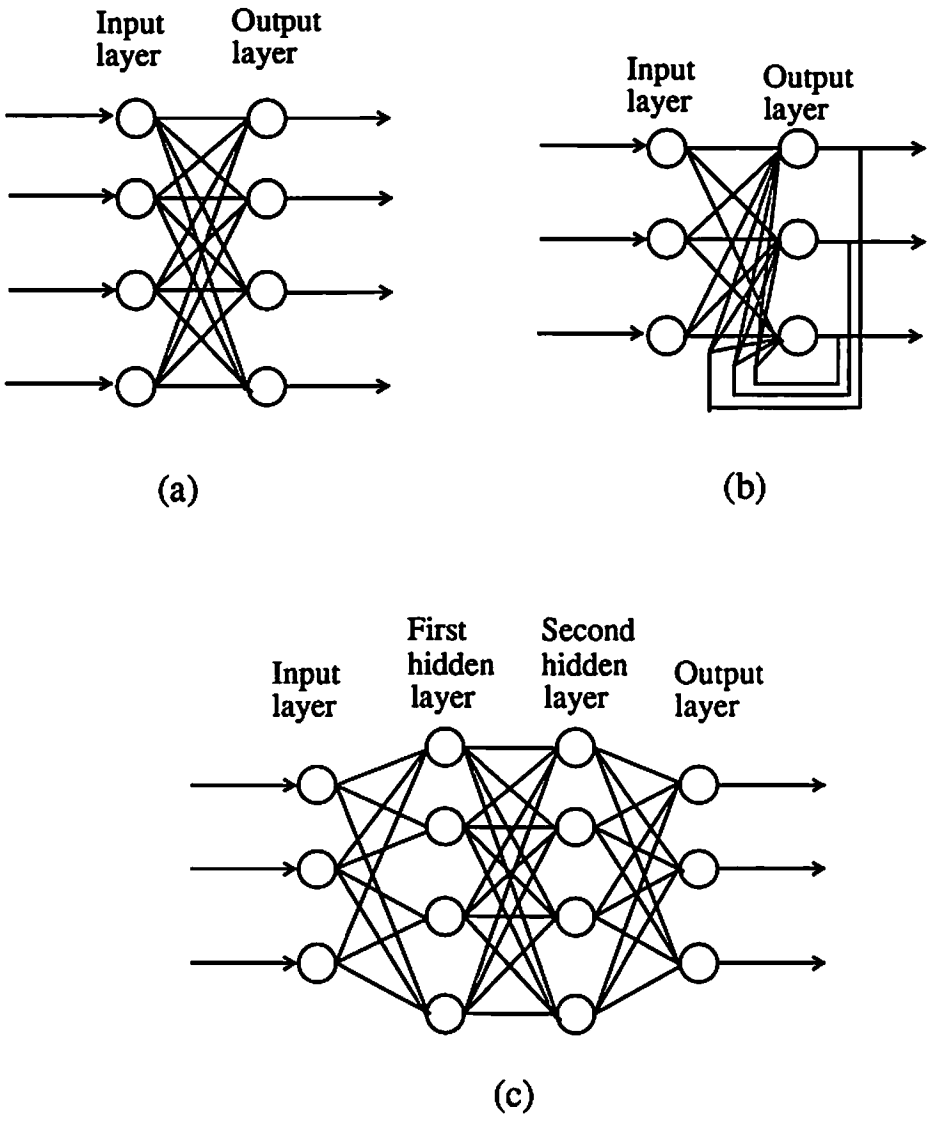


Fig. 2.5 Several neural networks.
(a) Single-layer network.
(b) Single-layer network with feedback.
(c) Three-layer network.

network. The neuron transfer function and threshold voltage characterize the retrieving process of an artificial neural network. Specific mathematical functions including sigmoid, step, Gaussian and Boltzmann functions [26] are widely used to model the neuron transfer function. The retrieving process can operate in either synchronous or asynchronous mode. In the synchronous mode, all neuron outputs are updated simultaneously. On the other hand, the neuron updating process in the asynchronous mode is random and independent of the other neurons. Most artificial neural networks in software computation operate synchronously, while the biological neural networks operate in the fully asynchronous mode. The training procedures can be divided into supervised learning and unsupervised learning [27]. In the supervised learning, synapse weightings are tuned by the difference between the retrieving patterns and expected patterns. In the unsupervised learning, the network classifies the input without references. The neural networks using unsupervised learning can detect the pattern regularities and group for each input patterns. The widely used learning rules include Hebbian rule [28], Delta rule, competitive learning rule, Boltzmann learning rule, Hopfield energy minimizing rule, and their derivatives. The input signal for an artificial neural network can be discrete or continuous values.

Neural network models reflect highly parallel, regular, and modular architectures that make them attractive for VLSI systems. The key issues for a successful integration of such systems are:

(a) The processing units (neurons) can be compactly arranged so that enough units appear on a chip to achieve the large networks needed to produce interesting features.

(b) The technology can offer solutions to the problem of complex connectivity between the processing units.

(c) The networks are capable of partitioning into smaller nets that fit on a chip.

(d) The synapses can store the multi-valued weights.

(e) The neural network model must take into account the feasibilities or restrictions of the technology.

Two kinds of hardware implementation approaches have been proposed to realize the true computing potential of massively parallel neural networks. The first one is general-purpose neural systems that consist of programmable processor arrays for emulating a range of neural network models. The second one is special-purpose neural systems that are dedicated hardware implementations of a specific neural network model.

The general-purpose neural systems can be further divided into coprocessor boards, and parallel processor arrays. Coprocessors boards usually consist of floating-point or signal-processing accelerator supplied with a large memory. These boards can be plugged into the backplane of an IBM PC or interface to a SUN workstation or a DEC VAX. Parallel processor arrays are cellular arrays [29] composed of a large number of primitive processing elements connected in a regular- and usually restricted- topology. These two categories of general-purpose systems differ basically in the number of the physical processing elements employed. Parallel processor arrays primarily attain high performance and real parallelism through an increase in the number of processing elements. On

the other hand, coprocessors attempt to achieve improved performance by strengthening the processing/storage capacities of standard microprocessors.

Hardware accelerators such as the HNC Anza [21], and the SAIC Sigma-1 [30] have allowed neural network experiments to be carried out over 100 times faster than the usual simulators of neural network models. These programmable systems can implement large networks of virtual processing elements with a limited number of hardware processors. The processing elements utilized in these systems are usually industry-standard signal-processing chips or microprocessors such as the MC68020 (and its MC68881 floating-point) interconnected through a standard parallel broadcast bus such as the VMEbus. Physical processors and interconnections are multiplexed across a large number of virtual processing elements and virtual interconnections and demand large memories represent them. Performance comparisons between these products involve capacity (the maximum size of the neural network) and speed (the time to process a neural network). Speed measurement usually takes the form of network updates per second for both the training and retrieving phases. For instance, the Anza Plus supports 1000 processing elements with 1,000 interconnections. It can perform 1,500 connection updates per second during training and 6,000 updates per second during retrieving.

General-purpose neurocomputers based on parallel processor arrays actually constitute a natural evolution from the coprocessors. Parallel processors arrays optimize neural processing by distributing the network through a large number of simpler processors. These neurocomputers usually consist of cellular arrays that are broadly similar to the Connection Machine [17]. They basically differ in

three aspects: the number of processors, the complexity of processor, and the interconnection geometry.

The Computation Network Environment (Cone) [31] developed at IBM, is an example of a system using parallel processor arrays. Cone is based on the Network Emulation Processor (NEP), a cascadable unit that acts as a coprocessor for a PC host. One can cascade NEP into as many as 256 NEPs in a unidirectional interprocessor communication network (NEPbus) to support a total of 1,000 virtual processing elements and 4,000 interconnections. NEP is based on the 5-MIPS (million instructions per second) TMS320 digital signal processor. The unit includes a 64K × 16-bit static RAM data memory, a 4K × 16-bit SRAM program memory, and three interfaces. Each NEP can simulate about 4K of virtual processing elements and 16K of interconnections and perform 30 to 50 network updates per second.

The Neural Network Environment Transputer System (NNETS), developed at The US National Aeronautics and Space Administration's Johnson Space Center in Houston, Texas, comprises forty 32-bit, 10-MIPS transputers. Each transputer with 256 Kbytes of memory. These transputers interconnect via four 10-Mbit, full-duplex serial links.

For special-purpose neurocomputers approach, a specific neural network model is directly implement in hardware to produce a very high performance system. Most implementations are based on the Kohonen [32] and Hopfield associative memory models because of their simplicity. Implementation technologies for the special-purpose neurocomputers can be classified into two categories. The first technology is the electronic implementation using analog, digital, or

mixed-signal circuit techniques. The analog circuit approach is quite attractive in terms of hardware size, power consumption, and speed. Analog neural networks were used as sensory devices to preprocess the real world data, as reported by Mead et al [33,34] and Abidi [35]. One major limitation of a pure analog neural network is the difficulty to solve problems that require more neurons than the network's physical size. To circumvent this limitation, extra analog switching circuits are used to enhance the reconfigurability and scalability of analog neural networks [36]. In addition, many other analog VLSI neural chips have been reported [37,38]. The digital circuit approach offers greater flexibility, scalability, and accuracy than the analog circuit approach. By using logic and memory, a large problem can be partitioned and processed by the digital neural networks. The major limitations of digital neural networks are in their larger silicon area, slower speed, and expensive cost of interconnections between processing elements. Many general-purpose digital VLSI neural chips were reported [39,40].

A mixed-signal circuit approach which can preserve the advantages of both digital and analog approaches and eliminate the limitations of them is very attractive. The local numeric computation is done by compact analog neurons and synapses. The fundamental computations required in the neural network, such as summation, multiplication and the nonlinear transfer function, can be implemented in a very simple and natural way. Long-distance communication is carried out in the digital format to preserve signal strength across the chip boundary and to achieve network scalability by using an array of neural chips.

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

2.4 Early Vision Processing

Early vision consists of a set of processes that recover physical properties of the visible three-dimensional surface from two-dimensional intensity arrays. Most early vision problems are ill-posed problem. A problem is ill-posed when the solution is not unique. Several examples include: edge detection, stereomatching, the computation of the optical flow, structure from motion, shape from shading and surface reconstruction. A central and common characteristic of such vision approaches is the formation of a cost or energy function which, when minimized, provides the desired solution. Because the function to be minimized is very complex, with large dimensionality and multiple local minima, sophisticated and computationally intensive minimization technique, such as simulated annealing [41] were required. The collective computational capability of neural network provides powerful new technique for solving such complex minimization problems rapidly.

The first demonstration of this capability was by Hopfield and Tank [42] for the traveling salesman problem. They showed that the Hopfield model can be

used to achieve good solutions to this problem within a few network time constants. A number of researchers have used this optimization capability of the Hopfield model for specific early vision optimization problems. Such work include implementations of shape from shading [43], and image restoration [44], and image segmentation [45].

On the other hand, at the early vision processing, the image is treated essentially as a set of samples or picture elements (pixels), without reference to the structures or objects contained within the image. Thus the operations act on the image on a pixel-by-pixel basis, as though it is a sampled waveform, with the sample organized in a two-dimensional grid. The image is treated as raw data rather than as a representation of some scene containing objects.

Algorithms that operate at this stage, in general, work on a large data set (say a 256×256 pixel array) but with a high degree of parallelism. Since, except possible near the image boundaries, the action of the algorithms should be independent of pixel position. In addition, for many applications, speed is of critical importance. For these types of algorithm conventional Von Neumann architectures, which execute a series of instructions sequentially on the data, are not well matched. Rather, one requires architectures with a high degree of parallelism but with relatively simple individual processing elements. Ideally this parallelism should be two-dimensional, so as to map onto the image data itself.

Many algorithms based on the neural network models have been developed for solving early vision problems. In this dissertation, the image restoration and computation of optical flow have been selected as two examples to demonstrate the feasibility of this approach. The system architecture and circuit level design

technique for these two examples will be described, following a brief introduction of the adopted algorithm, respectively.

References

[1] R. Cavin III, L. Sumney, R. Burger, " The semiconductor research corporation: Cooperative research," *Proc. of the IEEE*, vol. 77, no. 9, pp. 1327-1337, Sept. 1989.

[2] "Session 6: High-density DRAM," *Proc. of Intl. Solid-State Circuits Conf.* pp. 104-114, San Francisco, 1991.

[3] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, R. Riley, D. Hammerstrom, E. Means, "An 11-million-transistor digital neural network execution engine," *Proc. of Intl. Solid-State Circuits Conf.* pp. 180-181, San Francisco, 1991.

[4] "Session 3: High-speed RAM," *Proc. of Intl. Solid-State Circuits Conf.* pp. 46-54, San Francisco, 1991.

[5] J. Schutz, "A 100 MHZ CMOS microprocessor," *Proc. of Intl. Solid-State Circuits Conf.* pp. 90-91, San Francisco, 1991.

[6] T. Blyth, S. Khan, R. Simko, "A non-volatile analog storage device using EEPROM technology," *Proc. of Intl. Solid-State Circuits Conf.* pp. 192-193, San Francisco, 1991.

[7] A. Chiang, R. Mountain, J. Reinold, J. LaFranchise, J. Gregory, G. Lincoln, "A programmable CCD signal processor," *Proc. of Intl. Solid-State Circuits Conf.* pp. 110-111, San Francisco, 1990.

[8] Earl E. Swartzlander, *Wafer Scale Integration*, Kluwer Academic Publishers: Boston, MA, 1989.

[9] K. Hwang, F. Briggs, *Computer Architecture and Parallel Processing*, McGraw Hill: New York, NY, 1984.

[10] K. S. Fu, T. Ichikawa, *Special Computer Architectures for Pattern Recognition*, CRC Press: Boca Raton, Florida, 1982.

[11] H. T. Kung, "Why systolic architectures?" *Computer Magazine*, pp. 37-46, Jan., 1982.

[12] S. Y. Kung, *VLSI Array Processors*, Prentice Hall: Englewood Cliffs, NJ, 1988.

[13] L. Cordella, M. Duff, S. Levialdi, "An analysis of computational cost in image processing: a case study," *IEEE Trans. on Computer*, vol. 27, no. 10, pp. 904-910, 1978.

[14] D. Robinson, I. Parker, "A VLSI chip for real-time image processing," *IEEE Intl. Symp. on Circuits and Systems*, pp. 405-408, 1983.

[15] R. Davis, D.Thomas, "Geometric arithmetic parallel processor-systolic array chip meets the demands of heavy duty processing," *Electronic Design*, pp. 207-218, Oct. 1984.

[16] J. Potter, "Image processing on the massively parallel processor," IEEE, Computer Magazine, pp. 62-67, Jan. 1983.

[17] D. Hillis, *The Connection Machine*, MIT Press: Cambridge, MA, 1985.

[18] J. Rattner, "Concurrent processing: a new direction in scientific computing," *National Computer Conf.* 1985.

[19] C. Seitz, "The cosmic cube," *Communication of the ACM*, vol. 28, no. 1, pp. 22-33, 1985.

[20] M. Annaratone, "The Warp computer: architecture, implementation, and performance," *IEEE Trans. on Computers*, Dec. 1987.

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

[22] R. Lippman, "An introduction to computing with neural nets," *IEEE Acoustic, Speech, and Signal Processing Magazine*, pp. 4-22, April 1987.

[23] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat'l Academic Science*, vol. 79, pp. 2554-2558, 1982.

[24] D. Ackley, G. Hinton, T. Sejnowski, "A learning algorithm for Boltzmann Machines," *Cognitive Science*, No. 9, pp. 147-169, 1985.

[25] D. Rumelhart, J. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press: Cambridge, MA, 1986.

[26] B. Soucek, M. Soucek, *Neural and Massively Parallel Computers*, John Wiley & Sons: New York, NY, 1988.

[27] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Inc.: Berlin, 1984.

[28] D. Hebb, *Organization of Behavior*, Science Editions: New York, NY, 1961.

[29] C. Seitz, "Concurrent VLSI Architectures," *IEEE Trans. on Computers*, vol. C-33, No. 12, pp. 1247-1264, Dec. 1984.

[30] "DELTA/SIGMA/ANSim," *Neurocomputers*, vol. 2, 1988.

[31] C. Cruz, W. Hanson, J. Tam, "Neural network emulation hardware design considerations," *IEEE Proc. Int'l Conf. on Neural Networks*, pp. III 427-434, IEEE Press, June 1987.

[32] T. Kohonen, " New analog associative memories," *Int'l Joint Conf. on Artificial Intelligence*, pp. 1-8, Aug. 1973.

[33] M. Mahowald, C. Mead, "Silicon retina," in *Analog VLSI and neural systems*, editor, C. Mead, Addition-Wesley Publishing Company: Reading, MA, 1989.

[34] R. Lyon, C. Mead, "An analog electronic cochlea," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 36, pp. 1119-1134, July 1988.

[35] H. Kobayashi, J. White, A. Abidi, "An analog CMOS network for Gaussian convolution with embedded image sensing," *IEEE Proc. of Intl. Solid-State Circuits Conf.*, pp. 216-217, San Francisco, CA, 1990.

[36] S. Satyanarayana, Y. Tsividis, H. Graf, "A reconfigurable analog VLSI neural network chip," in *Advances in Neural Information Processing Systems 2*, editor, D. Tourelzky, Morgan Kaufmann Publishers Inc.: San Mateo, CA., 1990.

[37] B. Boser, E. Sackinger, "An analog neural network processor with programmable network topology," *IEEE Proc. of Intl. Solid-State Circuits Conf.*, pp. 184-185, San Francisco, CA, 1991.

[38] T. Morishita, Y. Tamura, T. Otsuki, "A BiCMOS analog neural network with dynamically updated weights," *IEEE Proc. of Intl. Solid-State Circuits Conf.*, pp. 142-143, San Francisco, CA, 1990.

[39] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, B. Riley, "An 11-million transistor neural network execution engine," *IEEE Proc. of Intl. Solid-State Circuits Conf.*, pp. 180-181, San Francisco, CA, 1991.

[40] Y. Arima, K. Mashiko, K. Okada, T. Yamada, A. Maeda, H. Notani, H. Kondoh, S. Kayano, "A 336-neuron 28-K synapse self-learning neural network chip with branch-neuron-unit architecture," *IEEE Proc. of Intl. Solid-State Circuits Conf.*, pp. 182-183, San Francisco, CA, 1991.

[41] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.

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

[43] C. Koch, "Analog neural networks in early vision," *Proc. Nat'l. Acad. Sci.*, vol. 83, pp. 4263-4267, June 1986.

[44] Y. Zhou, R. Chellappa, A. Vaid and B. Jenkins, "Image restoration using a neural network", *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. 36, pp. 1141-1151, July 1988.

[45] C. Cortes and J. Hertz, "A network system for image segmentation," *IEEE Proc. of Int. Joint Conf. on Neural Networks*, vol. I, pp. 121-125, Piscataway, NJ. 1989.

Chapter 3

Video Motion Estimation on Neural-Based VLSI Multiprocessors

Motion understanding is one of the most important visual functions, and has numerous applications in robotics and industrial automation. The information extracted from motion understanding process includes: segmentation of objects due to their motion, depth from motion, and motion estimation. A very broad set of applications are motivating a strong interest in sensing, interpretation, and description of motion via a sequence of images. The applications areas include [1-3]:

1. Bandwidth Compression

   To transmit sequences of images on channels with limited bandwidth is made possible if motion is extracted and coded. It can be applied to video conferencing, TV signal transmission, and so on.

2. Medical Applications

   Information about dynamic changes in size, shape, and position of working organs is of great interest, not only for detection of abnormalities, but also for detailed understanding of their normal functions. A sequence of X-ray or computer aided tomography images may be used.

3. Autonomous Navigation

   An autonomous land vehicle has important applications to military, space exploration, and metropolitan transportation. Even with sophisticated inertial navigation systems, the accumulation of position errors requires periodic

corrections. Visual information from the environment becomes an indispensable clue.

4. Industrial Automation and Robotics

   Motion analysis is necessary for hand-eye coordination of the robots working on the conveyer belts of assembly line. Moving robots depend on motion analysis to determine their trajectory, construct a model of their environment, and avoid obstacles.

5. Target Tracking

   It is of immense interest to the department of defense of every country. This technique can be also applied to traffic monitoring, movie coloring, and so on.

6. Meteorology

   Satellite images provide the opportunity for interpretation and prediction of atmospheric process through estimation of shape and motion parameters of atmospheric disturbances.

7. Human Movements Understanding

   The computation, characterization and understanding of human motion in the context of dancing and athletics is another field of endeavor.

The key to understand multiple images or image sequences lies in the analysis of differences and similarities between consecutive time frames. Many features from the images such as points, lines, curves, planar or curved surfaces, and optical flow, can be used to extract motion parameters. Optical flow is the apparent motion of the brightness patterns in an image. Generally, the optical

flow corresponds to the motion field [4], and provides important information about the spatial arrangement of the objects, the rate of change of this arrangement in a given scene and also the perceiver's own movements. Optical flow can thus be used for deriving relative depth of points [5,6], segmenting images into regions [7], and estimating the object motion in the scene [8].

According to the nature of the measured primitives, existing approaches to optical flow computing can be divided into two types: the image intensity based approach and the token based approach. The intensity based approach relies on the assumption that changes in intensity are strictly due to the motion of the object and uses the image intensity values and their spatial and temporal derivatives to compute the optical flow. By expanding the intensity function into a first-order Taylor series, Horn and Schunck [9] derived an optical flow equation using the brightness constancy assumption and spatial smoothness constraints. An iterative method for solving the resulting equation was also developed. The token based approach is to consider the motion of tokens such as edges, corners and linear features in an image. The key advantage of the token based approach is that tokens are less sensitive to variations of the image intensity. The token based approach provides the information of the object motion and shape at edges, corners, and linear features. An interpolation procedure has to be included when dense data are required.

Recently, several researchers used neural networks to conduct optical flow computing [10,11]. To prevent the smoothness constraint from taking effect across strong velocity gradients, a line process has been incorporated into the optical flow equation [11]. The resulting equation is nonconvex and includes the

cubic and some higher terms. Instead of using an annealing algorithm which is very time consuming, a deterministic algorithm was used to obtain a near-optimal solution. Convergence of such a network was obtained within a few iteration cycles. Basically, the mixed analog/digital neural network approach is to first use Horn's optical flow equation to find a smoothest solution and then to update the line process by lowering the energy function of the network repeatedly.

In order to obtain a dense flow field, the intensity based approach is preferable. However, the intensity value may be corrupted by noise appeared in natural images and partial deviatives of the intensity value are sensitive to rotation. It is difficult to detect the rotational objects in natural images based on such measurement primitives. Under the assumption that changes in intensity are strictly due to the motion of the object, Zhou et al [12] use the principal curvatures of the intensity function to compute the optical flow because they are rotation-invariant. The intensity values and their principal curvatures are estimated by using a polynomial fitting technique. Under the assumption of local rigid motion and the smoothness constraint, a neural network with maximum evolution function was developed to compute the optical flow. A deterministic decision rule was used for the updating of neurons states.

3.1 A Neural Network for Optical Flow Computation

3.1.1 The Optical Flow Algorithm

Let the velocity field consist of two components k and l. A set of $(2D_k + 1)(2D_l + 1)$ modules of neurons are used to represent the optical flow field, where D_k and D_l are the maximum values of velocity components in k and l directions, respectively. For the implementation purposes, the velocity component range is sampled using bins of size Q. As shown in Fig. 3.1, each module corresponds to a velocity value and contains $N_r \times N_c$ neurons if the images are of size $N_r \times N_c$. All neurons in the same module are self-connected and locally interconnected with other neurons in a neighborhood of size $\Gamma \times \Gamma$. Every pixel is represented by $(2D_k + 1)(2D_l + 1)$ mutually exclusive neurons which form a hypercolumn for velocity selection. When the neuron at the point (i, j) in the (k, l)-th module is 1, the actual velocities in the k and l directions at the point (i, j) are kQ and lQ, respectively.

Let $V = \{v_{i,j,k,l},\ 1 \le i \le N_r,\ 1 \le j \le N_c,\ -D_k \le k \le D_k,\ -D_l \le l \le D_l\}$ be a binary set of the neural network with $v_{i,j,k,l}$ denoting the state of the (i,j,k,l)-th neuron which is located at point (i, j) in the (k, l)-th module, $T_{i,j,k,l;m,n,k,l}$ be the synaptic interconnection strength from neuron (i,j,k,l) to neuron (m,n,k,l), and $I_{i,j,k,l}$ be the bias input.

At each step, the neuron (i,j,k,l) synchronously receives signals from itself and neighboring neurons and a bias input,

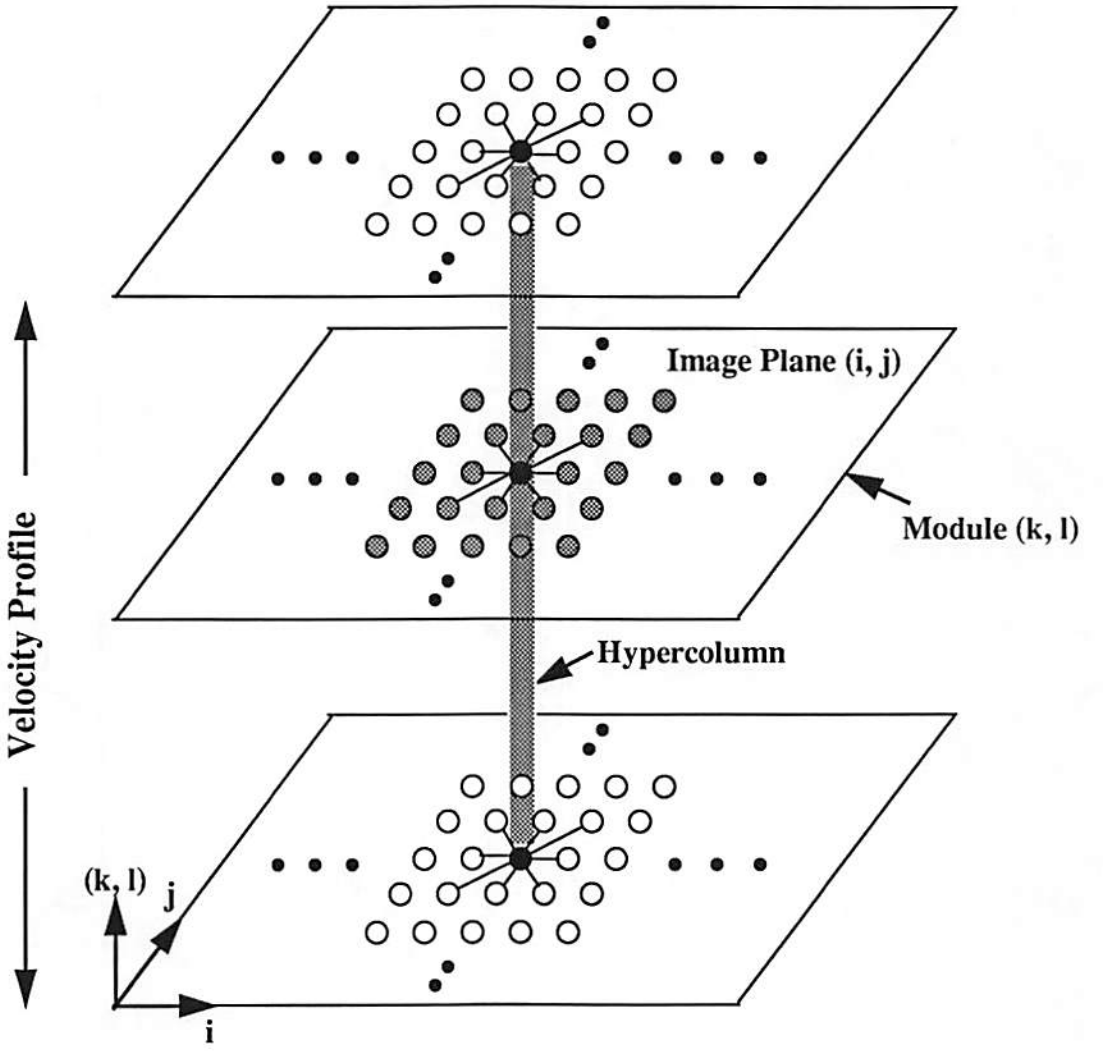


Fig. 3.1 A competitive neural network used for computation of optical flow.

$$u_{i,j,k,l} = \sum_{(m-i,n-j) \in S_0} T_{i,j,k,l;m,n,k,l}\ v_{m,n,k,l} + I_{i,j,k,l} \tag{3.1}$$

where S_0 is an index set for all neighbors in a $\Gamma \times \Gamma$ window centered at point (i, j). The $u_{i,j,k,l}$ is then processed by the maximum evolution circuitry to determine the velocity of the pixel,

$$v_{i,j,k,l} = g(u_{i,j,k,l}) \tag{3.2}$$

where $g(x_{i,j,k,l})$ is the maximum evolution function (it is also called the winner-take-all function)

$$g(x_{i,j,k,l}) = \begin{cases} 1 & \text{if } x_{i,j,k,l} = max(\, x_{i,j,p,q};\ -D_k \le p \le D_k,\ -D_l \le q \le D_l). \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

The network operation will be terminated if the network converges; i.e., the energy function of the network defined by

$$E = -\frac{1}{2} \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} \sum_{k=-D_k}^{D_k} \sum_{l=-D_l}^{D_l} \Big(\sum_{(m-i,n-j) \in S_0} T_{i,j,k,l;m,n,k,l} v_{i,j,k,l} v_{m,n,k,l}$$

$$+ I_{i,j,k,l} v_{i,j,k,l} \Big), \tag{3.4}$$

reaches a minimum.

Two important features of the network should be noted:

(i) The synaptic interconnection strength between neurons on different modules are zeros since only the neurons in the same module are connected, i.e.

$$T_{i,j,k,l;m,n,p,q} = 0, \quad \text{for } (k,l) \neq (p,q), \text{ if } (i,j) \neq (m,n). \tag{3.5}$$

(ii) A maximum evolution function is used to ensure that only one neuron which

has the maximum excitation is fired and the other $(2D_k + 1)(2D_l + 1) - 1$ neurons are turned off.

As reported in [12], a smoothness constraint is used for obtaining a smooth optical flow field and a line process is employed for detecting motion discontinuities. The line process consists of vertical and horizontal lines, L^v and L^h. Each line can be in either one of the two states: 1 for being active and 0 for being idle. The error function for computing the optical flow from a pair of image frames can be expressed as

$$E = \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} \sum_{k=-D_k}^{D_k} \sum_{l=-D_l}^{D_l} \Big[\{A[k_{11}(i,j) - k_{21}(i+k,j+l)]^2 + A[k_{12}(i,j) - k_{22}(i+k,j+l)]^2 + [g_1(i,j) - g_2(i+k,j+l)]^2\} v_{i,j,k,l} + \{\frac{B}{2} \sum_{s \in S} (v_{i,j,k,l} - v_{(i,j)+s,k,l})^2 + \frac{C}{2}[(v_{i,j,k,l} - v_{i+1,j,k,l})^2(1 - L^h_{i,j,k,l}) + (v_{i,j,k,l} - v_{i,j+1,k,l})^2(1 - L^v_{i,j,k,l})]\} \Big] \quad (3.6)$$

where $k_{11}(i,j)$ and $k_{12}(i+k,j+l)$ are the principal curvatures of the first image, $k_{21}(i,j)$ and $k_{22}(i+k,j+l)$ are the principal curvatures of the second image, $g_1(i,j)$ and $g_2(i+k,j+l)$ are the intensity values of the first and second images, respectively. Here, $S = S_0 - (0,0)$ is an index set excluding (0,0), A, B, and C are empirical constants.

The principal curvatures are defined as [13]

$$k_1(i,j) = M + (M^2 - G)^{\frac{1}{2}} \tag{3.7}$$

and

$$k_2(i,j) = M - (M^2 - G)^{\frac{1}{2}} \tag{3.8}$$

where $k_1(i,j)$ and $k_2(i,j)$ are the principal curvatures, G and M are the Gaussian and mean curvatures given by

$$G = \frac{\partial^2 g(i,j)}{\partial i^2} \cdot \frac{\partial^2 g(i,j)}{\partial j^2} - [\frac{\partial^2 g(i,j)}{\partial i \partial j}]^2 \quad , \tag{3.9}$$

and

$$M = \frac{1}{2}[\frac{\partial^2 g(i,j)}{\partial i^2} + \frac{\partial^2 g(i,j)}{\partial j^2}] \quad . \tag{3.10}$$

A polynomial fitting technique is used to estimate the derivatives.

In (3.6), the first term is to seek velocity values such that all points of two images are matched as closely as possible in a least-squares sense. The second term, which is weighted by B, is the smoothness constraint on the solution and the third term, which is weighted by C, is a line process to weaken the smoothness constraint and to detect motion discontinuities. The constant A in the first term determines the relative importance of the intensity values and their principal curvatures to achieve the best results. The line process weakens the smoothness constraints by changing the smoothing weights, resulting in space-variant smoothing weights. For example, if all lines are on, the weights will be $\frac{B}{2}$. If all lines

are off, the weights at the four nearest neighbors of the center point are increased by $\frac{C}{2}$.

By choosing the interconnection strengths and bias inputs as

$$T_{i,j,k,l;m,n,k,l} =$$

$$-[48B + C(4 - L^h_{i,j,k,l} - L^h_{i,j+(-1),k,l} - L^v_{i,j,k,l} - L^v_{i+(-1),j,k,l})]\delta_{i,m}\delta_{j,n}$$

$$+ C[(1 - L^h_{i,j,k,l})\delta_{i,m}\delta_{j+1,n} + (1 - L^h_{i,j+(-1),k,l})\delta_{i,m}\delta_{j+(-1),n}$$

$$+ (1 - L^v_{i,j,k,l})\delta_{i+1,m}\delta_{j,n} + (1 - L^v_{i+(-1),j,k,l})\delta_{i+(-1),m}\delta_{j,n}]$$

$$+ 2B \sum_{s \in S} \delta_{(i,j),(m,n)+s} \tag{3.11}$$

and

$$I_{i,j,k,l} = -A\{[k_{11}(i,j) - k_{21}(i+k,j+l)]^2 + [k_{12}(i,j) - k_{22}(i+k,j+l)]^2\}$$

$$-[g_1(i,j) - g_2(i+k,j+l)]^2 \tag{3.12}$$

where $\delta_{a,b}$ is the Dirac delta function, the error function in (3.6) is mapped into the energy function of the neural network in (3.4). Notice that the interconnection strengths consist of constants and line process only. The bias inputs contain all the information from images. When the network reaches a stable condition, the optical flow field is determined by the neuron states. The size of a typical smoothing window is 5 × 5.

3.1.2 Updating Scheme

Since the first and second terms in (3.6) do not contain the line process, the updating of the line process is prior to the updating of neuron states. Let $L^{v;new}_{i,j,k,l}$ and $L^{v;old}_{i,j,k,l}$ denote the new and old states of the vertical line $L^{v}_{i,j,k,l}$, respectively. Let $\psi_{i,j,k,l}$ be the potential of vertical line $L^{v}_{i,j,k,l}$ given by

$$\psi_{i,j,k,l} = \frac{C}{2}(v_{i,j,k,l} - v_{i+1,j,k,l})^2. \tag{3.13}$$

Then, the new state is determined by

$$L^{v;new}_{i,j,k,l} = \begin{cases} 1 & \text{if } \psi_{i,j,k,l} > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.14}$$

Whenever the states of neurons $v_{i,j,k,l}$ and $v_{i+1,j,k,l}$ are different, the vertical line $L_{i,j,k,l}$ will be active provided that the parameter C is greater than zero. If $C = 0$, then all lines are inactive, which means that no line process exists in the network operation. The choice of C is closely related to selecting the smoothness parameter B in (3.6). A similar updating scheme is also used for the horizontal lines. In the prototype neural chip design, computation for the terms which are weighted by the parameter C is not included.

The state of each neuron is synchronously evaluated and updated according to (3.1) and (3.2). The initial states of the neurons are set as

$$v_{i,j,k,l} = \begin{cases} 1 & \text{if } I_{i,j,k,l} = \max(\, I_{i,j,p,q};\ -D_k \le p \le D_k,\ -D_l \le q \le D_l) \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

where $I_{i,j,k,l}$ is the bias input. The initial conditions are completely determined by the bias inputs. If there are two maximal bias inputs at point (i, j), then

only the neuron corresponding to the smaller velocity is initially set at 1 and the other one is set at 0. This is consistent with the minimal mapping theory [14]. In the updating scheme, the minimal mapping theory is also used to handle the case of two neurons having the same largest inputs.

3.2 The Neural-Based Neuroprocessor Design

3.2.1 VLSI Architecture

To implement the electronics neural network, a VLSI architecture has been developed which maps the 3-dimensional neural network configuration onto a 2-dimensional plane. As shown in Fig. 3.2, each small frame represents one velocity-selective hypercolumn which contains $(2D_k + 1)(2D_l + 1)$ velocity-sensitive components. Each hypercolumn is locally interconnected with the $\Gamma \times \Gamma - 1$ neighboring hypercolumns. The hypercolumn is designed as a neuroprocessor within which the velocity selectivity of an image pixel can be conducted. Mixed analog-digital design technologies are utilized for the neuroprocessor design to achieve compact and programmable synapses and neurons for massively parallel neural computation [15].

To simplify the two-dimensional interconnection problem for computation of optical flow, the analog point-to-point interconnection for local communication and the digital common bus for global communication are used. Since velocity information of one pixel is affected by its neighbors, each neuroprocessor receives information from the neighboring neuroprocessors during the network operation. Data communication between these locally

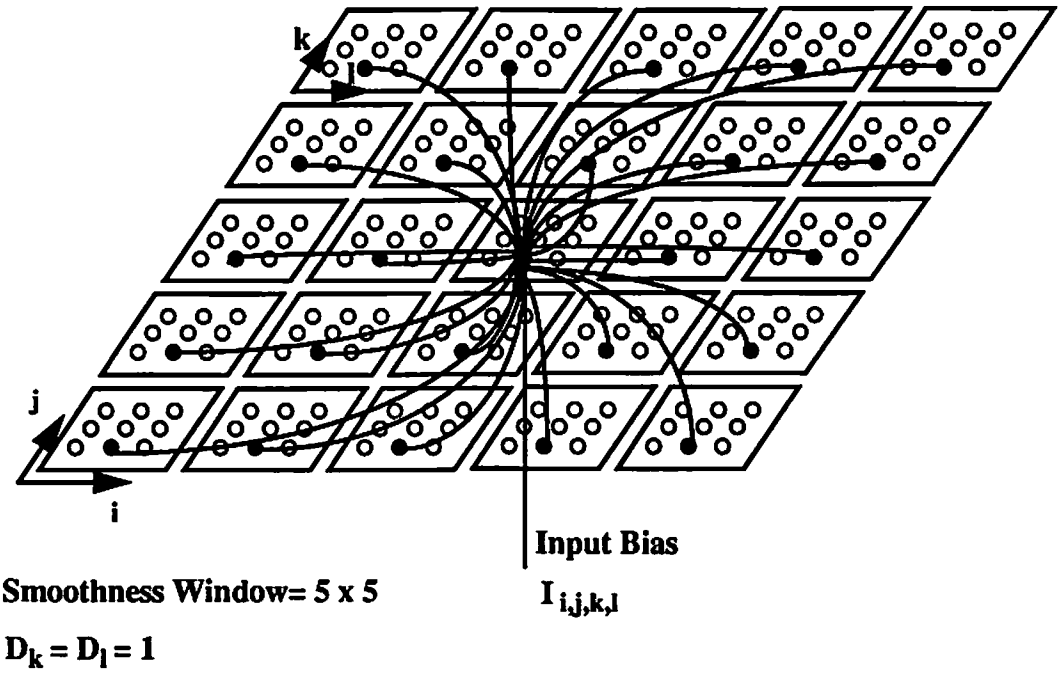


Fig. 3.2 A two-dimensional array of velocity-selective hypercolumn.

neuroprocessors is one key factor on the overall system performance. There are three different methods to accomplish the local data communication with trade-offs on the operation speed and silicon area.

The first method is to use the digital bit-parallel point-to-point interconnection. The $(2D_k + 1)(2D_l + 1)$-bit $v_{i,j,p,q}$'s, where $-D_k \leq p \leq D_k$, $-D_l \leq q \leq D_l$, are transmitted using the word-wide point-to-point interconnections. The data transfer speed is very fast. However, the total number of interconnection lines for each neuroprocessor is as large as $(2D_k + 1)(2D_l + 1)(\Gamma \times \Gamma)$. The silicon area for the interconnection routing is large. The required large pin count becomes a major constraint for hardware implementation.

The second method is to use the digital bit-serial point-to-point interconnection. The $v_{i,j,p,q}$'s are sent in a bit-serial order by using a time-multiplexing technique. The total number of interconnection lines is reduced by a factor of $(2D_k + 1)(2D_l + 1)$. However, the time required for data transfer increases with the same factor. In addition, the required hardware overhead for time-multiplexing includes a one-bit latch for each synapse cell, the multiplexing control signals, and the associated decoding circuitry.

The third method is to use the analog bit-parallel point-to-point interconnection. The $v_{i,j,p,q}$'s are converted to an analog value, and then sent to the neighboring neuroprocessors. The $v_{i,j,p,q}$'s are converted back into digital values at the receiving sites. This method is quite effective since it allows the network to operate at a very high speed and also achieve a compact layout. The required

I/O port and silicon area for interconnection routing are moderate for multi-neuroprocessor hardware implementation.

A functional diagram of the velocity-selective neuroprocessor is shown in Fig. 3.3. It includes a velocity-sensitive component array, and a data conversion block. The array has $(2D_k + 1)(2D_l + 1)$ velocity-sensitive components which are laterally connected through the winner-take-all circuit. The velocity of the neuroprocessor is determined by competition which is performed by the winner-take-all circuit. Only one velocity component which has the maximum excitation will be the winner to represent the velocity of that pixel. The data conversion block is used for analog point-to-point inter-processor interconnection.

As shown in Fig. 3.4, the velocity-sensitive component is constructed with one synapse array, one summing neuron, and one winner-take-all cell. The synapse array contains $\Gamma \times \Gamma + 1$ programmable synapses. The synapse weights $T_{i,j,k,l;m,n,k,l}$ are stored as charge packets on capacitors and must be refreshed periodically. The binary outputs $v_{m,n,p,q}$ from the neighboring neuroprocessors are routed to the corresponding mask ports of the synapse cells to conduct the network operation. A summing neuron functions as a parallel current adder. Each summing neuron with its associated programmable synapse array perform a complete inner-product computation. The winner-take-all cell contributes to a maximum evolution function on the analog outputs of summing neurons. The binary outputs of the winner-take-all circuit represent the velocity status.

The synapse weights and bias inputs are calculated by the host computer or a digital co-processor and stored in a digital static-RAM. The 8-bit D/A converter transforms the digital representation of the synapse weights into analog

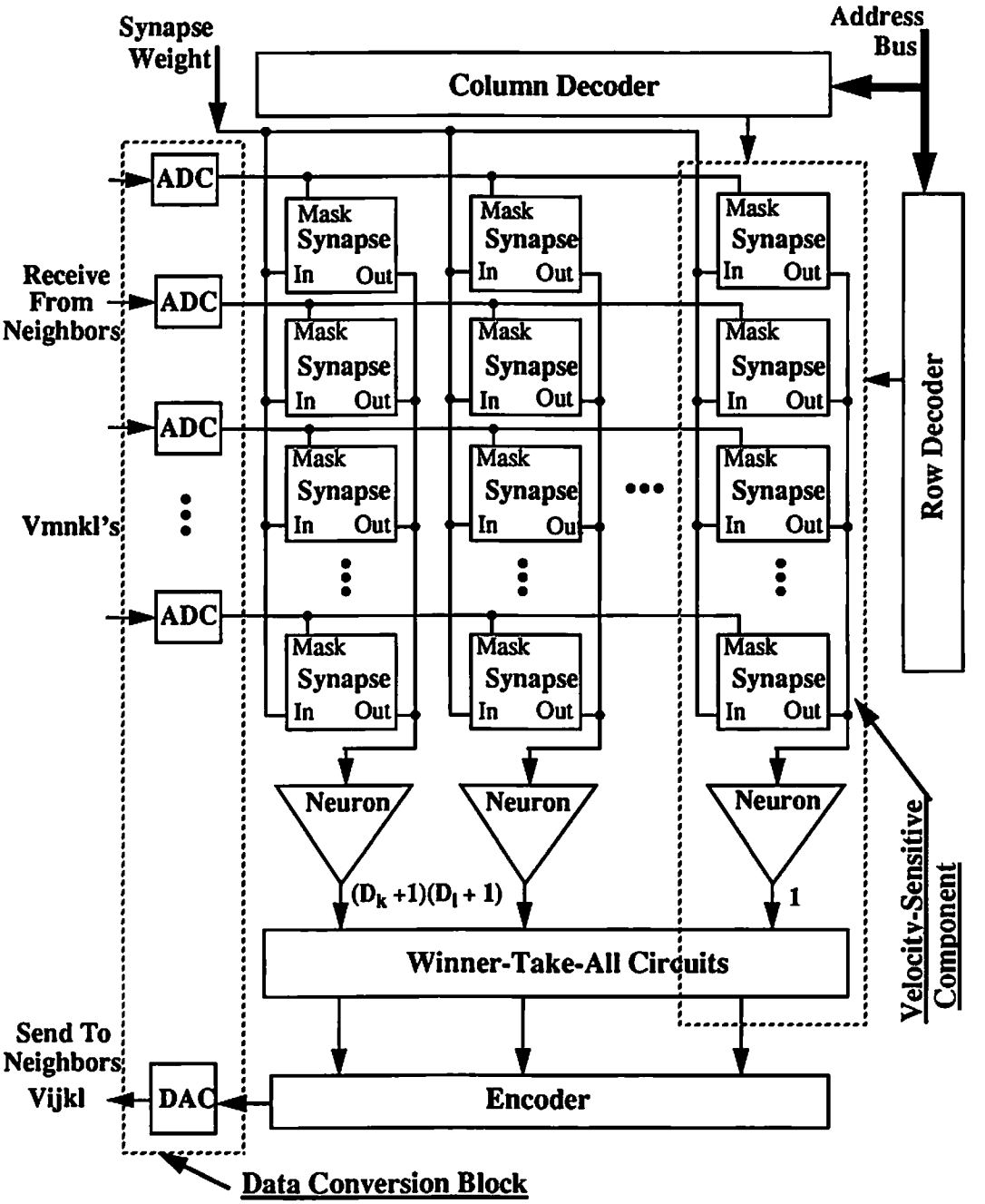
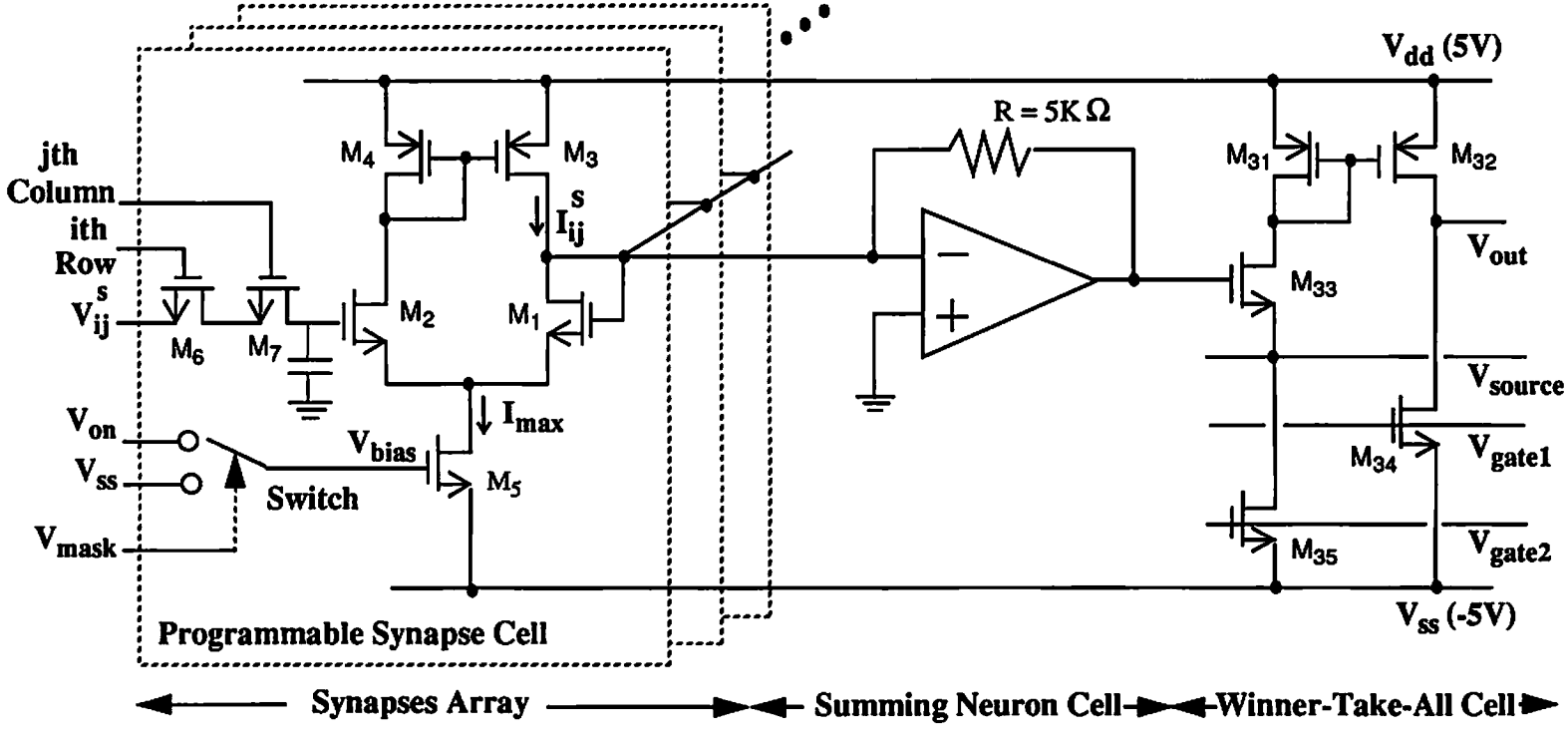


Fig. 3.3 Functional diagram of one velocity-selective neuroprocessor.

values for charging the weight-storage capacitances of the synapse matrix. A two-port static-RAM and differential amplifier-based synapse design allows network retrieving and learning processes to occur concurrently.

3.2.2 Detailed Circuit Design

In Fig. 3.4, a transconductance amplifier consisting of transistors M_1-M_5 produces synapse output current $I^s_{i,j}$ according to mask voltage V_{mask} and weight voltage $V^s_{i,j}$. The bias voltage V_{bias} controls the dynamic range of synapse cells by adjusting the bias current in the transconductance amplifiers. When the V_{mask} is at logic 1, the V_{bias} is connected to V_{on} to provide the amplifier with a specific bias current $I^{\max}$. When the V_{mask} is at logic 0, the V_{bias} is connected to the negative power supply so that no synapse output current is produced. Therefore, the V_{mask} performs a masking operation on the synapse weight voltage $V^s_{i,j}$. The mask voltage of each synapse cell is directly related to the value of the $v_{m,n,k,l}$ which represents the velocity information of the neighboring pixels. The maximum synapse conductance is decided by device sizes of the differential pair and the bias current $I^{\max}$, while the minimum synapse conductance is determined by the resolution of the weight value on the MOS capacitance. The synapse output currents are summed up and converted to the voltage format at the summing neuron. This compact synapse circuit performs a two-quadrant multiplication. The polarity of the synapse output depends on the value of weight voltage $V^s_{i,j}$. An 8-bit resolution can be supported in the DRAM-style synapse cell [16]. In the EEPROM-style synapse cell [17,18], at least a 6-bit resolution can be obtained.



Transistor	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_{31}	M_{32}	M_{33}	M_{34}	M_{35}
Size in um	3/39	3/39	4/4	4/4	5/3	4/2	4/2	8/4	8/4	8/2	8/2	20/8

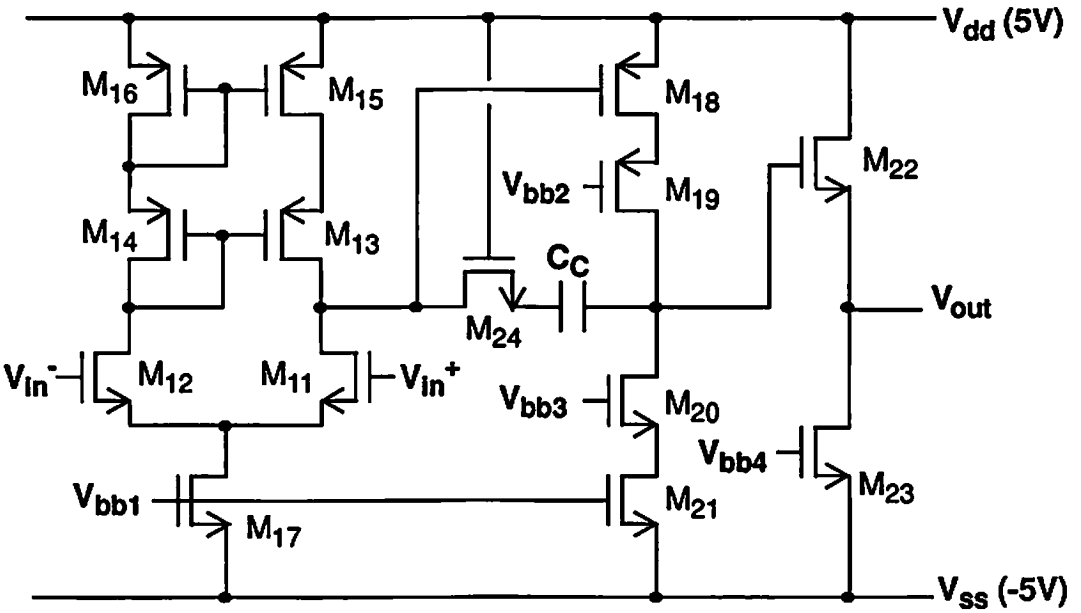
Fig. 3.4 Detailed circuit schematic of the velocity-sensitive component.

The summing neuron functions as a current-to-voltage converter and is realized by using a two-stage operational amplifier and a feedback resistor. Circuit schematic diagram of the two-stage operational amplifier is shown in Fig. 3.5. Transistors M_{13} and M_{14} form an improved cascode stage to increase the voltage gain and M_{24} operates as a resistor for proper frequency compensation. The amplifier voltage gain can be 100 dB.

The outputs of the winner-take-all circuit are binary values. Only one winner cell with the maximum input voltage will have the logic-1 output value. The other cells will have the logic-0 output value. The winner-take-all circuitry functions as a multiple-input comparator. Figure 3.6 shows the circuit schematic diagram of two winner-take-all cells. The high-resolution and expandability of this winner-take-all circuit make it suitable for many competitive learning neural networks [19-22].

After the winner-take-all circuit, the $(2D_k + 1)(2D_l + 1)$ binary outputs represent the velocity information of one image pixel. Combinational logic gates are used to encode these $(2D_k + 1)(2D_l + 1)$ binary signals and to store the result into a data latch. Figure 3.7 shows digital circuits of the data latch with the associated read/write control logic. The final velocity result is read by the host computer from the data latches through the digital common bus.

Figure 3.8 shows a voltage-scaling digital-to-analog converter which is used to convert the encoded binary code to the analog value and send it to the neighboring neuroprocessors. The voltage-scaling converter uses a series of resistors connected between V_{ref} and $-V_{ref}$ to provide intermediate voltage values. For an N-bit converter, the resistor string would have $2^N + 1$ resistor segments. In



Transistor	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
Size in um	16/2	16/2	6/2	6/2	20/4	20/4	40/4	46/4
Transistor	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	C_C	
Size in um	90/2	26/2	52/4	600/2	400/4	50/2	1.5pF	

Fig. 3.5 Circuit schematic of the operational amplifier used for summing neuron.

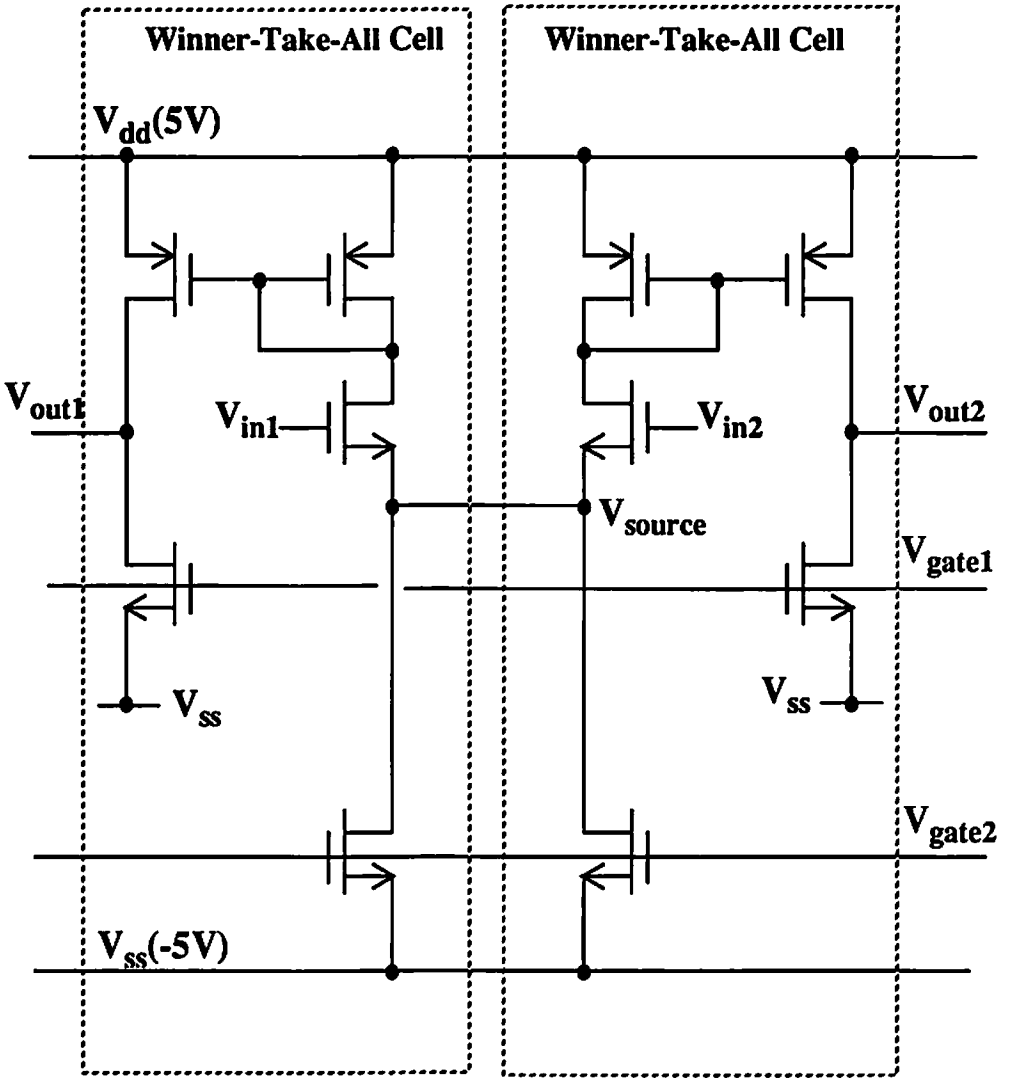


Fig. 3.6 Two winner-take-all cells are connected as a differential operational amplifier.

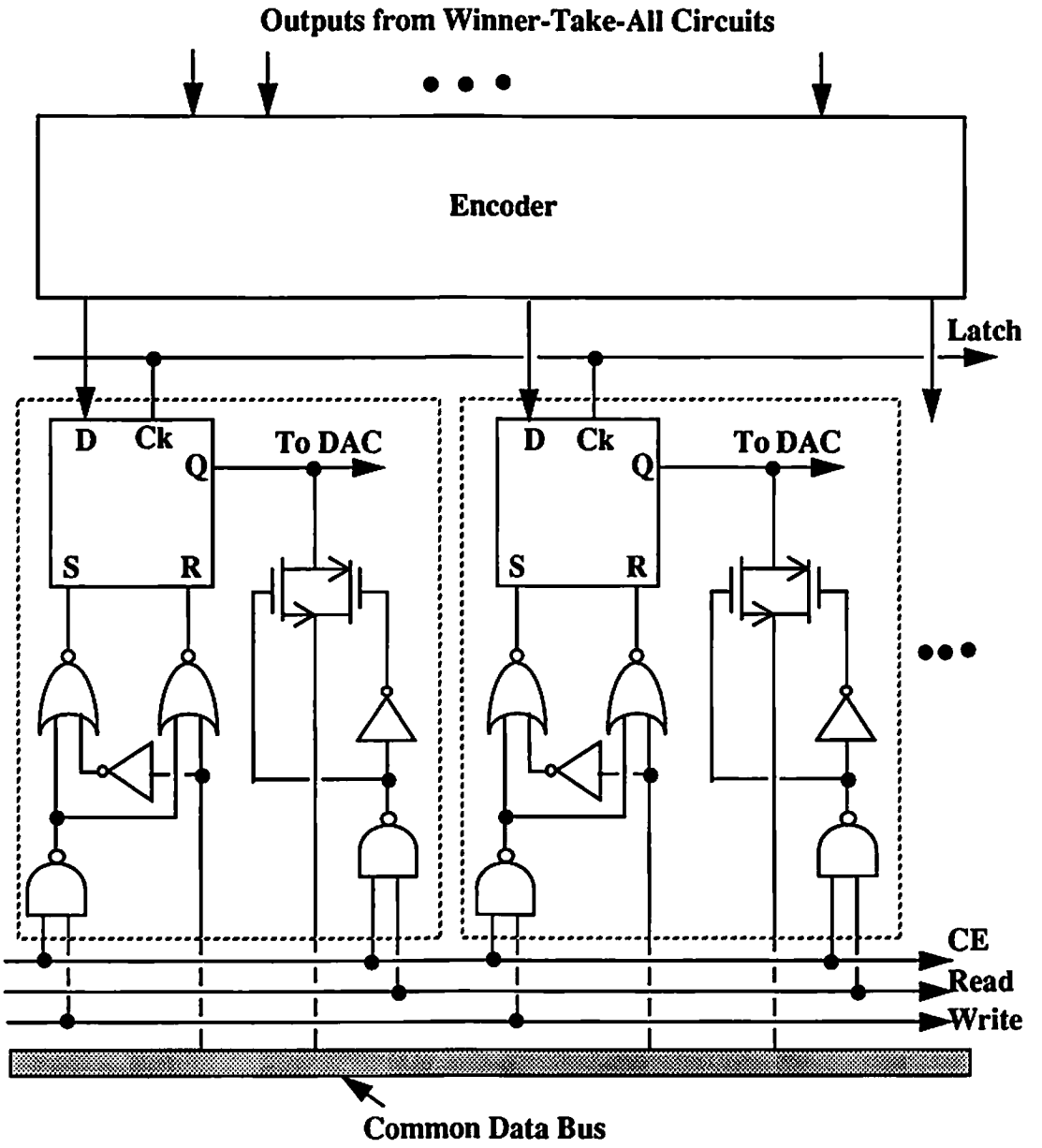


Fig. 3.7 Digital circuits of the data latch and its associated read/write control logic.

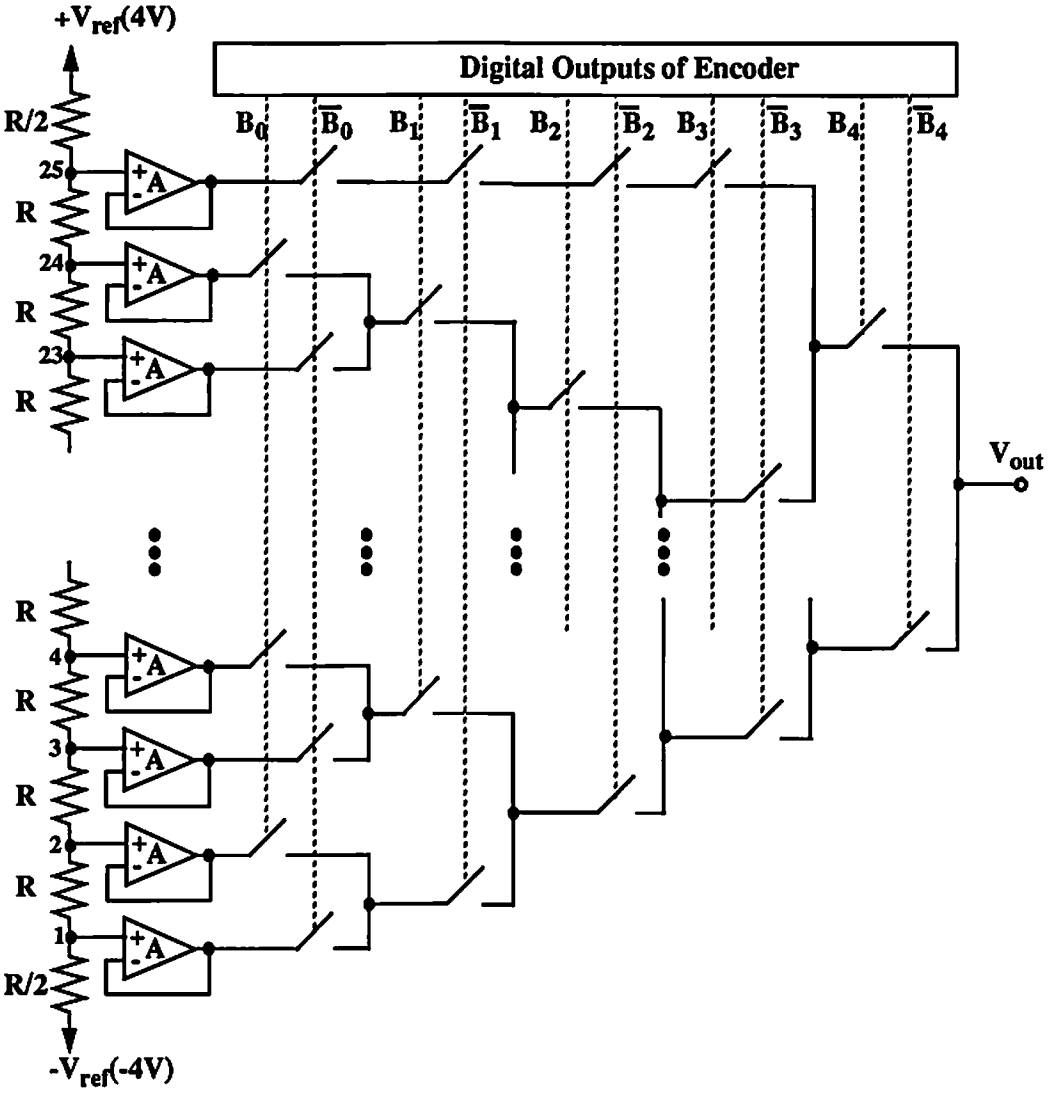


Fig. 3.8 A voltage-scaling digital-to-analog converter.

Fig. 3.8, a total of 26 resistor segments are used. The resistor is implemented in the P-well diffusion layer. The sheet resistance of the P-well layer is 2 KΩ/□ from the MOSIS 1.2-μm CMOS P-well process. Unity-gain followers are used to buffer the resistor string from conductive loading. Each tap is connected to a switching tree whose switches are controlled by the bits of the digital word. Each switch is implemented by a CMOS transmission gate.

When the analog velocity information from the neighboring neuroprocessors is received, a total of $\Gamma \times \Gamma - 1$ analog-to-digital converters are used to convert these analog values back to the binary values with $(2D_k + 1)(2D_l + 1)$ bits. Only one of these bits is logic-1 and the others are logic-0. To achieve high-speed performance and compact silicon area, a parallel and distributed analog-to-digital converter has been designed. One voltage scaling resistor-chain is used. As shown in Fig. 3.9, the comparators and the associated digital decoding circuitries are distributed into the synapse cells. The comparators included in the same velocity-sensitive component use the same reference voltage provided by the resistor-chain. The distributed decoding circuitries make sure that only one of $(2D_k + 1)(2D_l + 1)$ binary outputs is logic 1 and the others are inhibited to logic 0.

3.3 Experimental Results

3.3.1 Prototype Neural Chips

In the prototype neural chip design, $D_k = D_l = 2$ and a size of 5×5 smoothing window are used. The layout of the velocity selection neuroprocessor

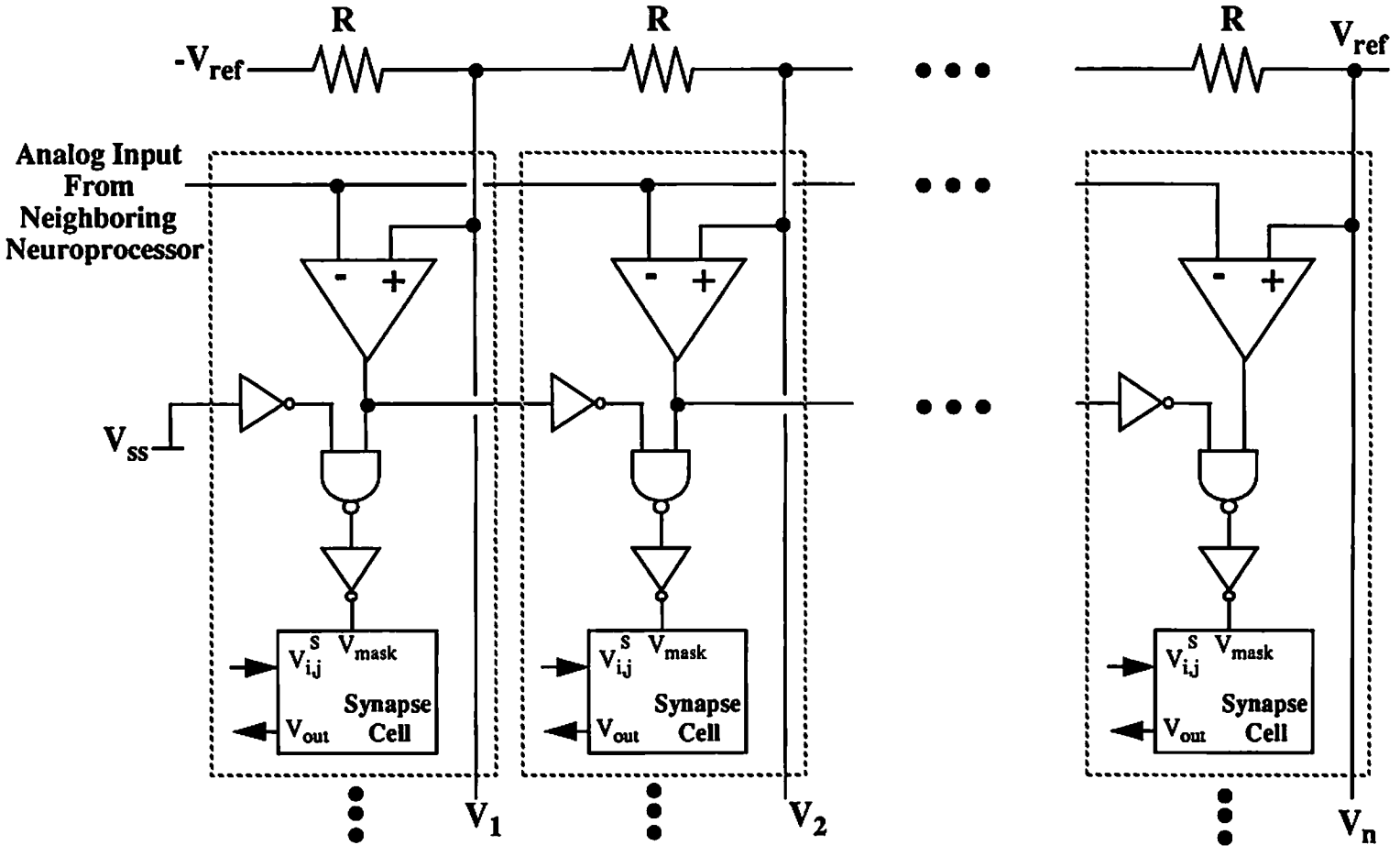


Fig. 3.9 A parallel and distributed analog-to-digital converter.

for one image pixel is shown in Fig. 3.10. It occupies an area of 2,482 × 5,636 λ^2 and contains 25 neurons, 25 × 27 synapse cells and is able to detect the moving object with 25 different velocities. In the hardware implementation, two rows of synapses are used to increase the resolution of synapse weights coming from the bias inputs and also to enhance the fault tolerance of the network. With an advanced 1.2-μm CMOS technology, 64 neuroprocessors can be accommodated into one VLSI neural chip of 1.5 × 2.8 cm^2 in size. The chip layout is shown in Fig. 3.11. It requires a 178-pin PGA package. The analog interprocessor data communication requires 128 pins. The detailed layout of interconnects among four neuroprocessors is shown in Fig. 3.12. The interconnection routing area occupies 23% of the chip area. A performance comparison against the digital bit-parallel point-to-point interconnection method is listed in Table 1. In the digital bit-parallel method, each data link requires 25 lines. Only 12 neuroprocessors can be accommodated in the same chip area and 85% of chip area will be used for the interconnection routing purpose.

With 128 VLSI neural chips and many supporting standard IC parts such as SRAMs and 8-bit DACs for storing the weight information and dynamically refreshing of the synapse cells, computation of optical flow from an image with 64 × 128 pixels and 256 gray levels can be performed at a rate of 30 frames per second. The system design for the fast motion detection using multiple VLSI neural chips is shown in Fig. 3.13.

To obtain the electrical properties of the basic circuit blocks, a test chip containing key circuit blocks has been fabricated with a 2-μm CMOS process from Orbit Semiconductor Inc. [23] through the MOSIS Service [24] and tested.

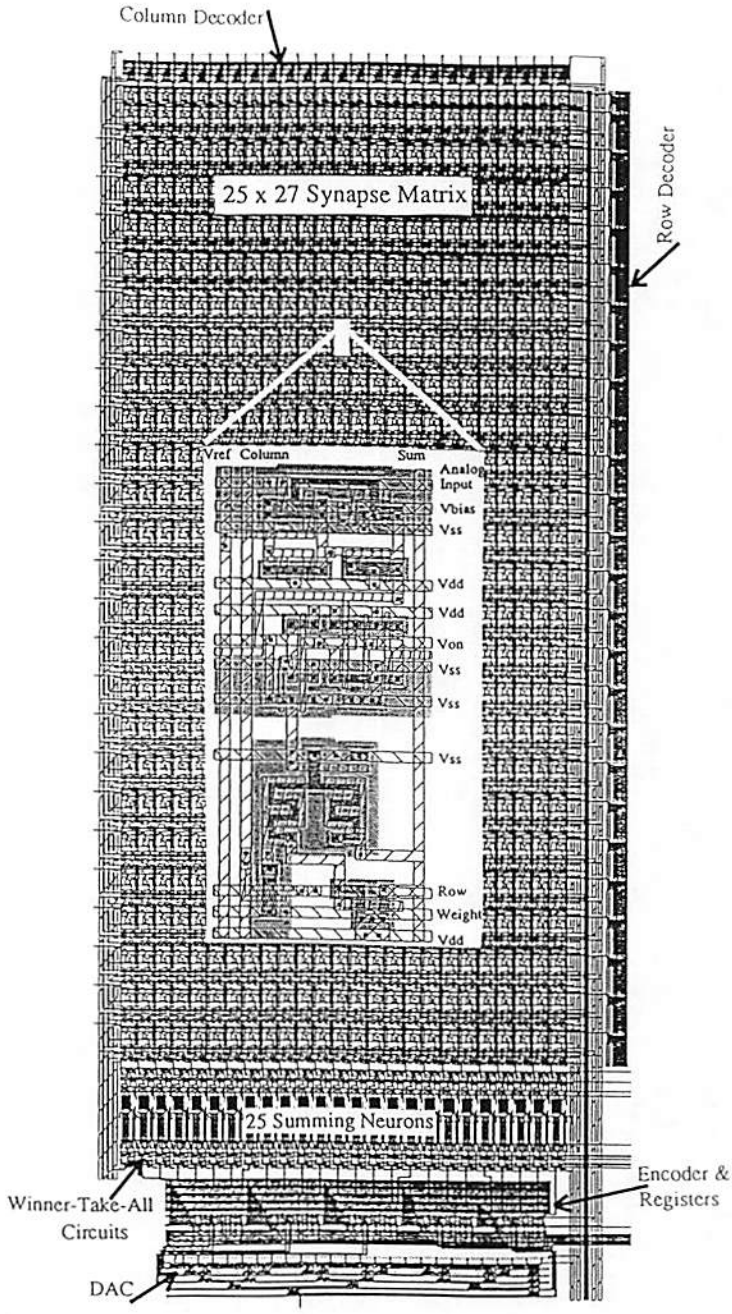


Fig. 3.10 The layout of one velocity-selective neuroprocessor. It contains 25 neurons, 25 x 27 synapse matrix and is able to detect the moving object with 25 different velocities.The synapse cell is shown in the insert.

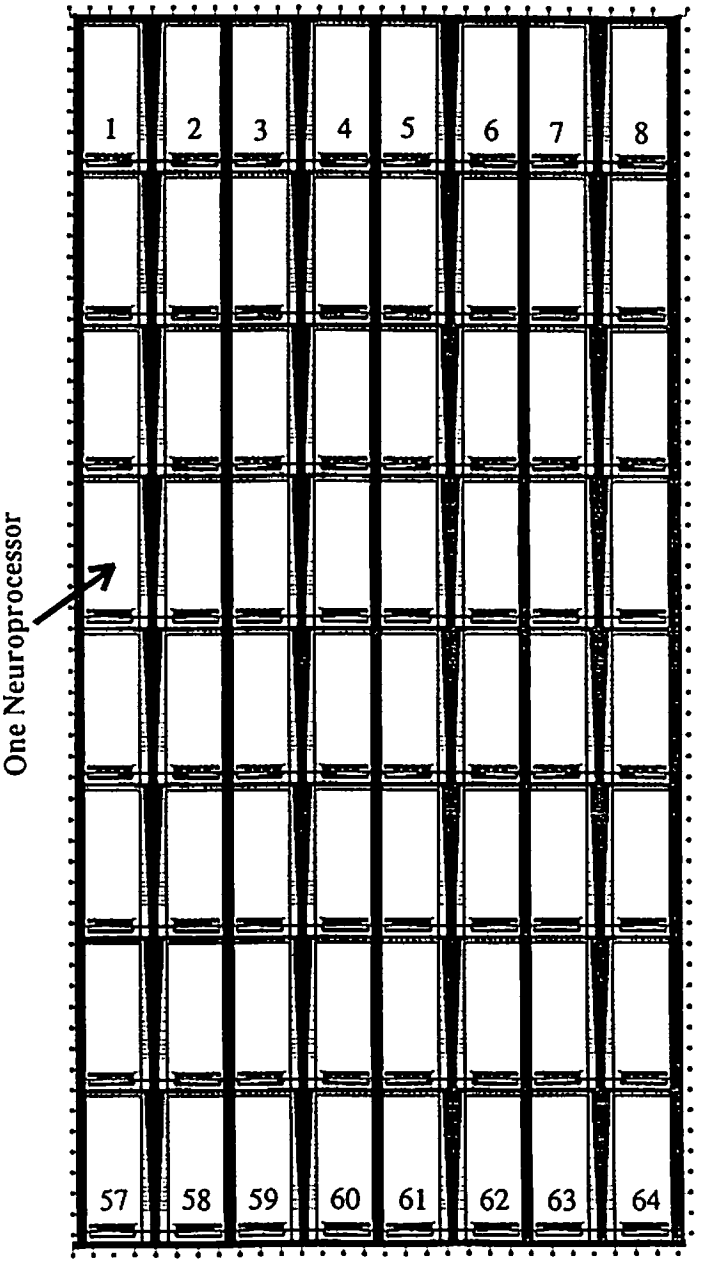


Fig. 3.11 The layout of VLSI neural chip. It consists of 64 neuroprocessors and occupies 1.5 x 2.8 cm^2 silicon area.

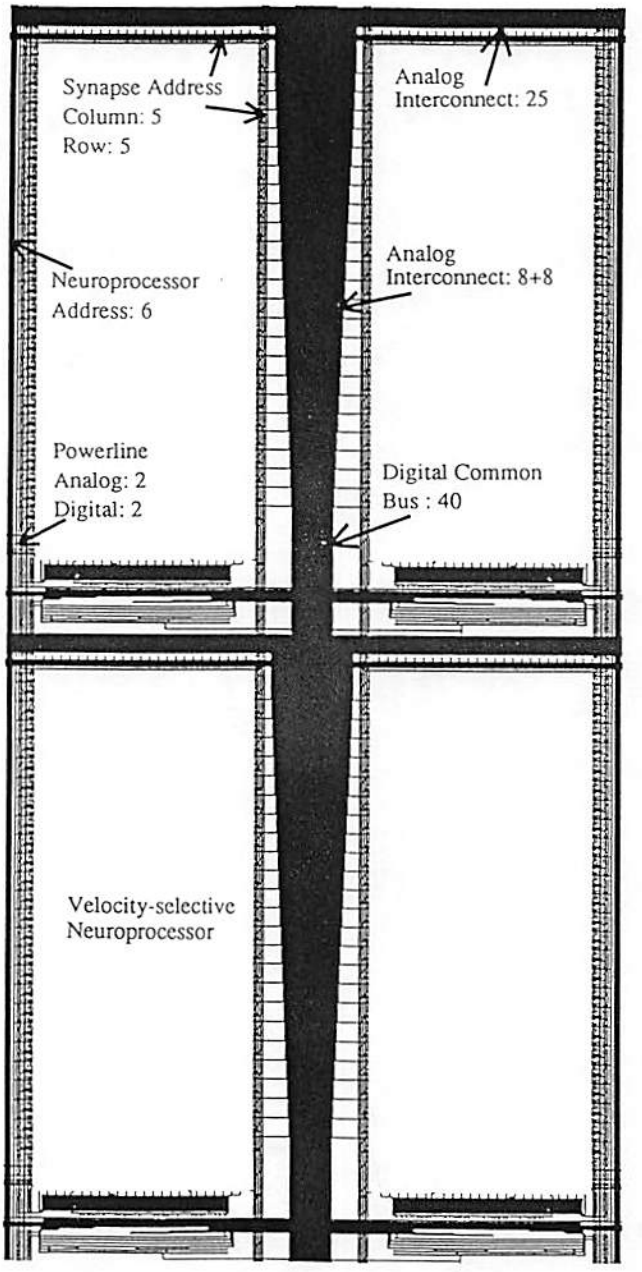


Fig. 3.12 The detailed layout of interconnects among four neuroprocessors.

Table 3.1 Performance Comparison of Two Interconnect Methods

Design Approach / Performance	Using Bit-Parallel Analog Interconnect Method	Using Bit-Parallel Digital Interconnect Method
Chip Size (cm^2)	1.5 x 2.8	1.5 x 2.8
Number of Neuro-processors	64	12
Interconnection Routing Area (%)	23	85
Pin Count	178	3,250
Network Iteration Time (nsec)	522	250
Speed Performance (connection/second)	8.03×10^{10}	2.08×10^{10}

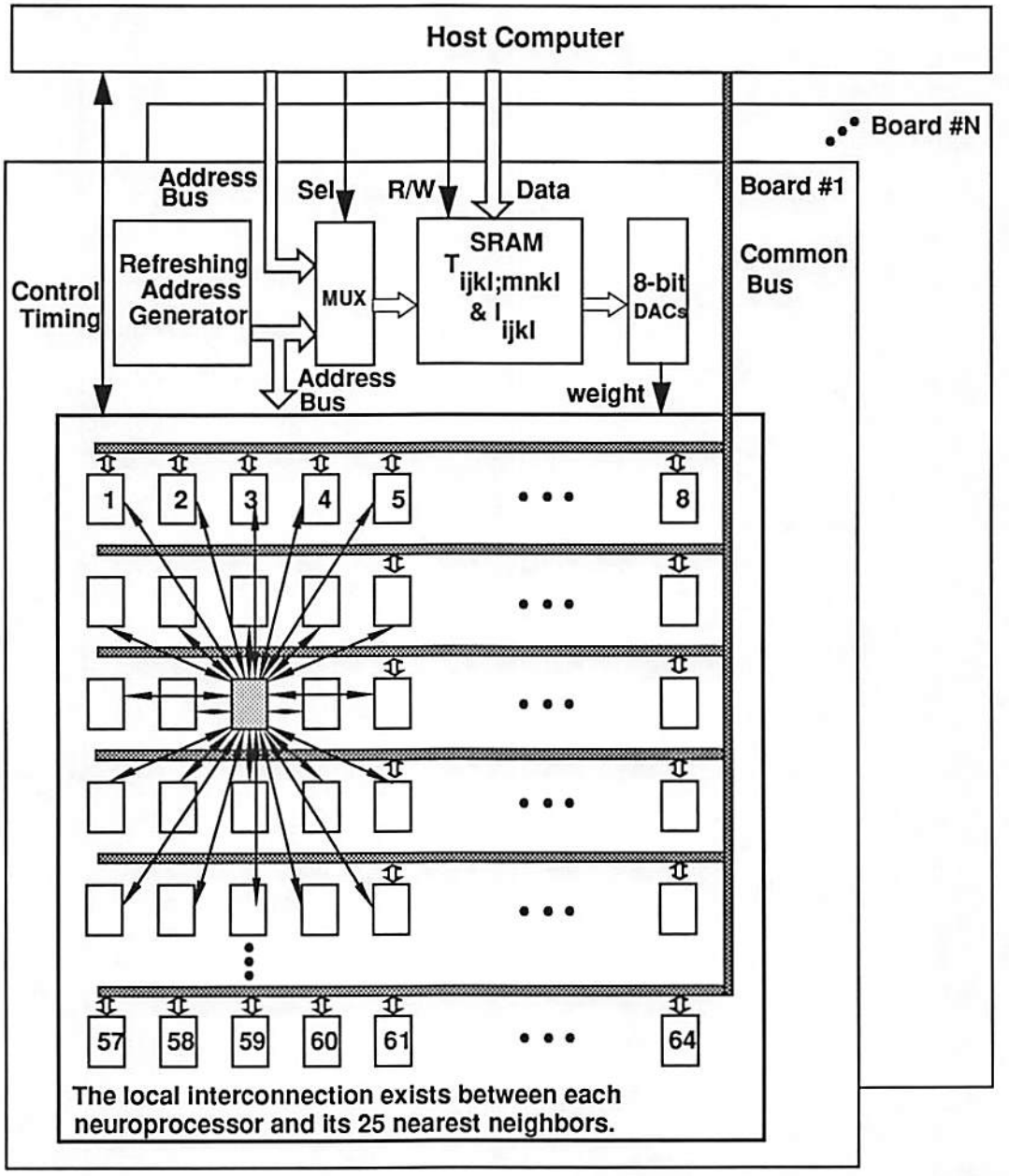
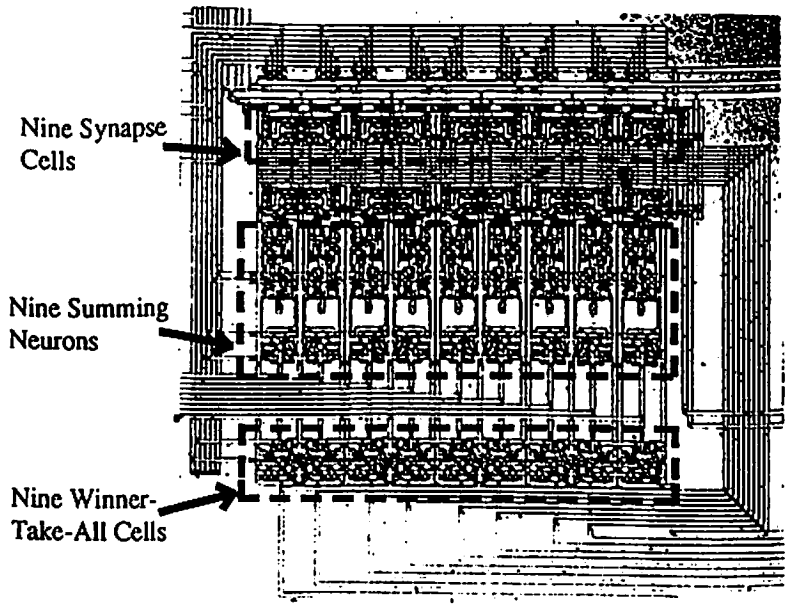


Fig. 3.13 System diagram for high-speed motion estimation using multiple VLSI neural chips.

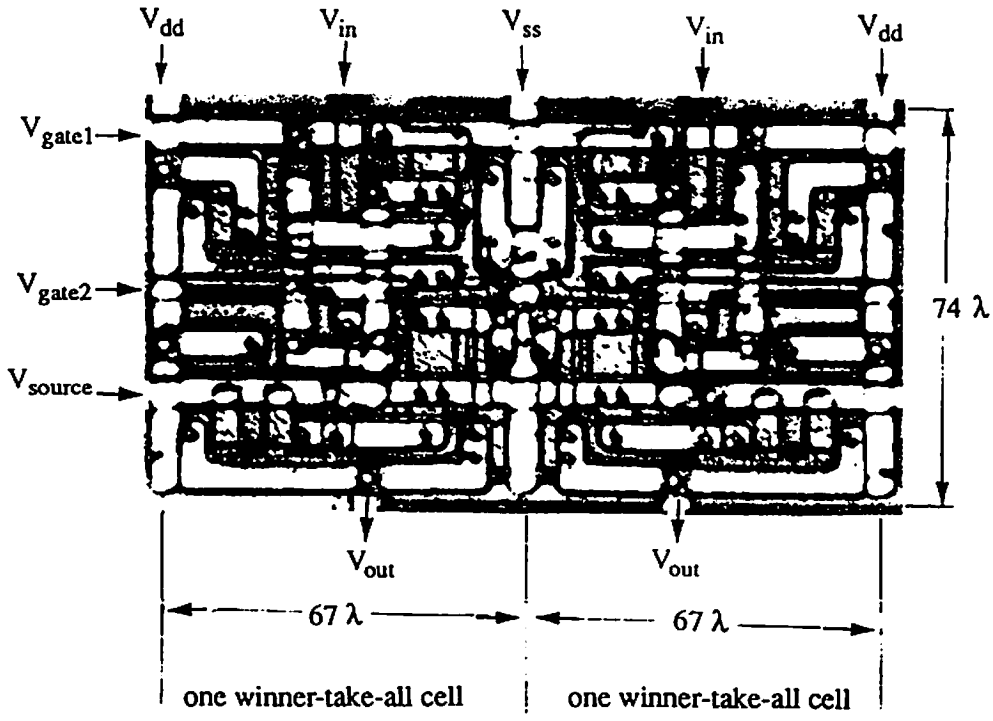
The die photo of the test chip is shown in Fig. 3.14(a). An enlarged die photo of two winner-take-all cells is shown in Fig. 3.14(b). The area of one winner-take-all cell is $67 \times 74\ \lambda^2$. Measured transfer curves of the synapse cell with different bias voltages are shown in Fig. 3.15. The dynamic range of the synapse cell is controlled by the bias voltage.

Experimental data on the winner-take-all circuit are shown in Figs. 3.16-3.18. Three experiments were conducted. In Fig. 3.16(a), a single-stage, nine-cell winner-take-all circuit is tested. Among of nine input analog signals, one input sweeps linearly from 1.47 V to 1.52 V, the second input is connected to 1.5 V, and the other seven inputs are kept at 1.475 V. By using a two-stage, nine-cell winner-take-all circuit, the corresponding output can be shown in Fig. 3.16(b). In Fig. 3.17(a), a single-stage, nine-cell winner-take-all circuit is tested. Among of nine input analog signals, one input sweeps linearly from -0.015 V to 0.035 V, the second input is connected to 0.015 V, and the other seven inputs are kept at -0.010 V. By using a two-stage, nine-cell winner-take-all circuit, the corresponding output can be shown in Fig. 3.17(b). In Fig. 3.18(a), a single-stage, nine-cell winner-take-all circuit is tested. Among of nine input analog signals, one input sweeps linearly from -1.53 V to -1.48 V, the second input is connected to -1.5 V, and the other seven inputs are kept at -1.525 V. By using a two-stage, nine-cell winner-take-all circuit, the corresponding output can be shown in Fig. 3.18(b). The winner-take-all function is successfully implemented with a resolution of 10 mV.

The processing time for one network iteration is around 522 $n$sec. Each iteration cycle includes synapse multiplication, neuron summing, winner-take-all



(a)



(b)

Fig. 3.14 Die photos of (a) Test module for optical flow computing, (b) Two winner-take-all cells.

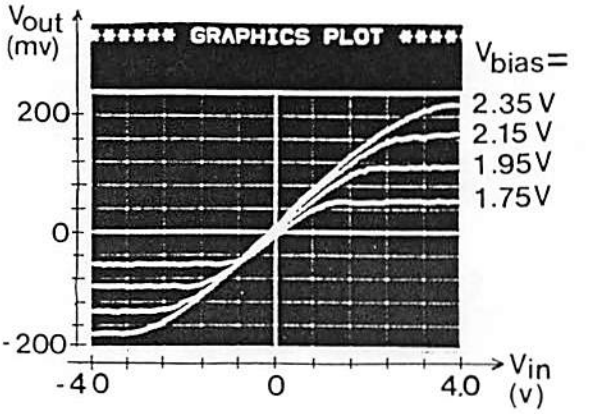
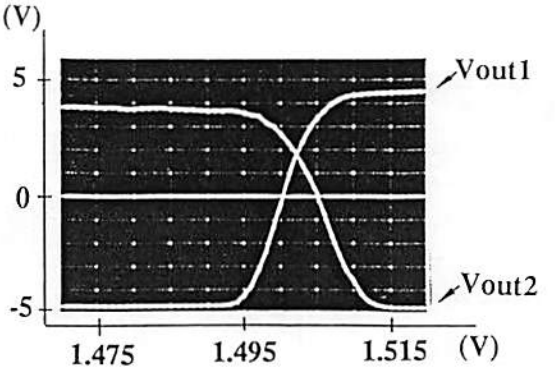
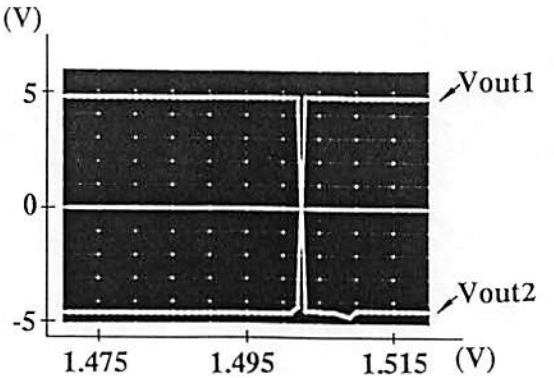


Fig. 3.15 Measured transfer curves of the synapse cell with different bias voltages.

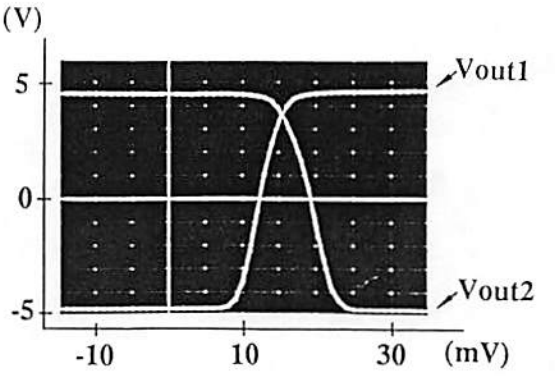


(a)

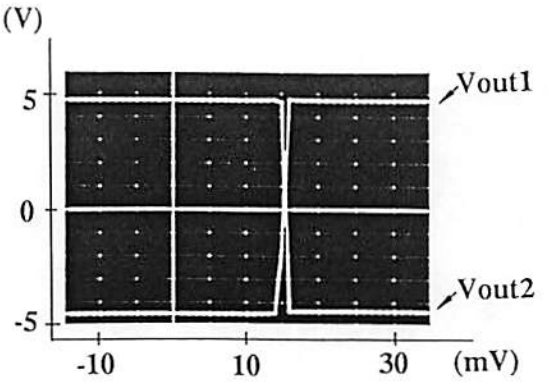


(b)

Fig. 3.16 Measured results of winner-take-all circuit with one input sweeps from 1.47V to 1.52V, the second input is connected to 1.5V, and the other seven inputs are kept at 1.475V.
(a) Output of one-stage winner-take-all circuit.
(b) Output of two-stage winner-take-all circuit.

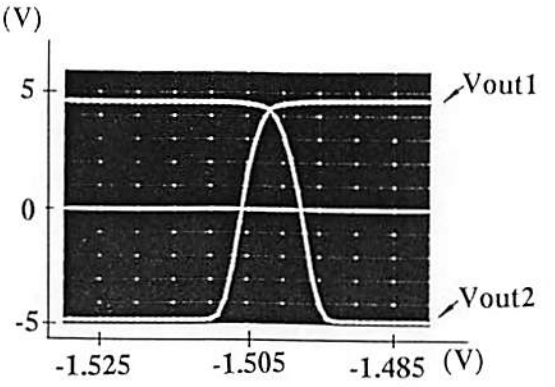


(a)

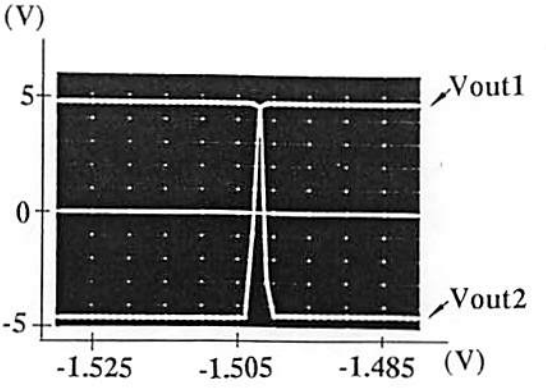


(b)

Fig. 3.17 Measured results of winner-take-all circuit with one input sweeps from -0.015V to 0.035V, the second input is connected to 0.015V, and the other seven inputs are kept at -0.010V.
(a) Output of one-stage winner-take-all circuit.
(b) Output of two-stage winner-take-all circuit.



(a)



(b)

Fig. 3.18 Measured results of winner-take-all circuit with one input sweeps from -1.53V to -1.48V, the second input is connected to -1.5V, and the other seven inputs are kept at -1.525V.
(a) Output of one-stage winner-take-all circuit.
(b) Output of two-stage winner-take-all circuit.

operation, data storage on latches, D/A and A/D conversion, and inter-processor data transfer. SPICE simulation results on various circuit blocks are listed in Table 3.2. The large response time of the synapse multiplication is due to the significant capacitance loading on the current-summation line. For the D/A conversion simulations, 5 pF and 50 pF effective capacitance loadings are estimated for inter-chip data communication and off-chip data communication, respectively. The major delay will come from the off-chip inter-processor data communication. The total of 8.32×10^{10} connections per second can be achieved by using one VLSI neural chip containing 1600 neurons, 41,600 synapses cells, and operated at a master clock rate of 2 MHz. Based on the results of Table 3.2, the speed comparison of a system using 128 VLSI neural chips with a Sun-4/75 SPARC workstation is listed in Table 3.3. The speedup factor is 24,242.

3.3.2 System-Level Analysis

System-level analysis on different images have been conducted to illustrate the performance of the motion estimation chip. The mismatch effect of analog synapse components has been included. Figure 3.19 shows the statistical distribution of measured synapse output conductances. A total of 300 synapses was measured. In Fig. 3.19(a), the synapse conductances can be described by a Gaussian distribution with a mean value of 14.07 $\mu A/V$ and a standard deviation of 0.042 $\mu A/V$ at weight voltage $V_{i,j}^{s} = 2\ V$. In Fig. 3.19(b), the synapse conductances can be described by a Gaussian distribution with a mean value of -13.69 $\mu A/V$ and a standard deviation of 0.036 $\mu A/V$ at weight voltage

Table 3.2 Circuit Response Time

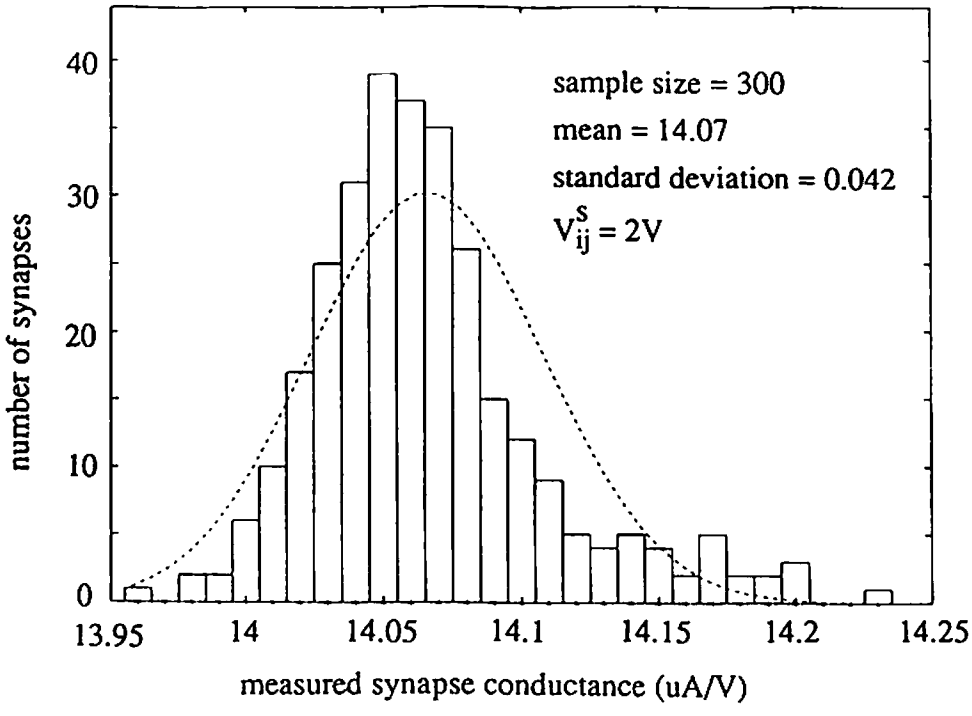
	Measured Results
Analog-to-Digital Conversion	73 ns
Synapse Multiplication	120 ns
Neuron Threshoding	20 ns
Winner-Take-All Operation	38 ns
Encoder & Data Latch	8 ns
Digital-to-Analog Conversion	84 ns + 263 ns *
Total	343 ns + 522 ns *

Note: T_{ox}= 202 Å in MOSIS 1.2-um CMOS technology.
+ with an output loading of 5 pF
* with an output loading of 50 pF

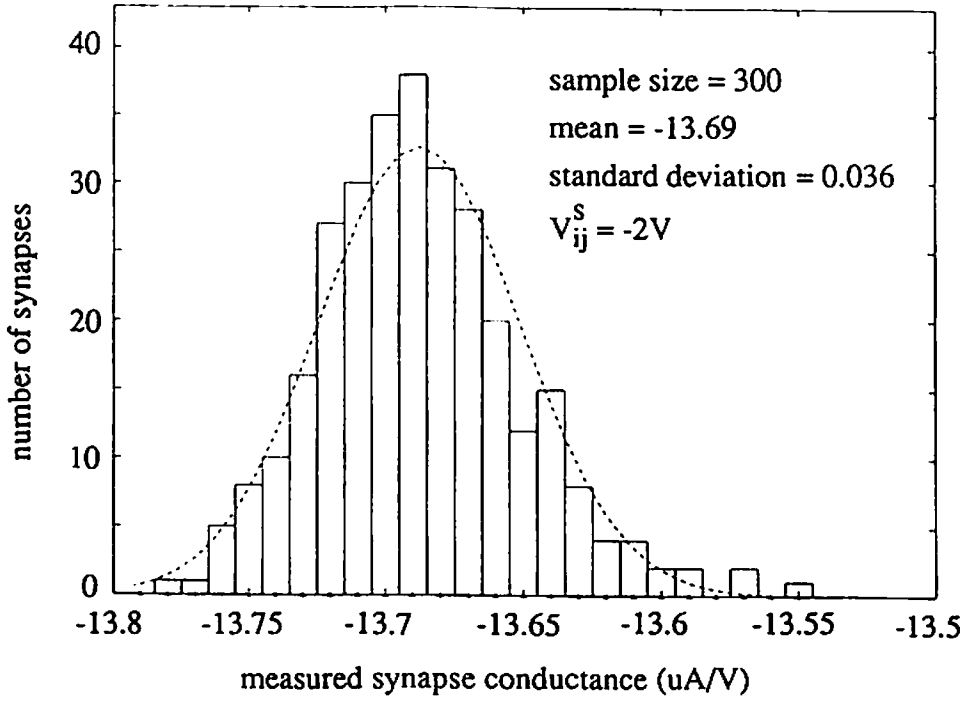
Table 3.3 Performance of VLSI Motion Estimation System

Synapse Weight Loading Time (into SRAM)	2,080 us (50ns per write)
Network Execution Time (for 36 iterations)	18.792 us
Neuron State Read Out Time	409.6 us (50 ns per read)
Total Processing Time	2.508 ms
Speed-Up Factor +	24,242

Note: + The comparison reference is Sun-4/75 SPARC-2 workstation.



(a)



(b)

Fig. 3.19 The statistical distribution of measured synapse output conductances.

$V^s_{i,j} = -2\ V$. During computer analysis, the effects of process variation on synapse weights are included through the Gaussian function.

Two sets of successive image frames directly produced by a Sony XC-77 CCD camera were used. To estimate the principal curvatures and the intensity values, a 5×5 window i.e., $\Gamma = 5$ was chosen and a third order polynomial was used for all frames. Figure 3.20(a)-(d) shows four successive image frames with a sedan moving from left to right against a stationary background. The size of each image frame is 100×135 pixels. The maximum displacement of the sedan between the time-varying image frame is 5 pixels. By setting $A = 4$, $B = 250$, $C = 0$, $D_k = 5$, and $D_l = 1$, the velocity field was obtained after 36 iterations. Figure 3.20(e) shows the final result with using synapse weights obtained by including the effects of process variation. Comparing with the result in Fig. 3.20(f), which the effects of process variation are not included, the motion information of the moving object still can be successful detected.

Figure 3.21(a)-(d) shows another four successive image frames with a mobile missile launcher moving from left to right have also been used for system-level analysis. The size of each image frame is 130×160 pixels. The maximum displacement of the mobile missile launcher between the time-varying image frame is 7 pixels. By setting $A = 4$, $B = 850$, $C = 0$, $D_k = 7$, and $D_l = 1$, the velocity field was obtained after 36 iterations. The final velocity fields of mobile missile launcher with and without including the device mismatch effects are shown in Fig. 3.21(e) and Fig. 3.21(f), respectively. The parameter A is set to be 4, because four successive image frames are used. The parameter B is chosen by using trial-and-error method. The parameter C is set to be 0 in

the prototype hardware implementation to simplify the neuron-state updating scheme of the network.

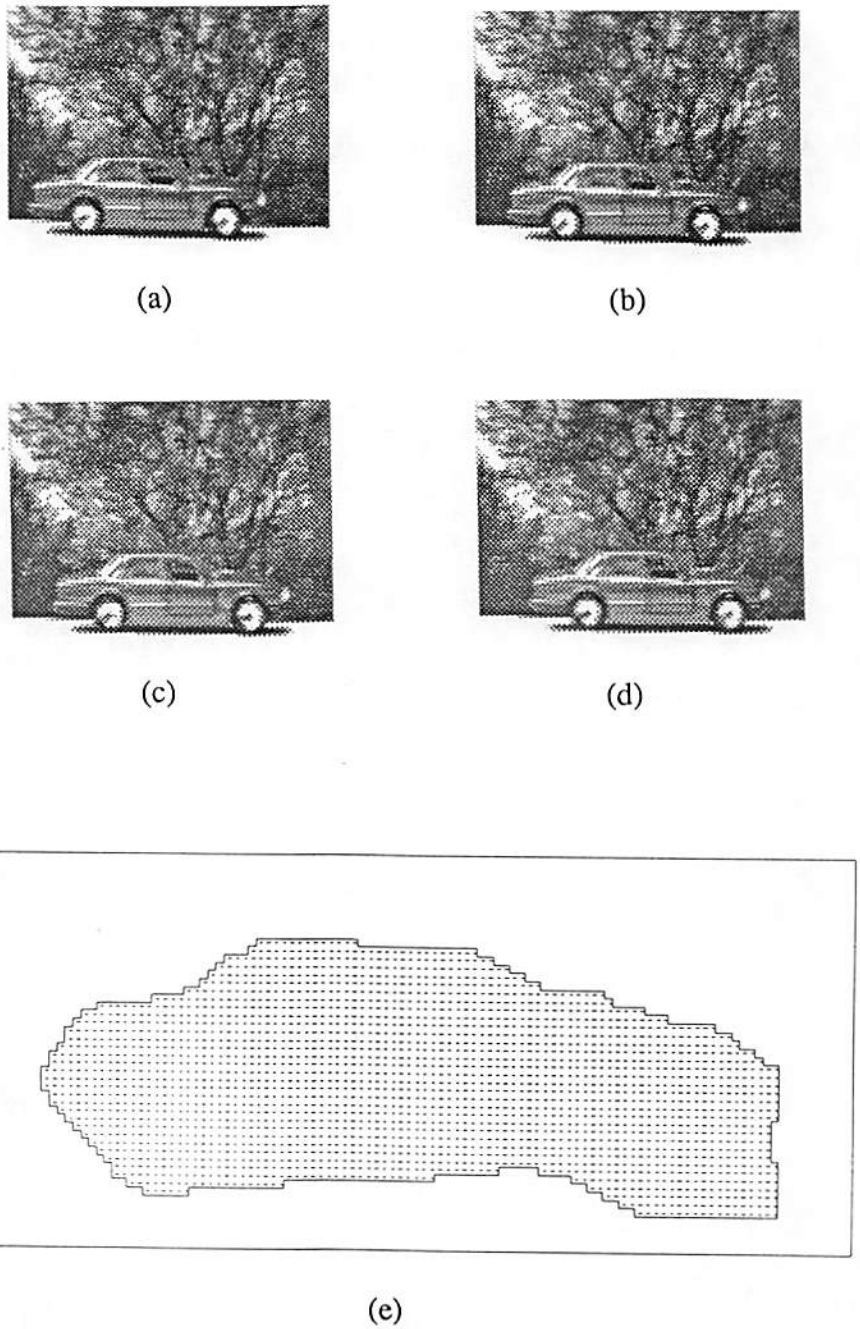
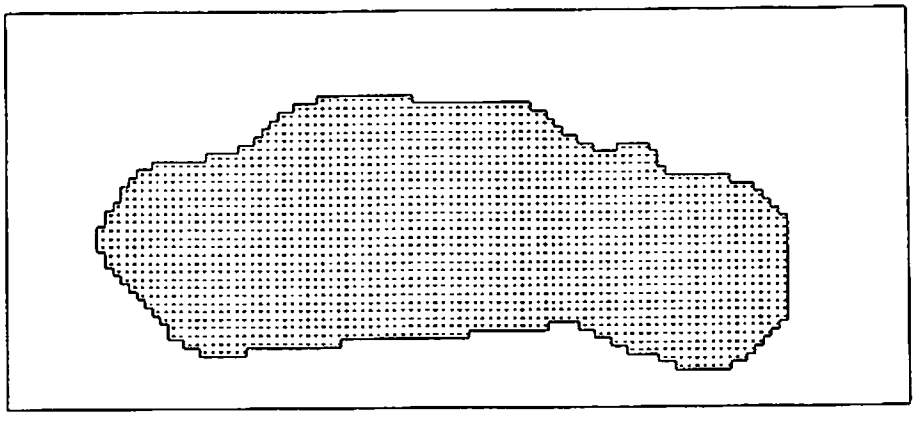


Fig. 3.20 System-level analysis on a sequence of four sedan images.
(a) The first frame. (b) The second frame.
(c) The third frame. (d) The fourth frame.
(e) By setting A = 4, B = 250, C = 0, D_k = 5, D_l = 1, and using synapse weights with device mismatch effect, the final result is obtained after 36 iterations.



(f)

Fig. 3.20 System-level analysis on a sequence of four sedan images.
(f) Using same conditions as (e) except the device mismatch effect has not been included.

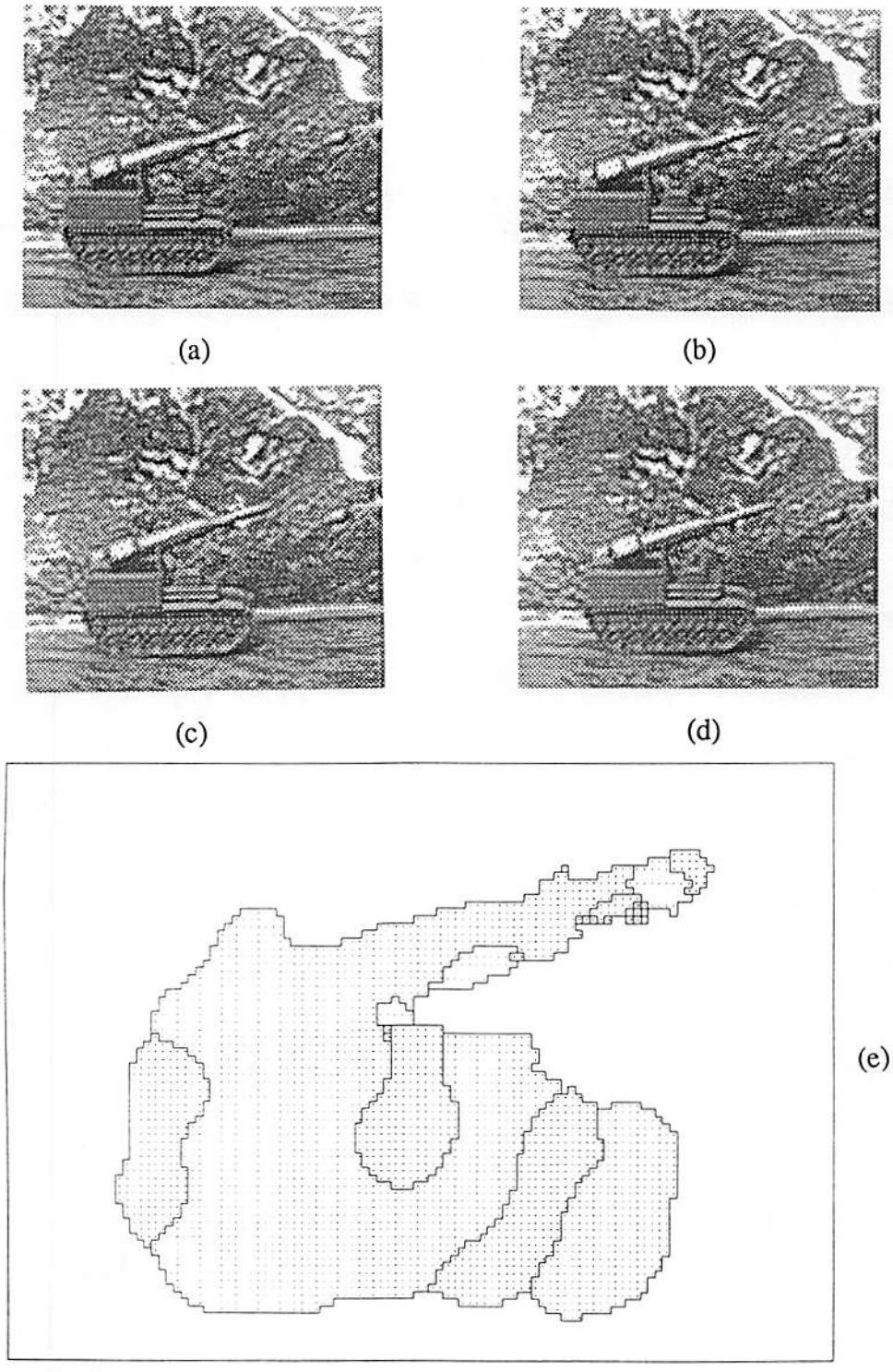
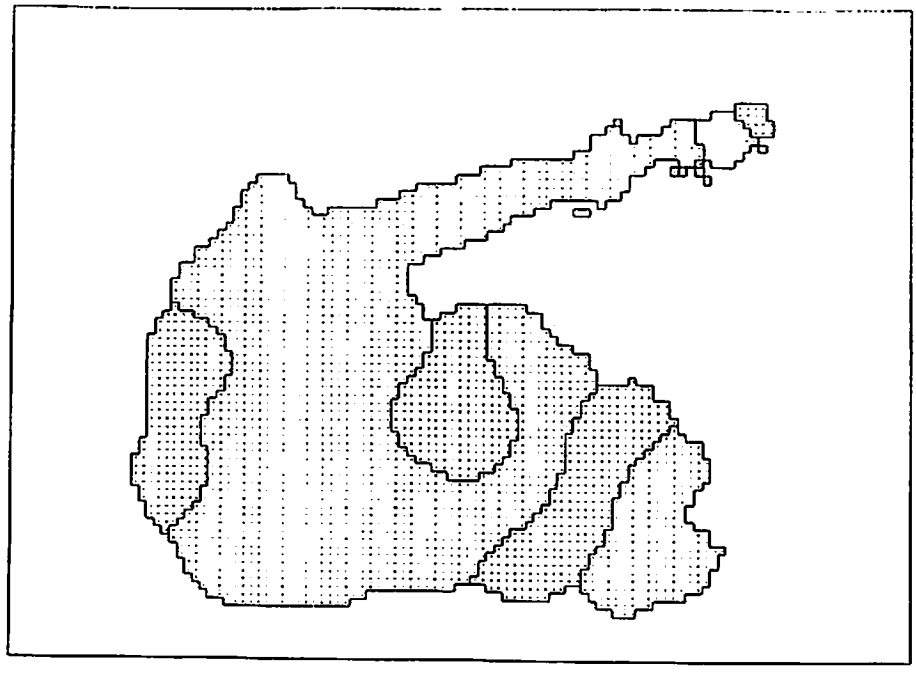


Fig. 3.21 System-level analysis on a sequence of four missile launcher images.
(a) The first frame. (b) The second frame.
(c) The third frame. (d) The fourth frame.
(e) By setting A = 4, B = 850, C = 0, D_k = 7, D_l = 1, and using synapse weights with device mismatch effect, the final result is obtained after 36 iterations.



(f)

Fig. 3.21 System-level analysis on a sequence of four missile launcher images.
(f) Using same conditions as (e) except the device mismatch effect has not been included.

References

[1] J. Aggarwal, N. Nandhakumar, "On the computation of motion from sequences of images - A review," *Proc. of the IEEE*, vol. 76, no. 8, pp. 917-935, Aug. 1989.

[2] W. Snyder, "Special issue on computer analysis of time-varying images," *IEEE Computer Magazine*, vol. 4, no. 8, Aug. 1981.

[3] T. Huang, *Image Sequence Analysis*, Springer-Verlag: New York, NY, 1981.

[4] B. Horn, *Robot Vision*, The MIT Press: Cambridge, MA, 1986.

[5] K. Prazdny, "Egomotion and relative depth map from optical flow," *Biological Cybernetics*, vol. 36, pp. 87-102, Springer-Verlag: Berlin, Germany, 1980.

[6] W. Simpson, "Depth discrimination from optical flow," *Perception*, vol. 17, pp. 497-512, 1988.

[7] G. Adiv, "Determining three-dimensional motion and structure from optical flow generated by several moving objects," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 384-401, July 1985.

[8] B. Ballard, O. Kimbal, "Rigid body motion from depth and optical flow," *Computer Graphics and Image Processing*, vol. 22, pp. 95-115, Academic Press: New York, NY, 1983.

[9] B. Horn, B. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185-203, North-Holland Publishing Co.: Amsterdam, Netherlands, 1981.

[10] N. Grzywacz, A. Yuille, "Massively parallel implementations of theories for apparent motion," *Technical Report AI Memo No. 888, CBIP Memo No. 016*, MIT Artificial Intelligence Lab. and Center for Biological Information Processing, June, 1987.

[11] J. Hutchinson, C. Koch, J. Luo, C. Mead, "Computing motion using analog and binary resistive networks," *IEEE Computer Magazine*, pp. 52-63, March, 1988.

[12] Y. Zhou, R. Chellappa, "Computation of optical flow using a neural network," *IEEE Proc. of Intl. Conf. on Neural Networks*, vol. 2, pp. 71-78, San Diego, CA, 1988.

[13] B. O'Neil, *Elementary Differential Geometry*, Academic Press: New York, NY, 1966.

[14] S. Ullman, *The Interpretation of Visual Motion*, The MIT Press: Cambridge, MA, 1979.

[15] B. W. Lee, B. J. Sheu, "A compact and general purpose neural chip with electrically programmable synapses," *IEEE Proc. of Custom Integrated Circuits Conf.*, pp. 26.6.1-26.6.4, Boston, MA, 1990.

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

[17] M. Holler, S. Tam, H. Castro, R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses," *IEEE/INNS Proc. of Intl. Joint Conf. on Neural Networks*, vol. 2, pp. 191-196, Washington D.C., 1989.

[18] B. W. Lee, B. J. Sheu, H. Yang, "Analog floating-gate synapses for general-purpose VLSI neural computation," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 6, June 1991.

[19] J. Dayhoff, *Neural Network Architectures*, pp. 97-114, Van Nostrand Reinhold: New York, NY, 1990.

[20] D. Rumelhart, D. Zipser, "Feature discovery by competitive learning," in *Parallel Distributed Processing*, editors, D. Rumelhart, J. McClelland, and PDP Group, pp. 151-193, The MIT Press: Cambridge, MA, 1986.

[21] R. Hecht-Nielsen, *Neurocomputing*, pp. 64-70, Addison-Wesley Publishing Co.: Reading, MA, 1990.

[22] J. Choi, B. J. Sheu, "Analog VLSI neural network implementations of hardware annealing and winner-take-all functions," *34th Midwest Symposium on Circuits and Systems*, Monterey, CA, May 1991.

[23] G. Lewicki, "Foresight: A fast turn-around and low cost ASIC prototyping alternative," *IEEE ASIC Seminar and Exhibit*, pp. 6.8.1-6.8.2, Rochester, NY, 1990.

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

Chapter 4

Digital Image Restoration on Neural-Based VLSI Multiprocessors

Restoration of a high-quality image from a degraded recording is an important problem in early vision processing. Image degradation due to optical aberration, atmospheric turbulence, motion, diffraction, and noise can be corrected. Various methods such as the inverse filter [1], Wiener filter [1], Kalman filter [2], and many other model-based approaches have been proposed for image restorations. One of the major drawbacks of most of the conventional image restoration algorithms is the computational complexity. Many simplifying assumptions such as wide-sense stationary property, availability of second-order image statistics are required to obtain computationally feasible algorithms. The inverse filter method works well for extremely high signal-to-noise images. The Wiener filter is usually implemented after the wide sense stationary assumption has been made for images. Furthermore, knowledge of the power spectrum or correlation matrix of the undegraded image is required. The Kalman filter approach can be applied to nonstationary image, but is computationally very expensive and the system dynamics of the undegraded image is also required. Therefore, it is desirable to develop a practical restoration algorithm that does not need much knowledge of the undegraded image. An artificial neural network that can perform extremely rapid computations is very attractive for image restoration [3].

In this chapter, an analog systolic architecture which employs multiple neuroprocessors for high-speed image restoration is designed to fully exploit the massively parallel computational power of the neural network. For a two-dimensional image, parallel processing is performed in the row direction and pipelined processing is performed in the column direction. In the VLSI architecture, each neuron is used to represent multiple bit information. The key is to use the neuron result as an increment/decrement information to control the pixel register. The pixel register content is combined in the synapse array with a distributed digital-to-analog conversion operation. By using this architecture, the number of neurons can be reduced by a factor of 2^M in the M-bit gray-level image processing [4,5].

4.1. Image Restoration Algorithm

Let Y and X denote the vectors of length L^2 obtained by lexicographically ordering the $L \times L$ degraded and original image arrays, respectively. A standard degradation model applicable to many space invariant imaging systems is given by the following expression [1,6]

$$Y = \theta ([H] X) + N \tag{4.1}$$

where H is an $L^2 \times L^2$ blur matrix constructed from the point spread function $h(k, l)$ of the imaging process, N is an independent noise field and θ is a pointwise transformation on $[H] X$. The problem of recovering X from Y is extremely ill-conditioned, having no unique solution. Most approaches to image restoration consequently attempt to find approximate solution to (4.1) by

optimizing some appropriately formulated criteria, with or without imposed constraints [7,8].

Conventional image restoration procedures assume the image degradation to be not only space-invariant but also linear. This assumption leads to a convolution model [9,10]. A further assumption of periodicity for the input functions results in a great simplification to this model by allowing the matrix H to be approximated by a matrix of block-circulant form.

By restricting θ in (4.1) to be the identity transformation, the linear image formation model can be obtained,

$$Y = [H]X + N. \tag{4.2}$$

If viewed from a deterministic standpoint, the restoration problem is one that solves a system of linear equations in the presence of noise. With no specific knowledge about the noise N, the criterion that an estimate $\hat{X}$ of the restored object should be sought is consistent with the norm of N being as small as possible. In the unconstrained optimization problem, the error function

$$E = (Y - [H]\hat{X})^t(Y - [H]\hat{X}) \tag{4.3}$$

is to be minimized with respect to $\hat{X}$, where the superscript t denotes the matrix transpose operation. For the nonlinear image system where the θ is not an identity matrix, similar procedures can be applied but more complicated numerical methods are needed [10,11].

Because the cost function is very complex and involves a large number of parameters, multiple local minima exist. Computationally intensive minimization techniques, such as simulated annealing [12,13] might be required. The collective

computational capability of analog neural networks provides a powerful new technique for solving such complex minimization problems rapidly [14,15].

Zhou et al [16] proposed a modified Hopfield network which uses redundant neurons to restore gray level images degraded by a known space-invariant blur function and noise. Based on the method by Takeda [17], the image gray levels are represented by the simple sum of neuron state variables which take binary values of logic-1 or logic-0. The restoration procedure consists of two stages: estimation of the parameters of the neural network model, and reconstruction of images. During the first stage, the parameters are estimated by comparing the energy function of the network with the constrained error function. The nonlinear restoration algorithm is then implemented using an iterative algorithm to minimize the energy function of the network.

In order to use the spontaneous energy-minimization process of the neural network, the image restoration problem can be solved by minimizing the error function,

$$E = \frac{1}{2}||Y - H\hat{X}||^2 + \frac{1}{2}\lambda||D\hat{X}||^2 \tag{4.4}$$

where $||Z||$ is the L^2 norm of Z and λ is a constant. Such a constrained error function is widely used in image restoration problems [1]. The first term in (4.4) is to find an $\hat{X}$ such that $H\hat{X}$ approximates Y in a least-squares sense. The second term is a smoothness constrain on the solution $\hat{X}$. The value of λ sets the amount of suppression on noise and ringing effects. For an image with low signal-to-noise ratio, a large λ is used to suppress effects due to noise. A common choice of D is a second-order Laplacian operator [16] which can be

approximated as a local window operator in the two dimensional discrete case. Typical value of λ lies in [0, 1].

In a Hopfield neural network which uses redundant neurons for representing the image gray level, $L^2 \times 2^M$ mutually interconnected neurons are needed for an $L \times L$ image with M-bit gray level. Let $V = \{v_{i,k}, \text{where } 1 \le i \le L^2,\ 1 \le k \le 2^M\}$ be a binary state set of the neural network with $v_{i,k}$ denoting the state of the (i,k)-th neuron, $T_{i,k;j,l}$ be the synaptic interconnection strength from neuron (i,k) to neuron (j,l) and $I_{i,k}$ be the bias input. The neuron (i,k) synchronously receives inputs from all neurons including itself and a bias input,

$$u_{i,k} = \sum_{j}^{L^2} \sum_{i}^{2^M} T_{i,k;j,l} v_{j,l} + I_{i,k} \tag{4.5}$$

The $u_{i,k}$ is then fed back to corresponding neurons after thresholding

$$v_{i,k} = g(u_{i,k}) \tag{4.6}$$

where $g(z)$ is a nonlinear function

$$g(z) = \begin{cases} 1 & \text{if } z \ge 0 \\ 0 & \text{if } z < 0. \end{cases} \tag{4.7}$$

The energy function of the network can be expressed as

$$E = -\frac{1}{2} \sum_{i=1}^{L^2} \sum_{j=1}^{L^2} \sum_{k=1}^{2^M} \sum_{l=1}^{2^M} T_{i,k;j,l} v_{i,k} v_{j,l} - \sum_{i=1}^{L^2} \sum_{k=1}^{2^M} I_{i,k} v_{i,k} \ . \tag{4.8}$$

In this model, the image is described by a set of gray level functions $\{x(i,j)$ where $1 \le i,j \le L\}$ with $x(i,j)$ denoting the gray level of the pixel (i,j). The image gray level function can be represented by a simple sum of the neuron state variables as

$$x(i,j) = \sum_{k=1}^{2^M} v_{m,k} \tag{4.9}$$

where $m = (i-1) \times L + j$.

By comparing (4.4) and (4.8), the synapse weights $\{T_{i,k;j,l}\}$ and the input bias currents $\{I_{i,k}\}$ of a Hopfield network are given as

$$T_{i,k;j,l} = -\sum_{p=1}^{L^2} h_{p,i} h_{p,j} - \lambda \sum_{p=1}^{L^2} d_{p,i} d_{p,j} \tag{4.10}$$

and

$$I_{i,k} = \sum_{p=1}^{L^2} y_p h_{p,i} \,. \tag{4.11}$$

Here, $h_{i,j}$ and $d_{i,j}$ are the elements of the matrices H and D, respectively. y_p is the element of the column vector Y.

Since self-feedback terms of the synapse matrix T are not zero, solutions of the modified Hopfield network operation are not guaranteed to move in the direction of lowering the energy function. From (4.10) and (4.11), the synapse weights and the input bias currents can be determined if the blur function is known and the signal-to-noise ratio of the image is high. For the image with low signal-to-noise ratio, a large λ is used to suppress effects due to noise.

The space-invariant blur function can be written as a convolution over a small window. For example, a uniform and space-invariant blur function with a 3×3 window size can be expressed as

$$h(k,l) = \frac{1}{9} \qquad \text{if } |k|,|l| \leq 1. \tag{4.12}$$

The corresponding H matrix is a block-circulant matrix and can be written as

$$H = \begin{bmatrix} H_1 & H_1 & O & O & \cdots & O & O & H_1 \\ H_1 & H_1 & H_1 & O & \cdots & O & O & O \\ . & . & . & . & & . & . & . \\ . & . & . & . & \cdots & . & . & . \\ . & . & . & . & & . & . & . \\ H_1 & O & O & O & \cdots & O & H_1 & H_1 \end{bmatrix} \tag{4.13}$$

where

$$H_1 = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & 0 & 0 & \cdots & 0 & 0 & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} & 0 & \cdots & 0 & 0 & 0 \\ . & . & . & . & & . & . & . \\ . & . & . & . & \cdots & . & . & . \\ . & . & . & . & & . & . & . \\ \frac{1}{9} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \tag{4.14}$$

and O is null matrix whose elements are all zeros.

For the image with a high signal-to-noise ratio, the λ value can be set to zero and the synapse weights for each pixel can be described as a 5×5 local window,

$$T = \begin{bmatrix} -\frac{1}{81} & -\frac{2}{81} & -\frac{3}{81} & -\frac{2}{81} & -\frac{1}{81} \\ -\frac{2}{81} & -\frac{4}{81} & -\frac{6}{81} & -\frac{4}{81} & -\frac{2}{81} \\ -\frac{3}{81} & -\frac{6}{81} & -\frac{9}{81} & -\frac{6}{81} & -\frac{3}{81} \\ -\frac{2}{81} & -\frac{4}{81} & -\frac{6}{81} & -\frac{4}{81} & -\frac{2}{81} \\ -\frac{1}{81} & -\frac{2}{81} & -\frac{3}{81} & -\frac{2}{81} & -\frac{1}{81} \end{bmatrix}. \tag{4.15}$$

Notice that each pixel has the self-feedback connection and is connected to the neighboring 24 pixels.

For most practical cases of image processing, a Gaussian blur function would be more appropriate. It takes the form

$$h(k,l) = \begin{cases} \frac{1}{2} & \text{if } k = 0, l = 0 \\ \frac{1}{16} & \text{if } |k|, |l| \leq 1, (k,l) \neq (0,0). \end{cases} \tag{4.16}$$

The corresponding H matrix is also a block-circulant matrix and can be written as

$$H = \begin{bmatrix} H_0 & H_1 & O & O & \cdots & O & O & H_1 \\ H_1 & H_0 & H_1 & O & \cdots & O & O & O \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ H_1 & O & O & O & \cdots & O & H_1 & H_0 \end{bmatrix} \tag{4.17}$$

where

$$H_0 = \begin{bmatrix} \frac{1}{2} & \frac{1}{16} & 0 & 0 & \cdots & 0 & 0 & \frac{1}{16} \\ \frac{1}{16} & \frac{1}{2} & \frac{1}{16} & 0 & \cdots & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \frac{1}{16} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{16} & \frac{1}{2} \end{bmatrix} \tag{4.18}$$

and

$$H_1 = \begin{bmatrix} \frac{1}{16} & \frac{1}{16} & 0 & 0 & \cdots & 0 & 0 & \frac{1}{16} \\ \frac{1}{16} & \frac{1}{16} & \frac{1}{16} & 0 & \cdots & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & & \cdot & \cdot & \cdot \\ \frac{1}{16} & 0 & 0 & 0 & \cdots & 0 & \frac{1}{16} & \frac{1}{16} \end{bmatrix} \tag{4.19}$$

The corresponding synapse matrix T can be described as

$$T = \begin{bmatrix} -\frac{1}{256} & -\frac{2}{256} & -\frac{3}{256} & -\frac{2}{256} & -\frac{1}{256} \\ -\frac{2}{256} & -\frac{18}{256} & -\frac{20}{256} & -\frac{18}{256} & -\frac{2}{256} \\ -\frac{3}{256} & -\frac{20}{256} & -\frac{72}{256} & -\frac{20}{256} & -\frac{3}{256} \\ -\frac{2}{256} & -\frac{18}{256} & -\frac{20}{256} & -\frac{18}{256} & -\frac{2}{256} \\ -\frac{1}{256} & -\frac{2}{256} & -\frac{3}{256} & -\frac{2}{256} & -\frac{1}{256} \end{bmatrix}. \tag{4.20}$$

To make effective use of a neuron, a modified algorithm is presented in [16,18]. Only one neuron is assigned to one pixel. A pixel register is added to store the gray level information. Each pixel register takes discrete values from 0 to 2^M and its value is increased by 1, decreased by 1, or remains unchanged according to the input of the neuron. Let u_i represent the input of the ith neuron, v_i represent the content of the ith pixel register. Then the transfer function of the neuron is given by

$$v_i^{(k+1)}(u_i) = \begin{cases} v_i^{(k)} - 1, & \text{if } u_i^{(k)} < -\frac{1}{2}c_i \\ v_i^{(k)}, & \text{if } -\frac{1}{2}c_i \le u_i^{(k)} \le \frac{1}{2}c_i \\ v_i^{(k)} + 1, & \text{if } u_i^{(k)} > \frac{1}{2}c_i \end{cases} \tag{4.21}$$

where

$$u_i^{(k)} = I_i + \sum_{j=1}^{L^2} T_{i,j} v_j^{(k)}, \quad \text{and} \quad c_i = |T_{ii}|. \tag{4.22}$$

The superscript k denotes the iteration number. For the case of a uniform, 3×3 blur function, the value of c_i is $\frac{1}{9}$. The data representation requires an eight-bit register for each pixel.

4.2. The Neural-Based Multiprocessor Design

4.2.1 Neuroprocessor Architecture

Figure 4.1 shows the functional block diagrams of the Hopfield neural network for digital image restoration. In the first configuration, which uses the simple sum number representation scheme, each pixel with M-bit gray level is represented by 2^M neurons. The output of neuron takes binary values of logic 1 or logic 0. The circuits in this configuration are not efficiently used. For a 256×256 image with 256 gray levels, $256 \times 256 \times 256$ neurons and $(256 \times 256 \times 256)^2$ synapses are needed. To overcome implementation difficulty on this architecture, a multiple-bit pixel register is used to incorporate the gray level information into the neural network. The binary neuron outputs are used to increment or decrement the gray level for each pixel via digital circuitry. The multiple-bit pixel register controls a multiplying digital-to-analog conversion circuitry to send the pixel information into the synapse matrix. Each information bit in the pixel register is connected to the synapse cell with the weight value scaled by the bit order. By using the second configuration, the number of neurons is reduced by a factor of 2^M and that of synapses is reduced by a factor of 2^{2M-3}, for an M-bit gray level image.

For a blur function of $w \times w$ window size, each neuron in the image restoration chip is connected to the $(2w-1) \times (2w-1)$ neighboring neurons. In our study, a blur function with the window size of 3×3 is used for illustration purposes. As shown in Fig. 4.2, an image neuroprocessor which can process a 5×5-pixel subimage is constructed with one synapse matrix, one neuron array,

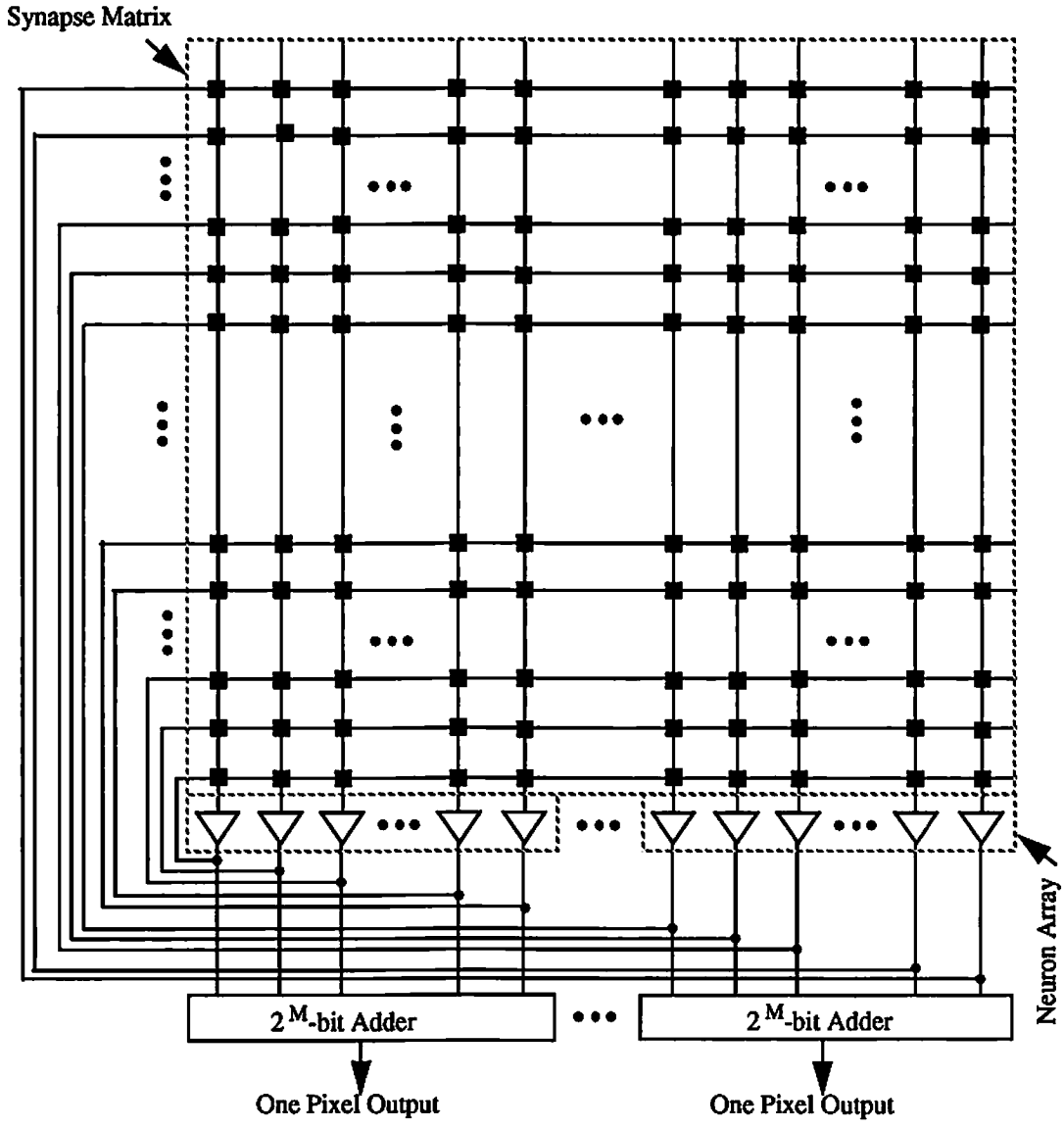


Fig. 4.1 Functional block diagrams of the Hopfield neural network for digital image restoration.
(a) The simple sum number representation scheme is used.

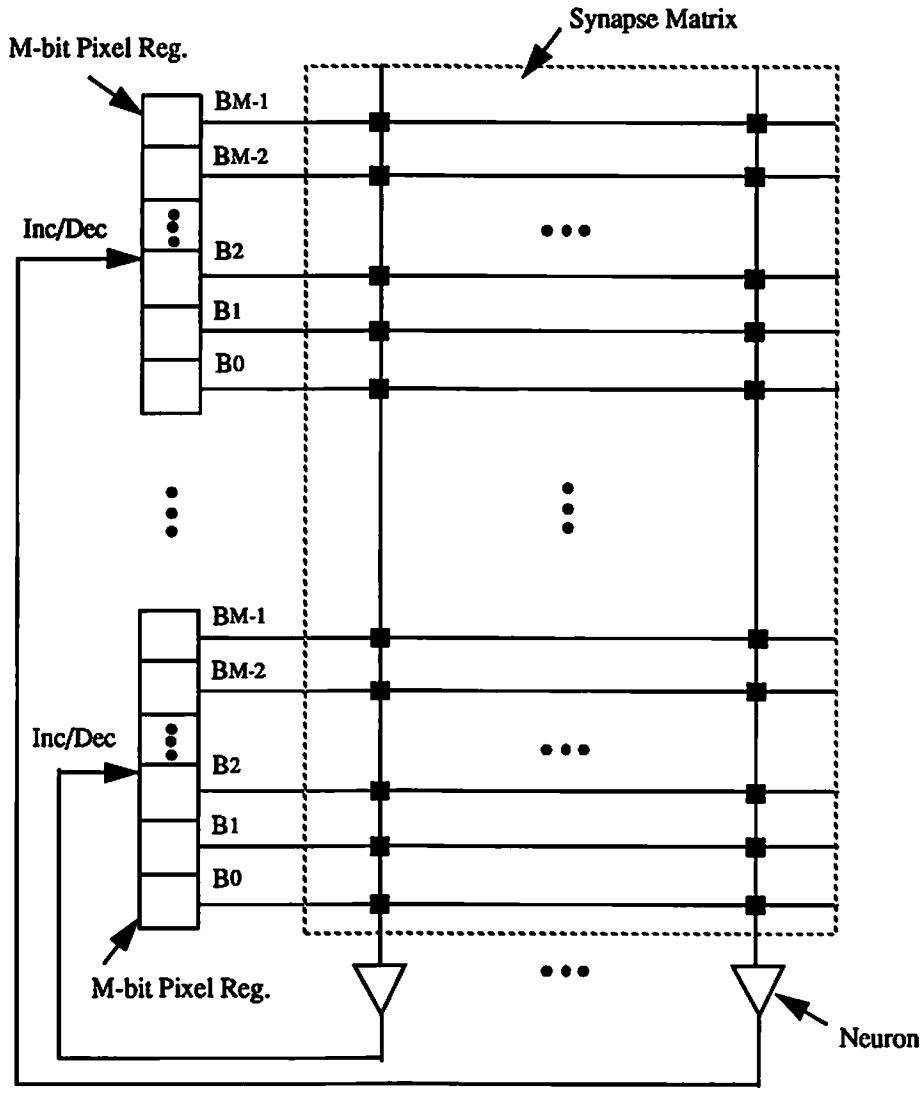


Fig. 4.1(b) Each pixel is representd by one neuron. The output state of neuron is used to increment/decrement the gray value of the pixel register.

and associated digital pixel registers. Each pixel-register set contains two shift-register chains. One is to store the degraded image data and the other is to store the updated image data. The original degraded image data are used as reference information during the neural network operation. They are used to provide the constant input bias $I_{i,k}$ according to (4.11). For a 3×3 uniform blur function, the input-bias of each neuron is the weighted sum of the gray levels from pixels in a neighborhood of size 3×3. Each pixel-register set is used to store the gray-level information for 5 pixels. During the network operation, only the middle pixel register in the updated shift-register chain will be increased or decreased according to the output state of its corresponding neuron. The other pixel registers function as shift registers.

Each neuroprocessor requires one image input port and one output port. The input port consists of 40 data lines to accept 5 rows of 8-bit image data. In each clock period, 40 bit data are piped into the shift registers. In total, 200 bit lines will be needed for the 5 data input ports and another 200 bit lines for the 5 output ports in a 5-neuroprocessor VLSI chip. In the initialization stage, it takes 5 clock cycles to fill up the 8-bit shift-register pipe. The synapse array is arranged for the center neuron in the neuroprocessor to obtain complete information from the surrounding pixels. Two types of synapse cells are shown in Fig. 4.2. The weight values of the synapse cells, which are marked as solid circles, are determined from the synapse matrix T in (4.15) or (4.20), depending on the type of the blur function. The synapse cells that are marked as cross signs provide the input bias information determined from the degraded image. For a 3×3 uniform blur function, the weight values for these input-bias synapse cells are all $\frac{1}{9}$.

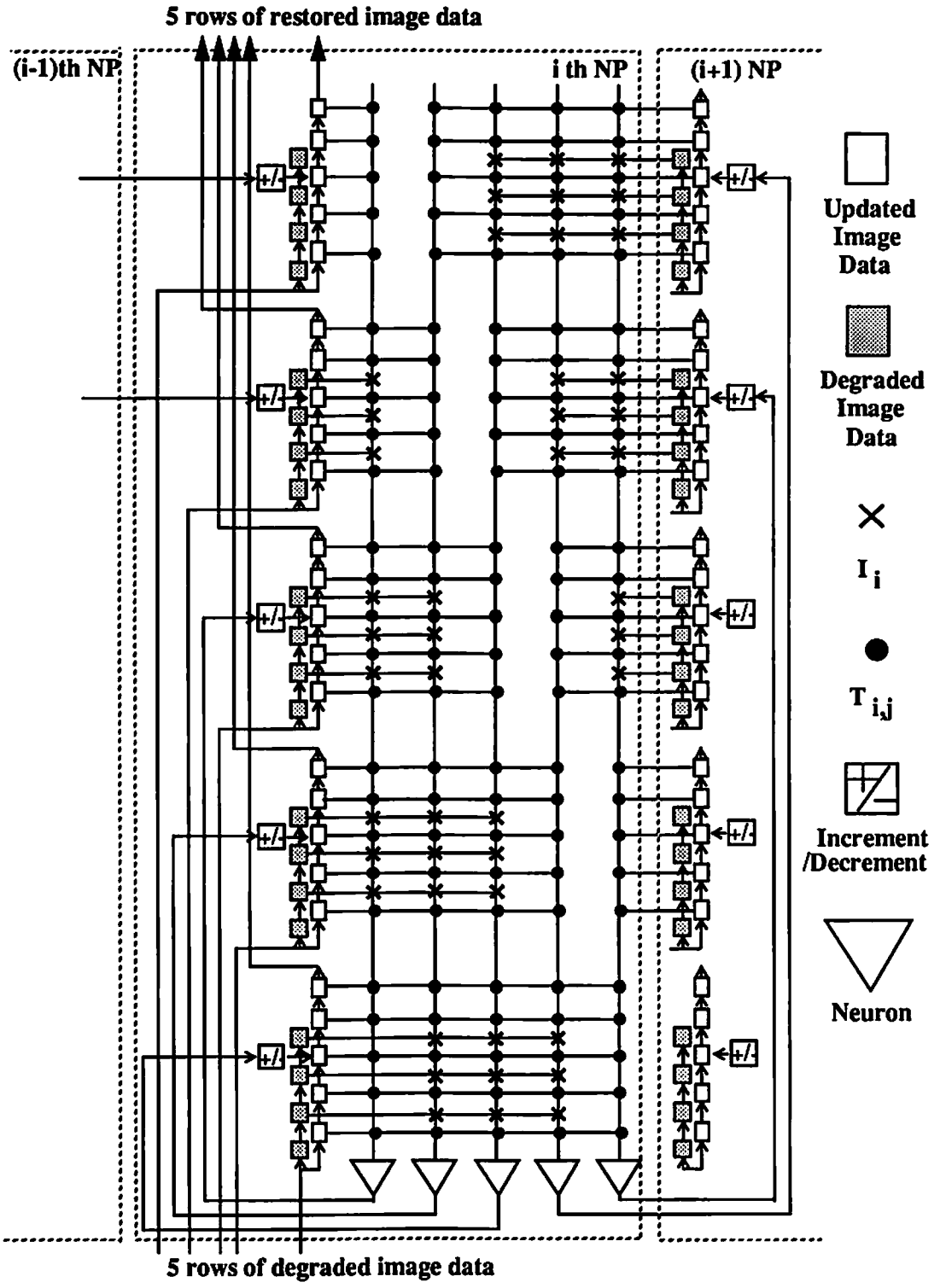


Fig. 4.2 Functional block diagram of image restoration neuroprocessor.

In the prototype neural chip, there are (34 × 8) synapses per neuron. Among of these synapse cells, (9 × 8) synapses are used for constant input-bias information.

The interaction of the pixels between two adjacent neuroprocessors can be accomplished through the data links in synapse array as shown in Fig. 4.3. By using this systolic architecture, image restoration process can be accomplished in parallel in the row direction. In the column direction, the image data are pipelined into the pixel registers and processed serially. The weight values of the synapse matrix for each subimage restoration neuroprocessor are the same.

Dynamic refreshing of the charge-storage capacitances in the synapse array for each neuroprocessor can be done without interrupting the normal network retrieving process. One two-port SRAM with 8000 words, one 8-bit digital-to-analog converter, and associated logic gates to generate the control timing and addresses for weight refreshing are used. The overall system configuration for high-speed image restoration using multiple VLSI boards is shown in Fig. 4.4. The synapse weight information is computed by the host computer and transferred to the SRAM. The 8-bit D/A converter converts the digital representation of the synapse weights into analog values in order to charge the dynamic capacitances of the synapse matrix. The two-port static-RAM and differential amplifier-based synapse design allow retrieving and learning processes to occur concurrently.

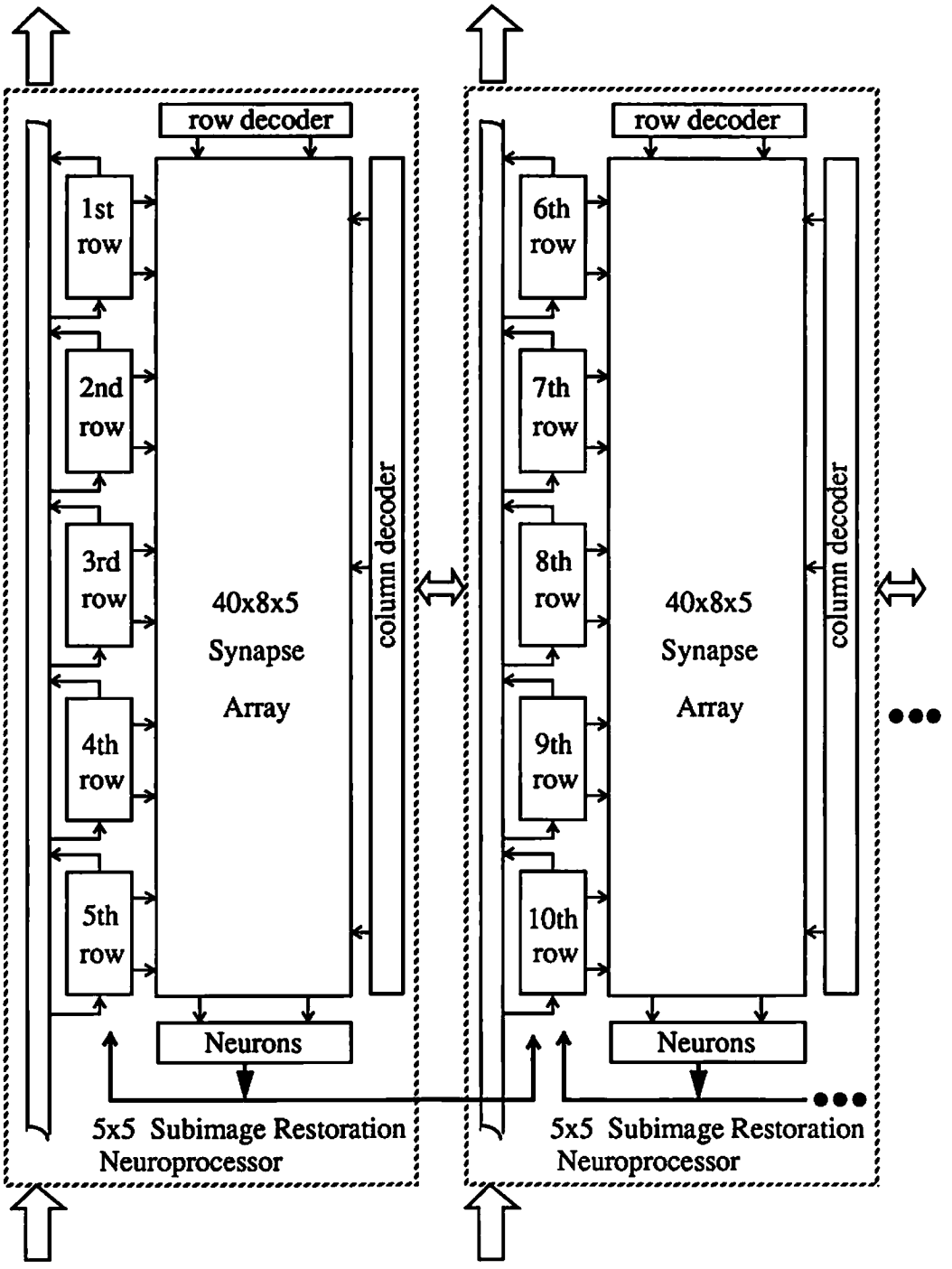


Fig. 4.3 Interaction of multiple image restoration neuroprocessors.

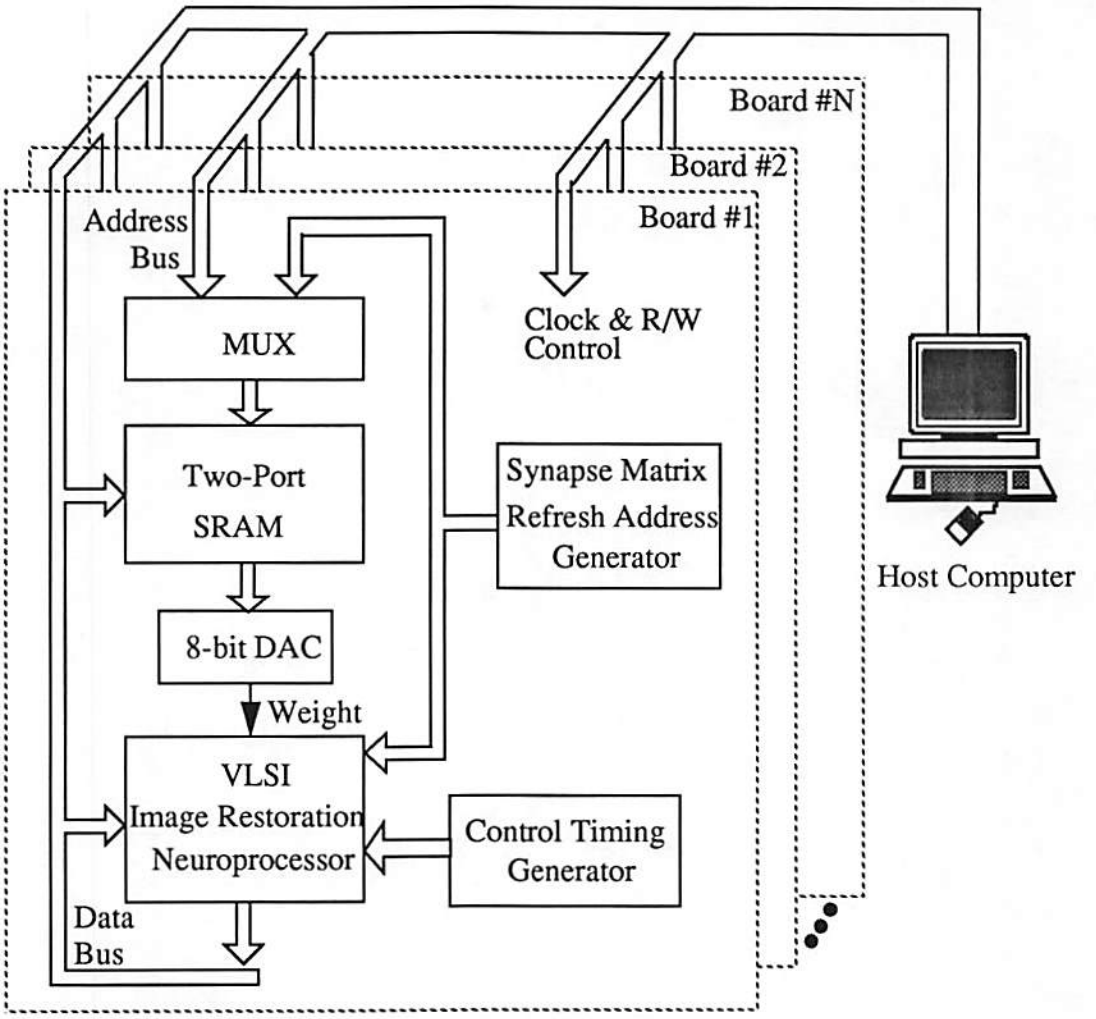


Fig. 4.4 System configuration for high-speed image restoration using multiple VLSI neural chips.

4.2.2 Detailed Circuit Design

Figure 4.5 shows the analog circuit realization of a neural model using the amplifier as neuron and resistors as synapses. The synapse weight $T_{i,j}$ is realized with a conventional resistor $R_{i,j}$. The dot products of input and weight are summed by taking advantage of Kirchoff's Current Law. After a current-to-voltage converter and voltage comparator, the neuron output can be described by

$$V_{oi} = A\left(\sum_{j=1}^{n} T_{i,j}V_j - V_\theta\right), \tag{4.23}$$

where V_θ is the neuron threshold voltage, and A is the voltage gain of the amplifier. Many programmable synapse cells and neurons have been reported [19-26].

In the prototype neural chip design, the synapse circuit in Fig. 4.6 is used. A transconductance amplifier consisting of transistors M_1-M_5 produces synapse output current $I^s_{i,j}$ according to neuron input voltage V_j and weight voltage $V^s_{i,j}$. The input voltage V_j steers the dynamic-range control current I^{max} in the input buffer cell. The transformed bias voltage is fanned out to multiple synapse cells. The maximum synapse conductance is decided by device sizes of the differential pair and the control current I^{max}, while the minimum synapse conductance is determined by the resolution of the weight value on the MOS capacitance.

The output neuron contains a current-to-voltage converter and a voltage comparator. It is realized by using two operational amplifier and a feedback resistor. Circuit schematic diagram of the operational amplifier is shown in Fig.

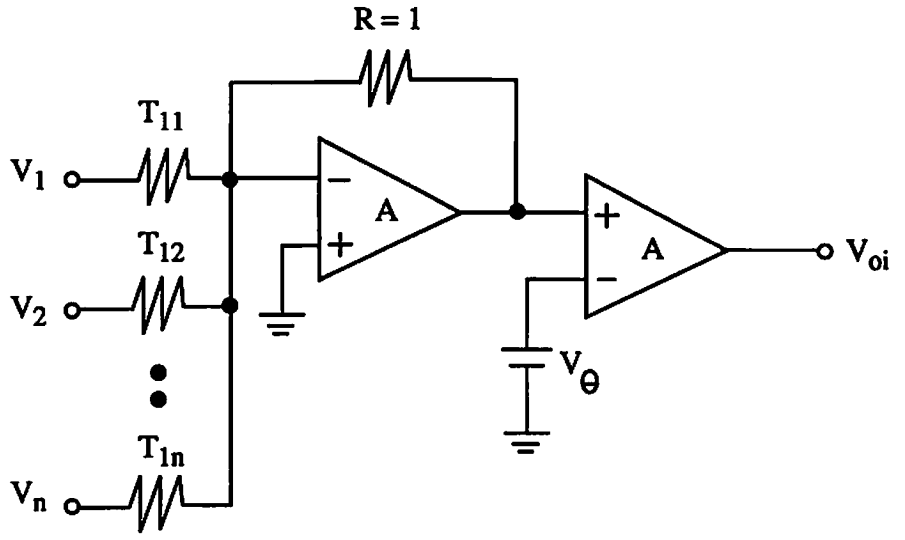
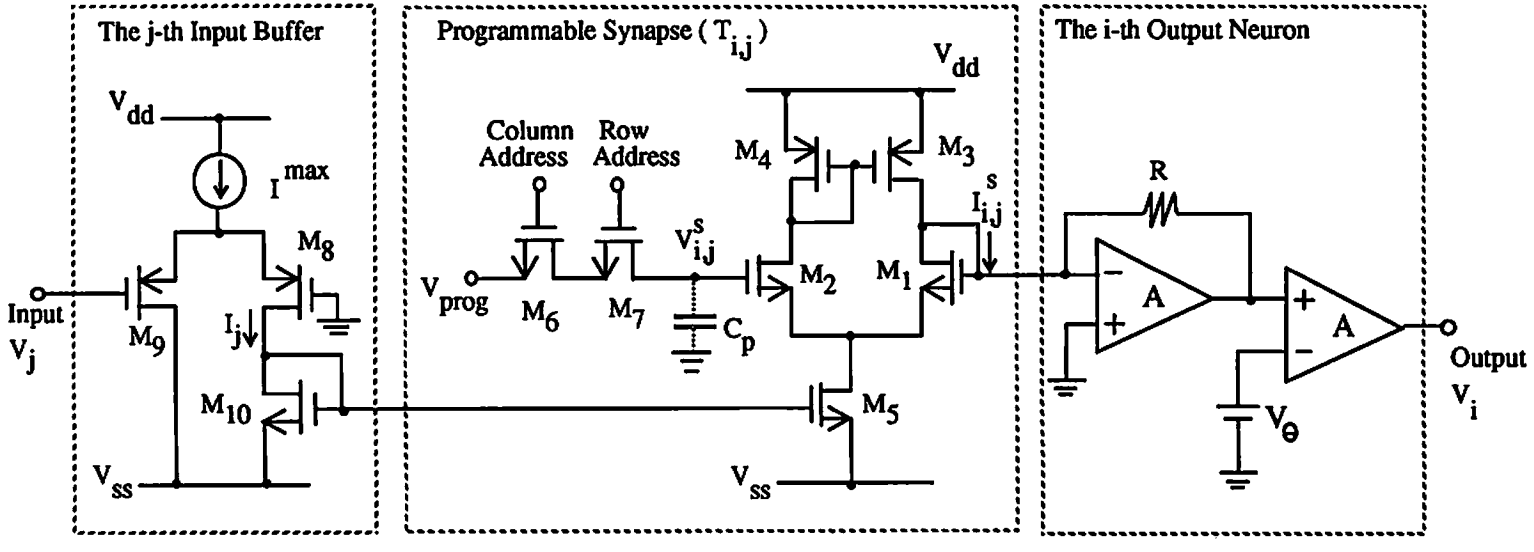
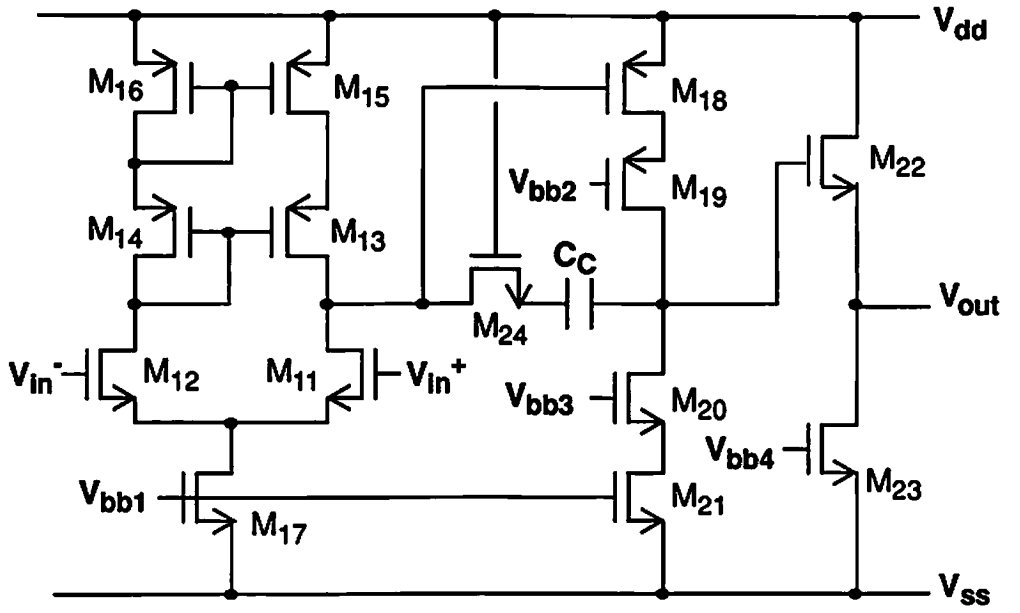


Fig.4.5 Analog circuit implementation with amplifiers as neurons and synthesized resistors as synapses.



Transistor	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	R
Size in um	3/39	3/39	4/4	4/4	5/3	4/2	4/2	5/3	5/3	5/3	5 K

Fig. 4.6 Circuit schematics of input buffer, programmable synapse, and output neuron.



Transistor	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
Size in um	16/2	16/2	6/2	6/2	20/4	20/4	40/4	46/4
Transistor	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}	C_C	
Size in um	90/2	26/2	52/4	600/2	400/4	50/2	1.5pF	

Fig.4.7 Circuit schematic diagram of the operational amplifier.

4.7. Transistors M_{13} and M_{14} form an improved cascode stage to increase the voltage gain and M_{24} operates as a resistor for proper frequency compensation.

For the image restoration, a half multiplier covering the first and the third quadrant is sufficient. The synapse weights can be stored either in the dynamic capacitors [21,23] or the EEPROM devices [24-26]. A typical 8-bit resolution can be achieved in the DRAM-style synapse cell. In the EEPROM-style synapse cell, a 6-bit resolution can be obtained. To implement the image restoration algorithm described in section 4.1, the required resolution of the synapse cell can be calculated in the following way. In (4.15), for a uniform, 3×3 blur function, the ratio of the maximum weight to the minimum weight is 9. In addition, there are 7 bits required for the scaling factor from the pixel register. Therefore, to restore an image with 8-bit gray-level, it requires at least 11-bit resolution for the synapse cell. By use of the DRAM-type synapse cell in Fig. 4.6, the image with 4-bit gray level will be processed in the prototype hardware implementation.

Figure 4.8 shows the digital circuitry for the pixel register, and increment/decrement logic gates. Eight cells are needed to construct the pixel register of the upgraded shift-register chain in each pixel register set. The increment/decrement function is implemented with a modified Manchester carry chain design [27]. The digital circuitry is synchronized by an external clock. Each network iteration cycle requires 185 $n$sec. When the CALC signal arrives, the neuroprocessors enter the retrieving process and the pixel-register contents start to change according to the neuron output values. When the CALC signal is turned off, the pixel registers function as shift registers to transmit the data.

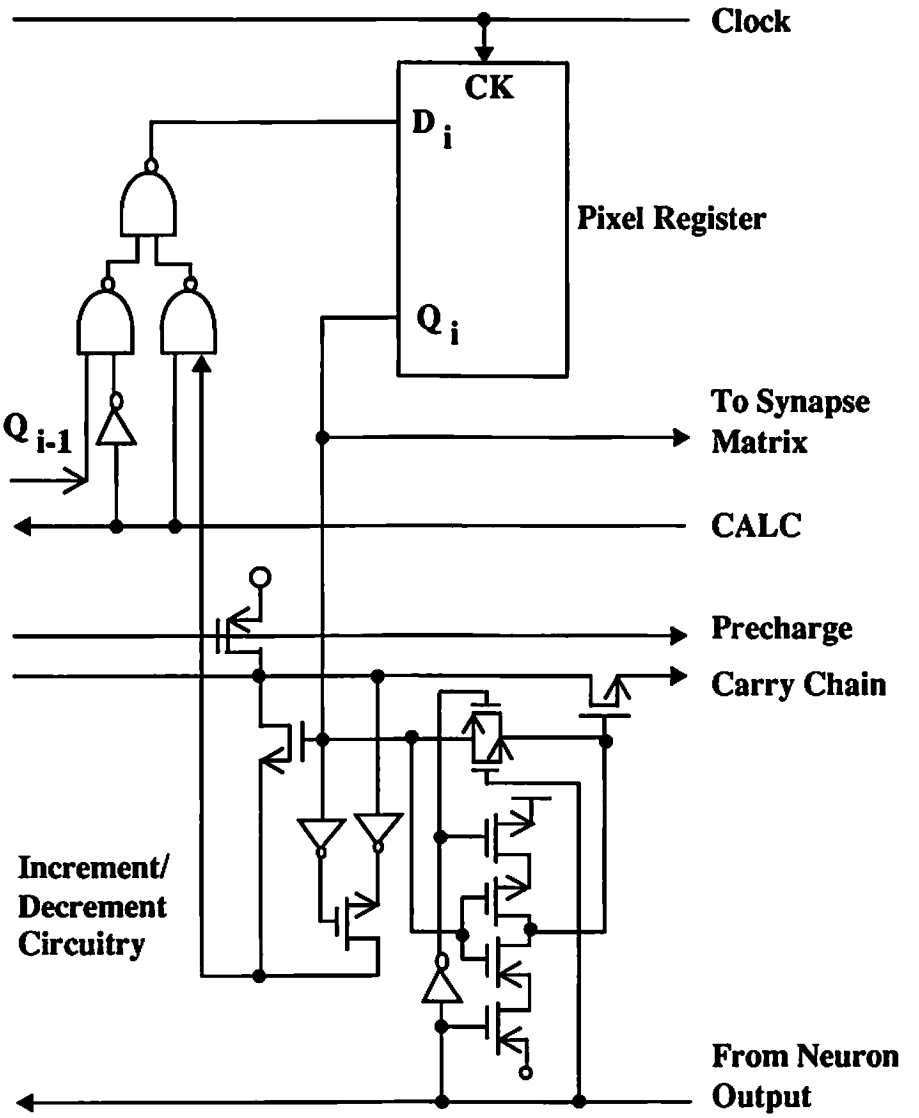


Fig. 4.8 One-bit pixel register and increment/decrement circuitry.

A programmable counter to control the iteration number of the network is also included in the chip.

4.3. Experimental Results

4.3.1 Prototype Neural Chip

By using the MOSIS [28] 1.2-μm CMOS technology, 5 neuroprocessor modules which include 25 neurons, 4,000 synapse cells, and 200 four-bit pixel registers can be realized in a $8.0 \times 6.0\text{-}mm^2$ chip. The layout of the image restoration chip is shown in Fig. 4.9. This chip requires a 236-pin PGA package which enables serial loading and unloading, of 25 rows of four-bit image data. A total number of 436 pins is required for the eight-bit pixel operation. The high-pin count could be avoided by using wafer scale integration approach [29]. To enhance the fault tolerance of the chip, a tri-level redundancy structure can be used. The first redundancy level disconnected one of five neuroprocessors. The interconnection between the neuroprocessor modules can be reconfigurated through programmable switching circuitry. The extra neuron columns and synapse rows can provide the second redundancy level and the third redundancy level, respectively. In prototype neural chip design, each neuroprocessor contains 5 neurons and 800 synapses. 24 rows of synapses are used to increase the resolution of synapse weights coming from the bias inputs and also to enhance the fault tolerance of the network.

To obtain the electrical properties of basic circuit blocks, a test chip containing 18 neurons and 30×18 synapse matrix has been fabricated and tested.

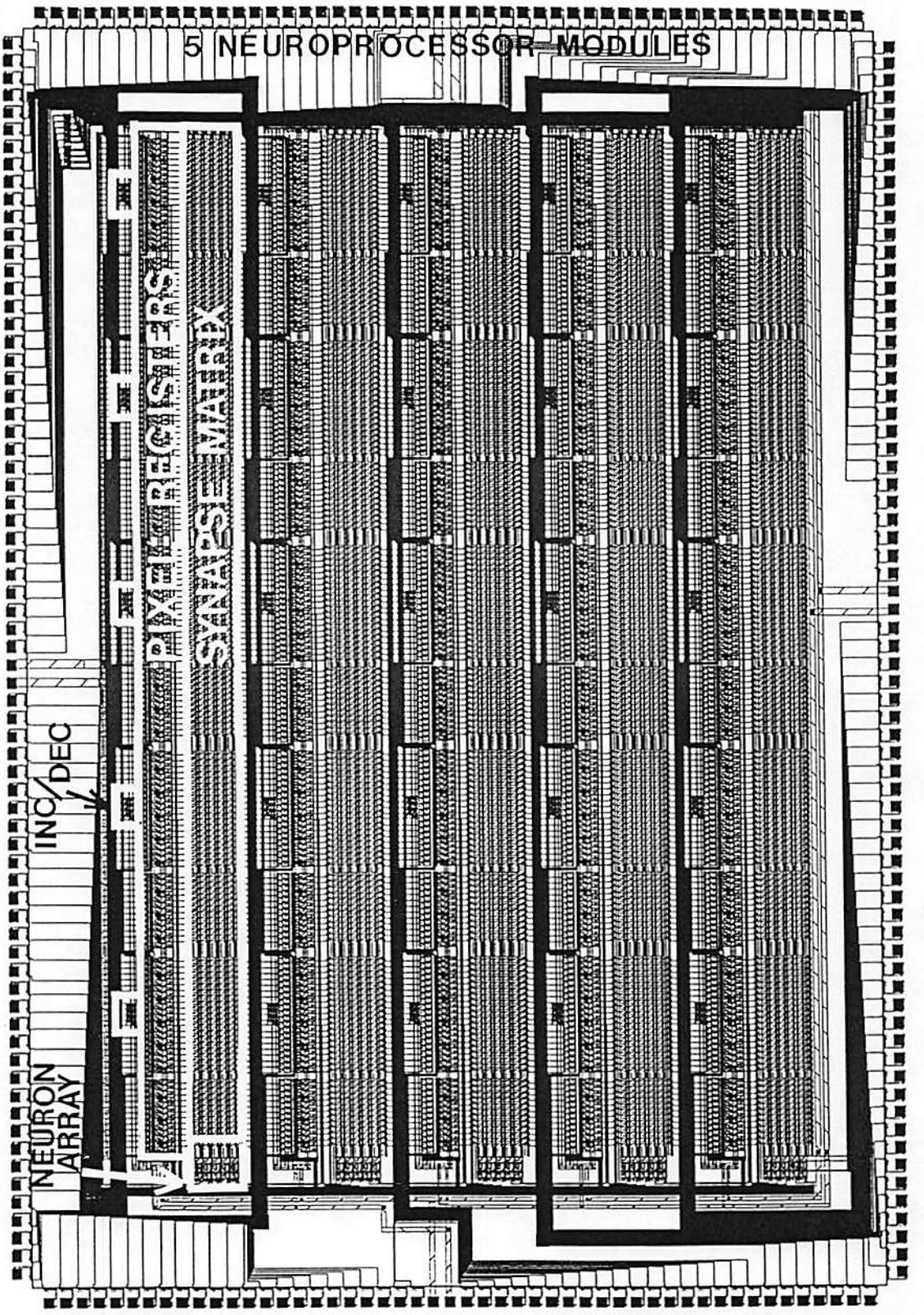


Fig. 4.9 Layout of the VLSI image restoration neural chip with 1.2-um CMOS technologies.

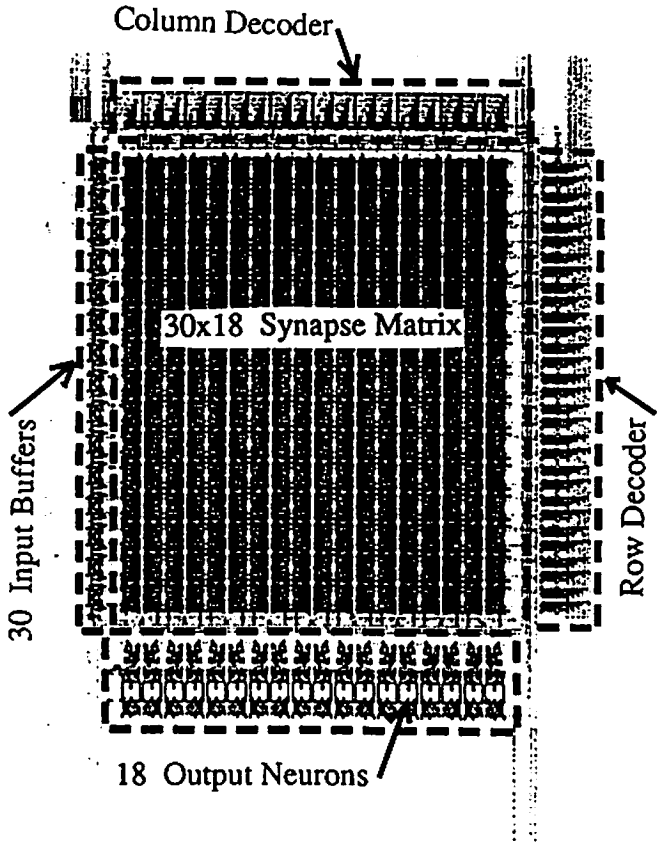


Fig. 4.10 Layout of the test module.

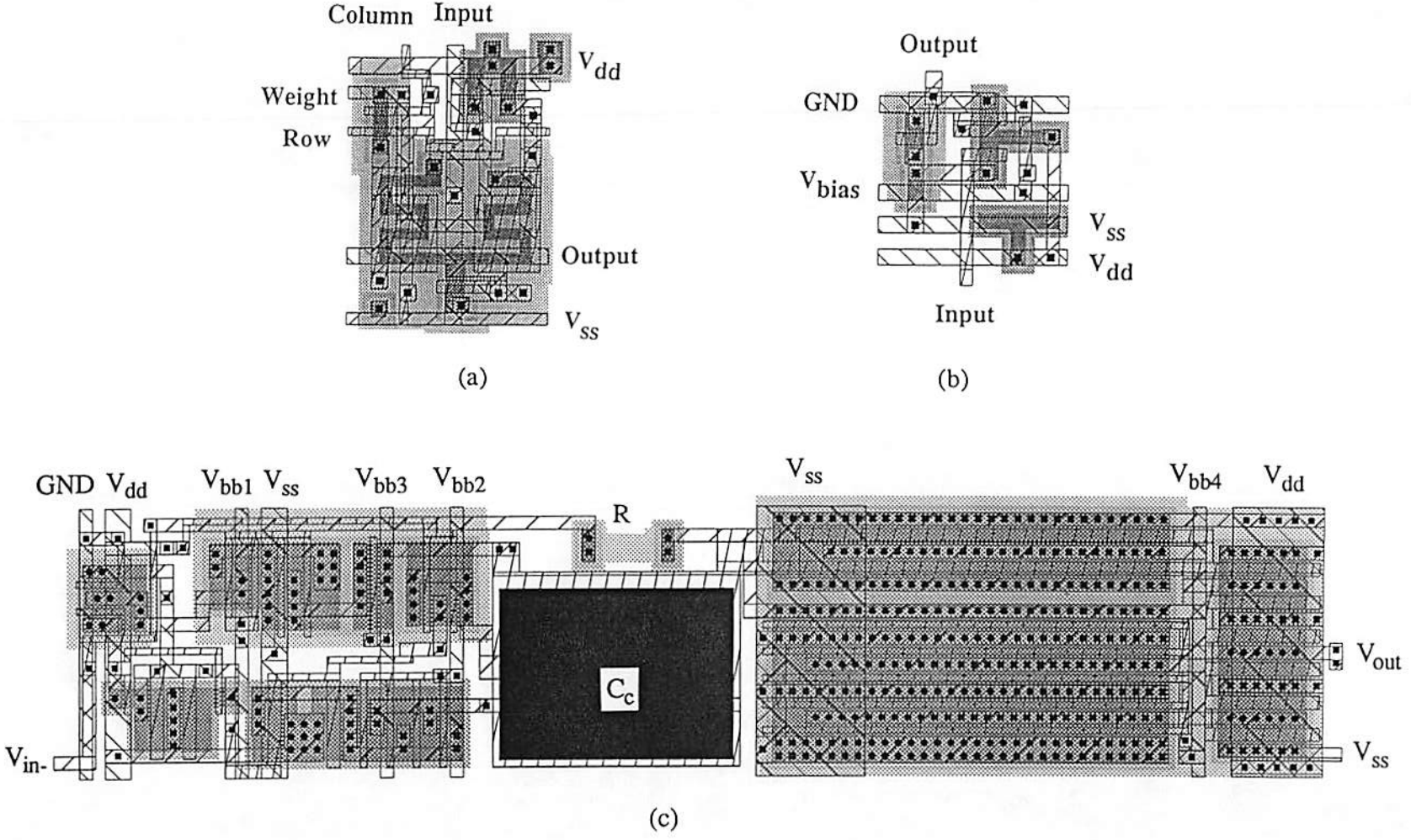
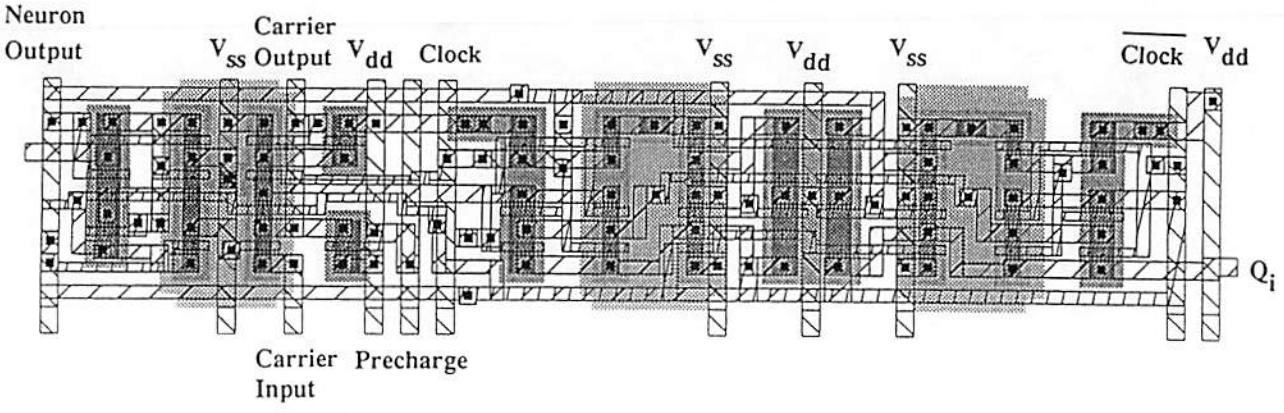


Fig. 4.11 Detailed layouts of (a) Synapse cell, (b) Input buffer, (c) Output neuron.



(d)

Fig. 4.11(d) Detailed layout of digital circuit.

The layout of the test chip is shown in Fig. 4.10. The detailed layout for the synapse cell, the input buffer, the output neuron, and the digital circuit are shown in Fig. 4.11. The transfer curves of the synapse cell with different bias voltages are shown in Fig. 4.12. The dynamic range varies with the bias voltage. The desired bias voltage is selected according to the resolution requirement of the weights. In charge retention experiment, the RC time constant at room temperature is about 50 sec as shown in Fig. 4.13. A refreshing cycle of 0.2 sec is adequate for 8-bit accuracy in synapse weights.

The processing speed for one network iteration is around 185 n sec. Each iteration cycle includes synapse multiplication, neuron thresholding, digital increment/decrement, pixel register updating and input buffer driving. Measurement results are listed in Table 4.1. The major delay comes from the synapse multiplication due to the significant capacitance loading from the current-summation line. The total 2.16×10^{10} connections per second can be achieved by using one VLSI neural chip containing 5 neuroprocessors, 4,000 synapses cells, and operated at 5.4 MHz. Using the results in Table 4.1, the speed comparisons of the 5-neuroprocessor chip with a Sun-4/75 SPARC workstation is listed in Table 4.2 for image restoration case. The speedup factor is 475 for the chip to be operated at 5.4 MHz master clock.

4.3.2 System-Level Analysis

System-level analysis at 8-bit resolution has been conducted to demonstrate the performance of the image restoration chip. The mismatch effect of analog synapse components has been included. The statistical distribution of synapse

Table 4.1 Circuit Response Time

Circuit Function	Measured Results
Synapse Multiplication	130 ns
Neuron Threshoding	25 ns
Digital Inc/Dec	10 ns
Pixel Register Update	8 ns
Input Buffer	8 ns

Note: T_{ox}= 202 Å in MOSIS 1.2-um CMOS technology.

Table 4.2 Performance of VLSI Image Restoration System *

Synapse weight loading time (into SRAM)	1.448 ms
Pixel register read/write time	0.556 ms
Network execution time for 30 iterations	16.68 ms
Total processing time for 256x256-pixel image	18.684 ms
Speed-up factor+	475

Note: * One VLSI neural chip with 5 neuroprocessors is used.

\+ The reference is software computation on Sun-4/75 SPARC-2 workstation.

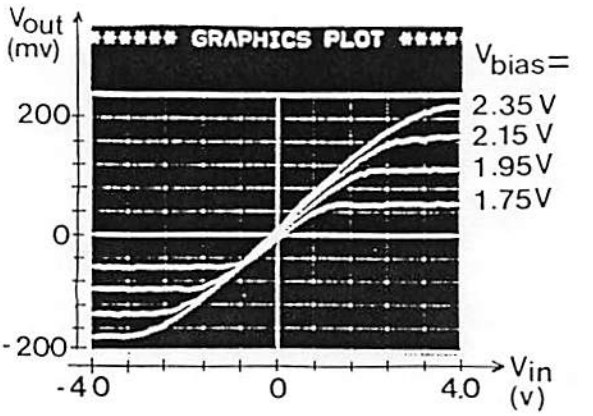


Fig. 4.12 Measured results of programmable synapse characteristics.

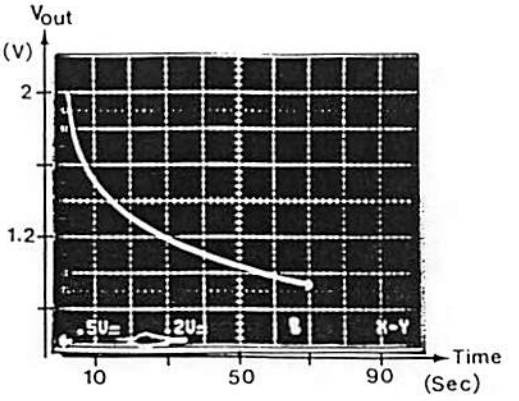
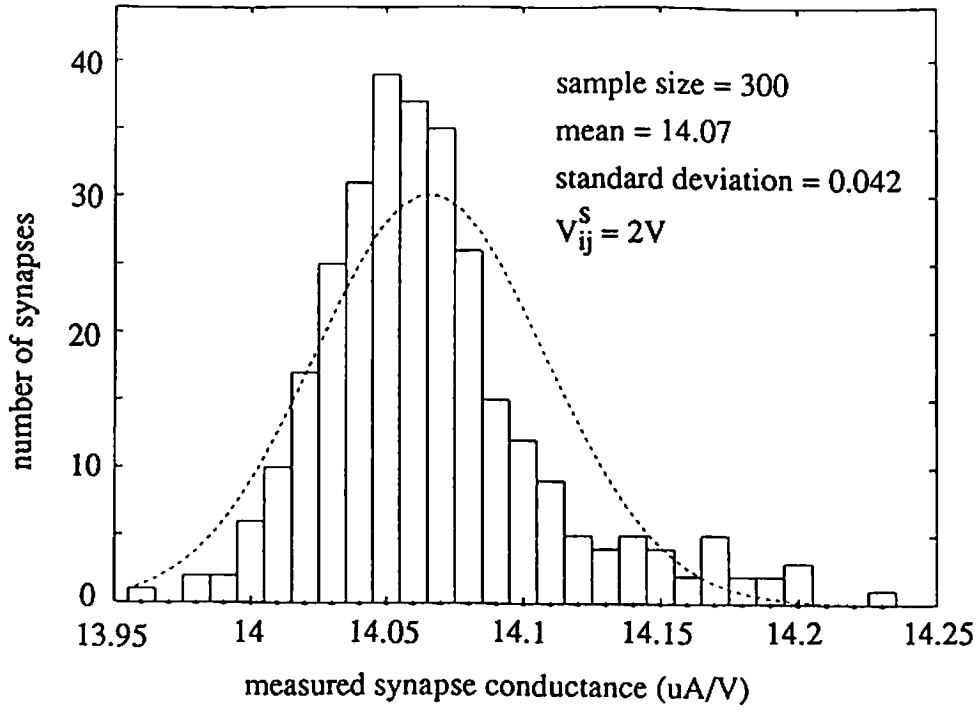
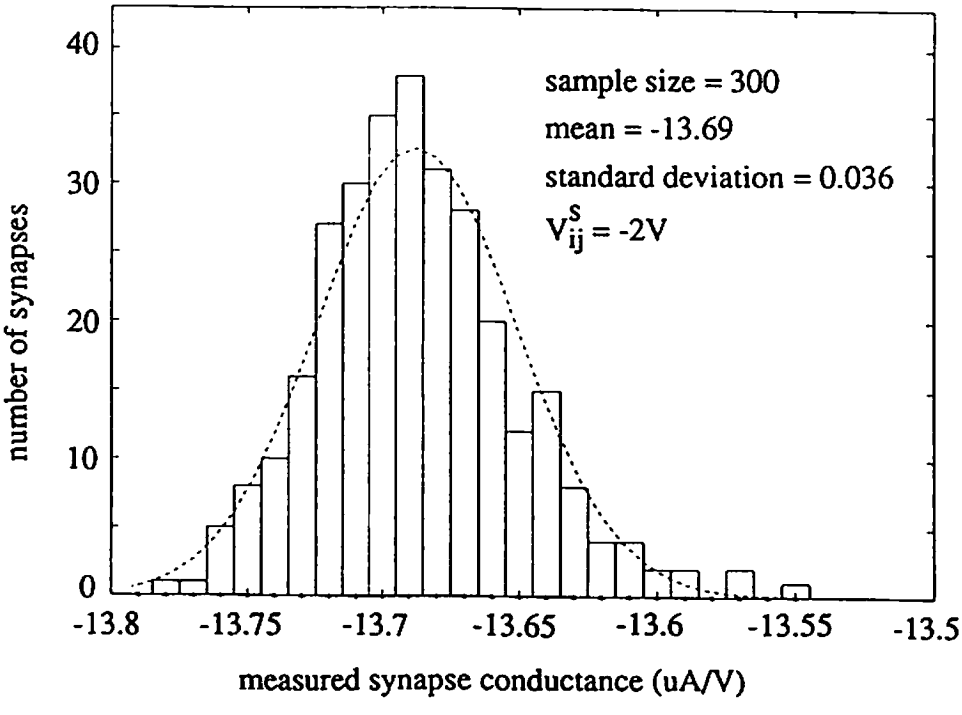


Fig. 4.13 Measured result of charge retention characteristic for the DRAM-type synapse cell.



(a)



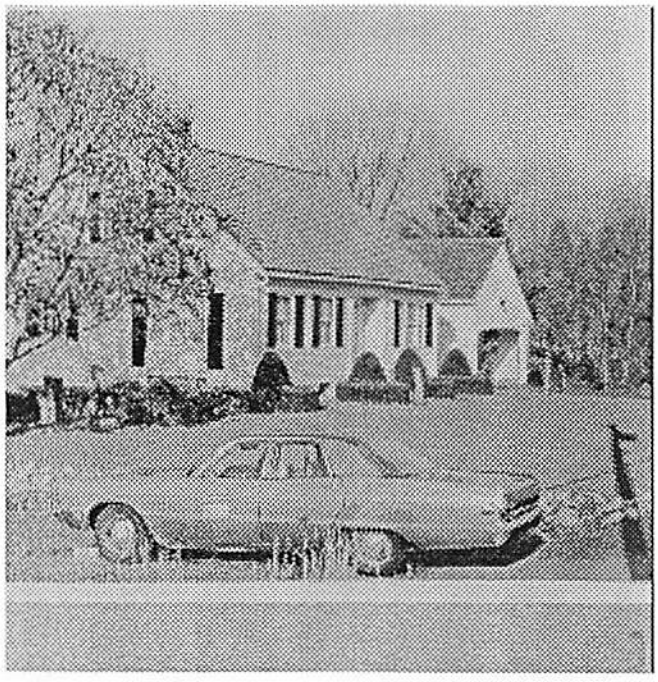
(b)

Fig. 4.14 The statistical distribution of measured synapse output conductances.

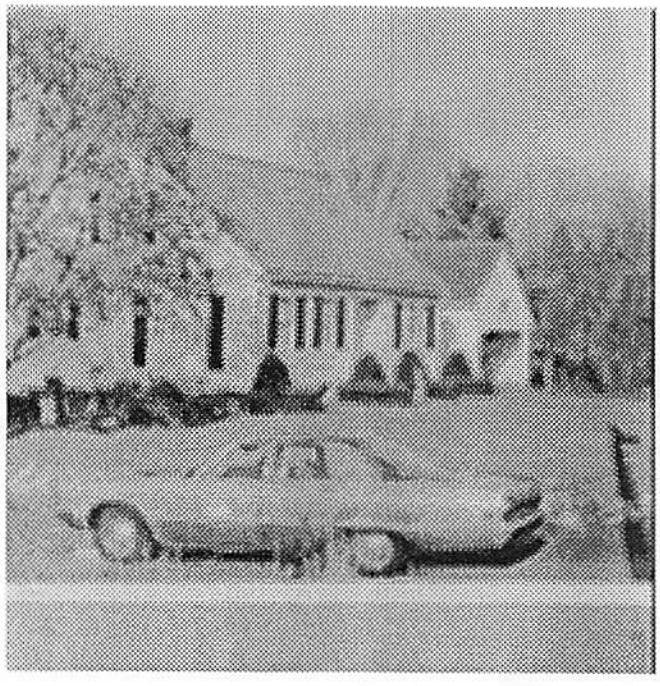
output conductances due to process variation can be described by a Gaussian function. A total of 300 synapses in test chip was measured. In Fig. 4.14(a), the synapse conductances can be described by a Gaussian distribution with a mean value of 14.07 $\mu A/V$ and a standard deviation of 0.042 $\mu A/V$ when the weight voltage $V_{i,j}^{s} = 2\ V$. In Fig. 4.14(b), the synapse conductances can be described by a Gaussian distribution with a mean value of -13.69 $\mu A/V$ and a standard deviation of 0.036 $\mu A/V$ when the weight voltage $V_{i,j}^{s} = -2\ V$.

A house image degraded by a 3×3 uniform space-invariant blur function was used as input image data during the computer analysis. The image size is of 256×256 pixels. The original undegraded image and blurred image are shown in Figs. 4.15(a) and 4.15(b), respectively. Two sets of weight values have been used to restore the image. In Fig. 4.15(c), the restored image using ideal weight values is obtained after 35 iterations. In Fig. 4.15(d), the restored image using weight values with the effect of device mismatch is obtained after 35 iterations. The ringing effects due to boundary problems of the whole image can be easily eliminated by replacing the first and last four columns and rows from the blurred image and updating the interior region only. The energy function defined as the mean square error (MSE) between the restored image and original undegraded image decreases after each network iteration. The energy curves for image restoration with and without considering the process variations are shown in Fig. 4.15(e). Generally speaking, the degraded image can be successfully restored by the neural chip with including the nonidealities of analog components.

An application of this image restoration network to enhance the quality of image edge detection operation is shown in Figs. 4.16 and 4.17. Figures 4.16(a) and 4.16(b) show the raw image directly produced by a Sony XC-77 CCD camera and the corresponding edge-detection result using Sobel operator. Figures 4.17(a) and 4.17(b) show the restored image and the corresponding edge-detection result after 30 iterations. A significant improvement in the edge image can be observed. Restoration was done using a 3×3, Gaussian, space invariant blur function. In general, a Gaussian blur function is suitable for this specific degradation in Fig. 4.16(a).

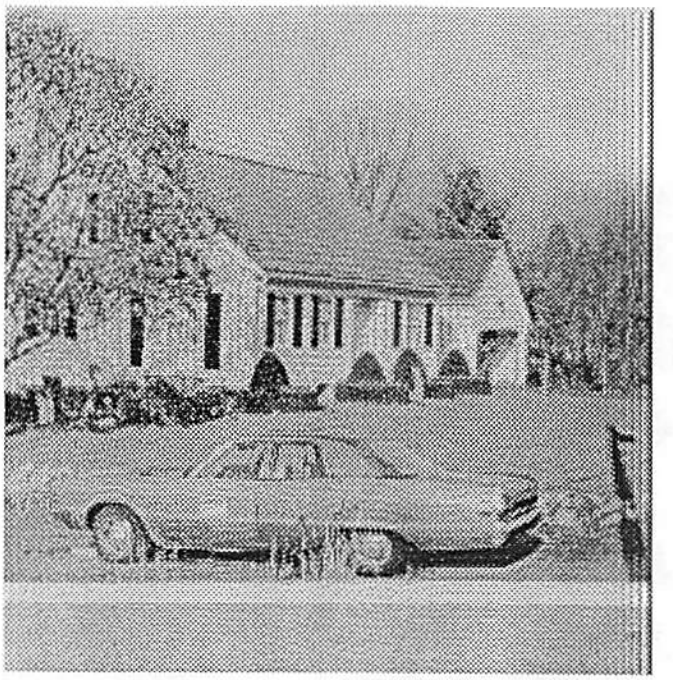


(a)

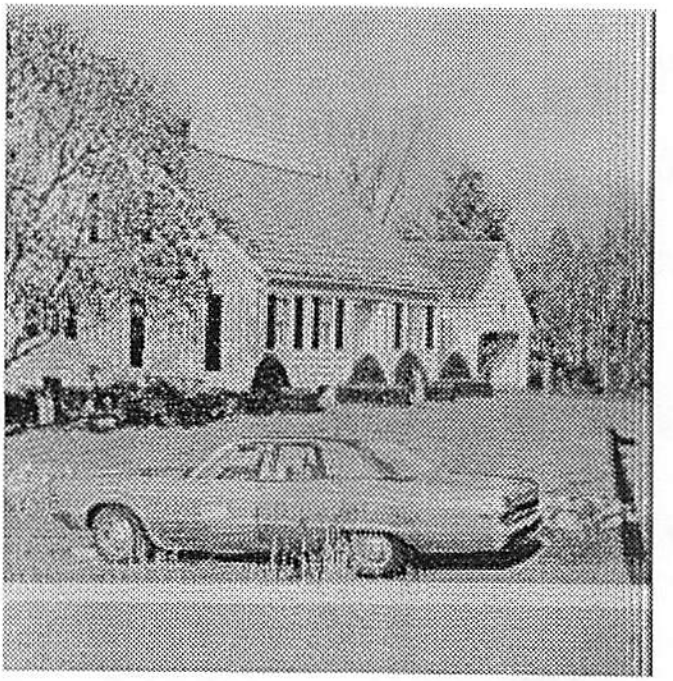


(b)

Fig. 4.15 System-level analysis on the house image.
(a) Original undegraded image.
(b) Image blurred by a 3x3 uniform space-invariant function.

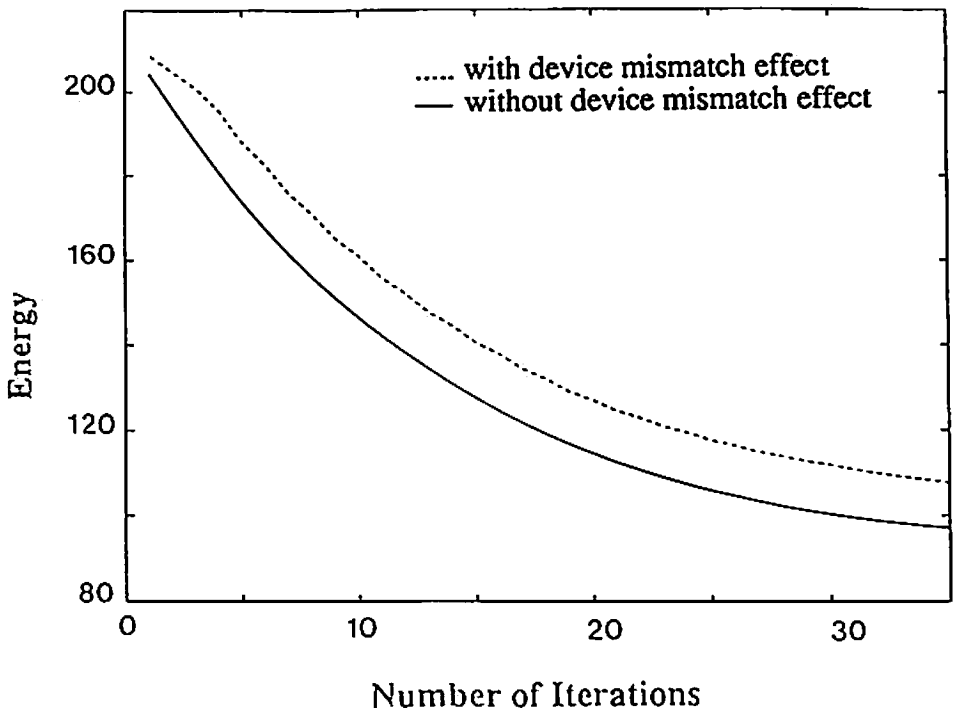


(c)



(d)

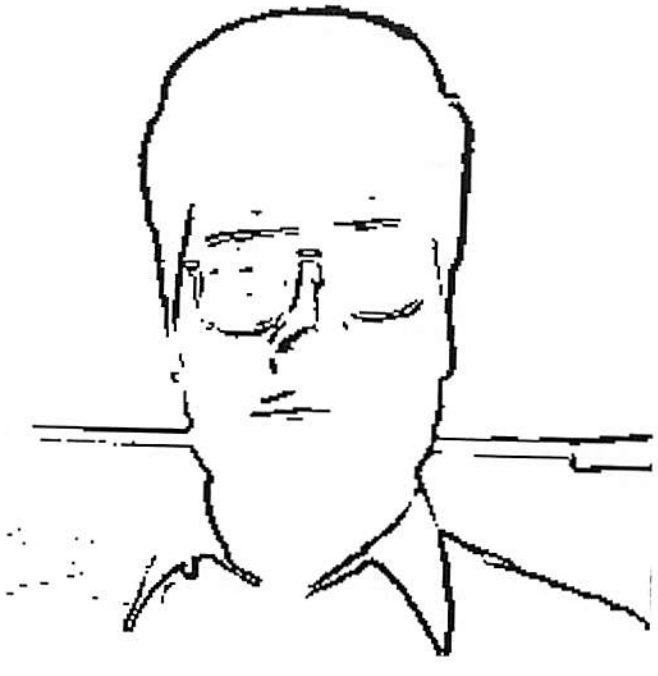
Fig. 4.15 System-level analysis on the house image.
(c) Restored image without device mismatch effect.
(d) Restored Image with device mismatch effect.



(e)

Fig. 4.15 System-level analysis on the house image.
(e) Energy function.

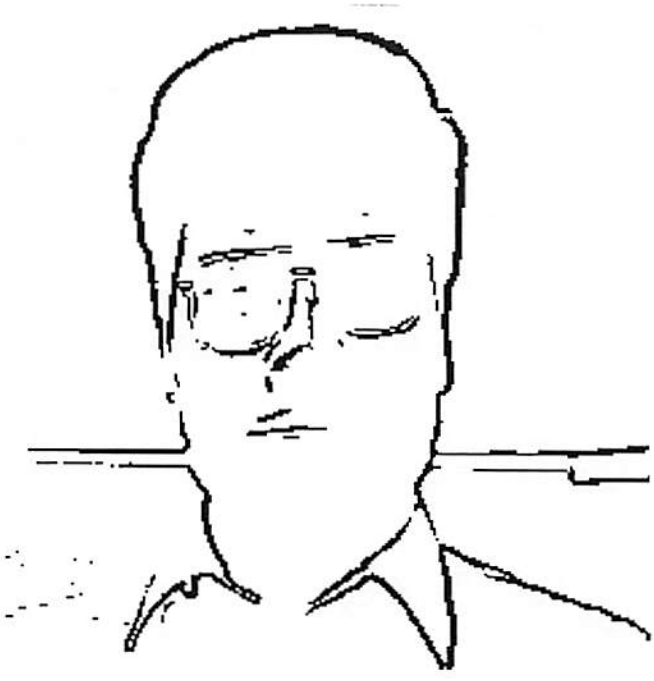
(a)



(b)

Fig. 4.17 Restoration result of Fig. 4.16 for a 3x3 Gaussian blur function.
(a) Restored image. (b) Corresponding edge detection result.

(a)



(b)

Fig. 4.16 Image obtained from a Sony XC-77 CCD camera
(a) Original image. (b) Edge detection result.

References

[1] H. Andrews, B. Hunt, *Digital Image Restoration*, Prentice Hall: Englewood Cliffs, NJ, 1977.

[2] J. Woods, V. Ingle, "Kalman filtering in two dimensions: Further results," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. ASSP-29, pp. 188-197, April 1981.

[3] R. Lippman, "An introduction to computing with neural nets," *IEEE Acoustics, Speech, and Signal Processing Magazine*, pp. 4-22, April 1987.

[4] J.-C. Lee, B. J. Sheu, "Parallel digital image restoration using adaptive VLSI neural chips," *IEEE Proc. of Intl. Conf. on Computer Design*, pp. 126-129, Cambridge, MA, Sept. 1990.

[5] J.-C. Lee, B. J. Sheu, "Analog VLSI Neuroprocessors for early vision processing," in *VLSI Signal Processing IV*, editors, H. Moscovitz, K. Yao, R. Jain, IEEE Press: Piscataway, NJ, 1991.

[6] W. Pratt, *Digital Image Processing*, John Wiley & Sons: New York, NY, 1978.

[7] T. Poggio, V. Torre, C. Koch, "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314-319, Sept., 1985.

[8] T. Poggio, C. Koch, "Ill-posed problems in early vision: from computational theory to analog networks," *Proc. Royal Soc. of London*, vol. B-226, pp. 303-323, 1985.

[9] R. Gonzalez, P. Wintz, *Digital Image Processing*, Addison-Wesley Publishing Company: Reading, MA, 1987.

[10] A. Jain, *Fundamentals of Digital Image Processing*, Prentice Hall: Englewood Cliffs, NJ, 1989.

[11] H. Sorenson, *Parameter Estimation: Principle and Problems*, Dekker, 1980.

[12] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671-680, 1983.

[13] S. Geman, D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721-741, Nov. 1984.

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

[15] J. Hopfield, "Neural networks and physical system with emergent collective computational abilities," *Proc. Nat. Acad. Sci. USA*, vol. 79, pp. 2554-2558, Apr. 1982.

[16] Y. Zhou, R. Chellappa, A. Vaid, B. Jenkins, "Image restoration using a neural network", *IEEE Trans. Acoustics, Speech and Signal Processing*, vol. ASSP-36, pp. 1141-1151, July 1988.

[17] M. Takeda, "Neural networks for computation: number representations and programming complexity," *Appl. Opt.*, vol. 25, pp. 3033-3046, Sept. 1986.

[18] J. paik, "Image restoration using the Hopfield network with nonzero autoconnection," IEEE Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing, pp. 1909-1912, Albuquerque, NM, 1990.

[19] B. W. Lee, B. J. Sheu, *Hardware Annealing in Analog VLSI Neurocomputing*, Kluwer Academic Publishers: Boston, MA, Dec. 1990.

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

[21] B. W. Lee, B. J. Sheu, "A compact and general purpose neural chip with electrically programmable synapses," *Proc. of IEEE Custom Integrated Circuits Conf.*, pp. 26.6.1-26.6.4, Boston, MA, May 1990.

[22] P. Hollis, J. Paulos, "Artificial neural networks using MOS analog multipliers," *IEEE Jour. of Solid-State Circuits*, vol. 25, no. 3, pp. 849-855, June, 1990.

[23] T. Morishita, Y. Tamura, T. Otsuki, "A BiCMOS analog neural network with dynamically updated weights, " *IEEE Proc. of Integrated Solid-State Circuits Conf.*, pp. 142-143, San Francisco, Feb. 1990.

[24] S. Bibyk, M. Ismail, "Issues in analog VLSI and MOS techniques for neural computing," in *Analog VLSI Implementation of Neural Systems*, editors, C. Mead and M. Ismail, Kluwer Academic Publishers: Boston, MA, 1989.

[25] M. Holler, S. Tam, H. Castro, R. Benson, "An electrically trainable artificial neural network (ETANN) with 10240 "floating gate" synapses," *Proc. of IEEE Int. Joint Conf. on Neural Networks*, vol. 2, pp. 191-196, Washington D.C. June 1989.

[26] B. W. Lee, B. J. Sheu, H. Yang, "Analog floating-gate synapses for general-purpose VLSI neural computing," *IEEE Trans. on Circuits and Systems*, vol. 38, no. 6, June 1991.

[27] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Chap. 8, pp. 322-326, Addison-Wesley Publishing Company: Reading, MA, 1984.

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

[29] Earl E. Swartzlander, *Wafer Scale Integration*, Kluwer Academic Publishers: Boston, MA, 1989.

Chapter 5

Conclusion

The processing of video signals often requires a tremendous computational capability which can only be achieved by using special parallel processing architectures. The inherent massive parallelism of artificial neural network architecture provides a new paradigm of video signal processing. In addition, rapid advances on VLSI technologies have resulted in orders of magnitude reduction in the hardware size, power consumption, and cost of many video systems and have made real-time video applications practical. A large number of VLSI neural chips are beginning to appear on the market. These neural chips can perform specific tasks in the areas of vision processing, speech recognition, and robotics control. In this dissertation, two specific VLSI neural chips for early vision processing have been developed.

In Chapter 3, a mixed-signal two-dimensional mesh-connected architecture for high-speed video motion estimation has been presented. A compact and efficient analog neuroprocessor which includes 25 neurons and 25×27 synapse cells is able to estimate the motion of each pixel with 25 different velocities. Multiple neuroprocessors can be connected as a two-dimensional array to fully exploit the massively parallel computational power of neural networks. In this architecture, the local computation is processed in analog neuroprocessor and the local data communication between the neuroprocessors is performed in parallel. A VLSI neural chip which occupies $1.5 \times 2.8\text{-}cm^2$ silicon area from a 1.2-μm CMOS technology can accommodate 64 neuroprocessors and operate at a

sustained rate of 83.2 Giga connections per second. The speed-up factor for a system of such 128 VLSI neural chips over the Sun-4/75 SPARC-2 workstation is 24,242.

In Chapter 4, an analog systolic architecture for efficient VLSI neurocomputing has been presented. In this architecture, the image data are processed in parallel in the row direction and in the pipelined fashion in the column direction. A compact and efficient VLSI neural chip which includes five neuroprocessors for high-speed digital image restoration has been constructed. The mixed-signal design technique uses analog circuits for local neurocomputation and digital circuits for decision-making functions and inter-processor communication. The multiprocessor chip allows the exploitation of massively parallel processing, programmability, and scalability of VLSI neurocomputing. This chip operates at a sustained rate of 21.6 Giga connections per second. The speed factor for a chip with 5-neuroprocessor over the Sun-4/75 SPARC-2 workstation is 475.

These VLSI neural chips, usually optimized for specific tasks, can be combined in a complete system architecture to exploit the collective attributes of these chips. Therefore the objective of this research is to develop neurocomputer architectures, incorporating advances in neural network research and chip technology, and to achieve supercomputer performance at a fraction of the size, power and cost of conventional supercomputers. One good application example has been shown in Fig. 1.1. In order to perform such research on the design and construction of a high performance integrated information system, the research will be interdisciplinary, with issues involving brain cortical structures, neural network architectures, VLSI design and fabrication, parallel systems

design, signal processing and system integration.

More studies on the data communication scheme to enhance the performance of a multiprocessor-based system are necessary. Wafer scale integration can be employed to accommodate multiple modules of VLSI neuroprocessors to avoid data communication bottleneck. With integration of different technologies, the sensing devices and processing elements can be directly coupled to alleviate the data communication problem and achieve full advantage of parallel architectures.

Appendix A

Program for Neural-Based Motion Estimation

To compile, please type "f77 -o of1_mis of1_mis.f libsipi.a libsplot.a ". To run the program, please type "of1_mis < of1_mis.dat > log_mis".

```
c ---------------------------------------------------------------------------
c for mis image (i.e. mobile missile launcher images).
c line process is included.
c a set of 4 successive image frames are used.
c device mismatch effect is included and modeled as a Gaussian function.
c---------------------------------------------------------------------------
      program of1_mis4
      dimension ix (160), xh3 (80, 113), xh4 (80, 113)
      dimmension bias (240, 904), iboun (80, 113)
      dimension xh1 (80, 113), xh2 (80, 113), itp3 (15, 15)
      dimension gsub (80, 113), gk1 (80, 113), gk2 (80, 113)
      dimension gsub3 (80, 113), gk3 (80, 113), gk4 (80, 113)
      dimension gsub4 (80, 113), gk5 (80, 113), gk6 (80, 113)
      dimension xk1 (80, 113), xk2 (80, 113), ivold (80, 113)
      dimension ihnun (80, 113), ivnun (80, 113), ihold (80, 113)
      dimension lvk1 (80, 113), lvl1 (80, 113)
      dimension lvk2 (80, 113), lvl2 (80, 113)
      dimension lhk1 (80, 113), lhl1 (80, 113)
      dimension lhk2 (80, 113), lhl2 (80, 113)
      common/neuron/ihold, ivold
      common/vline/lvk1, lvl1, lvk2, lvl2
      common/hline/lhk1, lhl1, lhk2, lhl2
      common/sig/xh1, xh2, xh3, xh4
      common/subdv/gsub, gk1, gk2
      common/subdv3/gsub3, gk3, gk4
      common/subdv4/gsub4, gk5, gk6
      common/xdv/xk1, xk2
      equivalence (xh3 (1, 1), ihnun (1, 1))
      equivalence (xh4 (1, 1), ivnun (1, 1))
      character*12 hinput, vinput, hfile, vfile
      character*12 firtimg, secdimg, thidimg, fourimg, outfile, boundfile
c---------------------------------------------------------------------------
      n=113
      nll=160
      nend=35
      len=80
      ixd=15
      snn=n*len
      idisp=7
      print 10
10    format (1x, 'input the standard deviation')
```

```
      read*, wcc
      print 11
11    format (1x,'input the first image file name')
      read*, firtimg
      print 12
12    format (1x,'input the second image file name')
      read*, secdimg
      print 13
13    format (1x,'input the third image file name')
      read*, thidimg
      print 14
14    format (1x,'input the fourth image file name')
      read*, fourimg
      print 15
15    format (1x,'use initial velocity files or not (1:yes; 0:no)')
      read*, inornot
      print 16
16    format (1x,'input initial vertical velocity file name')
      read*, vinput
      print 17
17    format (1x,'input initial horizontal velocity file name')
      read*, hinput
      print 18
18    format (1x,'output file name')
      read*, outfile
      print 19
19    format (1x,'need boundary file (1:yes; 0:no)')
      read*, iynbnd
      print 20
20    format (1x,'boundary file name')
      read*, boundfile
      print 21
21    format (1x, 'output file name for horizontal velocity data')
      read*, hfile
      print 22
22    format (1x, 'output file name for vertical velocity data')
      read*, vfile
      print 23
23    format (1x,'window size? (5)')
      read *, iwd
      print 24
24    format (1x, 'order of polynomials? (3,5)')
      read *, iord
      print 25
25    format (1x, 'number of sample points? (1)')
      read *, idv
      print 26
26    format (1x, 'parameter A?')
```

```
          read*, acoef
          print 27
27        format (1x, 'parameter B for k1 ?')
          read*, bcoef
          print 28
28        format (1x, 'parameter C for k2 ?')
          read*, ccoef
          print 29
29        format (1x, 'parameter D for intensity?')
          read*, dcoef
          print 30
30        format (1x, 'parameter E for line process (first) ?')
          read*, ecoef
          print 31
31        format (1x, 'parameter F for line process (second) ?')
          read*, fcoef
          print 32
32        format (1x, 'plot lines or not (1:yes: 0:no)')
          read*, lyn
          print 33
33        format (1x, 'Iterations?')
          read*, itera
c-------------------------------------------------------------------------------
          sp=1.0
          isearch=idisp
          isey=1
          isch1=isearch+1
          ivshd=isch1-isey
          ivsed=isch1+ise
          ihshd=isch1
          ihsed=isch1+isch1-1
          ka48=48*ifix(acoef)
          ka2=2*ifix(acoef)
          ke=ifix(ecoef)
          ke4=4*ke
          ke2=2*ke
          nline=ke*fcoef
          ncf=isch1
          nvf=isey*2+1
          nv=nvf*len
          nc=ncf*n
c-------------------------------------------------------------------------------
c         input image files
c-------------------------------------------------------------------------------
          read=0
          call dopen (firtimg, iread, 1, 4, nunit)
          if (nend.eq.0) go to 50
          do 51 i=1, nend
```

```
        call dread (nunit, ix, nll)
51      continue

50      do 60 i = 1, len
            call dread (nunit, ix, nll)
            do 61 j = 1, n
               xh1 (i,j) = float (ix (j + ixd))
61          continue
60      continue
        call dclose (nunit)

        call dopen (secdimg, iread, 1, 4, nunit)
        if (nend.eq.0) go to 70
        do 71 i = 1, nend
           call dread (nunit, ix, nll)
71      continue

70      do 72 i = 1, len
           call dread (nunit, ix, nll)
           do 73 j = 1, n
              xh2 (i, j) = float (ix (j + ixd) )
73         continue
72      continue
        call dclose (nunit)

        call dopen (thidimg, iread, 1, 4, nunit)
        if (nend.eq.0) go to 74
        do 75 i =1, nend
           call dread (nunit, ix, nll)
75      continue

74      do 80 i = 1, len
           call dread (nunit, ix, nll)
           do 81 j = 1, n
              xh3 (i,j) = float (ix (j + ixd) )
81         continue
80      continue
        call dclose (nunit)

        call dopen (fourimg, iread, 1, 4, nunit)
        if (nend.eq.0) go to 83
        do 84 i = 1, nend
           call dread (nunit, ix, nll)
84      continue

83      do 90 i = 1, len
           call dread (nunit, ix, nll)
           do 91 j=1,n
```

```
                xh4(i,j)=float(ix(j+ixd))
91          continue
90        continue
          call dclose (nunit)

92        if (iwd.lt.2) go to 100
          call wnd (iwd, n, idv, sp, iord, len)

100       do 101 i = 1, n
             do 101 j = 1, len
                ihnun (j, i) = isch1
                ivnun (j, i) = isch1
101       continue

          if (inornot.eq.0) go to
          open (34, file = hinput)
          do 111 i = 1, len
             read (34, 500) (ihnun (i, j), j = 1, n)
111       continue
          close (34)

          open (35, file = vinput)
          do 120 i = 1, len
             read (35, 500) (ivnun (i, j), j = 1, n)
120       continue
          close (35)

110       do 130 i = 1, n
             do 130 j = 1, len
                ihold (j, i) = ihnun (j, i)
                ivold (j, i) = ivnun (j, i)
130       continue
          do 140 i = 1, nc
             do 140 j = 1, nv
                bias (j, i) = 0.0
140       continue
          do 150 i = 1, len
             do 150 j = 1, n
                lvk1 (i, j) = isearch
                lvl1 (i, j) = isearch
                lvk2 (i, j) = isearch
                lvl2 (i, j) = isearch
                lhk1 (i, j) = isearch
                lhl1 (i, j) = isearch
                lhk2 (i, j) = isearch
                lhl2 (i, j) = isearch
150       continue
c-------------------------------------------------------------------------------
```

```
c          updat neuron states
c-------------------------------------------------------------------------------
           ibw=iwd
           if (iwd.lt.5) ibw = 5
           ihf = 3 * idisp + ibw/2
           ihs1 = 1 + ibw/2
           ihy = ibw/2
           ihy1 = ihy + 1
           ih1 = ihf + 1
           idvf1 = idv/2 + 1

           do 160 i = ihy1 + 1, len - ihy1
              ig = (i - 1) * nvf + 2
              do 161 j = ihs1 + 1, n - ihf
                 jg = (j - 1) * ncf + 1
                 do 162 k = -isey, isey, 1
                    ik = ig + k
                    ig11 = i + k
                    ig22 = ig11 + k
                    ig33 = ig22 + k
                    do 163 l = 0, isearch, 1
                       il = jg + l
                       lg11 = j + l
                       lg22 = lg11 + l
                       lg33 = lg22 + l
                  bias(ik,il)=dcoef*((xh1(i,j)-gsub(ig11,lg11))**2+
              $ (gsub(ig11,lg11)-gsub3(ig22,lg22))**2+(gsub3(ig22,lg22)
              $ -gsub4(ig33,lg33))**2)+
              $ bcoef*((xk1(i,j)-gk1(ig11,lg11))**2+
              $ (gk1(ig11,lg11)-gk3(ig22,lg22))**2+(gk3(ig22,lg22)-
              $ gk5(ig33,lg33))**2)
              $ +ccoef*((xk2(i,j)-gk2(ig11,lg11))**2
              $ +(gk2(ig11,lg11)-gk4(ig22,lg22))**2
              $ +(gk4(ig22,lg22)-gk6(ig33,lg33))**2)
163                 continue
162                 continue
161           continue

              do 164 j = n-ihf+1, n-ihf+idisp
                 jg = (j-1)  * ncf + 1
                 do 165 k = -isey, isey, 1
                    ik = ig + k
                    ig11 = (i-1) * idv + idvf1 + k
                    ig11 = i + k
                    ig22 = ig11 + k
                    do 166 l = 0, isearch, 1
                       il = jg + l
                       lg11 = j + l
```

```
                lg22 = lg11 + 1

              bias (ik, il) = 1.5 * (dcoef * ((xh1 (i,j) -
             $  gsub (ig11, lg11)) ** 2 +
             $ (gsub (ig11, lg11) - gsub3 (ig22, lg22)) ** 2) +
             $ bcoef * ((xk1 (i, j) - gk1 (ig11, lg11)) ** 2 +
             $ (gk1 (ig11, lg11) - gk3 (ig22, lg22)) ** 2)
             $ + ccoef * ((xk2 (i, j) - gk2 (ig11, lg11)) ** 2
             $ + (gk2 (ig11, lg11) - gk4 (ig22, lg22)) ** 2))
166             continue
165           continue
164         continue

            do 167 j = n-ihf+idisp+1, n-ihf+idisp+idisp
               jg = (j-1) * ncf + 1
               do 168 k = -isey, isey, 1
                  ik = ig + k
                  ig11 = i + k
                  do 169 l = 0, isearch, 1
                     il = jg + l
                     lg11 = j + l
              bias (ik, il) = 3.0 * (dcoef * (xh1 (i, j) - gsub (ig11, lg11)) ** 2
            $ + bcoef * (xk1 (i, j) - gk1 (ig11, lg11)) ** 2 +
            $ + ccoef * (xk2 (i, j) - gk2 (ig11, lg11)) ** 2)
169               continue
168            continue
167         continue
160       continue

          iend = n - ihf + 2 * idisp
          ibg = ihs1 + 1
          if (inornot.eq.1) go to 170
          do 171 i = ihy1+1, len-ihy1
             ig = (i - 1) * nvf + 2
             do 172 l = ibg, iend
                lg = (l - 1) * ncf + 1
                amax = bias (ig, lg)
                ip = 0
                iq = 0
                irr = 0
                do 173 j = -isey, isey, 1
                   do 174 k = 0, isearch, 1
                      sumh = bias (ig + j, lg + k)
                      if (amax.lt.sumh) go to 174
                      if (amax.gt.sumh) go to 175
                      irw = j * j + k * k
                      if (irw.ge.irr) go to 174
175                   amax = sumh
```

```
                    ip = j
                    iq = k
                    irr = irw
174               continue
173             continue
                ihnun (i, l) = iq + isch1
                ivnun (i, l) = ip + isch1
                ihold (i, l) = iq + isch1
                ivold (i, l) = ip + isch1
172           continue
171         continue

170         if (iynbnd.eq.0) go to 180
            open (33, file=boundfile, status='old')
            do 181 i = 1, len
               read (33, 500)(ix(j), j = 1, n)
                 do 182 j = 1, n
                    iboun (i, j) = ix (j)
                    if (ix(j).eq.0) go to 182
                    ig = (i - 1) * nvf + 2
                    lg = (j - 1) * ncf + 1
                    amin = 10000.0
                    do 183 k = -isey, isey, 1
                       do 184 l = 0, isearch, 1
                          temp = bias (ig + k, lg + l)
                          if (amin.lt.temp) go to 184
                          amin = temp
184                    continue
183                 continue
                    bias (ig, lg) = amin
                    ihold (i, j) = isch1
                    ivold (i, j) = isch1
                    ihnun (i, j) = isch1
                    ivnun (i, j) = isch1
182           continue
181         continue
            close (33)
180         do 200 l = 1, itera
               if (iynbnd.eq.0) go to 210
                 do 211 j = 1, len
                    do 212 k = 1, n
                       if (iboun (j, k).eq.0) go to 212
                       ihold (j, k) = isch1
                       ivold (j, k) = isch1
                       ihnun (j, k) = isch1
                       ivnun (j, k) = isch1
212                 continue
211               continue
```

```
210      if (ecoef.gt.0) call line (len, n, ecoef, fcoef, isearch)

           do 220 j = ihy1+1, len-ihy1
              jj = (j - 1) * nvf + 2
                do 230 k = ihs1, iend
                   kk = (k - 1) * ncf + 1
                   itph = ihold (j, k) - isch1
                   itpv = ivold (j, k) - isch1
                   it2h = itph
                   it2v = itpv
                   do 231 la = ihshd-1, ihsed, 1
                      do 232 lb = ivshd, ivsed, 1
                         itp3 (lb, la) = 0
232                   continue
231                 continue
                    jn = ivold (j, k)
                    kn = ihold (j, k)
                    icompt = - ke4
                    if ((jn.eq.lhl1(j,k)).and.(kn.eq.lhk1(j,k))) icompt = icompt + ke
                    if ((jn.eq.lhl1(j,k-1)).and.(kn.eq.lhk1(j,k-1))) icompt = icompt + ke
                    if ((jn.eq.lvl1(j,k)).and.(kn.eq.lvk1(j,k))) icompt = icompt + ke
                    if ((jn.eq.lvl1(j-1,k)).and.(kn.eq.lvk1(j-1,k))) icompt = icompt + ke
                    if ((jn.eq.lhl2(j,k)).and.(kn.eq.lhk2(j,k))) icompt = icompt + ke
                    if ((jn.eq.lhl2(j,k-1)).and.(kn.eq.lhk2(j,k-1))) icompt = icompt + ke
                    if ((jn.eq.lvl2(j,k)).and.(kn.eq.lvk2(j,k))) icompt = icompt + ke
                    if ((jn.eq.lvl2(j-1,k)).and.(kn.eq.lvk2(j-1,k))) icompt = icompt + ke
                    itp3 (jn, kn) = icompt

                    if ((jn.eq.lhl1(j,k)).and.(kn.eq.lhk1(j,k))) go to 240
                    itp3(ivold(j,k+1), ihold(j,k+1))=itp3(ivold (j,k+1), ihold(j,k+1)) + ke
240
                    if ((jn.eq.lvl1(j,k)).and.(kn.eq.lvk1(j,k))) go to 250
                    itp3(ivold(j+1,k), ihold(j+1,k))=itp3(ivold(j+1,k), ihold(j+1,k)) + ke

250                 if ((jn.eq.lhl2(j,k-1)).and.(kn.eq.lhk2(j,k-1))) go to 260
                    itp3(ivold(j,k-1),ihold(j,k-1))=itp3(ivold(j,k-1), ihold(j,k-1)) + ke

260                 if ((jn.eq.lvl2(j-1,k)).and.(kn.eq.lvk2(j-1,k))) go to 270
                    itp3(ivold(j-1,k),ihold(j-1,k))=itp3(ivold(j-1,k), ihold(j-1,k)) + ke

270        do 271 ir = -2, 2, 1
              do 272 ic = -2, 2, 1
                 itp3 (ivold(j+ir,k+ic), ihold(j+ir,k+ic)) =
           $ itp3 (ivold(j+ir,k+ic), ihold(j+ir,k+ic)) + ka2
272           continue
271        continue

         itp3 (ivold(j,k), ihold(j,k)) = itp3 (ivold(j,k), ihold(j,k)) - ka48 - ka2
```

```
c-------------------------------------------------------------------------------
c          Gaussian distribution
c-------------------------------------------------------------------------------
           wsig = itp3 (ivold(j,k), ihold(j,k)) * wcc
           wvar = wsig * wsig
           f0 = 0.5284163
           wh = 1
           wmpx = 1
           call grndm (wrn, wvar, f0, wh, wmpx)
           itp3 (ivold (j,k), ihold (j,k)) = wrn + itp3 (ivold(j,k), ihold(j,k))
c-------------------------------------------------------------------------------
           amax = float (itp3 (itpv+isch1, itph+isch1)) - bias (jj+itpv, kk+itph)
           old = amax
           do 280 imv = -isey, isey, 1
              do 281 imh = 0, isearch, 1
                 t3 = float (itp3 (imv+isch1, imh+isch1)) - bias (jj+imv, kk+imh)
                 if (amax.ge.t3) go to 281
290              amax = t3
                 itph = imh
                 itpv = imv
281           continue
280        continue

           eneg = old - amax + ka48 - icompt
           if  (eneg.ge.0.0) go to 230
              ihnun (j,k) = itph + isch1
              ivnun (j,k) = itpv + isch1
230        continue
220        continue

           isumh = 0
           isumv = 0
                do 295 j = 1, n
                   do 296 i = 1, len
                      ihhh = ihnun (i, j)
                      ivvv = ivnun (i, j)
                      ihold (i, j) = ihhh
                      ivold (i, j) = ivvv
                      isumh = isumh + (ihhh - isch1) * (ihhh - isch1)
                      isumv = isumv + (ivvv - isch1) * (ivvv - isch1)
296                continue
295             continue

                sumh = float (isumh) / snn
                sumv = float (isumv) / snn
                print 297, l
297             format (1x, i4, 'th iteration')
                print 298, sumh, sumv
```

```
298          format (1x, e16.8, e16.8)

             if ((sumh.eq.hold1).and.(sumv.eq.vold1)) go to 300
             if ((sumh.eq.hold2).and.(sumv.eq.vold2)) go to 300
             hold2 = hold1
             vold2 = vold1
             hold1 = sumh
             vold1 = sumv
200       continue

300       call line (len, n, ecoef, fcoef, isearch)
          open (21, file=hfile, status='new')
          do 310 i = 1, len
            write (21, 500) (ihnun (i, j), j = 1, n)
310       continue
          close (21)

          open (22, file=vfile, status='new')
          do 320 i = 1, len
             write (22, 500) (ivnun (i, j), j = 1, n)
320       continue
          close (22)

          print *, (ivnun (25, i), i = 1, 20)
          amax = 0
          do 330 i = 1, len
            do 330 j = 1, n
               temp = abs (ihnun (i, j) - isch1)
               if (temp.gt.amax) amax = temp
               temp = abs (ivnun (i, j) - isch1)
               if (temp.gt.amax) amax = temp
330       continue
          amax = 2.0 * amax
          print *, amax
          DIM = 1.333333 * (n + 2.0)
500       format (113 i3)
          do 340 i = 1, len
            do 340 j = 1, n
               xh2 (i, j) = (ihnun (i, j) - isch1) / amax
               xh1 (i, j) = (ivnun (i, j) - isch1) / amax
340       continue
          alen = n - len
          CALL BEGING (1, 0, -1)
          CALL CLEAR
          CALL WINDOW (-1.0, DIM, -1.0, (n+2.0))
          CALL VECABS
          CALL MOVE (0.0, alen)
          CALL DRAW (1.0 + n, alen)
```

```
      CALL DRAW (1.0 + n, 1.0 + n)
      CALL DRAW (0.0, 1.0 + n)
      CALL DRAW (0.0, alen)

      do 350 j = 1, n
        do 351 i = 1, len
          xco = j + xh2 (i, j)
          yco = n - i - xh1 (i, j) + 1
          ai = j
          aj = n - i + 1
          CALL MOVE (ai, aj)
          CALL DRAW (xco, yco)
351     continue
350   continue

      if (lyn.eq.0) go to 360
      do 361 j = 1, n-1
        do 361 i = 1, len-1
          if (lvk1 (i, j).ne.isearch) go to 362
          if (lvl1 (i, j).ne.isearch) go to 362
          if (lvk2 (i, j).ne.isearch) go to 362
          if (lvl2 (i, j).ne.isearch) go to 362
          go to 365
362       xco = j + 0.5
          yco = n - i + 0.5
          ai = j - 0.5
          aj = n - i + 0.5
          CALL MOVE (ai, aj)
          CALL DRAW (xco, yco)
365       if (lhk1 (i,j).ne.isearch) go to 367
          if (lhl1 (i,j).ne.isearch) go to 367
          if (lhk2 (i,j).ne.isearch) go to 367
          if (lhl2 (i,j).ne.isearch) go to 367
          go to 361
367       xco = j + 0.5
          yco = n - i + 1.5
          ai = j + 0.5
          aj = n - i + 0.5
          CALL MOVE (ai, aj)
          CALL DRAW (xco, yco)
361   continue

360   ai = n + 4
      aj = n - 2
      CALL MOVE (ai, aj)
      CALL TEXT (outfile)
      ai = n + 4
      aj = n - 12
```

```
      CALL MOVE (ai, aj)
      CALL BELL
      CALL ENDG

      stop
      end

c-----------------------------------------------------------------------------
      subroutine rnif (u, f0, k)
      real*8 f, f0, f1, f2

      if (k.eq.0) go to 10
      f1 = f0
      k = 0
10    f = 997.0 * f1
      f2 = f - idint (f)
      f1 = f2
      u = sngl (f2)
      return
      end
c-----------------------------------------------------------------------------
      subroutine grndm (u, var, f0, init, mpx)
      real*4 u, rnd1, rnd2, pi2, sq, u1, u2, var
      real*8 f0
      data pi/3.14159/

      if (init.eq.1) mpx = 1
      if (mpx.eq.2) go to 20
      call rnif (u1, f0, init)
      call rnif (u2, f0, init)
      sq=sqrt (-2.0 * var * alog (u1))
      pi2 = 2.0 * pi * u2
      rnd1 = sq * cos (pi2)
      rnd2 = sq * sin (pi2)
      u = rnd1
      mpx = 2
      return
20    u = rnd2
      mpx = 1
      return
      end
c-----------------------------------------------------------------------------
c     estimate principal curvatures
c-----------------------------------------------------------------------------
      subroutine wnd (iwd, n, idv, sp, iord, len)
      dimension x (80, 113), y(80, 113), z(80, 113), z2(80, 113)
      dimension xtemp (80, 113)
      dimension gsub (80, 113), gk1(80, 113), gk2(80, 113)
```

```
      dimension gsub3 (80, 113), gk3(80, 113), gk4(80, 113)
      dimension gsub4 (80, 113), gk5(80, 113), gk6(80, 113)
      dimension xk1 (80, 113), xk2 (80, 113)
      real*8 atcc2 (4), atrr2 (4), cz (5, 5), tempz, xrrz, xccz, xrcz
      real*8 temdz, xrrdz, xccdz, xrcdz, tempx, tempy
      real*8 tcc2 (4), trr2 (4), atc (5), atr (5), atrr (4), atcc (4)
      real*8 p (5, 5), a (5, 5), b (5, 5), u1, u2, u3, u4, trr (4), dz (5, 5)
      real*8 tc (5), tcc (4), tr (5), temp, aic, air, xrr, xcc, xrc
      common/sig/x, y, z, z2
      common/subdv/gsub, gk1, gk2
      common/xdv/xk1, xk2
      common/subdv3/gsub3, gk3, gk4
      common/subdv4/gsub4, gk5, gk6

      ntotal = n * idv
      ihalf = iwd / 2
      q0 = iwd
      q2 = 0
      g4 = 0
      g6 = 0
      do 10 i = 1, ihalf
         q2 = q2 + i * i
         q4 = q4 + i * i * i * i
         q6 = q6 + i * i * i * i * i * i
10    continue

      q2 = q2 * 2.0
      q4 = q4 * 2.0
      q6 = q6 * 2.0
      u1 = -q2 / q0
      u2 = -q4 / q2
      u3 = (q2 * q4 - q0 * q6) / (q0 * q4 - q2 * q2)
      u4 = (q2 * q6 - q4 * q4) / (q0 * q4 - q2 * q2)

      do 50 i = 1, iord
         do 51 j = 1, iord
            p (i, j) = 0
            a (i, j) = 0
            b (i, j) = 0
            cz (i, j) = 0
            dz (i, j) = 0
51       continue
50    continue

      do 30 i = -ihalf, ihalf, 1
         tr (1) = 1
         tr (2) = i * i
         tr (3) = (i * i + u1) * (i * i + u1)
```

```
            tr (4) = (i * i * i + u2 * i) * (i * i * i + u2 * i)
            tr (5) = (i * i * i * i + u3 * i * i + u4) * (i * i * i * i + u3 * i * i + u4)
            do 31 j = -ihalf, ihalf, 1
               tc (1) = 1
               tc (2) = j * j
               tc (3) = (j * j + u1) * (j * j + u1)
               tc (4) = (j * j * j + u2 * j) * (j * j * j + u2 * j)
               tc (5) = (j * j * j * j + u3 * j * j + u4) * (j * j * j * j + u3 * j * j + u4)
               do 35 k = 1, iord
                  do 36 l = 1, iord
                     p (k, l) = p (k ,l) + tr (k) * tc (l)
36                continue
35             continue
31          continue
30       continue

         atr (1) = 1
         atr (2) = 0
         atr (3) = u1
         atr (4) = 0
         atr (5) = u4
         atc (1) = 1
         atc (2) = 0
         atc (3) = u1
         atc (4) = 0
         atc (5) = u4
         atcc (1) = 2
         atcc (2) = 0
         atcc (3) = u3 * 2.0
         atrr (1) = 2
         atrr (2) = 0
         atrr (3) = u3 * 2.0
         atcc2 (1) = 1
         atcc2 (2) = 0
         atcc2 (3) = u2
         atcc2 (4) =0
         atrr2 (1) = 1
         atrr2 (2) = 0
         atrr2 (3) = u2
         atrr2 (4) = 0

         do 40 i = ihalf+1, len-ihalf
            iit = i
            do 41 j = ihalf+1, n-ihalf
               jjt = j
               do 65 k = 1, iord
                  do 66 l =1 , iord
                     a (k, l) = 0
```

```
                b (k, l) = 0
                cz (k, l) = 0
                dz (k, l) = 0
66            continue
65          continue

        do 60 ir = -ihalf, ihalf, 1
           tr (1) = 1
           tr (2) = ir
           tr (3) = ir * ir + u1
           tr (4) = ir * ir * ir + u2 * ir
           tr (5) = ir * ir * ir * ir + u3 * ir * ir + u4
           do 61 ic = -ihalf, ihalf, 1
              tc (1) = 1
              tc (2) = ic
              tc (3) = ic * ic + u1
              tc (4) = ic * ic * ic + u2 * ic
              tc (5) = ic * ic * ic * ic + u3 * ic * ic + u4
              do 62 k = 1, iord
                 do 63 l = 1, iord
                    a (k, l) = a (k, l) + tr (k) * tc (l) * x (i+ir, j+ic)
                    b (k, l) = b (k, l) + tr (k) * tc (l) * y (i+ir, j+ic)
                    cz (k, l) = cz (k, l) + tr (k) * tc (l) * z (i+ir, j+ic)
                    dz (k, l) = dz (k, l) + tr (k) * tc (l) * z2 (i+ir, j+ic)
63               continue
62            continue
61         continue
60      continue

        do 80 k = 1, iord
           do 81 l = 1, iord
              a (k, l) = a (k, l) / p (k, l)
              b (k, l) = b (k, l) / p (k, l)
              cz (k, l) = cz (k, l) / p (k, l)
              dz (k, l) = dz (k, l) / p (k, l)
81         continue
80      continue

        ir = 0
        ii = iit
        air = ir * sp
        tr (1) = 1
        tr (2) = 0
        tr (3) = u1
        tr (4) = 0
        tr (5) = u4
        trr (1) = 2
        trr (2) = 0
```

```
      trr (3) = u3 * 2.0
      trr2 (1) = 1
      trr2 (2) = 0
      trr2 (3) = u2
      trr2 (4) = 0
      ic = 0
      jj = jjt
      aic = 0
      tc (1) = 1
      tc (2) = 0
      tc (3) = u1
      tc (4) = 0
      tc (5) = u4
      temp = 0
      tempz = 0
      temdz = 0

      do 165 k = 1, iord
         do 166 l = 1, iord
            temp = temp + tr (k) * tc (l) * b (k, l)
            tempz = tempz + tr (k) * tc (l) * cz (k, l)
            temdz = temdz + tr (k) * tc (l) * dz (k, l)
166      continue
165   continue

      gsub (ii, jj) = temp
      gsub3 (ii, jj) = tempz
      gsub4 (ii, jj) = temdz
      xrr = 0
      xcc = 0
      xrc = 0
      xrrz = 0
      xccz = 0
      xrcz = 0
      xrrdz = 0
      xccdz = 0
      xrcdz = 0
      tcc (1) = 2
      tcc (2) = 0
      tcc (3) = u3 * 2.0

      do 180 k = 1, iord
         temp = 0
         tempz = 0
         temdz = 0
         do 181 l = 1, iord-2
            temp = temp + b (k, l+2) * tcc (l)
            tempz = tempz + cz (k, l+2) * tcc (l)
```

```
                temdz = temdz + dz (k, l+2) * tcc (l)
181           continue
              xrr = xrr + temp * tr (k)
              xrrz = xrrz + tempz * tr (k)
              xrrdz = xrrdz + temdz * tr (k)
180       continue

          do 190 k = 3, iord
              temp = 0
              tempz = 0
              temdz = 0
              do 191 l = 1, iord
                  temp = temp + b (k, l) * tc (l)
                  tempz = tempz + cz (k, l) * tc (l)
                  temdz = temdz + dz (k, l) * tc (l)
191           continue
              xcc = xcc + temp * trr (k - 2)
              xccz = xccz + tempz * trr (k - 2)
              xccdz = xccdz + temdz * trr (k - 2)
190       continue

          tcc2 (1) = 1
          tcc2 (2) = 0
          tcc2 (3) = u2
          tcc2 (4) = 0

          do 210 k = 1, iord-1
              temp = 0
              tempz = 0
              temdz = 0
              do 211 l = 1, iord-1
                 temp = temp + b (k+1, l+1) * tcc2 (l)
                 tempz = tempz + cz (k+1, l+1) * tcc2 (l)
                 temdz = temdz + dz (k+1, l+1) * tcc2 (l)
211           continue
              xrc = xrc + temp * trr2 (k)
              xrcz = xrcz + tempz * trr2 (k)
              xrcdz = xrcdz + temdz * trr2 (k)
210       continue

          temp = xrr + xcc
          tempz = xrrz + xccz
          temdz = xrrdz + xccdz
390       gk1 (ii, jj) = (temp + dsqrt ((xrr - xcc) * (xrr - xcc) + 4 * xrc * xrc)) / 2.0
380       gk2 (ii, jj) = temp - gk1 (ii, jj)
          gk3 (ii, jj) = (tempz + dsqrt ((xrrz - xccz) ** 2 + 4 * xrcz * xrcz)) / 2.0
          gk4 (ii, jj) = tempz - gk3 (ii, jj)
          gk5 (ii, jj) = (temdz + dsqrt ((xrrdz - xccdz) ** 2 + 4 * xrcdz * xrcdz)) / 2.0
```

```
      gk6 (ii, jj) = temdz - gk5 (ii, jj)

      temp = 0
      xrr = 0
      xcc = 0
      xrc = 0
      tempy = 0

      do 1182 k = 1, iord
         tempx = 0
         do 1183 l = 1, iord
            tempx = tempx + a (k, l) * atcc (l)
1183     continue
         tempy = tempy + tempx * atr (k)
1182  continue

      xtemp (i, j) = tempy

      do 1180 k = 1, iord
         temp = 0
         do 1181 l = 1, iord-2
            temp = temp + a (k, l+2) * atcc (l)
1181     continue
         xrr = xrr + temp * atr (k)
1180  continue

      do 1190 k = 3, iord
         temp = 0
         do 1191 l = 1, iord
            temp = temp + a (k, l) * atc (l)
1191     continue
         xcc = xcc + temp * atrr (k - 2)
 1190 continue

      do 1210 k = 1, iord-1
         temp = 0
         do 1211 l = 1, iord-1
            temp = temp + a (k+1, l+1) * atcc2 (l)
1211     continue
         xrc = xrc + temp * atrr2 (k)
1210  continue

      temp = xrr + xcc
      xk1 (i, j) = (temp + dsqrt ((xrr - xcc) * (xrr - xcc) + 4 * xrc * xrc)) / 2.0
      xk2 (i, j) = temp - xk1 (i, j)
41    continue
40    continue
```

```
          do 1185 i = 1, len
             do 1185 l = 1, n
                x (i, j) = xtemp (i, j)
1185      continue

          return
          end

c-------------------------------------------------------------------------------
c         line process
c-------------------------------------------------------------------------------
          subroutine line (len, n, ccoef, dcoef, isearch)
          dimension ihold (80, 113), ivold (80, 113)
          dimension lvk1 (80, 113), lvl1 (80, 113)
          dimension lvk2 (80, 113), lvl2 (80, 113)
          dimension lhk1 (80, 113), lhl1 (80, 113)
          dimension lhk2 (80, 113), lhl2 (80, 113)
          common/neuron/ihold, ivold
          common/vline/lvk1, lvl1, lvk2, lvl2
          common/hline/lhk1, lhl1, lhk2, lhl2

          do 10 i = 1, len-1
             do 10 j = 1, n-1
              k = ihold (i, j)
              l = ivold (i, j)
              kv = ihold (i+1, j)
              lv = ivold (i+1, j)
              kh = ihold (i, j+1)
              lh = ivold (i, j+1)
              k1 = lvk1 (i, j)
              l1 = lvl1 (i, j)
              k2 = lvk2 (i, j)
              l2 = lvl2 (i, j)
              kh1 = lhk1 (i, j)
              lh1 = lhl1 (i, j)
              kh2 = lhk2 (i, j)
              lh2 = lhl2 (i, j)
              if ((k.ne.kv).or.(l.ne.lv)) go to 34
              lvk1 (i, j) = isearch
              lvl1 (i, j) = isearch
              lvk2 (i, j) = isearch
              lvl2 (i, j) = isearch
              go to 220

34            if ((k.eq.k1).and.(l.eq.l1)) go to 50
              temp = dcoef - 0.5 * ccoef
              if (temp.ge.0.0) go to 120
              lvk1 (i, j) = k
```

```
            lvl1 (i, j) = l
            go to 120
50          temp = 0.5 * ccoef - dcoef
            if (temp.ge.0.0) go to 120
            lvk1 (i, j) = isearch
            lvl1 (i, j) = isearch
120         if ((kv.eq.k2).and.(lv.eq.l2)) go to 150
            temp = dcoef - 0.5 * ccoef
            if (temp.ge.0.0) go to 220
            lvk2 (i, j) = kv
            lvl2 (i, j) = lv
            go to 220
150         temp = 0.5 * ccoef - dcoef
            if (temp.ge.0.0) go to 220
            lvk2 (i, j) = isearch
            lvl2 (i, j) = isearch
220         if ((k.eq.kh).and.(l.eq.lh)) go to 40
            if ((k.eq.kh1).and.(l.eq.lh1)) go to 250
            temp = dcoef - 0.5 * ccoef
            if (temp.ge.0.0) go to 280
            lhk1 (i, j) = k
            lhl1 (i, j) = l
            go to 280
250         temp = 0.5 * ccoef - dcoef
            if (temp.ge.0.0) go to 280
            lhk1 (i, j) = isearch
            lhl1 (i, j) = isearch
280         if ((kh.eq.kh2).and.(lh.eq.lh2)) go to 290
            temp = dcoef - 0.5 * ccoef
            if (temp.ge.0.0) go to 10
            lhk2 (i, j) = kh
            lhl2 (i, j) = lh
            go to 10
290         temp = 0.5 * ccoef - dcoef
            if (temp.ge.0.0) go to 10
            lhk2 (i, j) = isearch
            lhl2 (i, j) = isearch
            go to 10
40          lhk1 (i, j) = isearch
            lhl1 (i, j) = isearch
            lhk2 (i, j) = isearch
            lhl2 (i, j) = isearch
10        continue

          return
          end
```

Appendix B

Program for Neural-Based Image Restoration

To compile, please type "f77 -o restore restore.f libsipi.a libsplot.a ". To run the program, please type "restore < restore.dat > log_res".

```
c-------------------------------------------------------------------------------
c  An uniform, space-invariant, and 3x3 blur function is used.
c  The subborder region = 0.
c  Neurons at the same column are updated synchronously.
c  Gaussian function of device mismatch effect is included.
c-------------------------------------------------------------------------------
          program restore
          real*8 f0
          real*4 xw, str, bia, v, ynew, y
          real*4 ppk, erx2
          real*4 x (256, 256)
          real*4 xbig (258, 258), bblu(3, 3),xesbg (258, 258)
          dimension ix (256)
          character*12 flin
          character*12 flre
          character*12 flblu
          common /observed/y (256, 256)
          common /sig/xw (65536)
          common/connect/str (5, 5)
          common/biase/bia (65536)
          common/state/v (65536)
          print 10
10        format (1x, 'image size= ?')
          read*,nk

          print 20
20        format (1x, 's/n sn= ?(dB) [ 500: no noise')
          read *, sn

          print 30
30        format (1x, 'constraint coefficient ? (lamda)')
          read *, bcoef

          print 40
40        format (1x, 'the number of iterations for inner loop')
          read *, loopin

          print 50
50        format (1x, 'the number of iterations for outer loop')
          read *, loopout
```

```
          print 60
60        format (1x, 'the standard deviation')
          read *, wcc

          print 70
70        format (' enter input image file name :')
          read *, flin

          print 80
80        format (' enter  restored image file name :')
          read *, flre

          print 90
90        format (' enter blurreded image file name :')
          read *, flblu

100       format (' error of estimation',i5)

c-------------------------------------------------------------------------------
          iwd=3
          n=nk*nk
          iwrite=1
c-------------------------------------------------------------------------------
c         input image file
c-------------------------------------------------------------------------------
          iread = 0
          call dopen (flin, iread, 1, 4, nunit)
          do 110 i = 1, nk
             call dread (nunit, ix, nk)
             do 111 j = 1, nk
             xbig (i+1, j+1) = float (ix (j))
             xw ((i-1) * nk + j) = float (ix (j))
111          continue
111       continue
          call dclose (nunit)

          call mean (bave, n)
          call varn (bave, avarn, n)

          do 120 j = 1, nk
             xbig (1, j+1) = xbig (2, j+1)
             xbig (nk+2, j+1) = xbig (nk+1, j+1)
             xbig (j+1, 1) = xbig (j+1, 2)
             xbig (j+1, nk+2) = xbig (j+1, nk+1)
120       continue

          xbig (1, 1) = xbig (2, 2)
          xbig (nk+2, 1) = xbig (nk+1, 2)
```

```
          xbig (1, nk+2) = xbig (2, nk+1)
          xbig (nk+2, nk+2) = xbig (nk+1, nk+1)

c-------------------------------------------------------------------------------
c         blur function
c-------------------------------------------------------------------------------
          hkk = 9.0
          hii = 1.0 / hkk
          hij = hii
          do 130 i = 1, iwd
             do 131 j = 1, iwd
             bblu (i, j) = hij
131          continue
130       continue

          do 140 i = 1, iwd
             print *, (bblu (i, j), j = 1, iwd)
140       continue

c-------------------------------------------------------------------------------
c         interconnection strength
c-------------------------------------------------------------------------------
          str (1, 1) = - 1.0 / 81.0
          str (1, 2) = - 2.0 / 81.0
          str (1, 3) = - 3.0 / 81.0
          str (1, 4) = - 2.0 / 81.0
          str (1, 5) = - 1.0 / 81.0

          str (2, 1) = - 2.0 / 81.0
          str (2, 2) = - 4.0 / 81.0
          str (2, 3) = - 6.0 / 81.0
          str (2, 4) = - 4.0 / 81.0
          str (2, 5) = - 2.0 / 81.0

          str (3, 1) = - 3.0 / 81.0
          str (3, 2) = - 6.0 / 81.0
          str (3, 3) = - 9.0 / 81.0
          str (3, 4) = - 6.0 / 81.0
          str (3, 5) = - 3.0 / 81.0

          str (4, 1) = - 2.0 / 81.0
          str (4, 2) = - 4.0 / 81.0
          str (4, 3) = - 6.0 / 81.0
          str (4, 4) = - 4.0 / 81.0
          str (4, 5) = - 2.0 / 81.0

          str (5, 1) = - 1.0 / 81.0
          str (5, 2) = - 2.0 / 81.0
```

```
          str (5, 3) = - 3.0 / 81.0
          str (5, 4) = - 2.0 / 81.0
          str (5, 5) = - 1.0 / 81.0
c-------------------------------------------------------------------------------
c         Gaussian distribution
c-------------------------------------------------------------------------------
          do 150 irow = 1, 5
             do 151 icol = 1, 5
                wsig = str (irow, icol) * wcc
                wvar = wsig * wsig
                f0 = 0.5284163
                wh = 1
                wmpx = 1
                call grndm (wrn, wvar, f0, wh, wmpx)
                str (irow, icol) = wrn + str (irow, icol)
                print *, str (irow, icol)
151          continue
150       continue
c-------------------------------------------------------------------------------
c         blur image
c-------------------------------------------------------------------------------
          do 160 i = 1, nk
             do 160 j = 1, nk
                x (i, j) = 0
                do 161 k = 1, iwd
                   do 161 l = 1, iwd
161                x (i, j) = x (i, j) + xbig (i+k-1, j+l-1) * bblu (k, l)
             x (i, j) = ifix (x (i, j) + 0.5)
160       continue

          do 170 i = 1, n
             xw (i) = 0.0
170       continue

          call dopen (flblu, iwrite, 1, 4, nunit)
          do 180 k = 1, nk
             do 181 j = 1, nk
                ix (j) = x (k, j)
                if (ix(j).lt.0) ix (j) = 0
                if (ix(j).gt.255) ix (j) = 255
181          continue
180       call dwrite (nunit, ix, nk)
          call dclose (nunit)

          cc = avarn * 10 ** (- (sn / 10.0))
          if (sn.gt.499) go to 200
          f0 = 0.5284163
          kh = 1
```

```
          mpx = 1

          do 210 i = 1, n
             call grndm (vuu, cc, f0, kh, mpx)
             xw (i) = vuu
210       continue

          call mean (bave, n)
          call varn (bave, avarn, n)
          vuu = sqrt (cc / avarn)
          do 220 i = 1, n
             xw (i) = xw (i) * vuu
220       continue
c-------------------------------------------------------------------------------
c         add noise
c-------------------------------------------------------------------------------
200       do 210 i = 1, nk
             do 210 j = 1, nk
             ppk = x (i, j) + xw ((i - 1) * nk + j)
             xw ((i-1) * nk + j) = ppk
             y (i, j) = ppk
210       continue
c-------------------------------------------------------------------------------
c         initial condition: v(0)
c-------------------------------------------------------------------------------
          do 220 i = 1, nk
             do 220 j = 1, nk
                klp = (i - 1) * nk + j
                v (klp) = xw (klp)
220       continue
          go to 230
c-------------------------------------------------------------------------------
c         bias input
c-------------------------------------------------------------------------------
230       do 240 i = 1, nk
             do 241 j = 1, nk
                xesbg (i+1, j+1) = y (i,j)
241          continue
240       continue
          do 250 j = 1, nk
             xesbg (1, j+1) = xesbg (2, j+1)
             xesbg (nk+2, j+1) = xesbg (nk+1, j+1)
             xesbg (j+1, 1) = xesbg (j+1, 2)
             xesbg (j+1, nk+2) = xesbg (j+1, nk+1)
250       continue
          xesbg (1, 1) = xesbg (2, 2)
          xesbg (nk+2, 1) = xesbg (nk+1, 2)
          xesbg (1, nk+2) = xesbg (2, nk+1)
```

```
      xesbg (nk+2, nk+2) = xesbg (nk+1, nk+1)

      do 260 i = 1, nk
        do 260 j = 1, nk
          jsum = (i-1) * nk + j
          bia (jsum) = 0
          do 261 l = 1, iwd
            do 261 k = 1, iwd
261         bia (jsum) = bia (jsum) + xesbg (i+k-1, j+l-1) * bblu(k,l)
260   continue

      ink = (nk-4) * (nk-4)

      do 270 jd = 1, loopout
        nstop = 0
        print 271, jd
271     format ('outloop=', i5)
        call update (nk, loopin, nstop)

270     format ('----------------------------------------')
        print 272
        print 273, jd
273     format (' the ',i5,'th iteration')

        do 274 i = 1, nk
          do 275 j = 1, nk
            xesbg (i+1, j+1) = v ((i-1) * nk + j)
275       continue
274     continue

        erx2 = 0
        do 275 i = 3, nk-2
          do 275 j = 3, nk-2
            ynew = 0
            erx2 = erx2 + ((xbig (i+1, j+1) - xesbg (i+1, j+1)) ** 2) / ink
275     continue
        print 276
276     format (/' error of estimation (x-es(x))')
        print *, erx2

        if (nstop.eq.0) go to 280
270   continue

280   call dopen (flre, iwrite, 1, 4, nunit)
      do 281 k = 1, nk
        do 282 j = 1, nk
          ix (j) = ifix (v((k-1) * nk + j) + 0.5)
          if (ix(j).lt.0) ix (j) = 0
```

```
                if (ix(j).gt.255) ix (j) = 255
282           continue
281         call dwrite (nunit, ix, nk)
            call dclose (nunit)
            stop
            end

c-------------------------------------------------------------------------------
            subroutine mean (ave, n)
            real*4 y
            common /sig/y(65536)

            ave = 0.0
            do 10 i = 1, n
10          ave = ave + y (i)
            ave = ave / n
            return
            end
c-------------------------------------------------------------------------------
            subroutine varn (ave, var, n)
            real*4 y
            common /sig/y(65536)

            var = 0.0
            do 10 j = 1, n
10          var = var + (y(j) - ave) ** 2
            var = var / n
            return
            end
c-------------------------------------------------------------------------------
            subroutine jgrg (varnc, dseed, n)
            real*4 y
            common /sig/y(65536)
            double precision dseed

            call mean (ave, n)
            call varn (ave, var, n)
            c = sqrt (varnc / var)
            do 15 j = 1, n
15          y (j) = y (j) * c
            return
            end
c-------------------------------------------------------------------------------
            subroutine rnif (u, f0, k)
            real*8 f, f0, f1, f2

            if (k.eq.0) go to 10
            f1 = f0
```

```
      k = 0
10    f = 997.0 * f1
      f2 = f - idint (f)
      f1 = f2
      u = sngl (f2)
      return
      end
c-----------------------------------------------------------------------------
      subroutine grndm (u, var, f0, init, mpx)
      real*4 u, rnd1, rnd2, pi2, sq, u1, u2, var
      real*8 f0
      data pi/3.14159/

      if (init.eq.1) mpx = 1
      if (mpx.eq.2) go to 20
      call rnif (u1, f0, init)
      call rnif (u2, f0, init)
      sq = sqrt (-2.0 * var * alog(u1))
      pi2 = 2.0 * pi * u2
      rnd1 = sq * cos (pi2)
      rnd2 = sq * sin (pi2)
      u = rnd1
      mpx = 2
      return
20    u = rnd2
      mpx = 1
      return
      end
c-----------------------------------------------------------------------------
      subroutine update (n, loopin, nstop)
      real*4 sux (65536), a, xw, bia, x, u(65536), tst
      common/connect/a(5,5)
      common/sig/xw(65536)
      common/biase/bia(65536)
      common/state/x(65536)

      do 100 i = 1, 65536
         sux (i) = 0
100   continue

      do 200 krow = 3, n-2, 1
        count = 0

210     do 220 kcol = 3, n-2, 1
           nout = (krow - 1) * n + kcol
           do 230 mrow = -2, 2, 1
              do 231 mcol = -2, 2, 1
                 lrow = krow + mrow
```

```
                lcol = kcol + mcol
                nin = (lrow - 1) * n + lcol
                sux (nout) = sux (nout) + a (mrow+3, mcol+3) * x (nin)
231           continue
230         continue

            u (nout) = bia (nout) + sux (nout)
            tst = u (nout)
            u (nout) = -1.0
            if (tst.gt.0.0) u (nout) = 1.0
220       continue

          do 250 lin = 3, n-2, 1
             nout = (krow - 1) * n + lin
             if ((x(nout).lt.1.0).and.(u(nout).lt.0.5)) go to 250
             if ((x(nout).gt.254.0).and.(u(nout).gt.0.5)) go to 250
             x (nout) = x (nout) + u (nout)
250       continue

          nstop = 1
          count = count + 1
          if (count.lt.loopin) go to 210
200     continue

        return
        end
```

Appendix C

SPICE Input files of basic circuit blocks

```
*-----------------------------------------------------
* find the operation time from synapse -> summing_neuron
*-----------------------------------------------------
.subckt out 7 6 9 11 14 17 16 99 100
*7=vin+ 6=vin- 9=vbb1 11=vbb2 14=vbb3 17=vbb4 16=vout
*99=vdd 100=vss
mo16p 2 2 99 99 pm 4u 20u ad=100p as=100p pd=28u ps=28u
mo15p 3 2 99 99 pm 4u 20u ad=100p as=100p pd=28u ps=28u
mo14n 4 4 2 99 pm 2u 6u ad=30p as=30p pd=16u ps=16u
mo13n 5 4 3 99 pm 2u 6u ad=30p as=30p pd=16u ps=16u
mo12n 4 6 8 8 nm 2u 16u ad=80p as=80p pd=26u ps=26u
mo11n 5 7 8 8 nm 2u 16u ad=80p as=80p pd=26u ps=26u
mo17n 8 9 100 100 nm 4u 40u ad=200p as=200p pd=50u ps=50u
mo18p 10 5 99 99 pm 4u 46u ad=230p as=230p pd=56u ps=56u
mo19p 12 11 10 99 pm 2u 90u ad=450p as=450p pd=100u ps=100u
mo20n 12 14 15 100 nm 2u 26u ad=130p as=130p pd=36u ps=36u
mo21n 15 9 100 100 nm 4u 52u ad=260p as=260p pd=62u ps=62u
mo22n 99 12 16 100 nm 2u 600u ad=3000p as=3000p pd=610u ps=610u
mo23n 16 17 100 100 nm 4u 400u ad=2000p as=2000p pd=410u ps=410u
mor 5 99 13 100 nm 2u 50u ad=250p as=250p pd=60u ps=60u
Cc 13 12 1.5pf
.ends
*-----------------------------------------------------
.subckt syn 6 3 4 99 100
*4=vij 3=vout 6=vin 99=vdd 100=vss
ms1p 2 2 99 99 pm 4u 4u ad=20p as=20p pd=14u ps=14u
ms2p 3 2 99 99 pm 4u 4u ad=20p as=20p pd=14u ps=14u
ms3n 2 4 5 100 nm 38u 3u ad=15p as=15p pd=13u ps=13u
ms4n 3 3 5 100 nm 38u 3u ad=15p as=15p pd=13u ps=13u
ms5n 5 6 100 100 nm 3u 4u ad=20p as=20p pd=14u ps=14u
.ends
*-----------------------------------------------------
* bias circuit for summing-neuron
*-----------------------------------------------------
mb21 21 21 100 100 nm w=20u l=4u
mb22 22 21 100 100 nm w=20u l=4u
mb23 23 23 21 100 nm w=8u l=10u
mb24 25 23 22 100 nm w=36u l=10u
mb25 23 24 99 99 pm w=19u l=4u
mb26 25 25 99 99 pm w=22u l=8u
v24 24 0 dc +3.8
v26 26 0 dc -3.7
v27 27 0 dc -3.55
*---------------------------------
```

```
xsyn1 27 31 71 99 100 syn
xsyn2 27 32 71 99 100 syn
xsyn3 27 33 71 99 100 syn
xsyn4 27 34 71 99 100 syn
xsyn5 27 35 71 99 100 syn
xsyn6 27 36 71 99 100 syn
xsyn7 27 37 71 99 100 syn
xsyn8 27 38 71 99 100 syn
xsyn9 27 39 71 99 100 syn
xsyn10 27 40 71 99 100 syn
xsyn11 27 41 71 99 100 syn
xsyn12 27 42 71 99 100 syn
xsyn13 27 43 71 99 100 syn
xsyn14 27 44 71 99 100 syn
xsyn15 27 45 71 99 100 syn
xsyn16 27 46 71 99 100 syn
xsyn17 27 47 71 99 100 syn
xsyn18 27 48 71 99 100 syn
xsyn19 27 49 71 99 100 syn
xsyn20 27 50 71 99 100 syn
xsyn21 27 51 71 99 100 syn
xsyn22 27 52 71 99 100 syn
xsyn23 27 53 71 99 100 syn
xsyn24 27 54 71 99 100 syn
xsyn25 27 55 71 99 100 syn
xsyn26 27 56 71 99 100 syn
v31 31 18 0v
v32 32 18 0v
v33 33 18 0v
v34 34 18 0v
v35 35 18 0v
v36 36 18 0v
v37 37 18 0v
v38 38 18 0v
v39 39 18 0v
v40 40 18 0v
v41 41 18 0v
v42 42 18 0v
v43 43 18 0v
v44 44 18 0v
v45 45 18 0v
v46 46 18 0v
v47 47 18 0v
v48 48 18 0v
v49 49 18 0v
v50 50 18 0v
v51 51 18 0v
v52 52 18 0v
```

```
v53 53 18 0v
v54 54 18 0v
v55 55 18 0v
v56 56 18 0v
xout1 0 18 21 25 22 26 20 99 100 out
rout 18 19 5k
v20 20 19 0v
cout 20 0 0.5p
vdd 99 0 5v
vss 100 0 -5v
v71 71 0 dc
.options gmin=1e-8 itl1=200 itl2=100 limpts=1000
.dc v71 -2.411 2.411 0.001
.width out=80
.print dc v(18) v(19) i(v31) i(v56) i(v20)
.nodeset v(18)=2.188e-02 v(19)=2
*-------------------------------------------------------
.MODEL nm NMOS level=2 LD=0.250000U TOX=395.000000E-10
+ NSUB=5.706000E+15 VTO=0.74457 KP=5.987000E-05 GAMMA=0.4978
+ PHI=0.6 UO=684.8 UEXP=0.185773 UCRIT=40591.3
+ DELTA=1.44455 VMAX=73636.6 XJ=0.150000U LAMBDA=3.557268E-02
+ NFS=4.667514E+12 NEFF=1 NSS=1.000000E+12 TPG=1.000000
+ RSH=28.940000 CGDO=3.278301E-10 CGSO=3.278301E-10 CGBO=7.168240E-10
+ CJ=1.201000E-04 MJ=0.675600 CJSW=5.545000E-10 MJSW=0.263600 PB=0.800000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 0.58 um
.MODEL pm PMOS level=2 LD=0.250000U TOX=395.000000E-10
+ NSUB=6.078000E+15 VTO=-0.723651 KP=2.404000E-05 GAMMA=0.5138
+ PHI=0.6 UO=274.99 UEXP=0.300442 UCRIT=35276.9
+ DELTA=1.000000E-06 VMAX=94066.1 XJ=0.050000U LAMBDA=5.530815E-02
+ NFS=2.697453E+12 NEFF=1.001 NSS=1.000000E+12 TPG=-1.000000
+ RSH=109.400000 CGDO=3.278301E-10 CGSO=3.278301E-10 CGBO=5.553545E-10
+ CJ=2.357000E-04 MJ=0.558900 CJSW=2.785000E-10 MJSW=0.316100 PB=0.800000
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is -0.16 um
*-------------------------------------------------------
.end
```

```
*-------------------------------------------------
* 2-stage winner-take-all circuits (25cells)
*-------------------------------------------------
x1 1 102 2 90 91 92 93 98 99 550 560 winner
x2 3 104 4 90 91 92 93 98 99 550 560 winner
x3 5 106 6 90 91 92 93 98 99 550 560 winner
x4 7 108 8 90 91 92 93 98 99 550 560 winner
x5 9 110 10 90 91 92 93 98 99 550 560 winner
x6 11 112 12 90 91 92 93 98 99 550 560 winner
x7 13 114 14 90 91 92 93 98 99 550 560 winner
x8 15 116 16 90 91 92 93 98 99 550 560 winner
x9 17 118 18 90 91 92 93 98 99 550 560 winner
x10 19 120 20 90 91 92 93 98 99 550 560 winner
x11 21 122 22 90 91 92 93 98 99 550 560 winner
x12 23 124 24 90 91 92 93 98 99 550 560 winner
x13 25 126 26 90 91 92 93 98 99 550 560 winner
x14 27 128 28 90 91 92 93 98 99 550 560 winner
x15 29 130 30 90 91 92 93 98 99 550 560 winner
x16 31 132 32 90 91 92 93 98 99 550 560 winner
x17 33 134 34 90 91 92 93 98 99 550 560 winner
x18 35 136 36 90 91 92 93 98 99 550 560 winner
x19 37 138 38 90 91 92 93 98 99 550 560 winner
x20 39 140 40 90 91 92 93 98 99 550 560 winner
x21 41 142 42 90 91 92 93 98 99 550 560 winner
x22 43 144 44 90 91 92 93 98 99 550 560 winner
x23 45 146 46 90 91 92 93 98 99 550 560 winner
x24 47 148 48 90 91 92 93 98 99 550 560 winner
x25 49 150 50 90 91 92 93 98 99 550 560 winner
*
x26 2 151 51 190 191 192 193 198 199 550 560 winner
x27 4 152 52 190 191 192 193 198 199 550 560 winner
x28 6 153 53 190 191 192 193 198 199 550 560 winner
x29 8 154 54 190 191 192 193 198 199 550 560 winner
x30 10 155 55 190 191 192 193 198 199 550 560 winner
x31 12 156 56 190 191 192 193 198 199 550 560 winner
x32 14 157 57 190 191 192 193 198 199 550 560 winner
x33 16 158 58 190 191 192 193 198 199 550 560 winner
x34 18 159 59 190 191 192 193 198 199 550 560 winner
x35 20 160 60 190 191 192 193 198 199 550 560 winner
x36 22 161 61 190 191 192 193 198 199 550 560 winner
x37 24 162 62 190 191 192 193 198 199 550 560 winner
x38 26 163 63 190 191 192 193 198 199 550 560 winner
x39 28 164 64 190 191 192 193 198 199 550 560 winner
x40 30 165 65 190 191 192 193 198 199 550 560 winner
x41 32 166 66 190 191 192 193 198 199 550 560 winner
x42 34 167 67 190 191 192 193 198 199 550 560 winner
x43 36 168 68 190 191 192 193 198 199 550 560 winner
x44 38 169 69 190 191 192 193 198 199 550 560 winner
```

```
x45 40 170 70 190 191 192 193 198 199 550 560 winner
x46 42 171 71 190 191 192 193 198 199 550 560 winner
x47 44 172 72 190 191 192 193 198 199 550 560 winner
x48 46 173 73 190 191 192 193 198 199 550 560 winner
x49 48 174 74 190 191 192 193 198 199 550 560 winner
x50 50 175 75 190 191 192 193 198 199 550 560 winner
*
vdd 550 0 dc +5
vss 560 0 dc -5
*
v1 1 0 dc
v3 3 0 dc -1.5
v5 5 0 dc -1.525
v7 7 0 dc -1.525
v9 9 0 dc -1.525
v11 11 0 dc -1.525
v13 13 0 dc -1.525
v15 15 0 dc -1.525
v17 17 0 dc -1.525
v19 19 0 dc -1.525
v21 21 0 dc -1.525
v23 23 0 dc -1.525
v25 25 0 dc -1.525
v27 27 0 dc -1.525
v29 29 0 dc -1.525
v31 31 0 dc -1.525
v33 33 0 dc -1.525
v35 35 0 dc -1.525
v37 37 0 dc -1.525
v39 39 0 dc -1.525
v41 41 0 dc -1.525
v43 43 0 dc -1.525
v45 45 0 dc -1.525
v47 47 0 dc -1.525
v49 49 0 dc -1.525
*
c51 51 0 0.05p
c52 52 0 0.05p
c53 53 0 0.05p
c54 54 0 0.05p
c55 55 0 0.05p
c56 56 0 0.05p
c57 57 0 0.05p
c58 58 0 0.05p
c59 59 0 0.05p
c60 60 0 0.05p
c61 61 0 0.05p
c62 62 0 0.05p
```

```
c63 63 0 0.05p
c64 64 0 0.05p
c65 65 0 0.05p
c66 66 0 0.05p
c67 67 0 0.05p
c68 68 0 0.05p
c69 69 0 0.05p
c70 70 0 0.05p
c71 71 0 0.05p
c72 72 0 0.05p
c73 73 0 0.05p
c74 74 0 0.05p
c75 75 0 0.05p
*
.options nomod limpts=10000 gmin=1e-10
.width in=80 out=132
.dc v1 -1.6 -1.4 0.005
.print dc v(102) v(104) v(106) v(108)
.print dc v(2) v(4) v(6) v(8)
.print dc v(151) v(152) v(153) v(154)
.print dc v(51) v(52) v(53) v(54)
.print dc v(90) v(92) v(190) v(192) v(91) v(98)
*-------------------------------------------------
* bias circuits
*-------------------------------------------------
ig1 550 91 dc 10u
ig2 550 93 dc 10u
mx1 91 91 560 560 nm w=20u l=8u
mx2 93 93 560 560 nm w=20u l=8u
mx3 98 98 560 560 nm w=8u l=2u
mx4 99 99 560 560 nm w=8u l=2u
is1 550 98 dc 10u
is2 550 99 dc 10u
*-------------------------------------------------
ig11 550 191 dc 10u
ig12 550 193 dc 10u
mx11 191 191 560 560 nm w=20u l=8u
mx12 193 193 560 560 nm w=20u l=8u
mx13 198 198 560 560 nm w=8u l=2u
mx14 199 199 560 560 nm w=8u l=2u
is11 550 198 dc 10u
is12 550 199 dc 10u
*-------------------------------------------------
.subckt winner 1 3 13 90 91 92 93 98 99 550 560
*1=vin 3=vout1 13=vout2 90=vcs1 91=vcs2 92=vg1 93=vg2
*98=vb1 99=vb2 550=vdd 560=vss
m1 2 1 90 560 nm w=8u l=2u ad=40p as=40p pd=18u ps=18u
m2 2 2 550 550 pm w=8u l=4u ad=40p as=40p pd=18u ps=18u
```

```
m3 3 2 550 550 pm w=8u l=4u ad=40p as=40p pd=18u ps=18u
m4 3 98 560 560 nm w=8u l=2u ad=40p as=40p pd=18u ps=18u
m5 90 91 560 560 nm w=20u l=8u ad=100p as=100p pd=30u ps=30u
m11 12 3 92 560 nm w=8u l=2u ad=40p as=40p pd=18u ps=18u
m12 12 12 550 550 pm w=8u l=4u ad=40p as=40p pd=18u ps=18u
m13 13 12 550 550 pm w=8u l=4u ad=40p as=40p pd=18u ps=18u
m14 13 99 560 560 nm w=8u l=2u ad=40p as=40p pd=18u ps=18u
m15 92 93 560 560 nm w=20u l=8u ad=100p as=100p pd=30u ps=30u
.ends winner
*---------------------------------------------------
.model nm nmos level=2 ld=0.250000u tox=395.000000e-10
+ nsub=5.706000e+15 vto=0.74457 kp=5.987000e-05 gamma=0.4978
+ phi=0.6 uo=684.8 uexp=0.185773 ucrit=40591.3
+ delta=1.44455 vmax=73636.6 xj=0.150000u lambda=3.557268e-02
+ nfs=4.667514e+12 neff=1 nss=1.000000e+12 tpg=1.000000
+ rsh=28.940000 cgdo=3.278301e-10 cgso=3.278301e-10 cgbo=7.168240e-10
+ cj=1.201000e-04 mj=0.675600 cjsw=5.545000e-10 mjsw=0.263600 pb=0.800000
* weff = wdrawn - delta_w
* the suggested delta_w is 0.58 um
.model pm pmos level=2 ld=0.250000u tox=395.000000e-10
+ nsub=6.078000e+15 vto=-0.723651 kp=2.404000e-05 gamma=0.5138
+ phi=0.6 uo=274.99 uexp=0.300442 ucrit=35276.9
+ delta=1.000000e-06 vmax=94066.1 xj=0.050000u lambda=5.530815e-02
*+ lambda=0
+ nfs=2.697453e+12 neff=1.001 nss=1.000000e+12 tpg=-1.000000
+ rsh=109.400000 cgdo=3.278301e-10 cgso=3.278301e-10 cgbo=5.553545e-10
+ cj=2.357000e-04 mj=0.558900 cjsw=2.785000e-10 mjsw=0.316100 pb=0.800000
* weff = wdrawn - delta_w
* the suggested delta_w is -0.16 um
*---------------------------------------------------
.end
```