USCIPI REPORT #203

VLSI Image Compression

by

Wai-Chi Fang

May 1992

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

Acknowledgments

I would like to express my deepest gratitude to Professor Bing J. Sheu, my research adviser, for his constant guidance and encouragement throughout the course of my Ph.D. research work. I wish to extend my sincere appreciation to Professor Irving S. Reed and Professor Dominic Cheung for serving on my dissertation committee. I would also like to thank them along with Professor Rama Chellappa and Professor B. Keith Jenkins for serving on my qualifying examination committee.

I am very grateful to Professor Hans H. Kuehl, Chairman of EE-Electrophysics Department; Professor Melvin A. Breuer, Chairman of EE-Systems Department; Professor Leonard Silverman, Dean of Engineering School; and Mrs. Ramona Gordon, Senior Administrative Assistant, for providing me the opportunity to pursue my Ph. D. studies under such a great research environment at USC. Encouragement from Professor Jerry M. Mendel, Director of Signal and Image Processing Institute (SIPI); Professor Mike Arbib, Director of Center for Neural Engineering (CNE); Professor Ramakant Nevatia, Director of Institute for Robotics and Intelligent Systems (IRIS); Professor Stephen R. Forrest, Director of National Center for Integrated Photonic Technology (NCIPT) has been very helpful. Assistance from Mr. Cesar Pina, Director of the MOSIS Service and other members of USC/Information Sciences Institute at Marina del Rey, CA is also highly appreciated.

I would like to thank Professor Sun-Yuan Kung of Princeton University, while he taught at USC, on VLSI array processors. Valuable discussions with my colleagues at Jet Propulsion Laboratory, especially Dr. Chi-Yung Chang and Dr. John C. Curlander on radar imagery compression, Dr. Jun-Ji Lee and Mr. Robert Rice on lossless data compression are very fruitful. I greatly enjoyed discussion with Professor B. Keith Jenkins and Dr. Chein-Hsun Wang on artificial neural networks, Dr. Yi-Tong Zhou and Professor Rama Chellappa on optical flow computing.

I would like to thank my fellow graduate students in the VLSI Signal Processing Laboratory for their constant discussions and friendships. I appreciate Dr. Bang Won Lee, Dr. Ji-Chien Lee, and Dr. Wen-Jay Hsu for discussions on VLSI neural chip design. I thank Oscal T.-C. Chen for his valuable contribution to the software simulation. I also thank Joongho Choi, Ms. Chia-Fen Chang, Ms. Min Chen, Sudhir Gowda, and Sa Hyun Bang for the circuit simulation and layout.

I wish to thank many people in Jet Propulsion Laboratory, California Institute of Technology; especially Dr. Edward C. Stone, Director; Dr. Charles Elachi, Assistant Laboratory Director; Dr. William J. Weber, Manager of Telecommunications Science and Engineering Division; Dr. Fuk K. Li, Manager of Radar Science and Engineering Section; Mr. D. Eisenman, Mr. R. H. Nixon, Mrs. Mimi Paller, and Mrs. Barbara Cline, for their encouragement and assistant. I am grateful for the tuition support and the Profession Development Program provided by Jet Propulsion Laboratory.

Finally, I would like to dedicate this work to my wife, Tsuey-Yuh, our children, Annie, Sonia, and Jonathan, my parents, Chung-Hsun Fang and Su-Yueh Lin Fang, and my parents-in-law, Jong-Tang Huang and Han-Hsiao Liao Huang, for their love, understanding, patience, support during my doctoral studies. The constant prayers of my mother and the cheerful faces of my children greatly help me to complete my Ph.D. studies.

This research has been performed since Summer 1989 while I keep a full-time position at Jet Propulsion Laboratory. The Interactive Television Program at the School of Engineering has been extremely valuable for most of my course work. The research was partially supported by DARPA under Contract No. MDA972-90-C-0037 and by Faculty Research and Innovation Fund No. 22-1502-9759 to Professor Bing J. Sheu from University of Southern California and Contributions from TRW, Inc., Samsung Electronics Co. and NKK Corp.

Table of Contents

List of Figures

List of Tables

Abstract

The increasing demands of speed and performance in data compression applications urge the VLSI data compression research. Our VLSI data compression research has been inspired by the neural networks, the parallel/pipelined processing, and the VLSI technologies. The primary objective was to develop effective image compression algorithm and their associated VLSI processors. The scope of our VLSI image compression research covers: (a) lossless image compression and VLSI processors, (b) lossy image compression and VLSI processors, and (c) neural network applications to image compression and VLSI neuroprocessors. The significant research results are presented in the following:

A high-speed image compression processor based on VLSI design of systolic binary tree-searched vector quantizer has been developed. Simulation results show that this design is applicable to many types of image data and capable of producing good reconstructed data quality at high compression ratios. Various design aspects for the binary tree-searched vector quantizer, that include the functionality, testability, and fault tolerance, were thoroughly investigated for VLSI implementation. A specific 8-level binary tree-searched vector quantizer can be realized on a custom VLSI chip that includes a systolic array of eight identical processors and a hierarchical memory of eight subcodebook memory banks. The total transistor count is about 300,000 and the die size is about 8.7 x 7.7 mm^2 in a 1.0 μm CMOS technology.

An adaptive electronic neural network processor that has been developed for high-speed image compression based upon a frequency-sensitive self-organization algorithm. Performances of this self-organization network and a conventional algorithm for vector quantization are compared. The proposed method is quite efficient and can achieve near-optimal results. The neural network processor includes a pipelined codebook generator and a paralleled vector quantizer, which obtains a time complexity *O(1)* for each quantization vector. A mixed-

signal design technique with analog circuitry to perform massively parallel computation and digital circuitry to handle multiple-bit address information is used. The prototype neural network processor chip for a 25-dimensional adaptive vector quantizer of 64 codewords was designed, fabricated, and tested. It includes 25 input neurons, 25 x 64 synapse cells, 64 distortion-computing neurons, a winner-take-all circuit block, and a digital index encoder. It occupies a silicon area of 4.6 x 6.8 mm^2 in a 2.0-μm scalable CMOS technology and provides a computing capability as high as 3.2 billion connections per second.

A neural network based motion compression algorithm and its associated VLSI neuroprocessor have been developed. The neuroprocessor design is based on a locally connected multiple competitive neural network developed for high performance optical flow computing systems. The proposed VLSI neuroprocessor design can achieve a high-speed wide-range motion estimation and thus an efficient image sequence compression by taking advantage of the massively parallel neural computing architecture and VLSI technology. An extendible VLSI *neuroprocessor* has been designed with a silicon area of 2,482 x 5,636 λ^2 in MOSIS scalable CMOS process. The mixed analog-digital design techniques are utilized to achieve compact and programmable synapses with gain-adjustable neurons and winner-take-all cells for massively parallel neural computation. Measured results of the programmable synapse, summing neuron, and winner-take-all circuitry are presented. A 1.25 x 1.17 cm^2 chip in a submicron CMOS technology can accommodate 128 velocity-selective neuroprocessors and achieve 166.4 Giga connections per second. Computing of optical flow using one neural chip can be accelerated by a factor of 379 than a Sun-4/260 workstation. Real-time motion estimation on industrial video images is practical using an extended array of VLSI neuroprocessors. Actual examples on moving vehicles are presented.

An efficient VLSI pipelined processor design for high-speed lossless compression based on Rice algorithm has been developed. The Rice algorithm is an adaptive lossless coding scheme that provides near-

optimal performance over a broad range of data entropies. The Rice algorithm is also an efficiently implementable scheme for VLSI realization. A VLSI pipelined architecture was designed to allow compact implementation of a single-chip VLSI compressor. This lossless compressor is named UNC-PSI14,K+ since it implements an advanced version of the Rice 's universal noiseless coding method called PSI14,K+. The chip layout was generated for a 1.0-micron CMOS technology. It occupies a compact chip area of 5.1 x 5.3 mm^2, with 49,000 transistors, 57 input/output pads, and 6 power/ground pads. The total power dissipation is 0.4 watts at the 40 MHz system clock with a 50% switching duty cycle. This compressor chip is mounted in an 84-pin pin-grid-array package. It can operate up to 40 Mpixels/sec. The potential applications of the proposed lossless compressor include database management systems, scientific instruments, CAE workstations, desktop computing machines, and the data systems that require high-speed compression without fidelity loss.

Chapter 1

Introduction

1.1 Motivation

The goal of data compression is to reduce the communication and storage costs for the data systems where reduction in the volume of transmitted or recorded data is important. The data compression field was intensively studied over the past four decades [1.1-1.11]. In the research field of data compression, important contributions have come from information sciences, computer sciences, signal and image processing, computer engineering, VLSI technology, etc. The increasing demands of speed and performance in data compression applications urge the VLSI data compression research. The major driving forces behind this trend include:

- More demand is placed on the communication and storage sub-systems as technology enhancement advances. Data compression techniques are employed in several key driving applications such as high definition TV, integrated information system, integrated service data networks, and the space-science data system.
- Technology improvements make compact integration of sophisticated data compression systems possible. These technology improvements include the advances of VLSI fabrication technologies and parallel processing architecture such as neural networks [1.12-1.16].

More efforts are still needed to develop high-speed high-performance data compression systems. To achieve the goal of efficient data compression, the following issues need to resolve:

- Due to different characteristics of various data sources, a sophisticated compression system is required to handle all types of source data such as speech, image, video, text, and binary files.
- Due to the intensive computation and real time processing requirements, the implementation of a high-speed data compression system is required to be with special-purpose parallel/pipelined processors.
- The characteristics of compressed data also inevitably add additional requirements to the data systems. Such requirements include more stringent communication quality and buffering management.

Figure 1.1 shows an end-to-end high rate data system diagram. To achieve a reliable high-speed communication and storage, the source data compression and channel coding become essential portions of the advanced high rate data system. Improvements in the data compression algorithm, architecture, software and hardware implementation, and associated system-level design issues are critical to effectively achieve the goal of data compression. Our VLSI image compression research has been motivated by a need of the high-speed high-performance data compression and inspired by the neural networks, the parallel/pipelined processing, and the VLSI technologies.

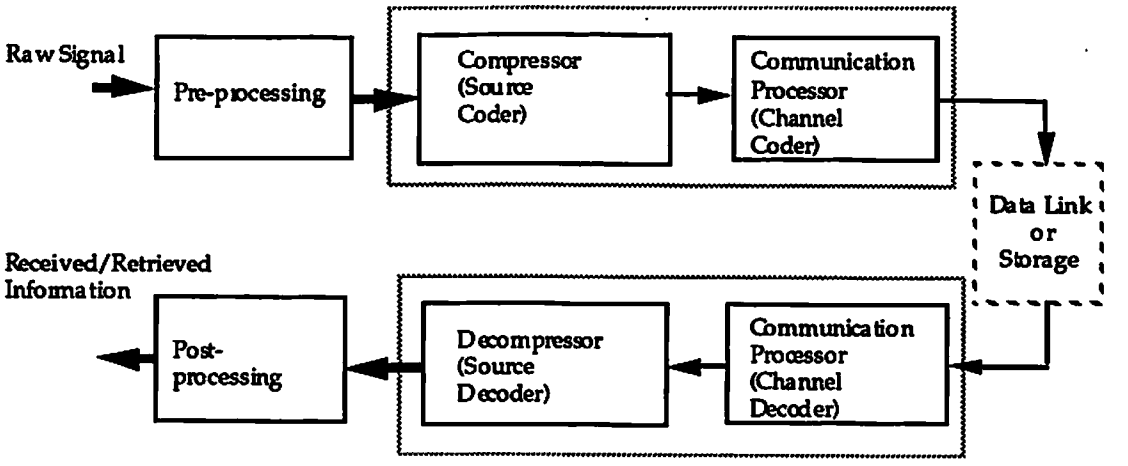


Fig. 1.1 An end-to-end high rate communication and/or storage system with the source and channel coding.

1.2 Scope and Objective

The scope of our VLSI image compression research covers:

(a) lossless image compression and VLSI processors,

(b) lossy image compression and VLSI processors, and

(c) neural network applications to image compression and VLSI neuroprocessors.

Our work in each above-mentioned research topic includes:

(a) system-level simulation to compare performances of the candidate algorithms for a targeted application,

(b) VLSI architecture design of the selected algorithm,

(c) VLSI functional and structural circuit and layout design,

(d) prototype chip design and fabrication, and

(e) testing and measurement.

The primary objective for each research topic was to develop an effective algorithm and its associated VLSI processor. The ultimate objective was to develop a high-speed high-performance image data compression system in special silicon chips or wafers.

1.3 Approach: An Integrated Study of VLSI Image Compression

Study on the subject of VLSI image compression involves a broad spectrum of disciplines that include application, algorithm, architecture, and implementation. Figure 1.1 depicts an integrated design methodology that is adopted for our VLSI image compression studies.

System Study The goal of the system study is to develop feasible system specifications. The scope of the system study covers: (a) understanding of the targeted application, (b) investigation of the implications of data compression on the overall system parameters such as link bandwidth, storage capabilities, compression needs, data quality, and so on, (c) system simulation on targeted applications, (d) system refinement to facilitate the feasibility, and (e) development of system specifications.

Algorithm Study The goal of the algorithm study is to develop effective algorithm specifications. The scope of the algorithm study covers: (a) understanding of the system specifications, (b) a survey of candidate algorithms, (c) algorithm analysis on targeted data sample sets, (d) system and algorithm refinement to facilitate the implementation, and (e) development of algorithm specifications.

Architecture Study The goal of the architecture study is to develop balanced hardware/software architecture specifications. The scope of the architecture study covers (a) understanding of the algorithm specifications, (b) parallelism extraction and mapping of the algorithm into architecture, (c) architecture evaluation with software/hardware implementation technologies, (d) algorithm and architecture refinement to facilitate the VLSI implementation, and (e) development of architecture specifications.

VLSI Design and Simulation The goal of the VLSI design and simulation is to develop functional chip specifications. The scope of this task covers (a) understanding of the architecture specifications, (b) select a chip design style/method and associated tools, (c) design functional building blocks and structural primitives to realize the targeted architecture, (d) simulate the

functional design, (e) verify the physical design, (f) design refinement to facilitate the speed-area tradeoffs, the worst-case timing margins, and the testability, and (g) generate a chip specification database.

VLSI Fabrication and Test The goal of the VLSI fabrication and test is to realize the chip specification onto a real chip. The MOSIS provides chip fabrication service through a variety of chip foundries [1.16]. The chip is tested to be functional by using a chip tester and a system testbed.

Design and implementation of the application-specific processors in VLSI are a highly challenging and iterative process. This integrated design method for our VLSI image compression research has been very successful. The success relies on a fundamental understanding of application, algorithm, architecture, and VLSI implementation,

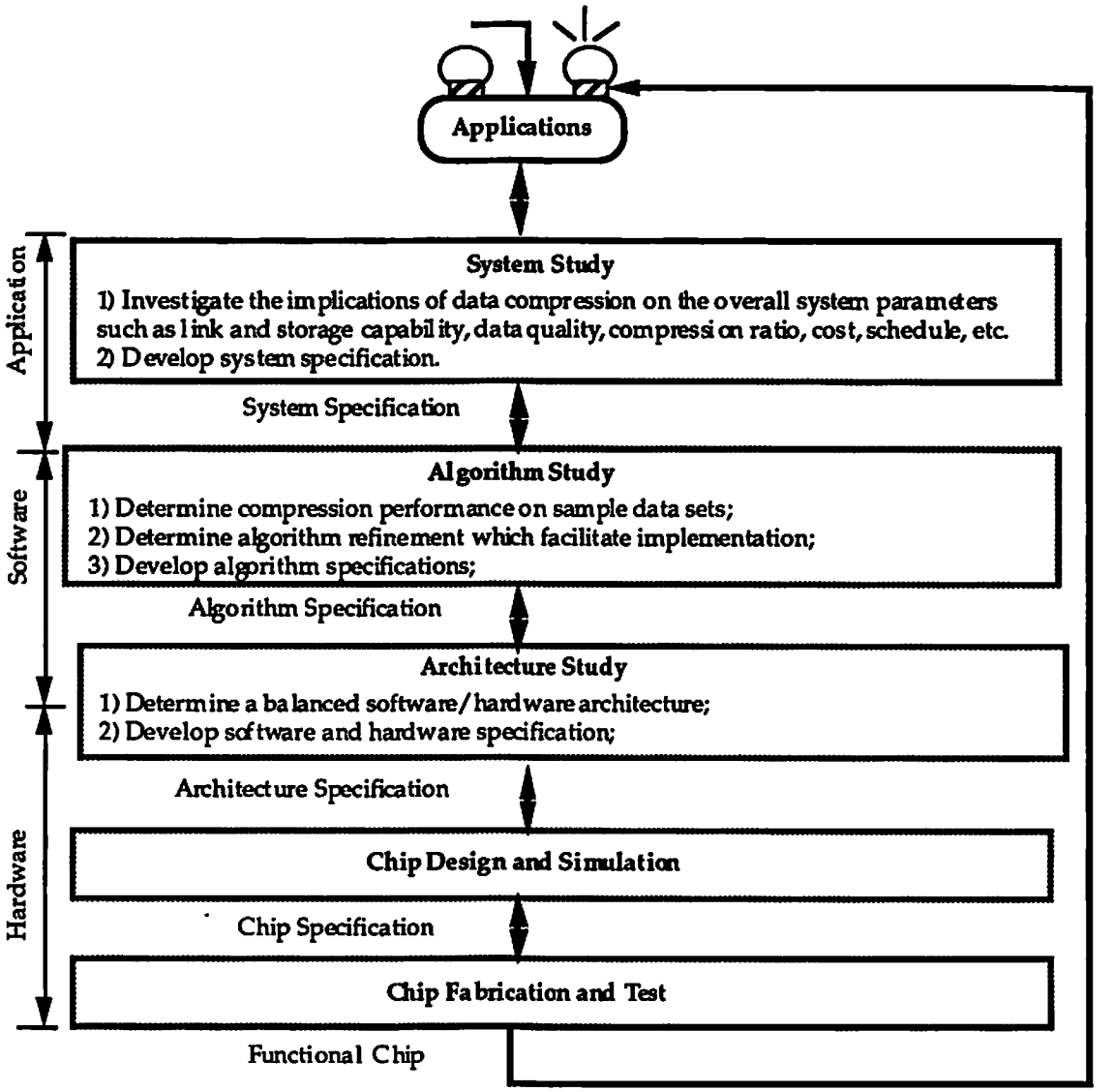


Fig. 1.2 Design methodology for the VLSI image compression processor.

1.4 Overview of the Dissertation

This dissertation contributes to real-time image compression research with emphasis on the hardware development, in particular to the high-speed high-performance VLSI image compressor design. Figure 1.3 shows an organization of various image compression research subjects discussed in this dissertation. The following selected chapters are covered to present our studies on the subject of VLSI image compression:

Chapter 2 starts with a brief introduction of widely used data compression algorithms. Data compression can be divided into two categories: lossless compression and lossy compression. Four lossy data compression algorithms were investigated: predictive coding, block truncation coding, transform coding, and vector quantization. Four types of losses coding algorithm were also investigated: Huffman codes, Rice codes, Limped-Ziv codes, and arithmetic codes. Furthermore, the image compression algorithm using neural-network paradigms were investigated. The limitations, assumptions, and applications for the algorithms were discussed.

Chapter 3 presents a high-speed image compression processor based on VLSI design of systolic binary tree-searched vector quantizer. Simulation results show that this design is applicable to many types of image data and capable of producing good reconstructed data quality at high compression ratios. Various design aspects for the binary tree-searched vector quantizer, that include the functionality, testability, and fault tolerance, were thoroughly investigated for VLSI implementation. A specific 8-level binary tree-searched

vector quantizer can be realized on a custom VLSI chip that includes a systolic array of eight identical processors and a hierarchical memory of eight subcodebook memory banks. The total transistor count is about 300,000 and the die size is about 8.7 x 7.7 mm^2 in a 1.0 μm CMOS technology. The throughput rate of this high-speed VLSI compression system is 25M pixels per second and its equivalent computation power is 400 MIPS. Portions of this chapter were presented at the 1990 IEEE *VLSI Signal Processing, Workshop* at San Diego, CA. [1.17], the 1991 Advanced *Research in VLSI Conference* at Santa Cruz, CA [1.18], and the 1991 IEEE *Workshop on Visual Signal Processing and Communications*, at Taiwan, Republic of China [1.19].

Chapter 4 presents an adaptive electronic neural network processor that has been developed for high-speed image compression based upon a frequency-sensitive self-organization algorithm. Performances of this self-organization network and a conventional algorithm for vector quantization are compared. The proposed method is quite efficient and can achieve near-optimal results. The neural network processor includes a pipelined codebook generator and a paralleled vector quantizer, which obtains a time complexity *O(1)* for each quantization vector. A mixed-signal design technique with analog circuitry to perform neural computation and digital circuitry to handle multiple-bit address information is used. The prototype neural network processor chip for a 25-dimensional adaptive vector quantizer of 64 codewords was designed, fabricated, and tested. It includes 25 input neurons, 25 x 64 synapse cells, 64 distortion-computing neurons, a winner-take-all circuit block, and a digital index encoder. It occupies a silicon area of 4.6 x 6.8 mm^2 in a 2.0-μm scalable CMOS technology and provides a

computing capability as high as 3.2 billion connections per second. The experimental results for this neural-based vector quantizer chip and the winner-take-all circuit test structure were also presented. Portions of this chapter were presented at *the 1991 IEEE first Data Compression Conference* at Snowbird, Utah [1.20], the *IEEE international Conference on Acoustics, Speech and Signal Processing*, at Toronto, Ontario, Canada [1.21], the 1991 *International Joint Conference on Neural Networks*, at Seattle [1.22], and the *IEEE Trans. on Neural Network*, May 1992 [1.23].

Chapter 5 describes a neural network based motion compression algorithm and its associated VLSI neuroprocessor design. The neuroprocessor design is based on a locally connected multiple competitive neural network developed for high performance optical flow computing systems. The proposed VLSI neuroprocessor design can achieve a high-speed wide-range motion estimation and thus an efficient image sequence compression by taking advantage of the massively parallel neural computing architecture and VLSI technology. An extendible VLSI *neuroprocessor* has been designed with a silicon area of 2,482 x 5,636 λ^2 in MOSIS scalable CMOS process. The mixed analog-digital design techniques are utilized to achieve compact and programmable synapses with gain-adjustable neurons and winner-take-all cells for massively parallel neural computation. Measured results of the programmable synapse, summing neuron, and winner-take-all circuitry are presented. A 1.25 x 1.17 cm^2 chip in a submicron CMOS technology can accommodate 128 velocity-selective neuroprocessors and achieve 166.4 Giga connections per second. Computing of optical flow using one neural chip can be accelerated by a factor of 379 than a Sun-4/260 workstation. Real-time

motion estimation on industrial video images is practical using an extended array of VLSI neuroprocessors. Actual examples on moving vehicles are presented. Portions of this chapter were presented at the 1990 *IEEE International Conference of Computer Design at* Cambridge, MA [1.24], the 1991 *IEEE international Conference on Acoustics, Speech and Signal Processing,* at Toronto, Ontario, Canada [1.25], and has been accepted for publication in the IEEE Trans. on Neural Networks [1.26].

Chapter 6 presents an efficient VLSI pipelined processor design for high-speed lossless compression based on Rice algorithm. The Rice algorithm is an adaptive lossless coding scheme that provides near-optimal performance over a broad range of data entropies. The Rice algorithm is also an efficiently implementable scheme for VLSI realization. A VLSI pipelined architecture was designed to allow compact implementation of a single-chip VLSI compressor. This lossless compressor is named UNC-PSI14,K+ since it implements an advanced version of the Rice 's universal noiseless coding method called PSI14,K+. The chip layout was generated for a 1.0-micron CMOS technology. It occupies a compact chip area of 5.1 x 5.3 mm^2, with 49,000 transistors, 57 input/output pads, and 6 power/ground pads. The total power dissipation is 0.4 watts at the 40 MHz system clock with a 50% switching duty cycle. This compressor chip is mounted in an 84-pin pin-grid-array package. It can operate up to 40 Mpixels/sec. The potential applications of the proposed lossless compressor include database management systems, scientific instruments, CAE workstations, desktop computing machines, and the data systems that require high-speed compression without fidelity loss. Portions of this chapter were presented at

the 1987 IEEE Aerospace EASCON Conference, Washington [1.27], the 1988 SPIE Conference, Los Angeles [1.28], NASA Technology Briefs, October 1989 [1.29], and *the 1991 IEEE First Data Compression Conference*, Snowbird Utah [1.30].

Chapter 7 discusses potential future research and development directions that are worth of further investigation.

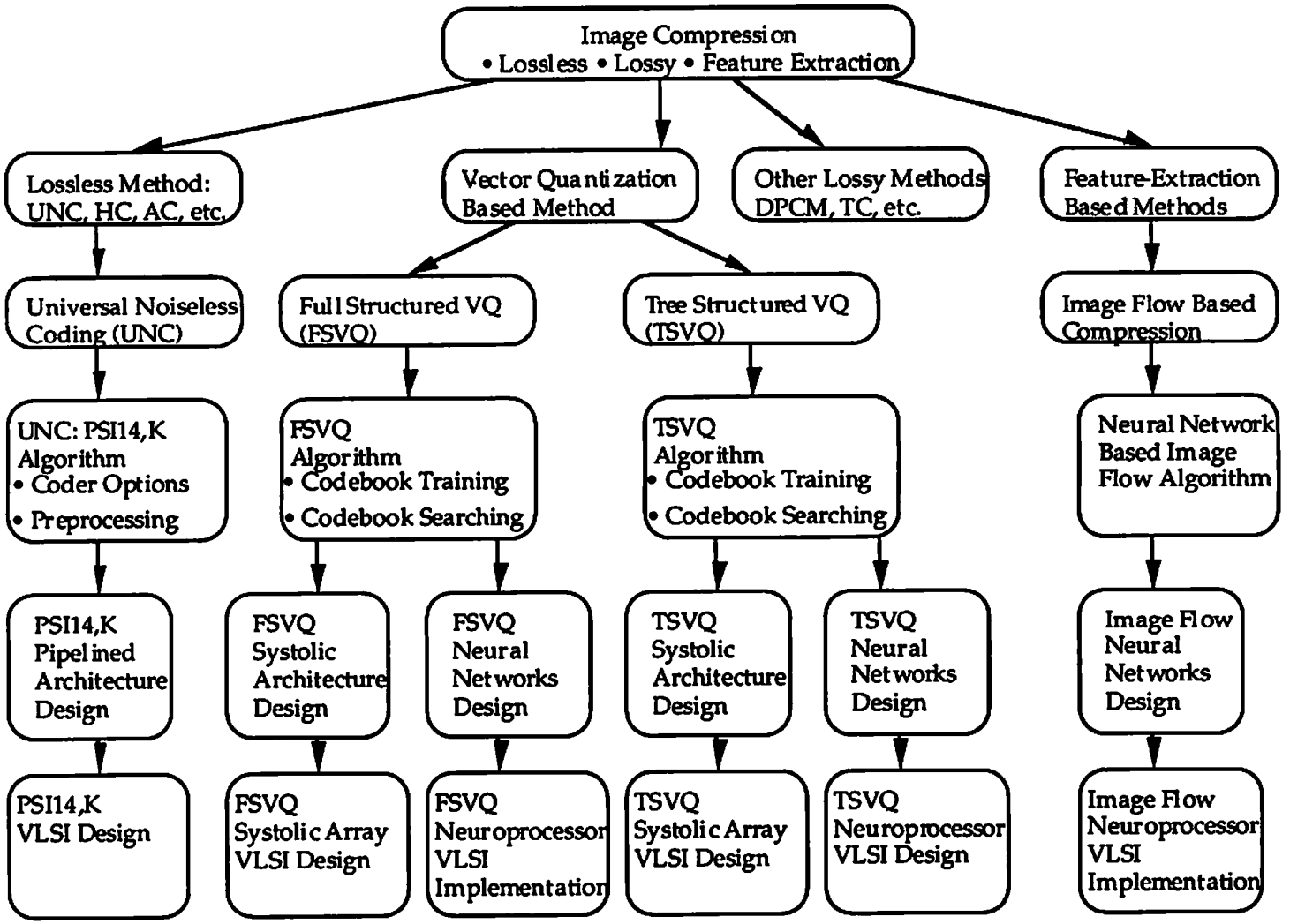


Fig. 1.3 Various image compression method discussed in this dissertation.

References

[1.1] R. M. Gray, *Source Coding Theory*, Kluwer Academic Publishers: Boston, MA, 1990.

[1.2] A. Gersho, R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers: Boston, MA, 1991.

[1.3] J. A. Storer, *Data Compression- Methods and Theory*, Computer Science Press, Rockville, Maryland, 1988.

[1.4] T. J. Lynch, *Data Compression- Techniques and Applications*, Lifetime Learning, Wadsworth, 1985.

[1.5] N. S. Jayant (Ed.), *Waveform Quantization and Coding*, IEEE Press, 1976.

[1.6] N.S. Jayant, P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, 1984

[1.7] R. J. Clarke, *Transform Coding of Images*, Academic Press, 1985.

[1.8] A. N. Netravali, B. G. Haskell, *Digital Pictures: Representation and Compression*, Plenum Press, 1988.

[1.9] H. Abut (Ed.), *Vector Quantization*, IEEE Press, 1990.

[1.10] M. Rabbani, P. Jones. *Digital Image Compression*, SPIE Publications, 1991.

[1.11] A. K. Jain, "Image data compression: a review," Proc. of IEEE, vol. 69, no. 3, pp. 349-389, Mar. 1981.

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

[1.13] C. A. Mead, *Analog VLSI and Neural Systems*, Addison Wesley: New York, NY, 1989.

[1.14] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design- A Systems Perspective*, Addison Wesley: Reading, MA, 1985.

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

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

[1.17] W.-C. Fang, C.-Y. Chang, B. J. Sheu, "Systolic tree-searched vector quantizer for real-time image compression," *VLSI Signal Processing, IV*, H. S. Moscovitz, Kung Yao, R. Jain (Eds.), pp. 352-361 (Chap. 34), IEEE PRESS: Piscataway, NJ, 1991.

[1.18] W.-C. Fang, B. J. Sheu, " VLSI adaptive image compression," *Advanced Research in VLSI: Proceedings of the 1991 University of California/Santa Cruz Conferences*, Carlo H. Sequin (Ed.), pp. 371-386, MIT Press, Cambridge, MA, 1991,

[1.19] O. T.-C. Chen, B. J. Sheu, W.-C. Fang, " Adaptive codebook construction and real-time hardware implementation for binary tree-searched vector quantizer," *IEEE Workshop on Visual Signal Processing and Communications*, pp. 35-38, Taiwan, Republic of China, June 1991.

[1.20] W.-C. Fang, B. J. Sheu, O. T.-C. Chen "A neural network based VLSI vector quantizer for real-time image compression," *IEEE Proceedings on the first Data Compression Conference*, Snowbird Utah, pp. 342-351, IEEE Computer Society Press: Los Alamitos, Apr., 1991.

[1.21] B. J. Sheu, W.-C. Fang "Real-time high-ratio image compression using adaptive VLSI neuroprocessor," *IEEE Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. Toronto, Ontario, Canada, May, 1991.

[1.22] W.-C. Fang, B. J. Sheu, O. T.-C. Chen, " A real-time VLSI neuroprocessor for adaptive image compression based upon frequency-sensitive competitive learning," *International Joint Conference on Neural Networks*, pp. 429-435, Seattle, WA, July, 1991.

[1.23] W.-C Fang, B. J. Sheu, O. T.-C. Chen, J. Choi "A VLSI neural processor for image data compression using self-organization networks," *IEEE Trans. on Neural Networks: Special Issue on Neural Network Hardware*, vol. 3, no. 2, May 1992.

[1.24] W.-C. Fang, B. J. Sheu, "Real-time computing of optical flow using adaptive VLSI neuroprocessors," *IEEE International Conference of Computer Design*, pp. 122-125, Cambridge, MA, Oct. 1990.

[1.25] W.-C. Fang, B. J. Sheu, J.-C. Lee, "A Neuroprocessor for real-time image flow computation," *IEEE Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. 2413-2416, Toronto, Ontario, Canada, May, 1991.

[1.26] J.-C. Lee, B. J. Sheu, W.-C. Fang, " VLSI neuroprocessors for video motion detection," IEEE Trans. on Neural Networks, (accepted for publication in April, 1992)

[1.27] J.-J. Lee, W.-C. Fang, R. F. Rice, "Real-time data compressor for EOS-class missions," Proceeding of the 1987 IEEE Aerospace EASCON Conference, Washington, D. C., Oct. 1987.

[1.28] J. -J. Lee, W.-C. Fang, R. F. Rice, "VLSI universal noiseless compressor for EOS-class missions," Proceeding of the 1988 SPIE Conference, Los Angeles, California, Feb. 1988.

[1.29] Robert F. Rice, Jun-Ji Lee, Wai-Chi Fang, " VLSI universal noiseless coder", *NASA Tech Briefs*, vol. 13, no.10, Oct. 1989

[1.30] R. Anderson, J. Bowers, W.-C. Fang, D. Johnson, J.-J. Lee, R. Nixon, "A very high speed noiseless data compression chip for space imaging applications", Proceedings IEEE Data Compression Conference, Snowbird, Utah, April 1991.

Chapter 2

Data and Image Compression

Efficient compression of data would significantly decrease both the communication and storage costs [2.1]. Data compression can be roughly divide into two categories, lossless compression and lossy compression. A *lossless* compression algorithm seeks to represent a given data set with the fewest number of bits possible without any fidelity loss while a *lossy* compression algorithm provides higher compression factors at the expense of some distortions in the reconstructed data set [2.2].

In this chapter, we focus on these algorithms have been investigated for an integrated information processing system [2.12] which has various signal representations including speech, image, video, text, binary file, etc. Section 2.1 review the traditional lossless data compression algorithms. Section 2.2 review the traditional lossy data compression algorithms. Section 2.3 investigated the neural network based data and image compression. The limitations, assumptions, and applications for each algorithm were examined and reported.

2.1 Lossless Compression

Lossless compression is usually required in such situations where the compression system must be designed without prior knowledge of the

structure or end use of the original data. Lossless coding is a data compression techniques that is employed to compress the data without inducing any distortion in the reconstructed data. Its maximum compression ratio is limited by the entropy of the source data. There are two types of lossless coding algorithm: Huffman coding [2.2,3] and universal noiseless coding [2.7,8]. The universal noiseless coding algorithm only needs to know the probability ordering of the source data while the Huffman coding algorithm requires the knowledge of the probability distribution. The assumption on the knowledge of the probability ordering is more realistic since the source data can be preprocessed (e.g. by taking difference of adjacent pixels). The preprocessed data (e.g. difference image) typically assumes some degree of probability ordering (e.g. higher probability for smaller difference in adjacent pixels).

For this reason, we focus only on the universal noiseless coding technique due to lack of any prior information on the source data statistics. The advantages of the universal noiseless coding are: 1) No distortion in the reconstructed data; 2) No need to know the probability distribution; and 3) simple encoding and decoding procedures. Its drawbacks are: 1) Algorithm performance is more sensitive to channel errors; 2) Output data rate (i.e., compression ratio) is not controllable; and 3) Maximum compression ratio is limited by the source entropy.

Among existing lossless data compression algorithms [2.3-8], the universal noiseless coding (UNC) is an effective method for producing reconstructed data without any fidelity loss. This adaptive algorithm provides excellent performance over a broad range of data contents. A high-

speed UNC coder is therefore proposed to implemented at a low hardware cost by using a VLSI pipelined architecture [2.9-10]. Notice that the noiseless coding technique alone is not feasible to reduce the data volume as required for the advanced data system. It is better suited as a post-processing stage for further reduction of data compressed by a lossy compression algorithm to maximize the compression to distortion noise ratio.

2.2 Lossy Compression

Four data compression algorithms were investigated for the advanced data system. There are predictive coding, block truncation coding, transform coding, and vector quantization algorithms. The limitations, assumptions, and applications for each algorithm were examined as the following:

2.2.1 Predictive Coding

The predictive coding technique is a simple coding algorithm that provides a relatively small compression ratio with reasonably good image quality [2.12]. Its major limitation is that it cannot compress data below 1 bit per pixel and its image quality at 1 bit per pixel is marginal. Another limitation is that it provides very limited flexibility in selection of compression ratio versus data quality. If the source data is stationary, the predictor need not be updated, which makes it possible for real-time applications. However, if the source data statistics vary with time, buffering

of the data and retaining of the prediction coefficients is required. Predictive coding is best suited for applications where very simple encoders and decoders are required and the compression ratio requirements are low (i.e., less than 4:1).

Compared to the block truncation coding algorithm, it offers about the same data quality but provides less flexibility in the compression ratio versus data quality trade-off. Also its prediction coefficients would require training of data sets if the source statistics vary significantly. Compared to the transform coding and vector quantization algorithms, the predictive coding algorithm cannot achieve as high a compression ratio and its data quality is usually 2 to 3 dB worse.

2..2.2 Block Truncation Coding Algorithm

The block truncation coding (BTC) algorithm [2.16] has the following advantages: 1) simple encoding and decoding procedures; 2) small memory requirements; 3) adaptivity to local image statistics; and 4) selectable compression ratios up to K:1. where K is the number of bits per sample in the original data. Its major limitation is similar to that of predictive coding, i.e., the maximum compression ratio is K:1 and the image quality at this compression ratio is marginal. Another limitation is that at the same compression ratio, the data quality using the BTC algorithm is about 2 to 3 dB worse than that of the adaptive transform coding or the vector quantization algorithm. For the algorithm to perform well, it assumes that the statistics of each data block are primarily characterized by its first two moments. If

higher order statistics exist, the reconstructed image quality degrades. In summary, the BTC algorithm appears to be a promising candidate for a data system requiring small compression ratios, low implementation cost, and some adaptivity to data statistics.

2..2.3 Transform Coding Algorithm

The adaptive transform coding algorithm is a technique capable of compressing the image data to a user specified compression factor given that the associated image quality degradation is tolerable [2.13,14,15]. It generally yields better image quality than both the predictive coding and block truncation coding algorithms at the same compression ratio. Its major limitation is that it is computationally intensive in both encoding and decoding, requiring a large number of two dimensional transforms. Another limitation is that it requires a large storage buffer from which the source statistics are derived. It assumes that the statistics of each data block can be well characterized by the lower frequency terms in the spectra such that the bits can be more efficiently assigned to these transform coefficients that the higher frequency terms.

In summary, the adaptive transform coding algorithm is promising for a single-encoder, single-decoder application such as a high-rate communication link that requires both high compression ratio and good data quality.

2.2.4 Vector Quantization Algorithm

Vector quantization (VQ) has been show as an effective method for speech waveform coding and image data compression [2.17,18]. It is capable of producing good reconstructed image quality at high compression ratios. As compared to the adaptive transform coding, the primary advantage of the VQ algorithm is its extremely simple decoding procedure, which makes it a promising technique for the single-encoder multiple-decoder data compression systems.

Vector quantization appears to be the most viable technique for the advanced data system. This judgment is based on the reconstructed image quality at the required high compression ratio, regular data flow pattern in the encoder, and a simple decoding procedure (table look-up). By combination of tree-searched VQ and systolic processing, a high throughput compressor can be realized at a low hardware cost.

2.2.5 Summary of Results

Four lossy compression algorithm were evaluated. Among them, predictive coding and block truncation coding are two simple coding algorithms that provide reasonably good image quality at small compression ratios. Their common major limitation is that they cannot compress the data below 1 bit per pixel and the image quality at 1 bit per pixel is marginal. Transform coding and vector quantization are more complex algorithm, however, they are capable of producing good reconstructive image quality at higher compression ratios. The transform coding algorithm achieves high

performance at the cost of high complexity in both the encoder and decoder. Although vector quantization also requires a computationally intensive process in encoding the data, its decoding procedure is a simple table look-up. Moreover, the regular data flow pattern in the vector quantization encoder allows efficient implementation using both parallel and pipelined processing techniques.

The results show that the vector quantization algorithm appears to be the most viable technique for an advanced data system. Besides, universal noiseless coding algorithm is suited as a post-processing stage for further reduction of data compressed by vector quantization algorithm to maximize the compression to distortion noise ratio.

2.3 Neural Networks for Data and Image Compression

Artificial neural networks serve an important purpose that is to extract and emulate the functions and operations from biological neural networks for our next generation artifacts. Artificial neural networks also serve as vehicles to study neuroscience by synthesizing low level finding and high level hypotheses together into a artificial system that can be simulated and compared with the behaviors of living systems.

Neural network approaches appear to be very promising for intelligent information processing [2.19-28] due to their massively parallel computing structures and self-organization learning schemes. A number of studies have been reported on using artificial neural networks for VQ and image compression applications [2.29-32]. The existing works in image

compression using neural networks share the same key idea which is performing the feature extraction or classification at the source data to achieve data compression.

2.3.1 Associative Memory

J. Solomon et al. applied associative model to do multispectral image compression [2.37]. The approach is based on the minimized energy search which is state dependent. Full information retrieval can be achieved by partial input. The percentage of the spurious state is high if the codewords are not orthogonal. The scheme proposed is similar to ordinary VQ. The codebook is prepaid and stored in the Hopfield Net before encoding. The way to generate codebook is similar to supervised learning. The desired set of primitives are used to train the associative memory. Then the trained associative memory serve as a classifier(encoder) to classify(encode) the image vectors. Whenever a image vector present, the list of stored codewords is searched, and the best match codeword and its index is produced.

2.3.2 Multi-layer Perceptron

G. W. Cottrell et al. used multi-layer perceptron with error back propagation (EBP) learning to do image compression [2.33]. Multi-layer perceptrons are feed-forward nets. The back-propagation algorithm uses a gradient search technique to minimize a cost function equal to the mean square difference between the desired and the actual net outputs.

EBP is a gradient search techniques that may find a local minimum in the LMS cost function instead of the desired global minimum. The local minima in VQ corresponds to clustering two or more disjoint codewords into one. The number of presentations of training data required for convergence is very large (more than 100 passes through all the training data). A three-layer perceptron can form a codebook and then work as an ordinary vector quantization. The way to generate codebook is the supervised BPE learning. Training and testing are separate into two phase.

2.3.3 Hierarchy Associative Memory

L. D. Jackel et al. reported the experiment result of image vector quantization using associative memory model and the concept of the hierarchy neural network [2.36] . The scheme proposed is based on associative memory model. Training and testing are separate into two phase. The codebook is prepared before encoding. The desired image vectors are used as training prototypes to train the associative memory. Then the trained associative memory serve as a classifier(encoder) to classify(encode) the image vectors. The VQ proceeds by first matching image vector to the best mothership and then matching to the appropriate rowboat by using hierarchy neural networks. The scheme is similar to an ordinary VQ. An unique point has be made is the use of neural net hierarchy in considering the practical memory size of ANN chip.

2.3.4 Winner-Take-all Competitive Network

J. G. Daugman et al. modeled the actually neurobiologically recorded receptive profiles with 2-D Gabor and treat these simulated receptive profiles as the visual primitive to do vector quantization with winner-take-all model. The generation of codebook is by supervised training the winner-take-all nets with the known 4,096 visual primitives. [2.34]. In this image data VQ compression scheme, each (8x8) pixel region is represented by a single 12-bit quantity which specifies the victor in the winner-take-all competition among 4,096 candidate visual primitives. The compression ratio about 40:1 (0.1875 bits/pixel) was reported.

2.3.5 Kohonen's Self-Organizing Feature Maps

Kohonen's algorithm creates a learning vector quantizer (LVQ) by adjusting weights from common input nodes to M output nodes arranged in a two dimensional grid as shown in Fig. 2.1. Output nodes are extensively interconnected with many local connections. Continuous-valued input vectors are presented sequentially in time without specifying the desired output. After enough input vectors have been presented, weights will specify cluster or vector centers that sample the input space such that the point density function of the vector centers tends to approximate the probability density function of the input vectors [2.24]. Kohonen's net does not perform the iterative K-means training algorithm (LBG algorithm). Instead, each new pattern is presented only once and weights are modified after each presentation. The Kohonen net does, however, form a pre-specified number

of clusters as in the K-means algorithm, where the K refers to the number of clusters formed. This algorithm can perform well in noise because the number of classes is fixed, weights adapt slowly, and adaptation stops after training. This algorithm is thus an ordinary VQ when the number of clusters desired can be specified before use and the amount of training data is large relative to the number of clusters desired. It is similar to the LBG algorithm in this respect.

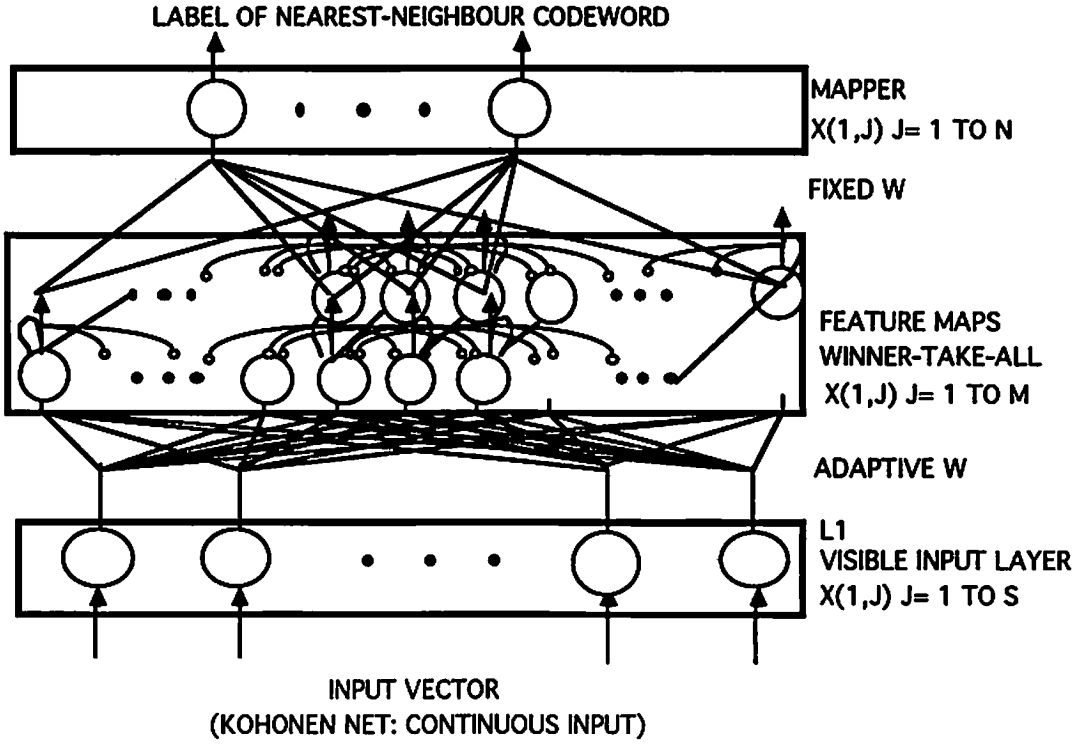


Fig. 2.1 Structure of Kohonen's self-organizing feature maps.

As shown in Fig. 2.1, the network consists of two layers of units, a linear input layer and an internal processing layer. The internal processing

layer contains inhibitory connections and provides a "top-down" training signal to connections between the two layers. Two dimensional array of output nodes used to form feature maps. Every input is connected to every output node via a variable connection weight. Kohonen's algorithm to perform VQ using self-organizing feature maps is described in the following:

Algorithm : Kohonen's self-organizing feature maps

Step 1: Initialize Weights

Initialize weights from N inputs to the M output nodes to small random values. Set the initial radius of the neighborhood of each node.

Step: Present New Input

(Note: No classification provided.)

Step 3: Compute Distance to All Nodes

Compute distance dj between the input and each output node j using

$$d_j = \sum_{i=0}^{N-1} [x_i(t) - w_{ij}(t)]^2$$

where x_i (t) is the input to node i at time t and w_{ij} is the weight from input node i to output node j at time t.

Step 4: Select Output Node with Minimum Distance (Winner)

Select the node j* as the output node with minimum d_j.

Step 5: Update Weights to Node j* and Neighbors

Weights are updated for node j^* and all nodes in the neighborhood defined by $NE_{j^*}(t)$. New weights are

$w_{ij}(t+1) = w_{ij}(t) + \eta(t)[x_i(t) - w_{ij}(t)]$

For j belong to NE_{j^*} and $0 \leq i \leq N-1$.

$\eta(t)$ is a gain term ($0 < \eta(t) < 1$) that decreases in time.

(Weights eventually converge and are fixed after the gain term is reduced to zero.)

Step 6: Repeat by Going to Step 2

2.3.6 Grossberg's Adaptive Resonance Theory (ART) Network

The Carpenter/Grossberg's Adaptive Resonance Theory (ART) network [2.26-28] consists of attentional subsystem and orienting subsystem. Familiar events are processed within an attentional subsystem, which is a two-layer competitive learning network.

ART network [2.28] forms clusters and is trained without supervision. This net implements a clustering algorithm that is very similar to the simple sequential leader clustering algorithm described in [2.41]. The ART is completely described using nonlinear differential equations, including extensive feedback. In typical operation, the differential equations can be shown to implement the clustering algorithm presented in the following:

Algorithm: Carpenter/Grossberg's Adaptive Resonance Theory (ART)

Step 1. Initialization

$t_{ij}(0) = 1$

$b_{ij}(0) = 1/(1+N)$

$0 \leq i \leq N-1$

$0 \leq j \leq M-1$

Set ρ, $0 \leq r \leq 1$

In these equations $b_{ij}(t)$ is the bottom up and $t_{ij}(t)$ is the top down connection weight between input node i and output node j at time t. These weights define the exemplar specified by output node j. The fraction ρ is the vigilance threshold which indicates how close an input must be to a stored exemplar to match.

Step 2. Apply New Input

Step 3. Compute Matching Scores

$$\mu_i = \sum_{i=0}^{N-1} b_{ij}(t)\, x_i \qquad 0 \leq j \leq M-1$$

In this equation μ_j is the output of output node j and x_i is element i of the input which can be 0 or 1.

Step 4. Select Best Matching Exemplar (Winner)

$$\mu_{j^*} = \max_j \{\mu_j\}$$

This is performed using extensive lateral inhibition as in the maxnet.

Step 5. Vigilance Test

$$||X|| = \sum_{i=0}^{N-1} x_i$$

$$||T.X|| = \sum_{i=0} t_{ij}^* x_i$$

If $||T.X|| / ||X|| > \rho$ then go to Step 6, otherwise goto Step 7.

(i.e. Number of one bits in common normalized by the number of one bits in input.)

Step 6 Disable Best Matching Exemplar

The output of the best matching node selected in Step 4 is temporarily set to zero and no longer takes part in the maximization of Step 4. Then go to Step 3.

Step 7A (Training). Adapt Best Matching Exemplar

$$t_{ij}^*(t+1) = t_{ij}^*(t)\, x_i$$

$$b_{ij}^*(t+1) = t_{ij}^*(t)x_i \;/\; \left(0.5 + \sum_{i=0}^{N-1} t_{ij}^*(t)x_i\right)$$

Step 7B (Testing). Output the Label of Best Matching Exemplar

Step 8. Repeat by Going to Step 2

(First enable any nodes disabled in Step 6 for new input data)

The structure of VQ using ART net is shown in Fig. 2.2. The adaptation of best matching exemplar only happen during codebook training phase. In the testing phase, the ART works as a self-adjusting search to find the best match from the learned codewords.

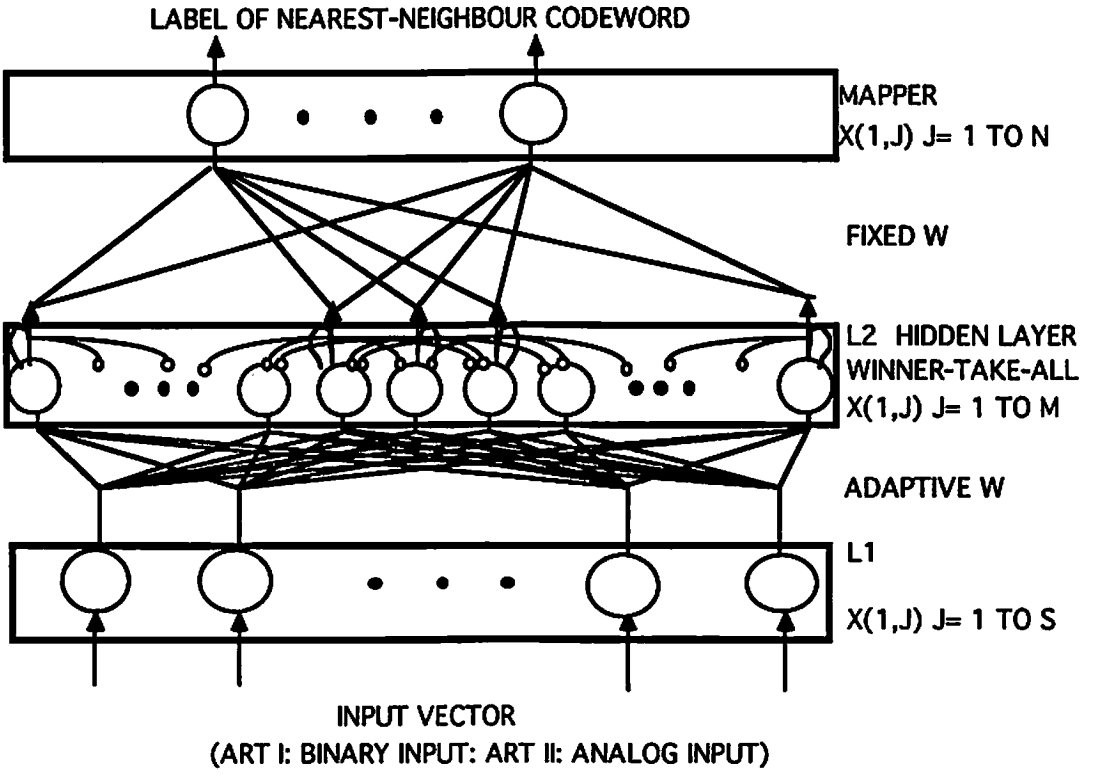


Fig. 2.2 Structure of VQ using ART.

2.4 Conclusion

For lossless data compression, the universal noiseless coding (UNC) is an effective method for producing reconstructed data without any fidelity loss. This adaptive algorithm provides excellent performance over a broad range of data contents. A high-speed VLSI design of the UNC coder will be presented in Chapter 6.

For lossy image compression, the vector quantization algorithm appears to be the most viable technique for an advanced data system. Various design aspects for the VLSI vector quantization, that include the functionality, testability, and fault tolerance, will be thoroughly investigated in Chapter 3.

Image compression and classification is an area that neural network models have been very successful. The progress in artificial neural network technology, especially the mixed-signal VLSI has shown the potential to support a real time adaptive vector quantization for image compression applications [2.42, 2.43]. The follow-up research reported in Chapter 4 will be emphasized on the use of adaptive self-organized neural networks to a VLSI neural network based image compressor.

References

[2.1] A. K. Jain, "Image data compression: A review," Proc. of IEEE, vol. 69, no. 3, pp. 349-389, Mar. 1981.

[2.2] R. M. Gray, *Source Coding Theory*, Kluwer Academic Publishers: Boston, MA, 1990.

[2.3] J. A. Storer, *Data Compression - Methods and Theory*, Computer Science Press: Rockville, Maryland, 1988.

[2.4] J. H. Witten, R. M. Neal, J. G. Cleary, "Arithmetic coding for data compression," Communications of the ACM, vol. 30, pp. 520-540.

[2.5] T. A. Welch, " A technique for high-performance data compression," IEEE Computer Magazine, pp. 8-18, 1984.

[2.6] J. Ziv, and A. Lempel, "Compression of individual sequences via variable-rate coding," IEEE Trans. on Information Theory, vol. IT-24, 1978.

[2.7] R. F. Rice, " Some practical universal noiseless coding techniques," JPL Publication 79-22, Jet Propulsion Laboratory, Pasadena, California, Mar. 15, 1979.

[2.8] R. F. Rice, J. Lee, " Some practical universal noiseless coding techniques, part II," JPL Publication 83-17, Jet Propulsion Laboratory, Pasadena, CA, Mar. 1, 1983.

[2.9] J. Lee, W. Fang, R. F. Rice, "Real-time data compressor for EOS-class missions," Proceeding of the 1987 IEEE Aerospace EASCON Conference, Washington, D. C., Oct. 1987.

[2.10] J. Lee, W. Fang, R. F. Rice, "VLSI universal noiseless compressor for EOS-class missions," Proceeding of the 1988 SPIE Conference, Los Angeles, CA, Feb. 1988.

[2.11] W. Pratt, "Hardmard transform image coding," Proc. of IEEE, Vol. 57, No.1, pp. 58-68, Jan. 1969.

[2.12] A. Habibi, "Comparison of the nth-order DPCM encoder with linear transformations and block quantization techniques," IEEE Trans, Comm, Tech., vol. COM-19, Dec. 1971, pp. 948-956.

[2.13] P. A. Wintz, "Transform picture coding," Proc. of IEEE, vol. 60, no. 7, pp. 809-820, July 1972 .

[2.14] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," IEEE Comput. Trans. C-23:90-93, 1974.

[2.15] W.H. Chen and H. Smith, "Adaptive coding of monochrome and color images," IEEE Trans. on Comm., vol. COM-25, vol. 11, pp. 1285-1292, Nov. 1977.

[2.16] D. J. Delp and O. R. Mitchell, "Image compression using block truncation coding," IEEE Trans. on Comm., Vol. COM-27, No. 9, , pp. 1335-1342, Sept. 1979.

[2.17] Y. Linde, A. Buzo, and R, M, Gray, "An algorithm for vector quantizer design," IEEE Trans. on Comm., vol. COM-28, , pp. 84-95, Jan. 1980.

[2.18] R. M. Gray, "Vector quantization," IEEE ASSP Magazine, pp. 4-29, April 1984.

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

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

[2.21] J. S. Denker (Ed.) AIP Conference Proceedings 151, *Neural Networks for Computing,* Snowbird Utah, AIP, 1986.

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

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

[2.24] T. Kohonen, *Self-Organization and Associative Memory,* 2nd Ed. New York: Spring-Verlag, 1988.

[2.25] T. Kohonen, K. Makisara, "Representation of sensory information in self-organization feature maps," in J. S. Denker (Ed.) AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986.

[2.26] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," Cognitive Sci., vol. 11, pp. 23-63, 1987.

[2.27] S. Grossberg, " Adaptive pattern classification and universal recording: I & II," Biol. Cybernetic, vol. 23, 1976.

[2.28] G. A. Carpenter, S. Grossberg, " ART 2: Self-organization of stable category recognition codes for analog input patterns," Apply Optics, vol. 26. Dec. 1987.

[2.29] N. M. Nasrabadi and . Feng, " Vector quantization of images based upon the Kohonen self-organizing feature maps," in Proc. IEEE Int. Conf.. Neural Networks, pp. 1101-1108, 1988.

[2.30] J. Naylor and K. P. IL, "Analysis of a neural network algorithm for vector quantization for speech parameters," in Proc. 1st Ann. INNS Meet., Boston, MA, p. 310, 1988.

[2.31] A. K. Krishnamurthy, S. C. Ahalt, D. Melton, and P. Chen, " Neural networks for vector quantization of speech and images," IEEE J. on Selec. Areas in Commun., vol. 8, no. 8, pp. 1449-1457, Oct. 1990.

[2.32] T. Lee and A. M. Peterson, " Adaptive vector quantization using a self-development neural network," IEEE J. Selec. Areas Commun., vol. 8, no. 8, pp 1458-1471, Oct. 1990.

[2.33] Cottrell, G. W., Munro, P. W., Zipser, D., "Image compression by back-propagation," Advances in Cognitive Science (Vol. 2) Sharkey, N. E., ed. Norwood, NJ, 1988.

[2.34] J. G. Daugman, D. M. Kammen, "Image statistics, gases, and visual neural primitives," IEEE first International Conference on Neural Networks, San Diego, CA, June 1987.

[2.35] H. Bourlard, Y. Kamp, "Auto-association by multilayer perceptrons and Singular Value decomposition," Neural Networks for Computing, Snowbird, Utah, April 1988.

[2.36] L. D. Jackel, R. DE. Howard, J. S. Denker, W. Hubbard, and S. A. Solla, "Building a hierarchy with neural networks: An example -image vector quantization," Applied Optics, Vol. 26, No. 23, p 5081-5084, Dec. 1987.

[2. 37] Jerry Solomon, W. Deitch, M. Yeates, "Neural network, spectral pattern recognition, and spectral mixing decomposition", Workshop on Neural Network Devices and Application, Jet Propulsion Laboratory, Feb. 1987.

[2.38] J. C. Platt and J. J. Hopfield, "Analog decoding using neural networks," in J. S. Denker (Ed.) AIP Conference Proceedings 151, Neural Networks for Computing, Snowbird Utah, AIP, 1986.

[2.39] A. Treisman, "Features and objects in visual processing," Science American, Nov. 1986.

[2.40] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning", Cognitive Science, Vol. 9, pp. 75-112, 1985.

[2.41 J. A. Hartigan, *Clustering Algorithm*, John Wiley & Sons: New York, 1975.

[2.42] Carver Mead, Mohammed Ismail (Ed.), *Analog VLSI Implementation of Neural Systems*, Kluwer Academic Publishers: Boston, MA, 1989.

[2.43] W.-C Fang, B. J. Sheu, O. T.-C. Chen, J. Choi "A VLSI neural processor for image data compression using self-organization networks," *IEEE Trans. on Neural Networks: Special Issue on Neural Network Hardware*, vol. 3, no. 2, May 1992.

Chapter 3

A High-Speed VLSI Systolic Vector Quantizer for Image Data Compression

Abstract- A high-speed image compression processor based on VLSI design of systolic binary tree-searched vector quantizer has been developed for the increasingly strong demands on large-volume data communication and storage requirements. Simulation results show that this design is applicable to many types of image data and capable of producing good reconstructed data quality at high compression ratios. Various design aspects for the binary tree-searched vector quantizer including the functionality, testability, and fault tolerance are thoroughly investigated for VLSI implementation in a 1-μm CMOS technology. A specific 8-level binary tree-searched vector quantizer can be realized on a custom VLSI chip that includes a systolic array of eight identical processors and a hierarchical memory of eight subcodebook memory banks. The total transistor count is about 300,000 and the die size is about 8.67 x 7.72 mm^2 in a 1.0 μm CMOS technology. The throughput rate of this high-speed VLSI compression system is 25M pixels per second and its equivalent computation power is 400 MIPS.

3.1 Introduction

Image compression is essential to reduce the image transmission or storage costs for broad areas of applications such as high-definition television, teleconferencing, remote sensing, radar, sonar, computer communication,

facsimile transmission, and image database management [3.1,3.2]. According to Shannon's source coding theorem, asymptotic optimal performance can be obtained by coding vectors instead of scalars [3.3-3.5]. Over the past decade, vector quantization (VQ) has developed from a theoretical possibility into a powerful technique for speech and image compression at medium to low bit rates [3.6-3.14]. Since late 1980s, a variety of design techniques have been developed for VQ [3.12-3.14] and some real-time hardwares have been designed [3.15-3.18]. The emphasis in VQ appears to have shifted from basic research to system development, in particular to high-speed VLSI implementation of speech and image coders at low and medium rates. The driving forces behind this trend are as follows. (1) Advances of technologies, especially progress in VLSI technology, make more sophisticated data compression systems possible. (2) The increasingly strong demands on large-volume data communication and storage requirements. (3) Several key driving applications require data compression techniques such as high definition TV, advanced multi-media information system, integrated service data networks, satellite imaging system, and the space science data and communication system.

The primary advantage of the VQ method, as compared to other high compression ratio methods such as the adaptive transform coding [3.19], is its extremely simple decoding procedure, that makes it a great potential technique for the single-encoder multiple-decoder data compression systems. The major limitation of the VQ method is the high complexity involved in the codebook search. To effectively reduce the codebook search complexity, the scheme of tree-searched vector quantization was proposed such that the

codebook search complexity only grows linearly rather than exponentially as the codebook size increases [3.8, 3.9]. VLSI design of the full-searched vector quantization algorithm has been reported in the literature [3.15-3.17]. In this chapter we present an algorithm-specific VLSI design based on the binary tree-searched VQ algorithm and its associated systolic architecture. By combining tree-searched VQ and systolic processing, a high throughput vector quantizer can be realized on a single VLSI chip using the 1.0 μm CMOS technology. A specific 8-level binary tree-searched vector quantizer has been designed which is applicable for the advanced image processing systems.

This chapter is organized as follows. Section 3.2 reviews the full-searched and the tree-searched vector quantization algorithms. Section 3.3 describes the systolic processing architecture. Section 3.4 discusses the detailed hardware design for the binary tree-searched vector quantization algorithm. Section 3.5 presents the VLSI implementation. Section 3.6 describes testability and fault tolerance. Section 3.7 presents an adaptive method to construct a codebook and the simulation results for binary tree-searched vector quantizer. The conclusion is given in Section 3.8.

3.2 Vector Quantization

3.2.1 Full-Searched Vector Quantization

Vector quantization can be viewed as a mapping Q from a M-dimensional vector space $\Re^M$ into a finite subset C of $\Re^M$:

$$Q: \Re^M \dashrightarrow C, \tag{3.1}$$

where $C = \{C_i \mid i = 0, 1, 2, ..., N\text{-}1\}$ is the set of reproduced vectors and N is the number of vectors in C. Each C_i in C is called a *codevector* and C is called the *codebook* for the vector quantizer.

Figure 1 shows the functional block diagram of the vector quantizer. In the encoding phase, every source vector $\mathbf{X}(t)$ is compared with each codevector C_i in C and the codevector of the smallest distortion is chosen to represent $\mathbf{X}(t)$. The index of the codevector of the minimum distortion is used as the encoded output and transmitted through the channel. That is if $D(\mathbf{X}(t), \mathbf{C}_{\hat{i}}) < D(\mathbf{X}(t), \mathbf{C}_i)$, $0 \le \hat{i}, i \le N\text{-}1$, $\hat{i} \ne i$, then $\hat{i}(t)$ represents $\mathbf{X}(t)$. It is denoted by

$$\hat{i}(t) = \min_{0 \le i \le N-1}^{-1} D(\mathbf{X}(t), \mathbf{C}_i), \tag{3.2}$$

where $D(*,\bullet)$ being the distortion function, $D(\mathbf{X}(t), \mathbf{C}_i)$ being the distortion between $\mathbf{X}(t)$ and $\mathbf{C}_i$, and t being the time index. For most distortion measures, such as the mean-squared error, the vector distortion can be written as the sum of the scalar distortion, i.e.,

$$D_i(t) = D(\mathbf{X}(t), \mathbf{C}_i) = \sum_{j=0}^{M-1} \left(X_j(t) - C_{ij}\right)^2 \tag{3.3}$$

where $X_j(t)$ being the j^{th} component of the source vector $\mathbf{X}(t)$, C_{ij} being the j^{th} component of the i^{th} codevector $\mathbf{C}_i$.

On the other side of the channel, the received indices are used to select the codevectors from the codebook to reproduce the source vectors during the decoding phase. This process is a table look-up operation and can be expressed as

$$\hat{\mathbf{X}}(t) = \mathbf{C}_{\hat{i}(t)}. \tag{3.4}$$

3.2.2 Tree-Searched Vector Quantization

The codebook search efficiency is an important issue in vector quantization design and can be solved by incorporating tree structure into the codebook [3.14]. The codebook search complexity of the tree-structured VQ is a logarithmic function of the size of the codebook. This technique divides the codebook into consecutive subcodebooks. Codebook bit-length is $n = n_1+n_2+ ...+n_L$ bits. Codevector index assignment is n_1 bits for the first level, n_2 bits for the second level,... , and n_L bits for the L^{th} level. Codebook size is $N= N_1N_2 ...N_l...N_L$ where $N= 2^n$ and $N_l = 2^{n_l}$. Codevector notation is $\mathbf{C}_i^1$ for the first level, $\mathbf{C}_i^2$ for the second level,... , and $\mathbf{C}_i^L$ for the L^{th} level. For the tree-searched VQ, the encoding procedure is executed by comparing the source vector $\mathbf{X}(t)$ with the codevectors in the consecutive subcodebooks. The encoded index of the source vector $\mathbf{X}(t)$ is $\hat{i}_1(t)$ for the first level, $\hat{i}_2(t)$ for the second level, ..., and $\hat{i}_L(t)$ for the L^{th} level. That is denoted by

$$\hat{i}_1(t) = \min_{0 \le i_1 \le N_1-1}{}^{-1} D(\mathbf{X}(t),\mathbf{C}_{i_1}^1)$$

$$\hat{i}_2(t) = \min_{0 \le i_2 \le N_2-1}{}^{-1} D(\mathbf{X}(t),\mathbf{C}^2_{\hat{i}_1(t)\, i_2}) \qquad (3.5)$$

...

$$\hat{i}_L(t) = \min_{0 \le i_L \le N_L-1}{}^{-1} D(\mathbf{X}(t),\mathbf{C}^L_{\hat{i}_1(t)\, \hat{i}_2(t)... \hat{i}_{L-1}(t) i_L})$$

The final encoded index for the source vector $\mathbf{X}(t)$ is

$$\hat{\mathbf{i}}(t) = \hat{i}_1(t)\hat{i}_2(t)... \hat{i}_L(t) \qquad (3.6)$$

The decoding procedure is a table lookup,

$$\hat{\mathbf{X}}(t) = \mathbf{C}^{L}_{\hat{\mathbf{i}}(t)} = \mathbf{C}^{L}_{\hat{i}_1(t)\hat{i}_2(t)\ldots\hat{i}_L(t)} \tag{3.7}$$

3.2.3 Binary Tree-Searched Vector Quantization

Among tree-searched VQs, the *binary* tree-searched scheme is the most efficient way of searching through the codebook to find the near optimum reproduction codevectors for the source vectors. The number of the binary tree level L is same as the codebook bit-length n.

The codebook search of the binary tree-searched VQ can be expressed as

$$\begin{aligned}
\hat{i}_1(t) &= \min^{-1}_{i_1 = 0,\,1} D\left(\mathbf{X}(t), \mathbf{C}^{1}_{i_1}\right) \\
\hat{i}_2(t) &= \min^{-1}_{i_2 = 0,\,1} D\left(\mathbf{X}(t), \mathbf{C}^{2}_{\hat{i}_1(t)\, i_2}\right) \\
&\ldots\ldots \\
\hat{i}_n(t) &= \min^{-1}_{i_n = 0,\,1} D\left(\mathbf{X}(t), \mathbf{C}^{n}_{\hat{i}_1(t)\,\hat{i}_2(t)\ldots\hat{i}_{n-1}(t) i_n}\right)
\end{aligned} \tag{3.8}$$

The final encoded index for the source vector $\mathbf{X}(t)$ is

$$\hat{\mathbf{i}}(t) = \hat{i}_1(t)\,\hat{i}_2(t)\ldots\hat{i}_n(t) \tag{3.9}$$

The decoding procedure is still a table lookup,

$$\hat{\mathbf{X}}(t) = \mathbf{C}^{n}_{\hat{\mathbf{i}}(t)} = \mathbf{C}^{n}_{\hat{i}_1(t)\hat{i}_2(t)\ldots\hat{i}_n(t)} \tag{3.10}$$

3.2.4 Vector Quantization Algorithm Trade-Offs

Image compression is achieved by using vector quantization since fewer bits are needed to represent the codevector indices than the source vectors. VQ algorithm trade-offs are summarized in Table 3.1.

For full-searched VQ, the rate is $R = n/M$ bits per pixel where n being the codebook bit-length. The codebook search complexity is N operations per pixel for a codebook of size N. Each operation includes one substation, one multiplication, one addition, and one memory access. The decoding complexity is one memory access. The codebook memory size is NMK bits.

For tree-searched VQ, the compression ratio is KM/n. The codebook search complexity is $2^{n}1+ 2^{n}2+ \ldots +2^{n}L$ operations per pixel. The decoding complexity is one memory access. The total codevector number of the encoder is $2^{n}1+ 2^{n}1^{+n}2+ \ldots +2^{n}1^{+\ldots+n}L$. The effective codebook for the decoding is $\{ C_{\mathbf{i}}^{L}, \mathbf{i} = i_1 i_2 \ldots i_L \}$.

For binary tree-searched VQ, the compression ratio is KM/n. The codebook search complexity is $2n$ operations per pixel. The decoding complexity is one memory access. The total codevector number of the encoder is $2(N-1)$. The effective codebook for the decoding is $\{ C_{\mathbf{i}}^{n}, \mathbf{i} = i_1 i_2 \ldots i_n \}$.

3.3 Systolic Architecture for Vector Quantization

3.3.1 Systolic Architecture for Full-Searched Vector Quantization

The codebook search and encoding procedure of the full-searched VQ shown in (2) can be expressed in a general matrix-vector multiplication form, where the multiplication operator represents the evaluation of scalar

distortion and the addition operator is the sum of the scalar distortions. Therefore, (2) can be implemented using systolic processing since matrix type computations are well suited for this technique [3.20-21]. A dependence graph of the full-searched VQ algorithm is shown in Fig. 3. 2 to illustrate the dependence of the computations.

Distortion-Stay Systolic Array

If the dependent graph of the full-searched VQ is projected along the *j*-direction, then it leads to a linear systolic architecture as shown in Fig. 3.3. In this *distortion-stay* systolic array, the distortion value D_i is associated with processing element *i* that the distortion is computed, where $0 \leq i \leq N\text{-}1$. The D_i accumulates the intermediate result as the codevector component C_{ij} moves downward and the source vector component X_j synchronously moves to the right. After *M* clock cycles, D_i will contain the distortion between the source vector and the i^{th} codevector. To perform (2), two variables, *I* and *D*, are required to record the index and distortion of the codevector of the current minimum distortion. The variable *D* is initialized to be a large number. Both *I* and *D* enter processing element 0 following determination of D_0. They move down the array one processing element per clock cycle. At processing element *i*, *D* is compared with D_i. If $D_i < D$, then $I = i$ and $D = D_i$. At the output of processing element *N-1*, *I* will contain the codevector index of the minimum distortion, representing the quantized source vector.

For the continuous data encoding, the next source vector with its own (*I*, *D*) pair immediately follows the current source vector such that the data are continuously piped through the array. This can be achieved by

cycling the codevector components C_{ij} into processing element i as the input data flows into the array. Each D_i is reset after the vector distortion is determined.

For this systolic architecture, N processing elements are required. The codebook search and encoding speed is increased by a factor of N over a single processing element architecture. The pipeline latency is $N+M-1$ clock cycles. The throughput rate is constant at 1 pixel/clock for any vector dimension and codebook size.

Distortion-Move Systolic Array

As shown in Fig. 3. 4, a *dual* systolic architecture for the full-searched VQ is derived by projecting the dependence graph along the i-direction. In this *distortion-move* systolic array, the distortion value D_i moves while the source vector component X_j stays in the processing element j. The input data X_j is loaded into processing element j, where $0 \leq j \leq M-1$. As the codevector component C_{ij} moves downward, the distortion value D_i moves synchronously to the right. Parameter D_i accumulates the intermediate result of the distortion as it moves toward the end of the array. After D_i flows out of processing element $M-1$, it will contain the distortion between the source vector $\mathbf{X}(t)$, and the i^{th} codevector, $\mathbf{C}_i$. To perform (2), a comparator is added to the right end of the array, which keeps track of the codevector of the minimum distortion. This array structure was first presented by Capello, et al., for the design of a real-time speech compression system [3.16]. It was later revised by Dianysian and Baker into a two-level systolic structure [3.17].

For the encoding of a continuous source vector sequence, the next source vector components are sequentially loaded into the processing elements immediately following computation of the distortion between X_0 and $C_{N-1,0}$ the time when D_{N-1} flows out of processing element 0 and into processing element 1. The same data flow pattern of C_{ij} is repeated for each source vector. This can be realized by storing the codevector components C_{ij} in processing element j and cycling these components as the input data flows into the array. The comparator is reset following final comparison of all distortions of the current source vector.

For this type of architecture, M (vector dimension) processing elements and one comparator are required which results in an encoding speed gain factor of M over a single processing element. The pipeline latency is $N + M - 1$ clocks. The throughput rate can be as fast as M pixels per N clocks, where the clock represents the internal clock of the systolic array. Since M is normally less than N, the throughput rate is slower than 1 pixel/clock.

3.3.2 Systolic Architecture for Tree-Searched Vector Quantization

Equation (5) shows that the tree-searched VQ encoder is simply a cascaded series of the full-searched VQ encoders. The key is to correctly address the next level subcodebook. This can be realized by relating the index of the current tree level to the indices of the previous tree levels. The combined indices are then used to address the next level subcodebook. In this section, we extend the systolic architecture from the full-searched VQ to encode the data for the tree-searched VQ. We focus the study on the

architecture presented in Fig. 3.3. The design based on Fig. 3. 3 has the following advantages for the tree-searched VQ: 1) Ease of expandability; 2) Improved throughput/complexity ratio; and 3) High speed uniform rate throughput for a tree-structured codebook.

Figure 3.5 shows an example of a systolic architecture for the tree-searched VQ. It is essentially a concatenation of L systolic arrays of the full-searched VQ shown in Fig. 3.3, where L being the number of tree-levels. Each stage corresponds to one tree-level. The codevectors of each subcodebook are arranged as follows. Codevector $C_{i_1 \ldots i_l}$ are allocated to processing element i_l of the l^{th} stage array. There are $N_1 \ldots N_{l-1}$ codevectors in each processing element of the l^{th} stage array. During VQ codebook searching, the codevectors are addressed by the combined indices of the previous stages, $i_1 \ldots i_{l-1}$. For this systolic architecture, the l^{th} stage contains N_l processing elements, which amounts to $\Sigma_{l=1}^{L} N_l$ processing elements. The pipeline latency is $\Sigma_{l=1}^{L} (N_l - 1) + LM$ clock cycles. The system throughput rate is 1 pixel/clock which is constant for any tree-structured codebook. This can be compared with a systolic architecture for the tree-searched VQ based on Fig. 3.4 that requires ML systolic processing elements with throughput rate of $M / \left[\Sigma_{l=1}^{L} N_l\right]$ pixel/clock. The architecture based on Fig. 3.3 is more favorable when $N_l \ll M$ which is generally true for the tree-searched VQ.

3.3.3 Systolic Architecture for Binary Tree-Searched Vector Quantization

Figure 3.6 shows a binary tree-searched VQ codebook structure. The codebook searching operation can be organized as finding a path from the root to a leaf in a binary tree. The computation at each level as specified

in (8) can be performed by one processor that comprises two systolic processing elements. A binary tree of depth n can be mapped into a linear array of n processors as shown Fig. 3. 7. The source vector sequence { X(t)} continuously flows through the array. At each stage, the source vector is compared with two codevectors. After the index of the current tree-level is obtained, it is tagged to the indices from the previous tree-levels to address the next level subcodebook. The index is attained at a rate of one bit per stage. At the end of the array, the concatenated indices are formed to represent the coded data. This final indices are n bits in length. Recall that the tree depth L is equal to the codebook bit-length n for the binary tree-searched VQ. The overall system requires $2n$ systolic processing elements. The pipeline latency equals $n(1+M)$ clock cycles. The throughput rate is 1 pixel per clock cycle.

3.3.4 Systolic Architecture for Binary Tree-Searched Vector Quantization with Difference Codebook

Using the mean-squared-error (MSE) criterion as a distortion measure, the distortion computation between the source vector $\mathbf{X}$ and the codevectors at the same binary tree level (C_0 and C_1) for a binary tree-searched VQ is

$$D_0(t) = \sum_{j=1}^{M} [X_j^2(t) + C_{0j}^2] - 2\sum_{j=1}^{M} X_j(t)C_{0j}$$
$$D_1(t) = \sum_{j=1}^{M} [X_j^2(t) + C_{1j}^2] - 2\sum_{j=1}^{M} X_j(t)C_{1j} \qquad (3.11)$$

The binary tree-searched VQ algorithm can be further simplified by computing the distortion based on the difference of the codevector pair

[3.18]. This results in significant reduction of hardware complexity. The key idea is to combine and simplify the distortion computation between the source vector X and the codevectors at the same binary tree level: C_0 and C_1. i.e.,

$$\frac{D_0(t) - D_1(t)}{2} = \sum_{j=1}^{M} \frac{C_{0j}^2 - C_{1j}^2}{2} - \sum_{j}^{M} X_j(t)[C_{0j} - C_{1j}]$$
$$= \Delta - \sum_{j=1}^{M} X_j(t)\, \delta_j \qquad (3.12)$$

where

$$\Delta = \sum_{j=1}^{M} \frac{C_{0j}^2 - C_{1j}^2}{2} \qquad (3.13)$$
$$\delta_j = [C_{0j} - C_{1j}].$$

Instead saving of C_0j and C_1j, δj and Δ are stored in the subcodebook. Here δj and Δ are the 1st-order difference and the 2nd-order difference, respectively. This technique is called the *difference-codebook* binary tree-searched VQ design. For this difference-codebook design, the codebook search complexity is reduced from $2n$ to n operations per pixel which is about half of the raw-codebook (i.e. original) scheme. The total memory size is reduced from $2(N-1)\,MK$ bits to $(N-1)\,[M(K+1) + (2K + \log_2 M)]$ bits. For instance, the total memory size for an 8-level 16-dimensional 8-bit-per-pixel binary tree-searched VQ is reduced from 2x(256-1)x16x8 = 65,280 bits to (256-1)x[16x(8+1)+(2x8+log$_2$16] = 41,820 bits by using the difference-codebook scheme instead of raw-codebook scheme. Thus a 64% reduction in the memory size is achieved. For the systolic binary tree-searched VQ with difference-codebook, the pipeline latency equals to nM clock cycles. The throughput rate is 1 pixel per clock cycle and the rate is n/M bits per pixel.

3.3.5 Comparisons for Various Systolic VQ Schemes

The performance comparisons for variant systolic VQ schemes are shown in Table 3.2. Among them, the systolic design for the binary tree-searched VQ is to be the most viable technique for realizing a high throughput vector quantizer at a relatively low hardware complexity. This architecture demonstrates that by a proper combination of tree-searched VQ and systolic processing a high throughput vector quantizer can be attained by using a small number of processors. Such an architecture has the advantages of modularity, regular data flow, simple interconnection, localized communication, simple global control, and pipeline processing. Hence it is well suited for VLSI implementation. A detailed VLSI design of this systolic binary tree-searched VQ is presented in the following .

3.4 Array Processors Design for Binary Tree-Searched Vector Quantization

As shown in Fig. 3.8, there are three major functional blocks of the systolic binary tree-searched vector quantizer (BTSVQ): 1) Array controller; 2) Systolic processing element (PE) array; 3) Codebook memory banks. The array controller interprets control information from the host system to set up the BTSVQ quantizer and generates timing and control signals to operate it. The array controller is implemented by using the Motorola DSP56000 signal processor. The codeboook memory banks and processing elements design for the raw-codebook BTSVQ and the difference-codebook BTSVQ are described as follows, respectively.

3.4.1 Raw-Codebook Binary Tree-Searched VQ Design

Detailed functional designs of the raw-codebook memory and processing element of the binary tree-searched VQ are shown in Fig. 3.9 and Fig. 3.10, respectively.

Raw-Codebook Memory Design

The hierarchical codebook memory banks stores the n-bit codebook that is divided into n subcodebooks. Each codevector pair in level-l are stored in the l-th subcodebook. The size of subcodebook memory of the l-th processor is 2^lMK bits. Although the subcodebook memories differ in size, they assume a regular structure in terms of memory cell design. The subcodebook memory can be programmed by the host computer during the set-up mode. While in the codebook searching and encoding mode, each subcodebook memory is controlled by its associated PE.

Raw-Codebook PE Design

The major functional blocks of each systolic processing element (PE) of the BTSVQ are: 1) source vector pipeline buffer; 2) distortion computing data path; and 3) index generator.

The source vector pipeline buffer is K-bit wide and M-word deep. It serves a first-in-first-out buffer for every source vector of M elements. The output of vector that is pipelined through the array enable the systolic processing.

The distortion computing of the raw-codebook PE design is primarily two mean square error operations. During the VQ encoding, the codevector-pair components are addressed by the combined indices of the previous PEs, $\hat{i}_1\hat{i}_2 \cdots \hat{i}_{l-1}$. An accumulator collects the intermediate result as

the codevector pair $C_{odd,j}$ and $C_{even,j}$ moves downward and the input data X_j moves to the right synchronously. After M clock cycles, The accumulator will consecutively contain the inner product between the input data vector and the selected codevector-pairs.

The index generator compares the distortion measurement $D_1$and D_0. If $D_1 \geq D_0$., then the $\hat{i}_l$ is assigned to be 0. Otherwise, $\hat{i}_l$ is assigned to be 1. Index $\hat{i}_l$ is tagged to the indices of the previous tree-levels to correctly address the next level subcodebook. At the end of the array, the concatenated indices, n bits in length, are formed to represent the coded data.

3.4.2 Design of Systolic Difference-Codebook Binary Tree-Searched VQ

Detailed functional designs of the difference-codebook memory and processing element is shown in Fig. 3.11 and Fig. 3.12, respectively.

Difference-Subcodebook Memory Design

The hierarchical codebook memory banks stores the n-bit codebook that is divided into n difference-subcodebooks. The first-order difference, δj, and second-order difference, Δ, of each codevector pair in level-l are stored in the subcodebook l. The size of difference subcodebook memory in level l is 2^{l-1} [M(K+1) + (2K+ log M)] bits. Although the subcodebook memories differ in size, they assume a regular structure in terms of memory cell design. The subcodebook memory can be programmed by the host computer during the set-up mode. While in the codebook searching and encoding mode, each subcodebook memory is controlled by its associated PE.

Difference-Codebook PE Design

The major functional blocks of each systolic processing element (PE) of the BTSVQ are: 1) source vector pipeline buffer; 2) distortion computing data path; and 3) index generator.

The source vector pipeline buffer is *K*-bit wide and *M*-word deep. It serves a first-in-first-out buffer for every source vector of M elements. The output of vector that is pipelined through the array enable the systolic processing.

The distortion computing data path of the DCPE design is primarily an inner product operator. During the vector quantization, the difference-codevector components are addressed by the combined indices from the previous PEs, $\hat{i}_1\hat{i}_2 \ldots \hat{i}_{l-1}$. An accumulator collects the intermediate result Δ' as the difference codevector component δj moves downward and the input data Xj moves to the right synchronously. After *M* clock cycles, The accumulator will consecutively contain the inner product between the input data vector and the selected difference codevector.

The index generator compares Δ with Δ'. Here, Δ is the pre-stored threshold, and Δ' is the value computed by the data path in the DCPE array. If Δ is greater than Δ', the $\hat{i}_l$ is assigned to be 1. Otherwise, $\hat{i}_l$ is assigned to be 0. Index $\hat{i}_l$ is tagged to the indices from the previous tree-levels to correctly address the next level subcodebook. At the end of the array, the concatenated indices, which are*n* bits in length, are formed to represent the quantized vector. Notice that the index generator can be effectively implemented by using an pre-loaded accumulator that executes a successive *addition* of $x(j)\ \delta(j)$ to the pre-loaded - Δ, where $j = 0, 1, \ldots, M-1$. The most significant bit (MSB) of the accumulator register represents the encoded bit $\hat{i}_l$. If the computing result

is equal or less than zero, the $\hat{i}_l$ is assigned to be 1. Otherwise, $\hat{i}_l$ is assigned to be 0.

3.5 VLSI Implementation

In this section, we consider a detailed VLSI design of the systolic difference-codebook BTSVQ. For image compression application, the binary tree-searched VQ design with an 8-bit codebook of 4x4-pixels vector dimension is selected for evaluation which results in a compression ratio of 16:1. Here, each pixel and codevector element are represented with 10-bit 2's complement number.

The main blocks of the VLSI chip are an 8-PE systolic array and a hierarchical memory of 8 subcodebook memory banks. A functional diagram of the processing element is shown in Fig. 3.12. The major functional unit of each PE are the 10-b x 10-b multiplier and the 24-bit adder. The multiply-and-accumulate operations are performed in a single processor cycle that is overlapped with the hierarchical memory read cycle. The operation timing is shown in Fig. 3.13. The silicon area for each PE is 1171 μm x 1629 μm in a 1-μm CMOS technology. The n-level hierarchical memory banks based on RAMs. For the l-th memory bank, 2^{l+3} x 10 RAM cells and 2^{l-1} x 24 RAM cells are used to store the first-order difference and the second-order subcodebook respectively. The silicon area for the 8-th level memory of the first-order and second-order difference-subcodebook is 2283 μm x 2815 μm and 800 μm x 1558 μm, respectively. A chip layout is shown in Fig. 3.13. The chip size is about 8.7 x 7.7 mm^2. The area, power, and the longest delay for the chip and its major building blocks are summarized in Table 3.5.

For the BTSVQ implementation, the targeted encoding rate is 25 M pixels per second and the end-to-end pipeline latency equals to 136 clock cycles. It provides a computing capability of 400 MIPS for the high-speed image compression systems. The encoding rate is constrained by the longest delay path of the multiplier. Table 3.6 summarizes the 1.0 μm 8-PE BTSVQ chip information. Use other faster multiplier circuit can greatly increase the speed. Use of the 0.5-μm VLSI technology will double the performance.

The subcodebook memory can be programmed by the host system via the controller during the training mode. While in the encoding mode, each subcodebook memory can only be accessed by its associated PE. In the training mode, the first-order codevector differences and the second-order codevector differences are derived and stored in the subcodebook memory. In the encoding mode, the source vectors are received via the array controller. The PE performs an inner product between the source vectors and the codevector differences. The inner product result is compared with the second-order codevector difference which is loaded in the threshold register at the rising edge of each vector clock. An one-bit index is generated and concatenated with indices from previous PEs to address the subcodebook of the next PE. The final concatenated indices are formed in the last PE to represent the coded data.

3.6 Testability and Fault Tolerance Design

It is important to achieve a long lifetime for the compressor with a high confidence value. A fault tolerant architecture is required to achieve these goals. By combination of architectural fault tolerance and inherent error

detection capability, a highly reliable VQ encoder can be attained. As shown in Fig. 3.14, the linear systolic array of the VQ encoder is augmented with a spare PE and dynamic reconfiguration switches (RS). Two switch designs, type A and type B, are presented to support the fault tolerance reconfiguration. If there is a permanent fault in any active PE, the faulty PE will be detected and isolated. Meanwhile, the spare PE will be activated via the reconfiguration switches. The reconfiguration switches are controlled by the fault-detection flag register which is a part of the array controller. The preliminary step for any reconfiguration policy aiming at fault tolerance is the testing and diagnosis of the system itself. In the following, we first discuss the fault model followed by the testing procedure to locate the faulty PE. A single computation unit (such as multiplier or adder) fault model is used [3.19], where we assume that at most one computation unit could be faulty within a given period of time which will be reasonably short compared with the mean time between failures. Two basic mechanisms can be applied to detecting faults in this type of system: on-line concurrent error detection and periodic self-test. On-line single error correction for arithmetic operations can be accomplished by arithmetic codes such as AN code or Residue code [3.20]. Temporary distortion of images due to transient faults may be tolerable for applications. Hence the second error, if any, can be detected by periodic self-test which is performed during power-up and periodically during operation by temporarily halting the data compression task.

For the DCPE design, predetermined test inputs can be applied. Since there is only one data path, pre-computed results corresponding to the inputs need to be pre-stored. The comparator then compares the generated

results with the stored values. If the two are the same, the PE is fault-free; otherwise, the same input is reapplied to determine if the fault is permanent or transient. If a permanent fault is detected, reconfiguration is required to replace the faulty PE. Following isolation of the faulty PE, the spare PE is switched in to maintain the integrity of the PE array.

The hardware overhead of the proposed self-test and reconfiguration scheme is about 20%. In the PE level, the overhead hardware includes two reconfiguration switches, one multiplexer, two registers, two comparators, one flag register, one n-input OR gate, one control line, n input lines, and one output line. In the system level, only one spare PE is required. It has been shown that error correction using arithmetic code is also effective [3.20]. The arithmetic code introduces redundant bits in the number representation. A proportional hardware increase takes place in register array and data path. The estimated hardware overhead is from 20% to 40%.

The reliability improvement by the use of fault tolerance scheme can be addressed as follows. If each PE has a reliability of R, then the reliability of 8 PEs is R^8. For the reconfigurable array with one spare PE, the reliability becomes $R^9 + 9\,R^8\,(1 - R)$. For example, if R = 0.950, the reliability of non-redundant PE array is 0.663 while the reliability of one spare PE array is 0.929. This represents a 40% increase in reliability. To handle the second PE failure, the PE array can be reconfigured to effectively reduce the codebook bit-length by one bit. In other words, the last tree level is bypassed. This approach will slightly degrade the image quality since smaller number of codevectors are available. However, the reliability can be further improved to 98%.

3.7 System Simulation Results

3.7.1 Algorithm for Constructing Codebook

An adaptive method to construct a codebook for binary tree-searched vector quantization has been developed. In computer simulation, the dynamic scalar training ratios are used to adjust the codebook through the backpropagation technique [3.21]. This achieves a better performance than traditional LBG splitting-2 methods [3.6]. If the dynamic vector training ratios are used, the best binary tree-searched structure that is achievable in this method can be reached. After the training step, a good binary tree-searched codebook can be obtained. To speed up the training and encoding processes, the codebook construction can also be implemented by using the presented systolic architecture as the core computation element.

The vector quantizer of binary tree-searched scheme were first proposed by R. M. Gray et al. [3.8] and is a natural extension of the Splitting-2 algorithm for generating initial code guesses in the LBG algorithm [3.6]. Our method begins with a full-searched codebook and then finds a better tree-searched scheme for the codebook. The Splitting-2 Relationship (S2R) algorithm records the tree-structure relationship. The Back Propagation Centroid (BPC) algorithm first groups the codevectors into closely disjoint pairs and then forms centroids of the pairs as the node labels of the immediate ancestors of the pairs. It works backward through the tree and groups close pairs. The Adaptive Training Ratio (ATR) algorithm has been

developed to combine the S2R and the BPC algorithms and finds a good adaptive training ratio for constructing the tree-searched codebook.

From the LBG and the Splitting-2 methods, the representative codevectors of different layers are

0^{th} *layer*: $Y_0^0 = [Y_{1,}^0, Y_{0,2}^0, ..., Y_{0,M}^0]$

1^{st} *layer*: Y_0^1, Y_1^1

...

l^{th} *layer*: $Y_0^l, Y_1^l, ..., Y_{2^l-1}^l$.

...

L^{th} *layer*: $Y_0^L, Y_1^L, ..., Y_{2^L-1}^L$.

The desired i^{th} tree-searched codevectors at the l^{th} layer of the S2R method are:

$$C_i^l = Y_i^l, \quad l = 1, ..., L.$$

The desired i^{th} tree-searched codevectors at the l^{th} layer of the BPC method are:

$$C_i^l = Y_i^l, \quad l = L$$
$$C_i^l = \frac{1}{2}(Y_{2i}^{l+1} + Y_{2i+1}^{l+1}) \quad l = L\text{-}1, ..., 1.$$

Here, L is the number of layers, M is the vector dimension, Y_i^l is the i^{th} codevector at the l^{th} layer, and C_i^l is the desired i^{th} tree-searched codevector at the l^{th} layer. The Y_{2i}^{l+1} and Y_{2i+1}^{l+1} are considered to be the closest codevectors because they are obtained from the same parent in the process of the Splitting-2 method.

The desired i^{th} tree-searched codevectors at the l^{th} layer of the ATR method are:

$$C_i^l = Y_i^l, \quad l = L$$
$$C_i^l = Y_i^l - R_i^l[Y_i^l - \frac{1}{2}(Y_{2i}^{l+1} + Y_{2i+1}^{l+1})] \quad l = L\text{-}1, ..., 1.$$

The adaptive training ratio $R_i^l = \{R_{ij}^l \mid 0 \le R_{ij}^l \le 1;\ j = 1, ..., M\}$ is a vector at the i^{th} codevector at the l^{th} layer. If all the element of R_i^l are the same, R_i^l is used as a scalar. In image processing, most of the neighboring pixels have similar gray levels. If the vector dimension of vector quantization is not too big, the scalar training ratio could be used to achieve a good balance between the distortion error and training time. The range of R_{ij}^l is from 0 to 1. If R_{ij}^l equals to zero for all i, j, and l, the ATR method is the same as the S2R method. If it equals to one, the ATR method is the same as the BPC method. In order to find the best training ratio, all possible cases have to be considered. These are $(2^L\text{-}2)xMxS$ adjustments of the codevectors and retrievals of the training data where S is the number of samples in the range [0,1]. If the least-mean-squared error is used as the criterion for data retrieval, the best training ratio set can be found. If the elements of the vector data are similar, the scalar training ratio can be used. The total training time is $(2^L\text{-}2)xS$ time units. If a constant is used as a training ratio then the training time is S time units.

3.7.2 Simulation Results

The criteria used to characterize the distortion, the radiometric error, induced by the data compression are the mean-squared error (MSE) and the signal-to-distortion ratio (SDR). The signal-to-distortion ratio is defined as the average intensity of the reconstructed image data divided by the mean- squared error between the original and reconstructed image data.

A constant training ratio for all codebooks was used in the computer simulation in Table 3.4 and Table 3.5. For both images, the performance of the ATR method is the best. In the "Girl" image, the second best is the BPC method. In the "Moon" image, the S2R method is better than the BPC method. It is obvious from Fig. 3.16(a) and 3.17(a) that the more layers in the tree can produce larger a training ratio. That is to say, the performance of the BPC method will be close to that of the ATR method at a large number of tree layers. Furthermore, a good training ratio exists between 0 and 1. The curves shown in Figures 3.16 (b) and 3.17(b) monotonically decrease to the lower bound and then increase monotonically. By comparing the original "Moon" image with the reconstructed "Moon" image in Fig. 3.18 , the quality of the ATR binary tree-searched VQ at a compression ratio 16 is observed to be very good. If a different vector dimension is considered, the small vector dimension would give better performance but the compression ratio would be smaller. The performance for different vector dimensions is shown in Fig. 3.19 for the ATR binary tree-searched VQ. The reproduced Girl image by a 10-level binary tree-searched vector quantizer using S2R, BPC, and ATR methods is shown in Fig. 3.20. The reproduced image quality is very good.

3.8 Conclusion

By combining the binary tree-searched algorithm and the systolic processing architecture, a high-speed VLSI vector quantizer has been developed for advanced information processing system where reduction of data communication or storage volume is important. An adaptive method to

construct a codebook for binary tree-searched vector quantization was also presented. The reconstructed image quality at a compression ratio 16 is observed to be very good. The 8-level binary tree-searched vector quantizer chip includes a systolic array of eight identical processing elements and a hierarchical memory of eight subcodebook memory banks. The total transistor count is about 300,000 with a pin count of 84. The die size is about 9.37 x 9.27 mm^2 in a 1-μm CMOS technology. The projected throughput rate is more than 20 M samples per second. A fault tolerant architecture was also presented. Fault tolerance is included in the system design to eliminate a single failure in the reconfigurable linear array structure. It is shown that by the use of an additional spare processing element and effective reconfiguration mechanism the reliability of the entire system can be improved by over fifty percent. Recent advances in communication, computer, and consumer electronics have created tremendous needs for efficient and high-speed data compression. This VLSI compressor chip meets low-power, light-weight, small-volume, high-speed, and user-transparent requirements and can play a crucial role in the high-performance data processing systems.

References

[3.1] A. K. Jain, "Image data compression: a review," *Proc. of IEEE*, vol. 69, no. 3, pp. 349-389, Mar. 1981.

[3.2] N. M. Nasrabadi, R. A. King, "Image coding using vector quantization: a review," *IEEE Tran. on Comm.*, vol. 36, no. 8, pp. 957-971, Aug. 1988.

[3.3] C. E. Shannon, " A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 379-423 and 623-656, 1948.

[3.4] R. E. Blahut, *Principle and Practice of Information Theory*, Addison-Wesley: Reading, MA, 1987.

[3.5] R. M. Gray, *Source Coding Theory*, Kluwer Academic Publishers: Boston, MA, 1990.

[3.6] Y. Linde, A. Buzo, R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Comm.*, vol. COM-28, no. 1, pp. 84-95, Jan. 1980.

[3.7] A. Buzo, R. M. Gray, "Speech coding based upon vector quantization," IEEE Trans. on ASSP, vol. ASSP-28, no. 5, pp. 562-574, Oct. 1980.

[3.8] R. M. Gray, H. Abut, "Full searched and tree searched vector quantization of speech waveforms," Proc. of IEEE ICASSP'82, pp. 256-260, Paris, France, May 1982.

[3.9] A. Gersho, " On the structure of vector quantizers," *IEEE Trans. on Inform. Theory*, vol. 28, no. 2, pp. 157-162, Mar. 1982.

[3.10] A. Gersho, V. Cuperman, "Vector quantization: a pattern matching technique for speech coding," IEEE Comm. Magazine, vol. 21, pp. 15-21, Dec. 1983.

[3.11] R. M. Gray, "Vector quantization," *IEEE Acoustics, Speech, and Signal Processing Magazine*, pp. 4-29, Apr. 1984.

[3.12] J. Makoul, S. Roucos, H. Gish, "Vector quantization in speech coding," Proceedings of the IEEE, vol. 73, no. 11, pp. 1551-1587, Nov. 1985.

[3.13] N. Nasrabadi, R. King, "Image coding using vector quantization: a review," IEEE Trans. on Comm., vol. 36, no. 8, pp. 957-971, Aug. 1988.

[3.14] A. Gersho, R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers: Boston, MA, 1991

[3.15] G. Davidson, A. Gersho, "Application of a VLSI vector quantization processor to real time speech coding," IEEE Journal on Selected Areas in Comm., vol. SAC4, pp. 112-124, Jan. 1986.

[3.16] P. R. Cappello, G. Davidson, A. Gersho, C. Koc, V. Somayazulu, "A systolic vector quantization processor for real-time image coding," Proc. of ICASSP, pp. 2143-2146, Tokyo, Japan, 1986.

[3.17] R. Dianysian, R. Baker, "A VLSI chip set for real-time vector quantization of image sequences," Proc. of Int. Symp. on Circuits and Systems, pp. 124-128, May 1987.

[3.18] W. -C. Fang, C. Y. Chang, B. J. Sheu, "Systolic tree-searched vector quantizer for real-time image compression," *VLSI Signal Processing IV*, H. S. Moscovitz, K. Yao, R. Jain (Eds.), pp. 352-361, IEEE Press: Piscataway, NJ, 1991.

[3.19] W.H. Chen, H. Smith, "Adaptive coding of monochrome and color images," IEEE Trans. on Comm., vol. COM-25, no. 11, pp. 1285-1292, Nov. 1977.

[3.20] H. T. Kung, "Why systolic architectures?", IEEE Computer Magazine, vol. 15, no. 1, pp. 37-46, Jan. 1982.

[3.21] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1988.

[3.20] J. A. Abraham, W. K. Fuchs, "Fault and error models for VLSI," Proc. of the IEEE, vol. 74, pp. 639-654, May 1986.

[3.21] D. K. Pradhan (Ed.), *Fault Tolerant Computing, Volume 1*, Prentice Hall: Englewood Cliffs, NJ, 1986.

[3.22] O. T.-C Chen, B. J. Sheu, W. -C. Fang, "Adaptive codebook construction and real-time hardware implementation for binary tree-searched vector quantization," IEEE Workshop Proc. on Visual Signal Processing and Communication, pp. 35-38, June 1991, Taiwan, R.O.C.

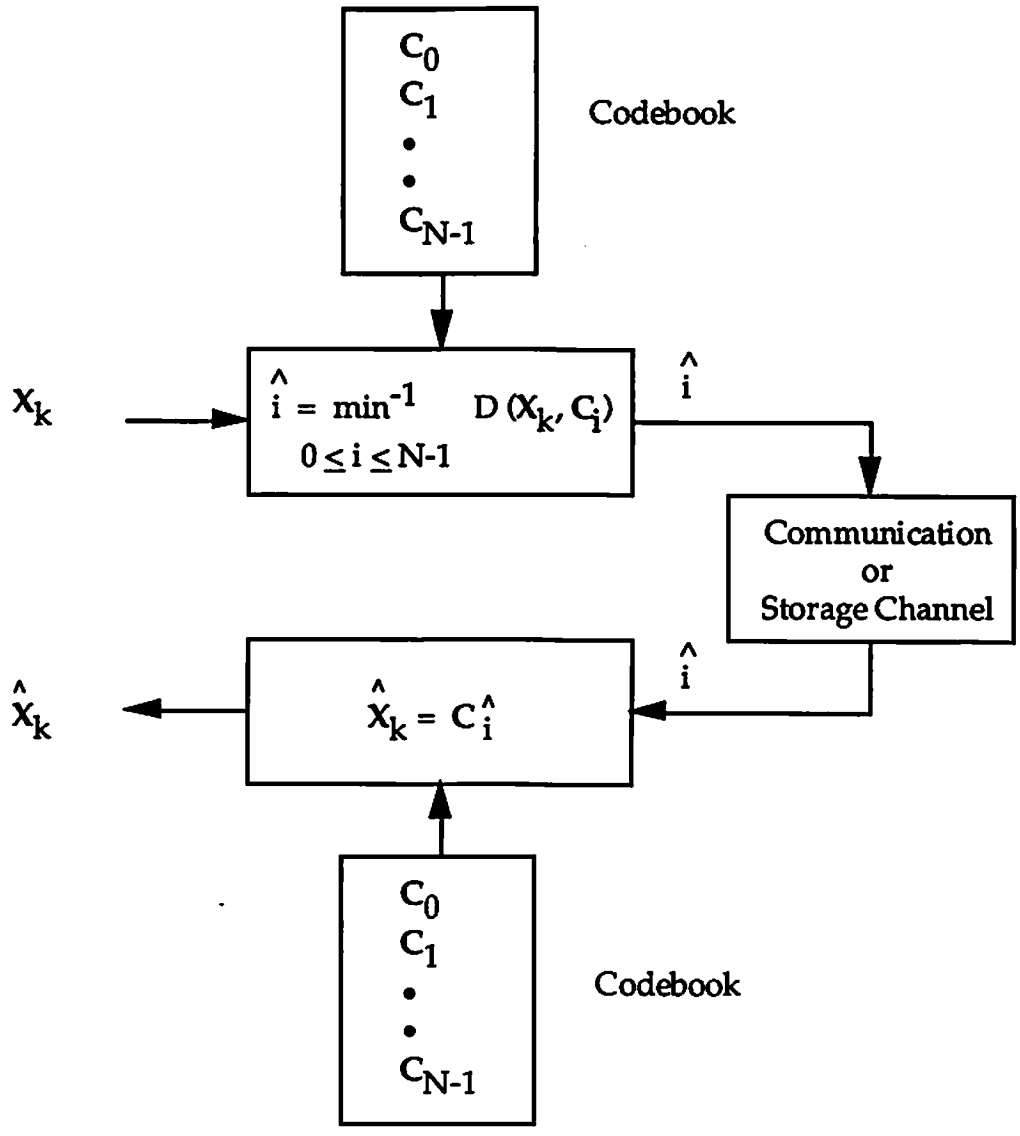


Fig. 3.1 Functional block diagram of the vector quantization.

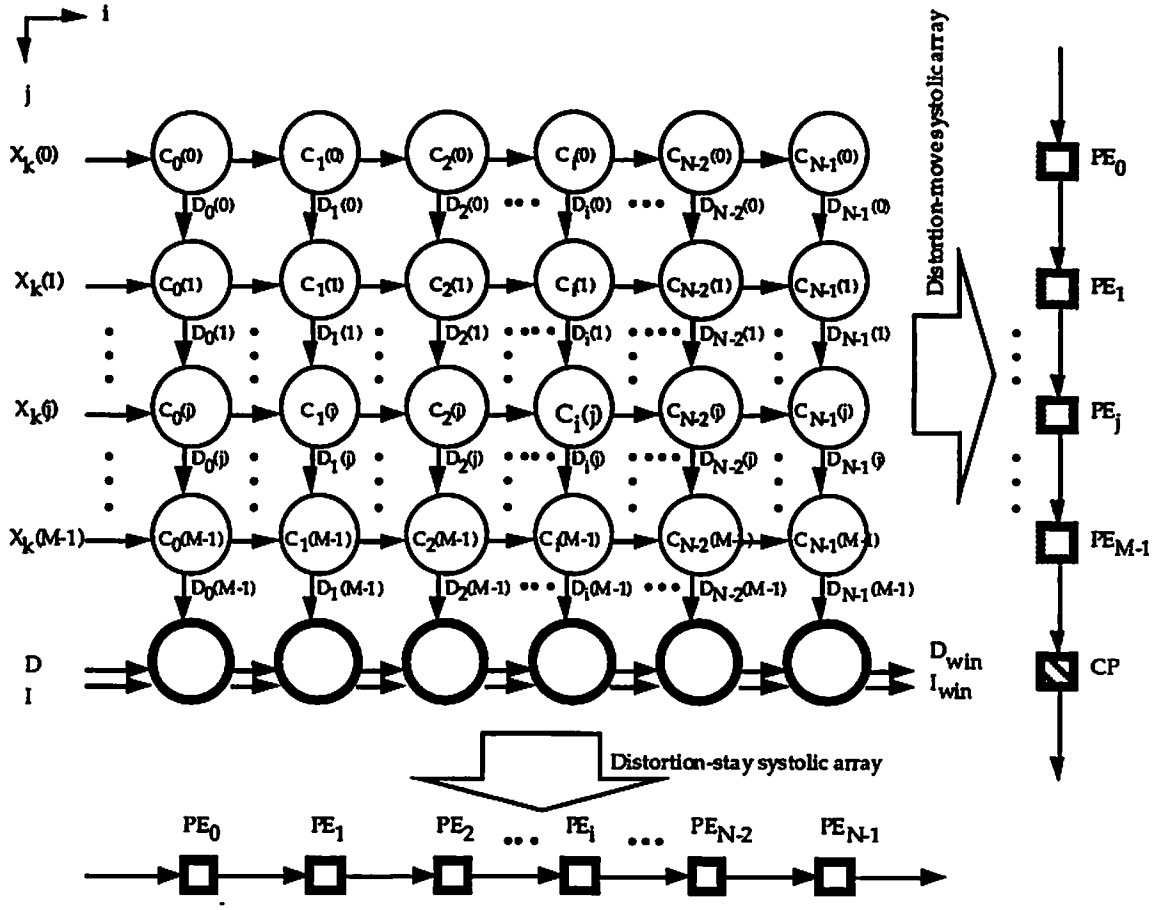


Fig. 3.2 Dependence graph of the full-searched vector quantization algorithm.

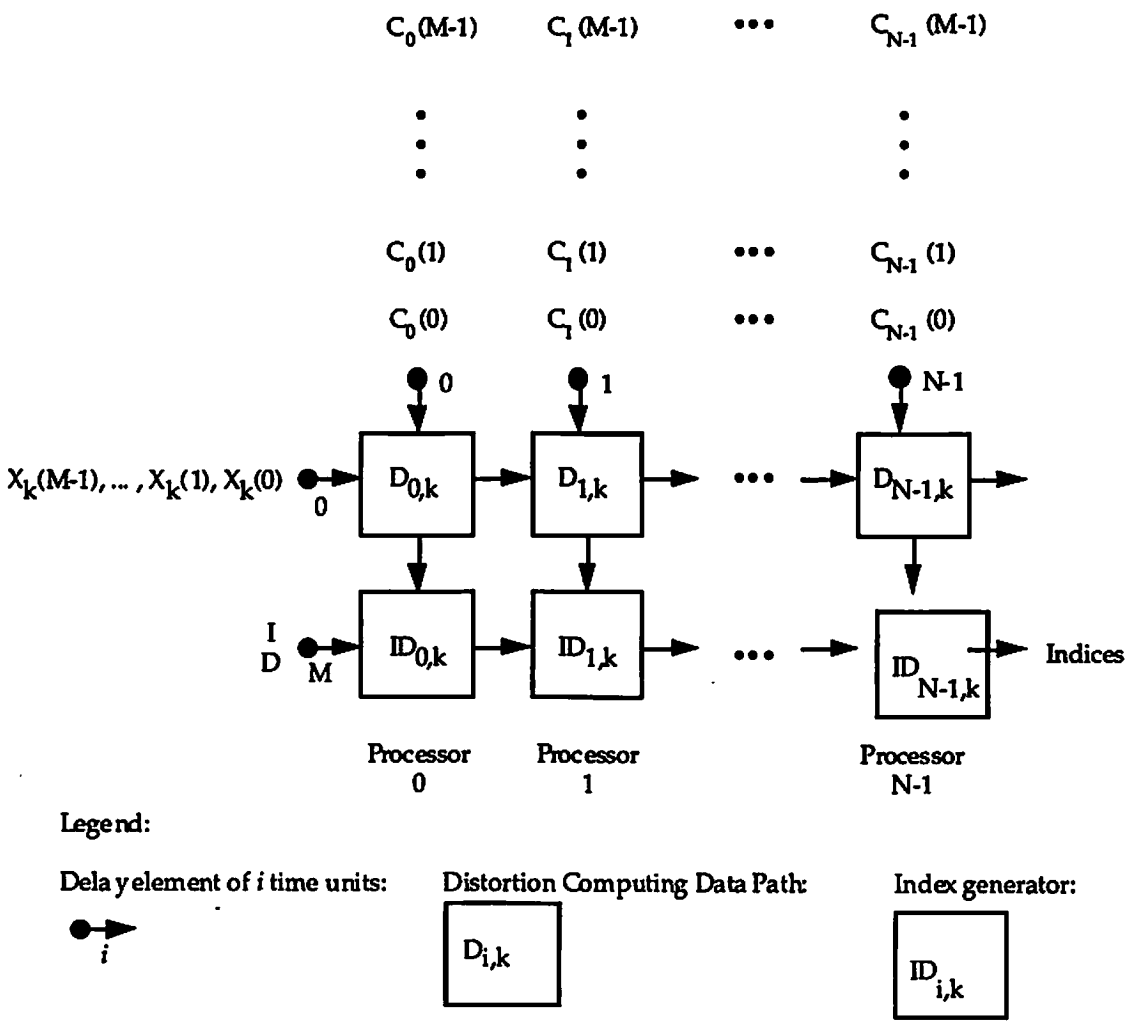


Fig. 3.3 Distortion-stay systolic architecture for the full-searched vector quantizer.

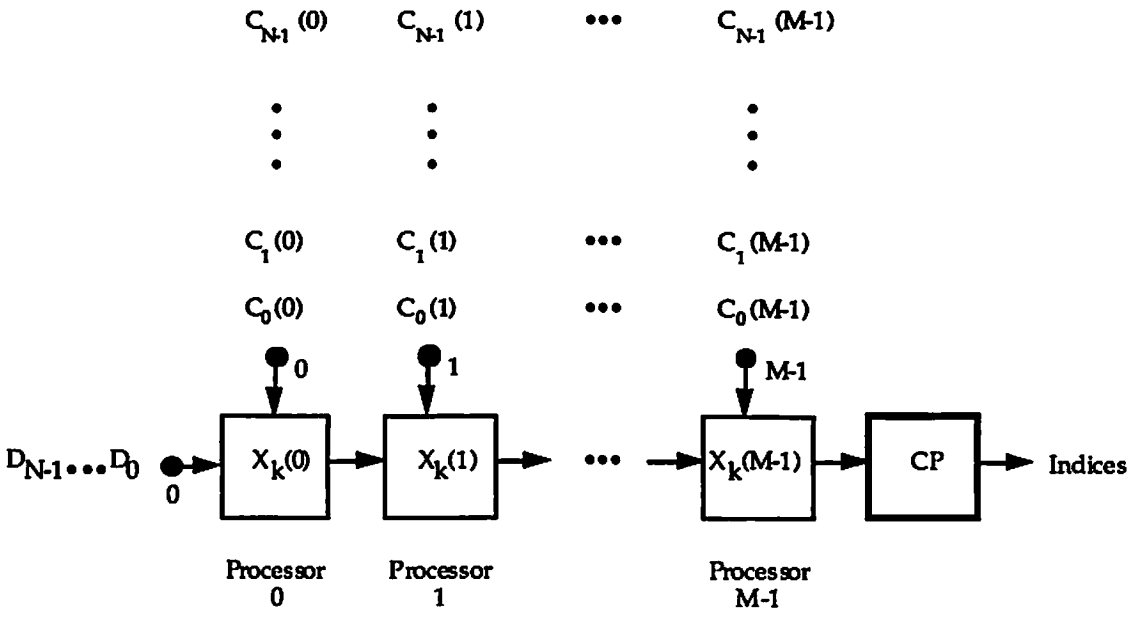


Fig. 3.4 Distortion-move systolic architecture for the full-searched vector quantizer.

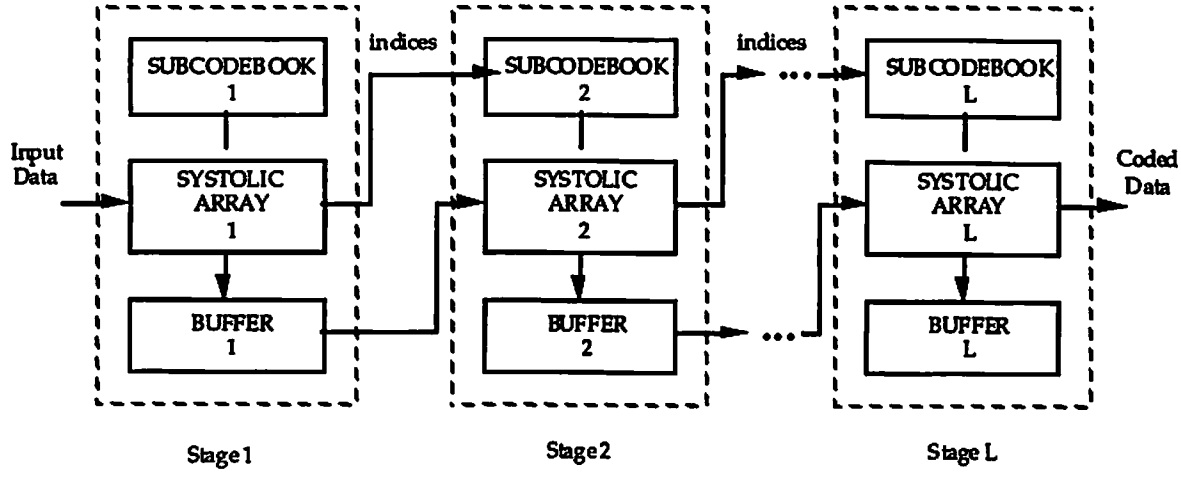


Fig. 3.5 Systolic architecture for the tree-searched vector quantizer.

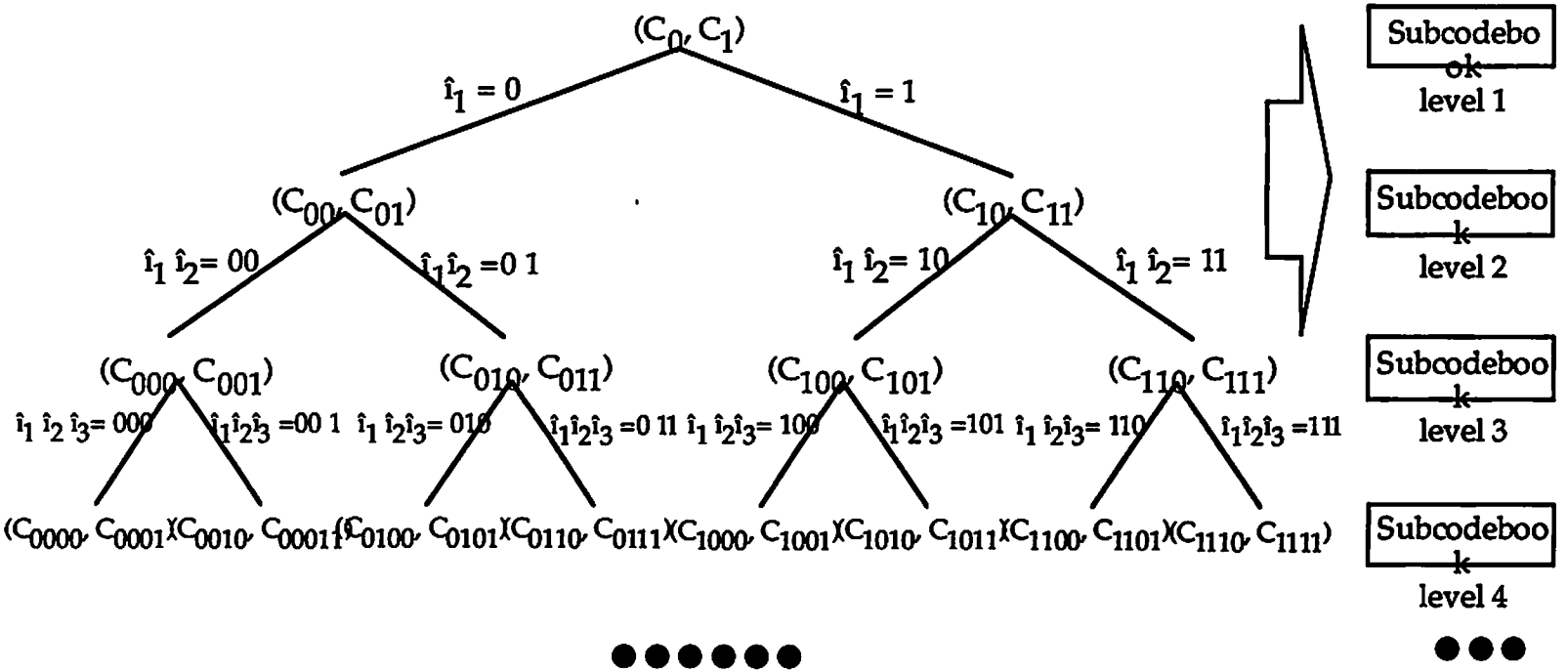


Fig. 3.6 The codebook structure for the binary tree-searched vector quantization.

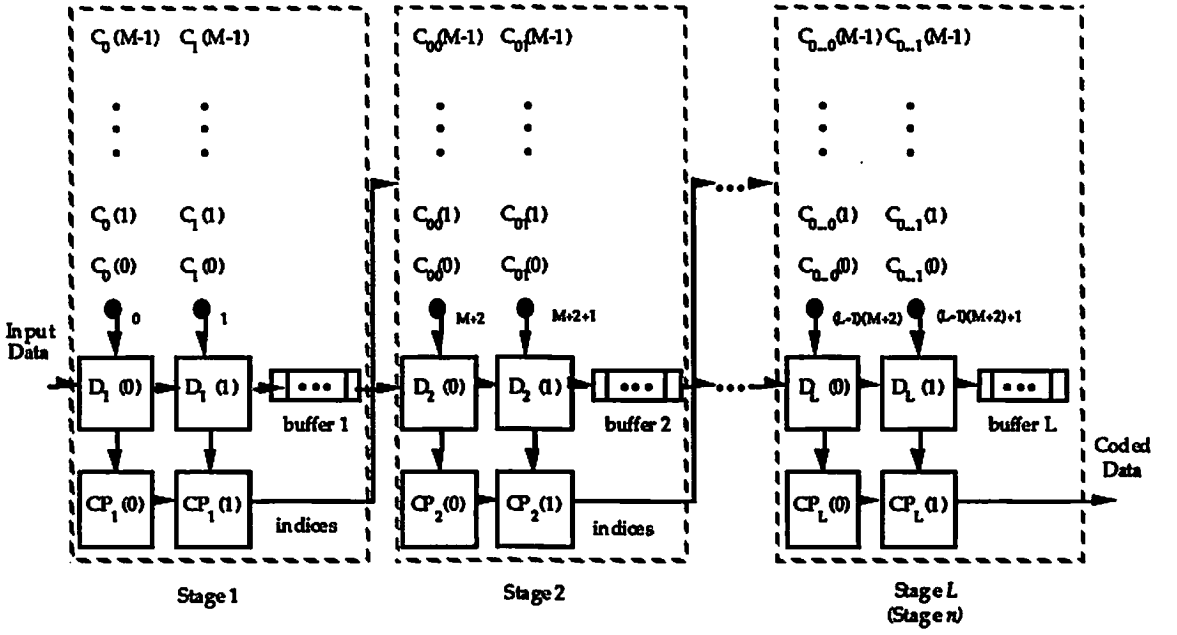


Fig. 3.7 Systolic architecture for the binary tree-searched vector quantizer.

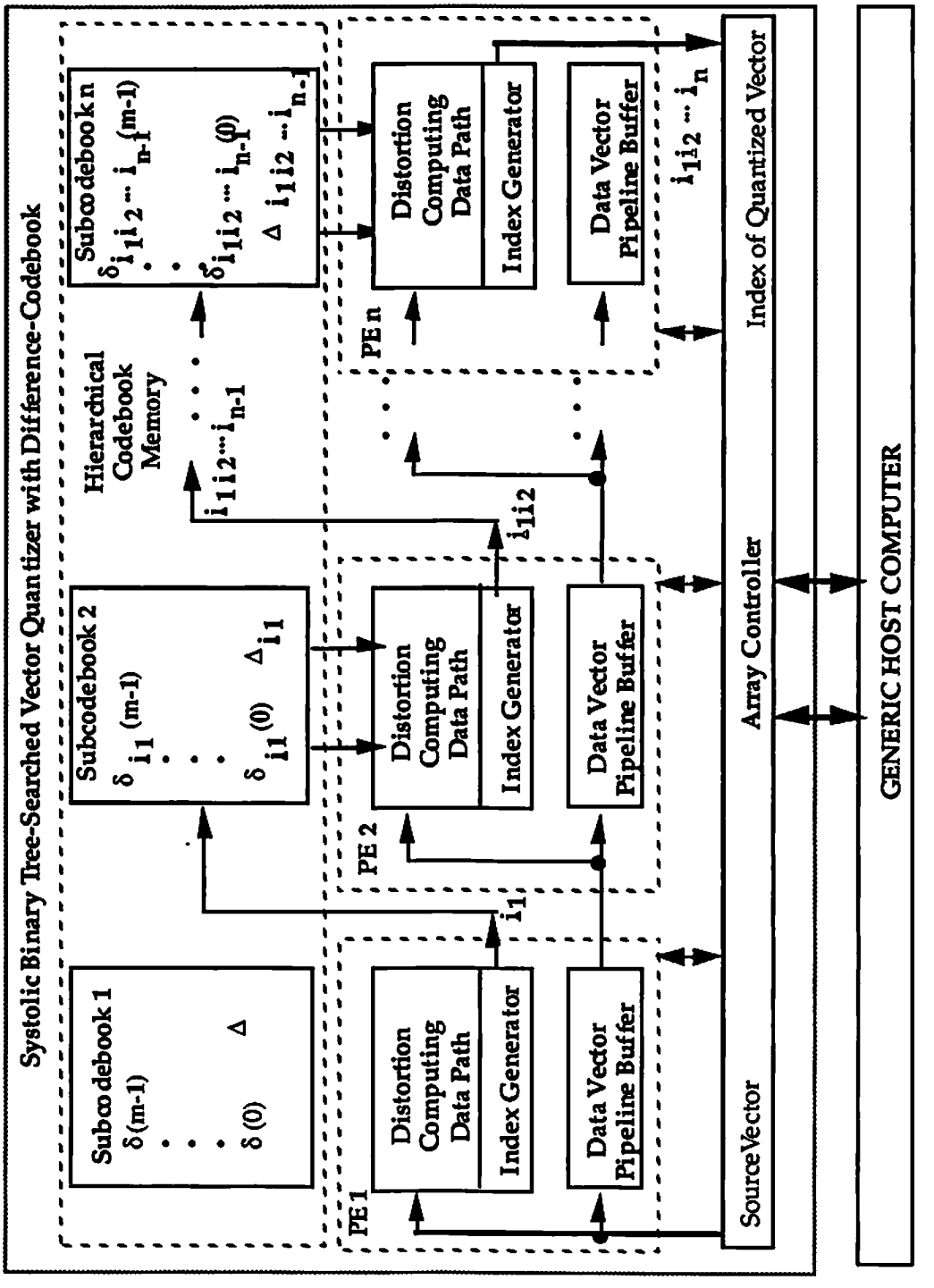


Fig. 3.8 System configuration for the difference-codebook BTSVQ.

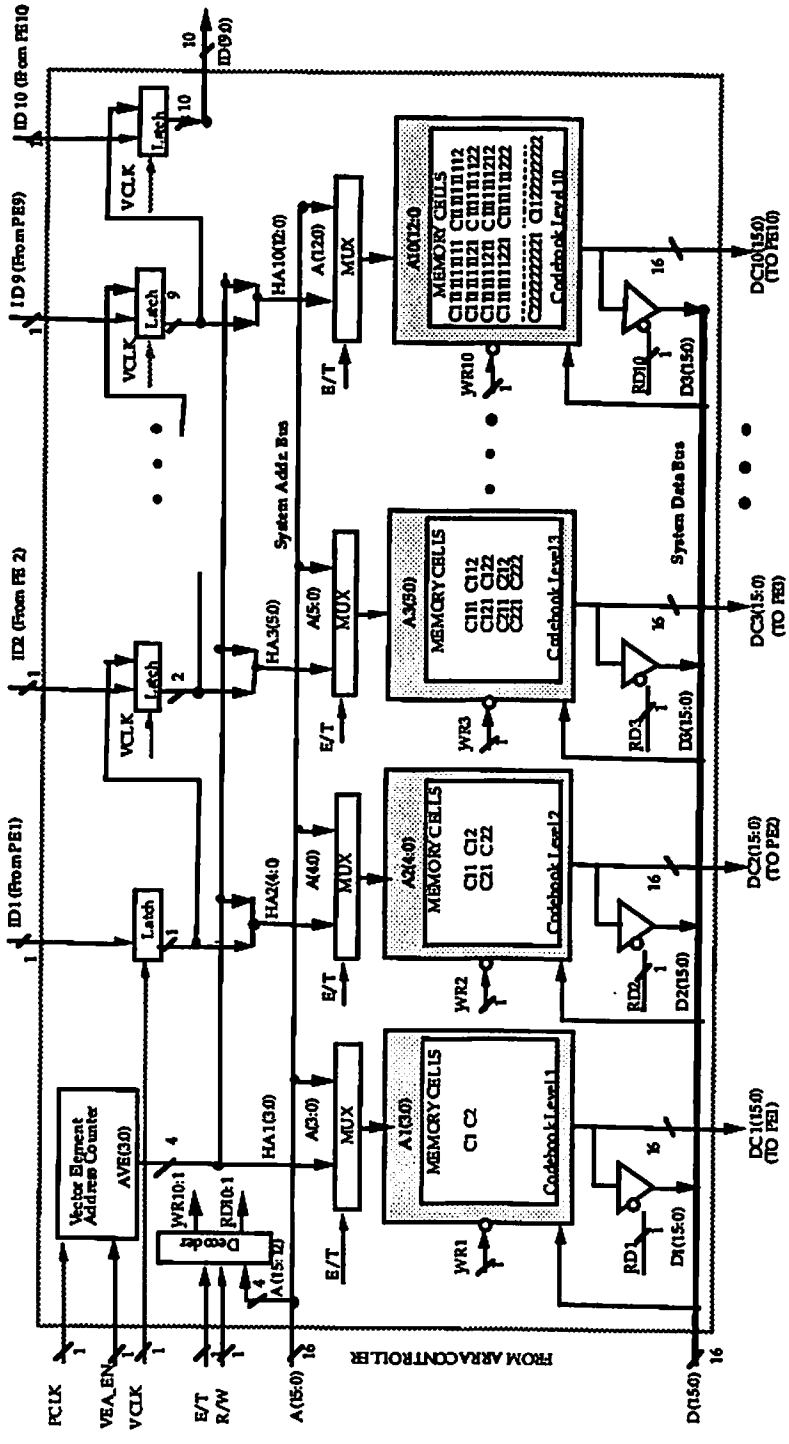


Fig. 3.9 Functional block diagram of hierarchical memory banks design of the raw-codebook BTSVQ.

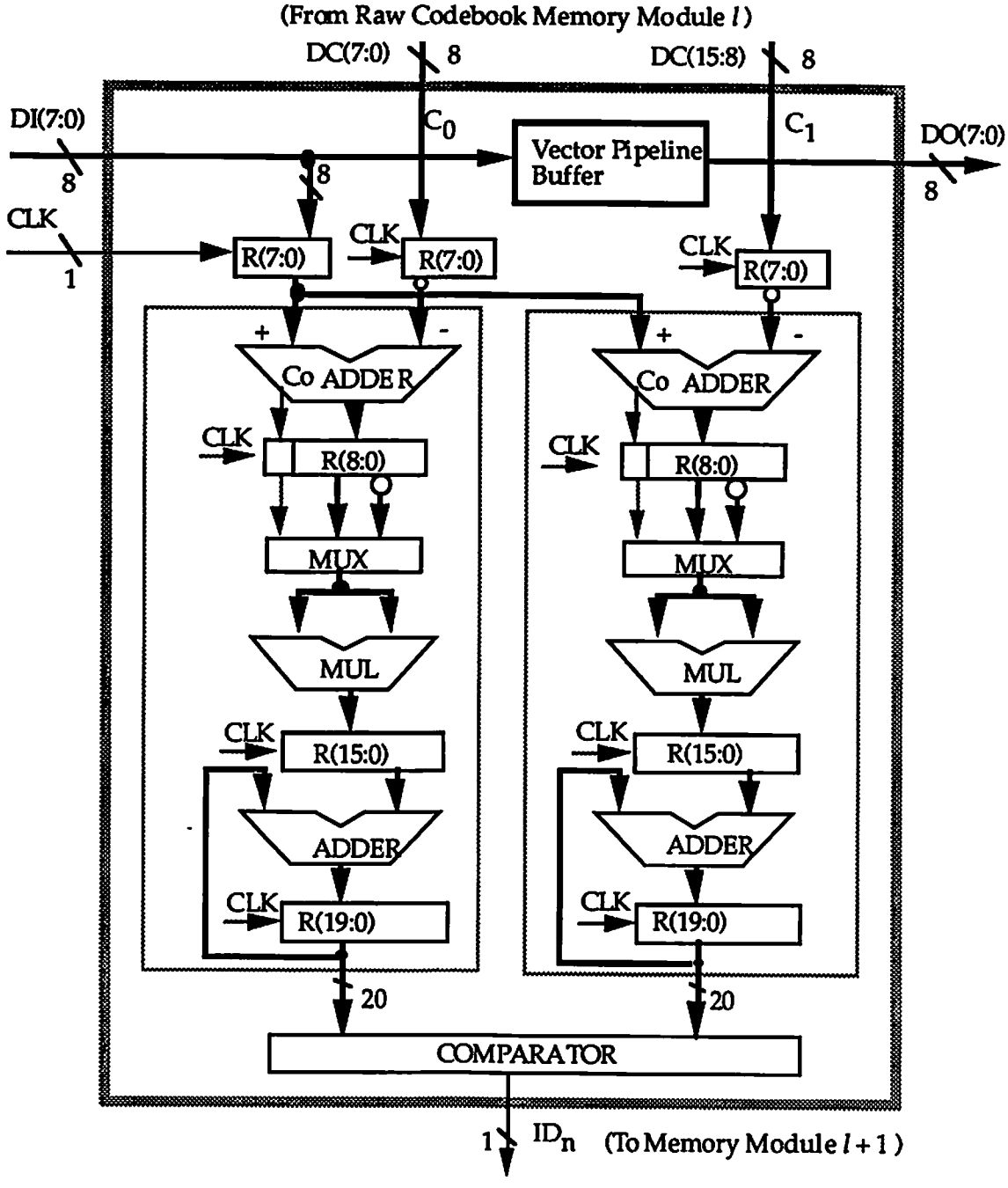


Fig. 3.10 Functional block diagram of the raw-codebook PE of the BTSVQ.

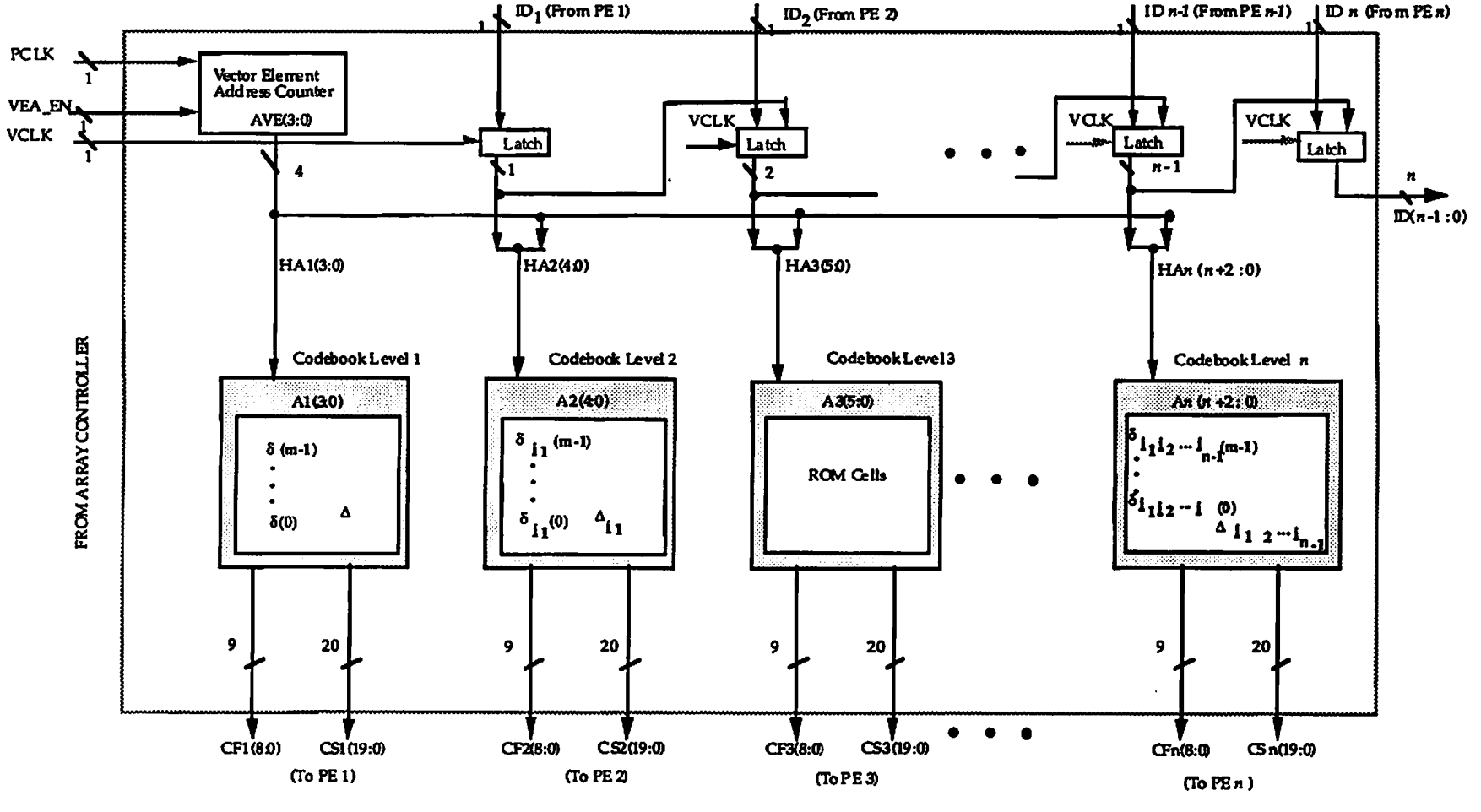


Fig. 3.11 ROM-based raw-codebook memory design of the n-level binary tree-searched VQ.

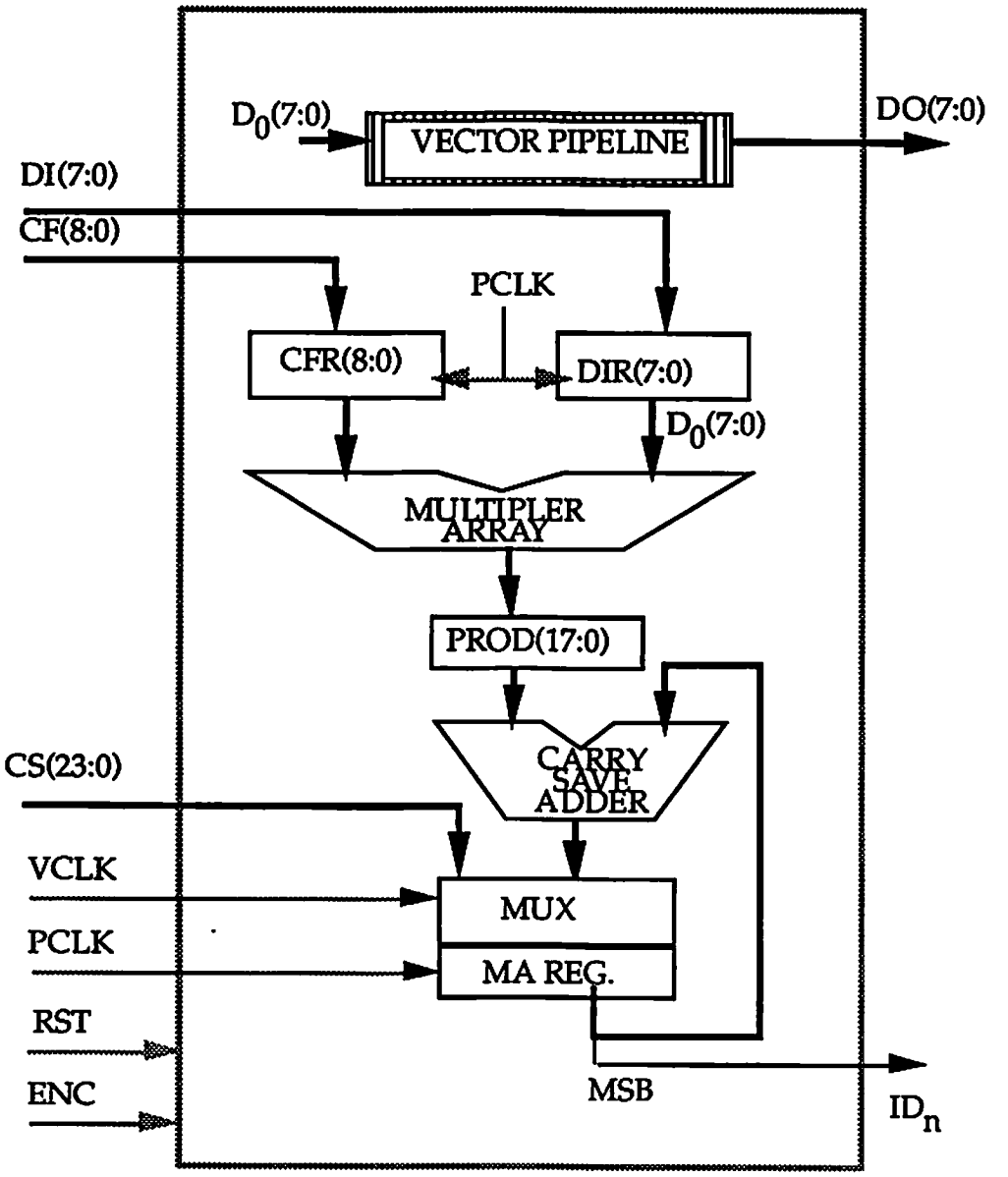


Fig. 3.12 Functional block diagram of the PE design of the difference-codebook BTSVQ.

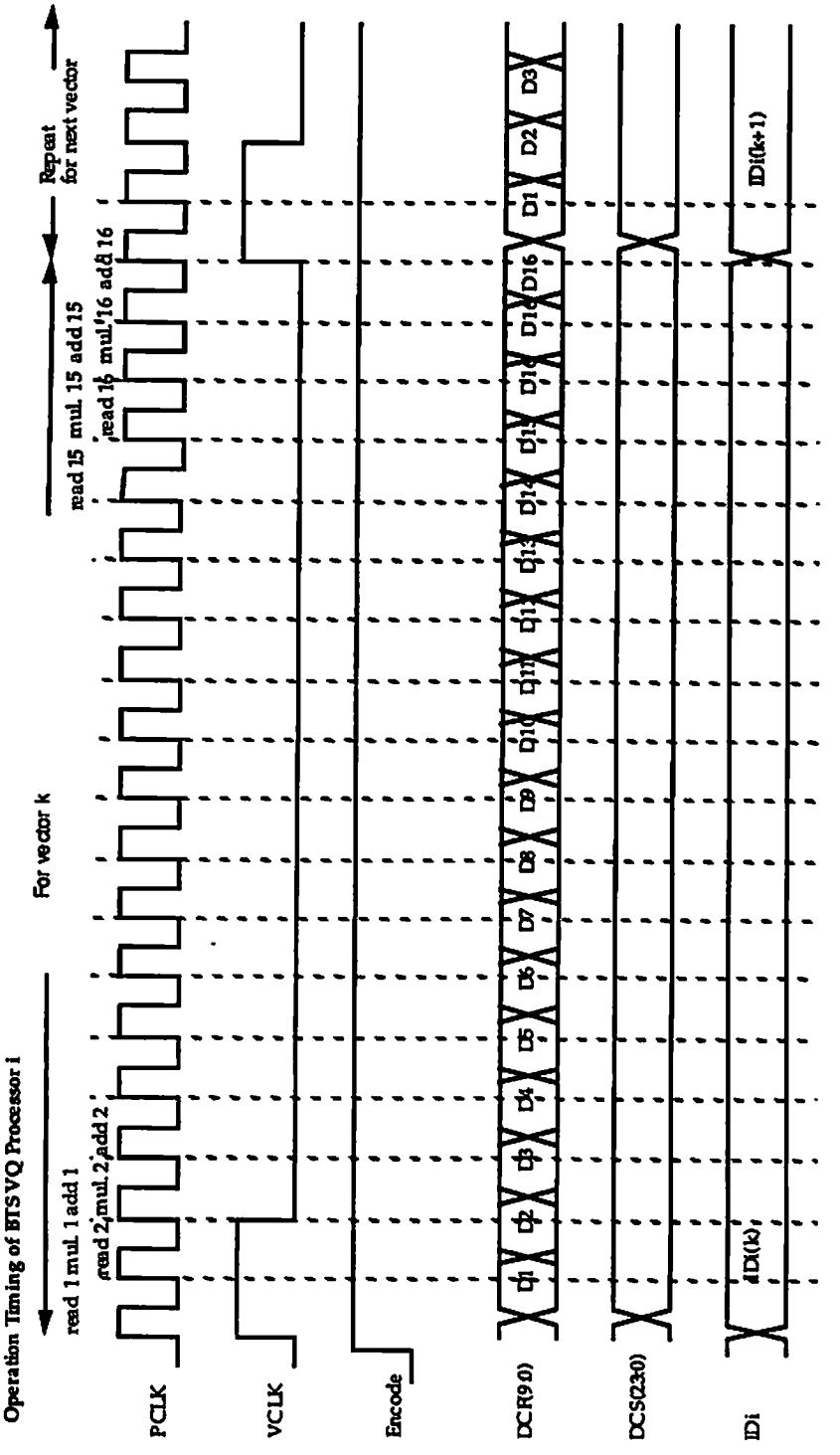


Fig. 3.13 Operation timing of the binary tree-searched vector quantizer.

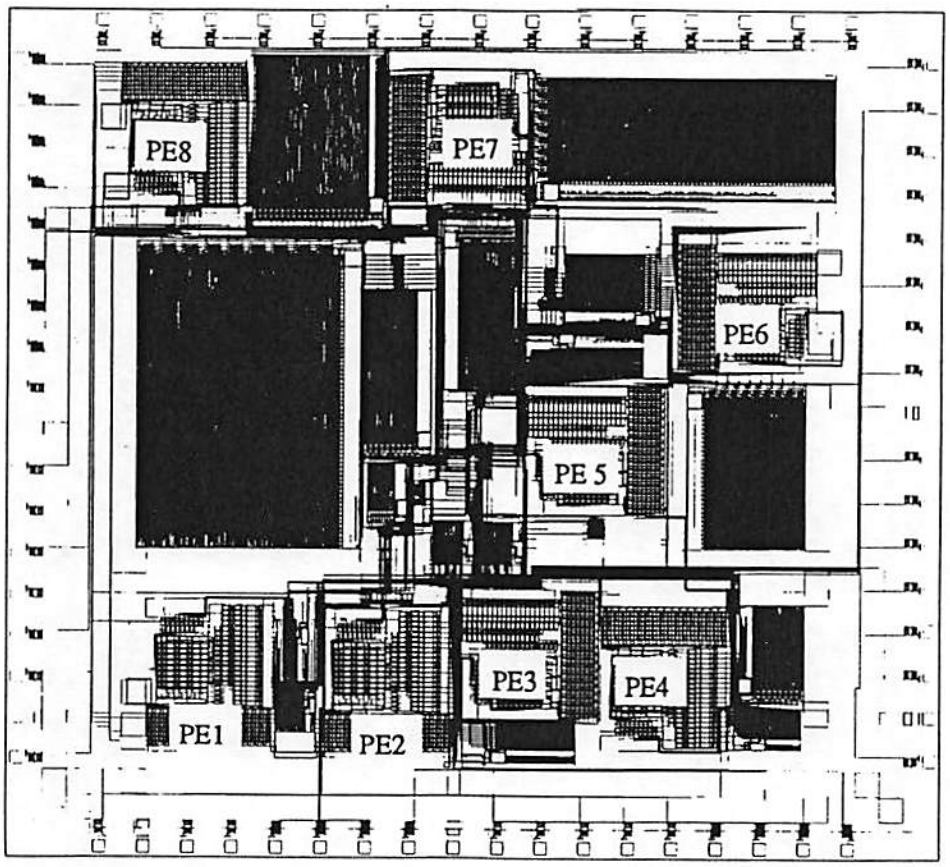


Fig. 3.14 A GDS-II layout of the systolic 8-level BTSVQ chip.

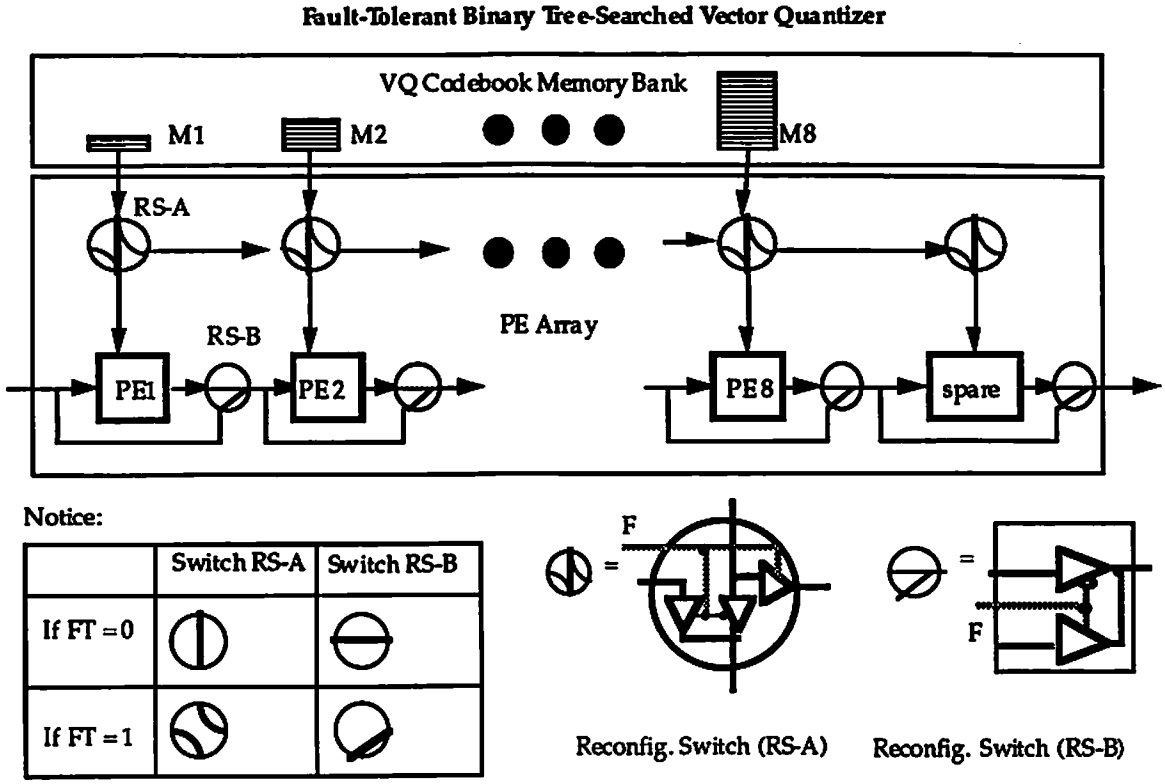


Fig. 3.15 Fault-tolerant systolic array architecture of the BTSVQ.

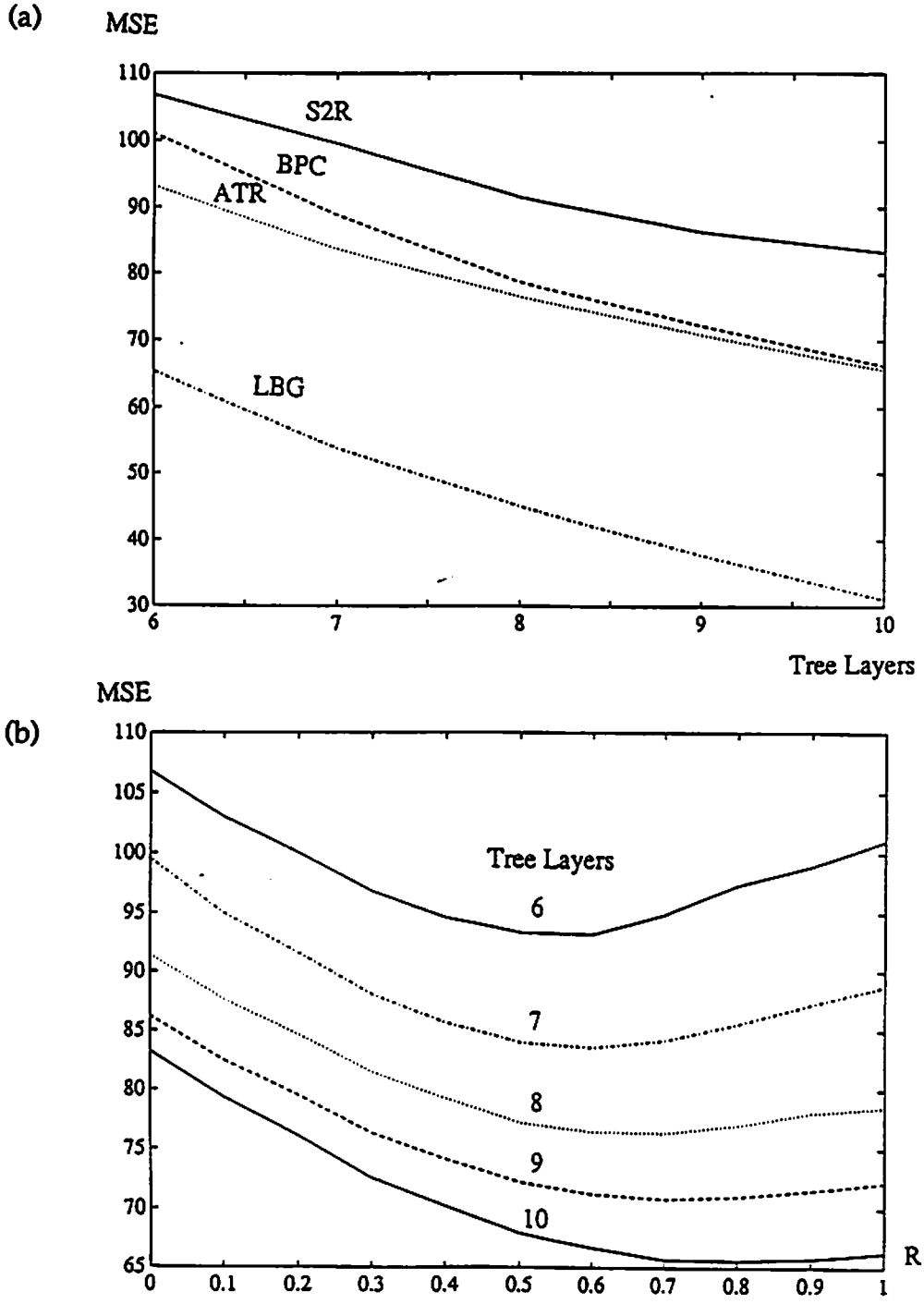


Fig. 3.16 (a) MSE vs. tree layer plot of S2R, BPC, ATR, and LBG for a 4x4 window on the 512x512-pixel 256-gray-level girl image. (b) MSE of the ATR method with scalar training ratio changes from 0 to 1.

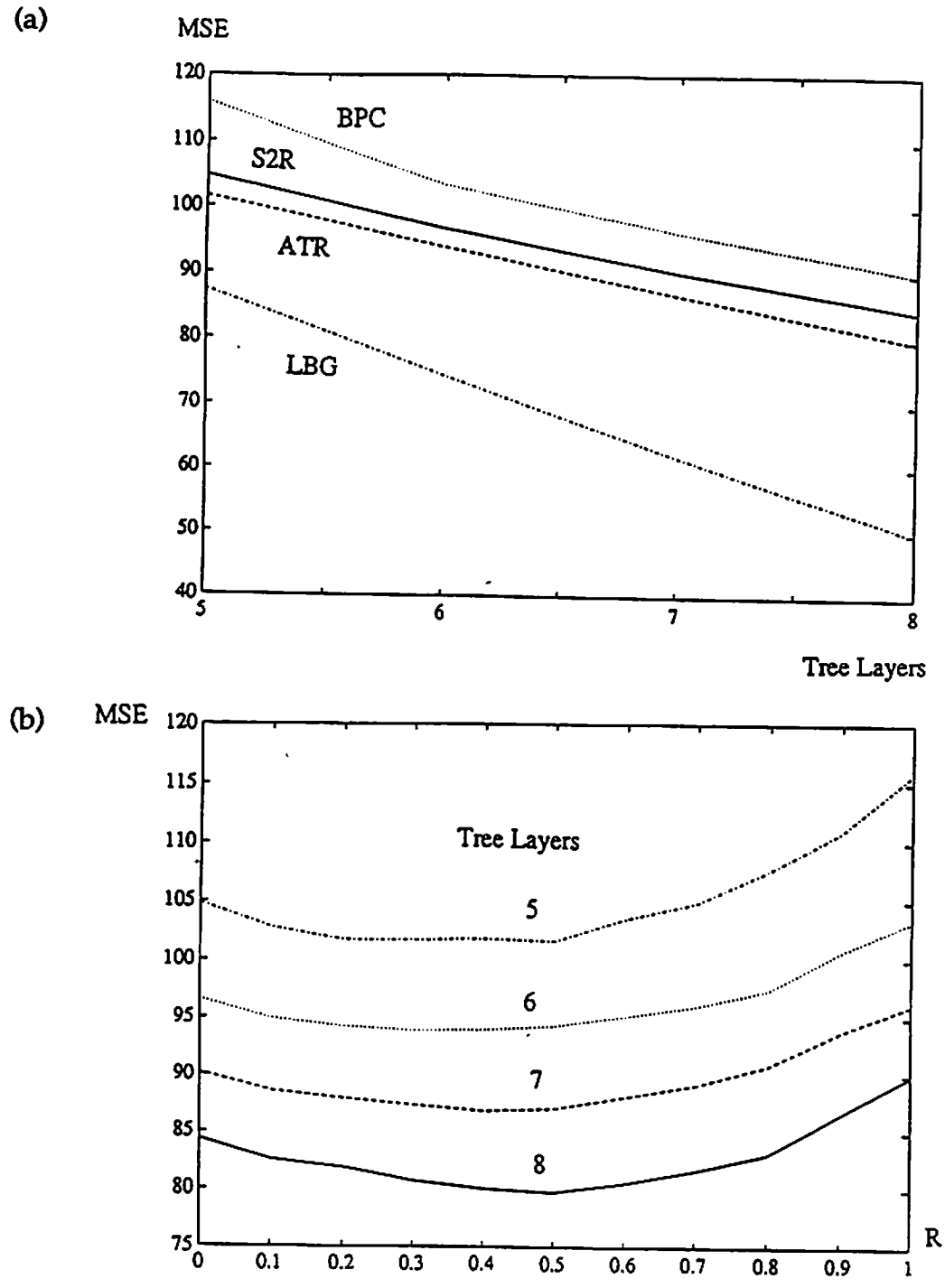
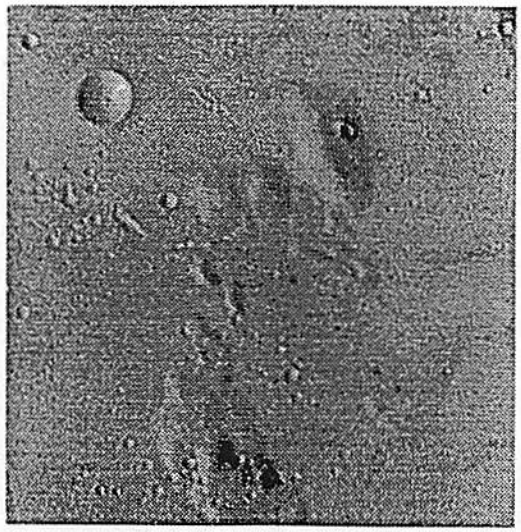
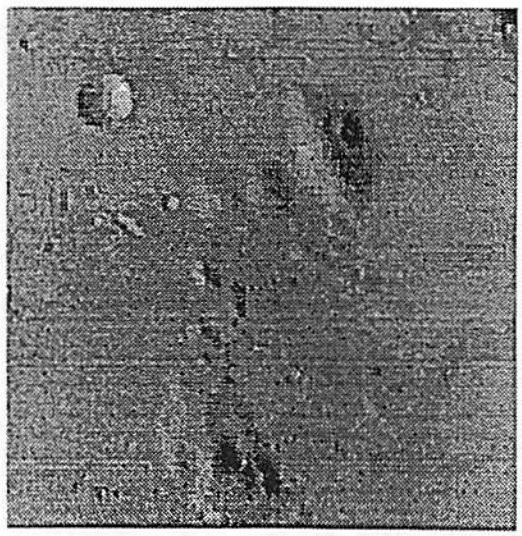


Fig. 3.17 (a) MSE vs. tree layer plot of S2R, BPC, ATR, and LBG for a 4x4 window on the 256x256-pixel 256-gray-level moon image. (b) MSE of the ATR method with scalar training ratio changes from 0 to 1.



(a) Original image



(b)

Fig. 3.18 (a) The original 256x256-pixel 256-gray-level moon image.
(b) The reproduced 256x256-pixel 256-gray-level moon image using 8-level binary tree-searched vector quantizer and ART method; MSE = 79.76; CR =16.

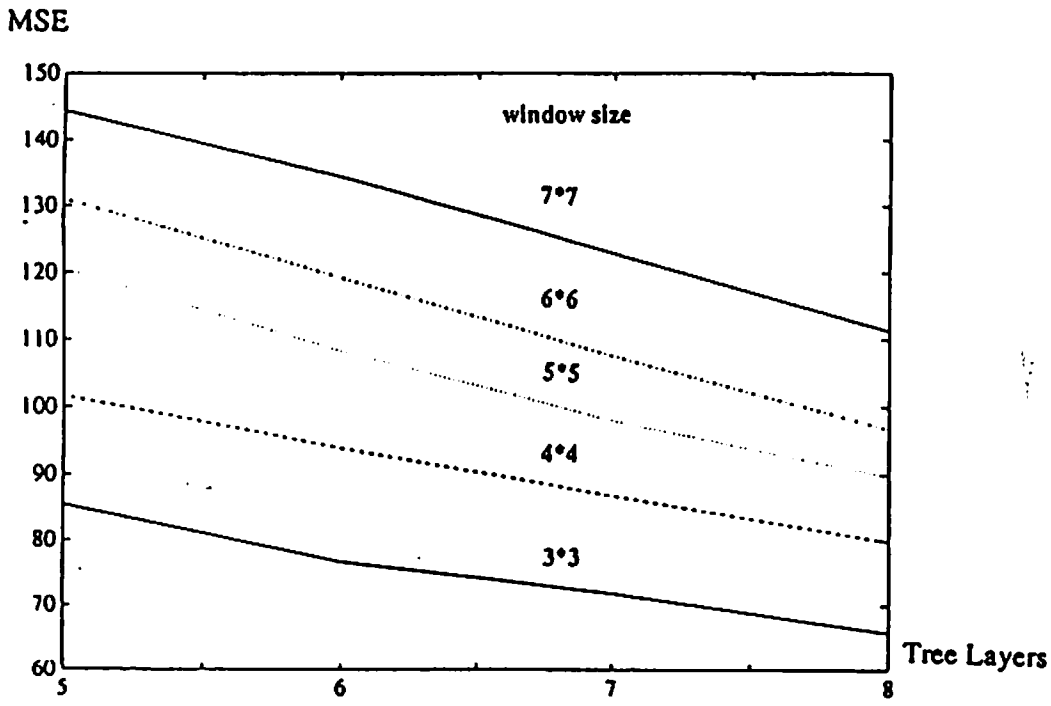


Fig. 3.19 MSE of ATR for various windows on the 256x256-pixel 256-gray-level moon image.

(a) S2R method ; MSE : 83.26

(b) BPC method ; MSE : 66.27

(c) ATR method ; MSE : 65.63

Fig. 3.20 Reproduced Girl image by 10-level binary tree-searched vector quantizer for a 4 x 4 window on the 512 x 512-pixel 256-gray-level Girl image.

(a) S2R method; MSE = 83.26; CR = 12.8.

(b) BPC method; MSE = 66.27; CR = 12.8.

(c) ATR method; MSE = 65.63; CR = 12.8.

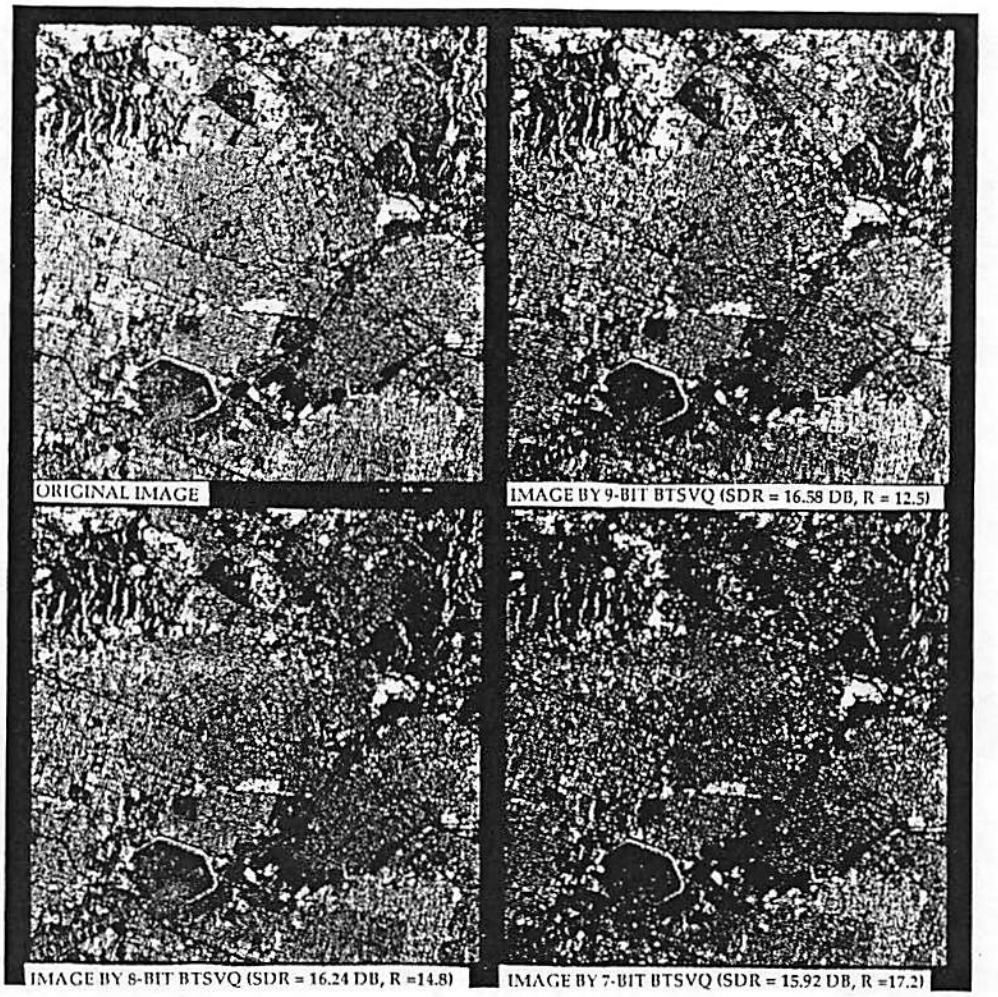


Fig. 3.21 Reconstructed image by using the binary-tree searched vector quantizer and ATR method for a 4x4 window on the 1024x1024-pixel 256-gray-level Synthetic Aperture Radar image of Los Angeles, CA.

(a) 7-level tree; MSE = 148.85; CR = 18.3.

(b) 8-level tree; MSE = 140.87; CR = 16.

(c) 9-level tree; MSE = 135.2; CR = 14.2.

Table 3.1 Hardware performance comparison for variant systolic VQ designs.

	Full-Searched VQ (Input-Vector Projected)	Full-Searched VQ (Codevector Projected)	Tree-Searched VQ (Input-Vector Projected)	Tree-Searched VQ (Codevector Projected)
# of Processors	N	M	$\sum_{l=1}^{L} N_l$	ML
bits of Memory	NMK	NMK	$\left[\sum_{l=1}^{L} N_l^l\right] MK$	$\left[\sum_{l=1}^{L} N_l^l\right] MK$
Throughput Rate (pixel/clock)	1	M/N	1	$(ML)/\left[\sum_{l=1}^{L} N_l\right]$
Pipeline Latency (clocks)	$M+N-1$	$M+N-1$	$\sum_{l=1}^{L} [(N_l - 1)+M]$	$\sum_{l=1}^{L} [M - 1)+N_l]$

Note: N: total number of codevectors; N_l : number of codevectors at level l; n: codebook bit-length; M: vector dimension; K: number of bits per pixel; L: number of tree-levels.

Table 3.2 Hardware performance comparison for variant binary tree-searched VQ designs (codevector projected).

	Binary Tree-Searched VQ w/ Raw Codebook	Binary Tree-SearchedVQ w/Diffference Codebook
# of Processors	2n	n
bits of Memory	2(N-1)MK	(N-1)[M(K+1)+ (2K+log M)]
Throughput Rate (pixel/clock)	1	1
Pipeline Latency (clocks)	n(1+M)	nM

Note: N : total number of codevectors; N_l : number of codevectors at level l; n: codebook bit-length; M: vector dimension; K: number of bits per pixel; L: number of tree-levels; p: pipeline delay of PE (assumed = 1).

Table 3.3 Pin definition of the raw-codebook binary tree-searched VQ.

Signal	Type	Description
MEMORY BANK:		
PCLK	Input	System clock at pixel rate.
VEA_EN	Input	To enable the vector element address generator.
VCLK	Input	System clock for the vector.
A(15:0)	Input	System address bus.
E/T	Input	To select either training mode or encoding mode.
R/W	Input	To select either memory read or memory write.
D(15:0)	Input	System data bus.
DCn(15:0)	Output	16-bit output port of subcodebook # n.
IDn	Input	1-bit index input form PE # n.
ID(n-1,0)	Output	n-bit encoded data for source vector.
PROCESSING ELEMENT # n:		
DCn(15:0)	Input	Codeword-pairs from subcodebook # n.
PCLK	Input	System clock at pixel rate.
VCLK	Input	System clock for the vector.
DI(7:0)	Input	8-bit input image data.
DO(7:0)	Output	8-bit 16-stage pipelined image data.
IDn	Output	1-bit index of source vector generated at PE # n.
RESET	Input	System reset.

Table 3.4 Pin definition of the raw-codebook binary tree-searched VQ.

Signal	Type	Description
MEMORY BANK:		
PCLK	Input	System clock at pixel rate.
VEA_EN	Input	To enable the vector element address generator.
VCLK	Input	System clock for the vector.
A(15:0)	Input	System address bus.
E/T	Input	To select either training mode or encoding mode.
R/W	Input	To select either memory read or memory write.
D(19:0)	Input	System data bus.
DCFn(8:0)	Output	8-bit output port of 1st order subcodebook # n.
DCSn(19:0)	Output	20-bit output port of 2nd order subcodebook # n.
IDn	Input	1-bit index input form PE # n.
ID(n-1:0)	Output	n-bit encoded data for source vector.
PROCESSING ELEMENT # n:		
DCFn(8:0)	Input	8-bit input port from 1st order subcodebook # n.
DCSn(19:0)	Input	20-bit input port from 2nd order subcodebook #n.
PCLK	Input	System clock at pixel rate.
VCLK	Input	System clock for the vector.
DI(7:0)	Input	8-bit input image data.
DO(7:0)	Output	8-bit 16-stage pipelined image data.
IDn	Output	1-bit index of source vector generated at PE # n.
RESET	Input	System reset.

Table 3.5 Size, power, and speed of BTSVQ-8 chip and its building block.

	Size (microns	Power (microwatts	Longest Path Delay (ns)
BTSVQ-8	8672x7719	451311	37.08
PE	1171 x 1629	25778	37.08
10x10-mult.	610 x 664	7334	35
ram128x24	800 x 1558	-	14.5
ram2048x10	2283 x 2815	-	18.19
nand	17.2 x 46.2	-	0.74
Note: The process isNational Semiconductor Corporation's 1 micron N-well silicon gate CMOS process.			

Table 3.6 Chip information

Chip Name	BTSVQ-8
Application	Digital Image Compression
Algorithm	Binary tree-searched vector quantization
Architecture	Systolic Array: 8 PEs
Compression ratio	16
Input format	4x4-pixel block
Output format	8-bit index for each vector
Design Method	Full custom using cell compiler
Process	1.0 micron CMOS
Die Size	8672 x 7719 microns
Total # of Device	300,000
No. of Pads	68
Package	84-pin PGA
Power	0.5 watts
Speed	25 M samples/sec

Table 3.7 The SNR and MSE of S2R, BPC, ATR, and LBG for a 4x4-pixel window on the 512x512-pixel 256-gray-level girl image.

Algorithms / No. of Tree Layers	Binary Tree Search							Full Search	
	S2R		BPC		ATR			LBG	
	SNR	MSE	SNR	MSE	R	SNR	MSE	SNR	MSE
6	22.88	106.81	23.12	101.06	0.6	23.47	93.19	25.01	65.37
7	23.18	99.54	23.68	88.77	0.6	23.94	83.59	25.85	53.84
8	23.55	91.39	24.21	78.56	0.7	24.33	76.37	26.63	45.01
9	23.81	86.23	24.58	72.18	0.7	24.66	70.83	27.40	37.73
10	23.96	83.26	24.95	66.27	0.8	24.99	65.63	28.24	31.05

*SNR: Signal to Noise Ratio **MSE: Mean Square Error ***R: scalar training ratio

Table 3.8 The SNR and MSE of S2R, BPC, ATR, and LBG for a 4x4-pixel window on the 256x256-pixel 256-gray-level moon image.

Algorithms / No. of Tree Layers	Binary Tree Search							Full Search	
	S2R		BPC		ATR			LBG	
	SNR	MSE	SNR	MSE	R	SNR	MSE	SNR	MSE
5	22.12	104.84	21.69	115.86	0.5	22.26	101.62	22.91	87.47
6	22.48	96.63	22.18	103.34	0.3	22.60	93.83	23.62	74.29
7	22.78	90.18	22.50	96.15	0.4	22.94	86.79	24.41	61.88
8	23.07	84.36	22.79	89.95	0.5	23.31	79.76	25.35	49.82

*SNR: Signal to Noise Ratio **MSE: Mean Square Error ***R: scalar training ratio

Chapter 4

A VLSI Neural Processor for Image Data Compression Using Self-Organization Networks

An adaptive electronic neural network processor has been developed for high-speed image compression based upon a frequency-sensitive self-organization algorithm. Performances of this self-organization network and a conventional algorithm for vector quantization are compared. The proposed method is quite efficient and can achieve near-optimal results. The neural network processor includes a pipelined codebook generator and a paralleled vector quantizer, which obtains a time complexity *O(1)* for each quantization vector. A mixed-signal design technique with analog circuitry to perform neural computation and digital circuitry to process multiple-bit address information is used. The prototype neural network processor chip for a 25-dimensional adaptive vector quantizer of 64 codewords was designed, fabricated, and tested. It includes 25 input neurons, 25 x 64 synapse cells, 64 distortion-computing neurons, a winner-take-all circuit block, and a digital index encoder. It occupies a silicon area of 4.6 x 6.8 mm^2 in a 2.0-μm scalable CMOS technology and provides a computing capability as high as 3.2 billion connections per second. The experimental results for this neural-based vector quantizer chip and the winner-take-all circuit test structure are also presented.

4.1 Introduction

Image compression is essential to reduce the image transmission or storage costs for broad areas of applications such as high-definition television, teleconferencing, remote sensing, radar, sonar, computer communication, facsimile transmission, and image database management [4.1]. According to Shannon's source coding theorem, asymptotic optimal performance can be obtained by coding vectors instead of scalars [4.2]. Over the past decade, vector quantization (VQ) has developed from a theoretical possibility into a powerful technique for speech and image compression at medium to low bit rates [4.3]-[4.6]. However, a high-speed VQ adapting to the changing-source data statistics is difficult to implement using the popular Linde-Buzo-Gray (LBG) algorithm [4.6], which requires that the entire training data be processed in a batch mode. Neural network approaches appear to be very promising for intelligent information processing [4.7]-[4.11] due to their massively paralleled computing structures and self-organization learning schemes. A number of studies have reported using artificial neural networks for VQ applications [4.12]-[4.15].

In this chapter, a modified self-organization algorithm and its associated VLSI neural network processor have been developed for adaptive vector quantization. Section 4.2 describes the frequency-sensitive self-organization algorithm and system-level analysis results. Section 4.3 presents a massively paralleled VLSI neural network hardware to implement this algorithm. Section 4.4 discusses the detailed circuit design of the neural network processor chip. Section 4.5 presents the experimental results. The conclusion is given in Section 4.6.

4.2 The Learning Algorithm

The fundamental theory of self-organizing networks was presented by Grossberg [4.7]-[4.9], Kohonen [4.10, 4.11], and other researchers [4.16]-[4.19]. One major challenge of using a basic self-organization network is that some of the neural units may be under-utilized. Various modifications have been proposed to address this problem [4.7, 4.16, 4.20]. Our frequency-sensitive self-organization (FSO) method modifies Grossberg's variable-threshold competitive learning method [4.7, 4.8, 4.9] by applying a winning frequency and its associated upper-threshold value to the centroid-based learning rule. It systematically distributes the codevectors in the vector space R^n to approximate the unknown probability density function of the training vectors. Codevectors quantize the vector space and converge to cluster centroids. This FSO method can produce near-optimal results, which will be shown later.

A synapse weight vector is stored as a codevector. In the one-iteration FSO scheme, the training data must pass once in constructing the codebooks. It is a fast and powerful scheme for adaptive vector quantization due to its relatively low computing requirement and massively paralleled computing structure. The one-iteration FSO scheme for adaptive vector quantization is described as follows:

1) Initialize the codevectors $\mathbf{W}_i$ and the winning frequency F_i for each distortion-computing neuron:

$$\mathbf{W}_i(0) = \mathbf{R}(i), \tag{4.1}$$

$$F_i(0) = 1,\ i = 1,\ ...,\ N,$$

where **R**(.) is a random vector-number generation function, M is the number of vector components, N is the number of codevectors, and $\mathbf{W}_i(0) = [W_{i1}(0), W_{i2}(0), ..., W_{iM}(0)]$. Notice that the first N input vectors can also be used as the initial codevectors instead of using results generated from **R**(.).

2) Compute the distortion $D_i(t)$ between an input vector $\mathbf{X}(t)$ and all codevectors:

$$D_i(t) = d(\mathbf{X}(t), \mathbf{W}_i(t)) = \sum_{j=1}^{M} (X_j(t) - W_{ij}(t))^2, \tag{4.2}$$

where t is the training time index.

3) Select the distortion-computing neuron with the smallest distortion and set its output $O_i(t)$ to high:

$$O_i(t) = \begin{cases} 1 & \text{if } D_i(t) < D_j(t),\ 1 \le i, j \le N,\ i \ne j, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

4) Update the codevectors with a frequency-sensitive training rule and the associated winning frequency:

$$\mathbf{W}_i(t+1) = \mathbf{W}_i(t) + S(t)\, O_i(t)[\mathbf{X}(t) - \mathbf{W}_i(t)], \tag{4.4}$$

$$S(t) = \begin{cases} \dfrac{1}{F_i(t)} & \text{if } 1 \le F_i(t) \le F_{th}, \\ 0 & \text{otherwise,} \end{cases} \tag{4.5}$$

$$F_i(t+1) = F_i(t) + O_i(t), \tag{4.6}$$

where $S(t)$ is the frequency-sensitive learning rate, and F_{th} is the upper-threshold frequency. Notice that only the winning codevector is updated. The training rule moves the winning codevector toward the training vector by a fractional amount which decreases as the winning frequency increases. If F_i

is larger than F_{th}, then set S(t) to zero and no further training will be performed for this neural unit.

5) Repeat steps (2) through (4) for all training vectors.

Use of the upper-threshold frequency can avoid codevector under-utilization during the training process for an inadequately chosen initial codebook. The selection of the upper-threshold frequency is heuristic and depends on source data statistics and training sequence. Empirically, an adequate F_{th} is chosen to be two to three times larger than the average winning frequency. The impact of different upper-threshold frequency values on mean-squared errors is illustrated in Fig. 4.1. The initial codebooks are created from a random number generation function. A good upper-threshold frequency value lies between 350 and 450 for a synthetic aperture radar (SAR) Ice image and between 250 and 350 for a Girl image.

The performance of the one-iteration FSO method can be incrementally improved by using iteration to adjust codevectors into better cluster centroids. The codebook obtained from the previous iteration is used as the initial values of the current iteration. After the first iteration, the upper-threshold frequency is not needed, because a good initial codebook is available. This method is called the multiple-iteration FSO method.

In the LBG method, the initial codebook could be obtained from the splitting-2 algorithm [4.6]. The iteration of grouping and calculating centroids in the LBG method is similar to that of updating the closest codevector for each incoming data vector through the centroid technique in the FSO method. Therefore, the iterating FSO method without the use of upper-threshold frequency asymptotically approximates the LBG method. If the learning

process in the FSO method is repeated with the same termination criterion for the LBG method, the result of the multiple-iteration FSO method appears very close to that of the LBG method.

The original and reconstructed SAR Ice images using the one-iteration and two-iteration FSO methods for the 10-bit codebook are shown in Fig. 4.2. Histograms of the reconstructed images are almost identical to that of the original image as illustrated in Fig. 4.3. The mean-squared error (MSE) measure is used to evaluate the reconstructed image quality,

$$MSE = \sum_{x=0}^{N_1-1} \sum_{y=0}^{N_2-1} \frac{\| I(x, y) - I'(x,y) \|^2}{N_1 N_2} \qquad (4.7)$$

where I is the original image of size $N_1 \times N_2$ and I' is its reconstructed image. The MSE values of images using the one-iteration, two-iteration FSO methods and the LBG method are listed in Table 4.1 and also plotted in Fig. 4.4. The performance figures given for the three algorithms are obtained by training the same picture and measuring MSE with Eq. (4.7). Performance of the FSO method is very close to that of the LBG method. The reconstructed images using the FSO method on 5 x 5-pixel subimage blocks are reasonably good.

The large dynamic range of images requires that the effective compression algorithms are adaptive to the image statistics. For the vector quantization approach, edge degradation is very severe if no adaptation is allowed for different scene characteristics. If the codebook trained from the FSO method for the SAR Ice image shown in Fig. 4.2(a) is used to encode and decode the Girl image without any modification, the mean-squared error is 1063, as shown in Fig. 4.5(a). After training this inadequate codebook by using the FSO method, a much smaller MSE value of 60 can be achieved as

shown in Fig. 4.5(b). This result illustrates that the codebook can be successfully adjusted according to the statistical change of the source data.

In an electronic system design, the resolution limit for the synapse matrix by analog circuitry is a very important factor. The simulation results from the FSO method for different signal resolutions are shown in Fig. 4.6. Performance of the 8-bit resolution case is reasonably close to that of floating point computation. If the codebook size is large, the performance for the 8-bit resolution and that for the floating point computation are not distinguishable. In a large codebook, each codevector is trained from a small portion of source data and, thus, the error induced by finite resolution is also small.

4.3 VLSI Neural Processor Architecture

The proposed very large-scale integration (VLSI) neurocomputing architecture for adaptive image compression using the frequency-sensitive self-organization network is shown in Fig. 4.7. The FSO network consists of two layers: an input layer and a competitive layer.

The input layer consists of M input neurons, which correspond to the elements of the M-dimensional input vector. Each input neuron gets its input from the external data bus and distributes the buffered signal to N distortion-computing neural units in the competitive layer through the synapse matrix. Each distortion-computing neuron calculates a square of Euclidean distance between its codevector and the input vector. The competitive process is performed throughout the whole layer by the winner-take-all operation. The winning neural unit is determined according to the

minimum distortion criterion. The synapse weights are then updated according to the FSO learning rule as specified in Eqs. (4.4), (4.5), and (4.6).

By using the massively paralleled neural computing paradigm and the mixed-signal VLSI design technique, the FSO network can be implemented on VLSI. The block diagram of a VLSI design of the FSO neural network processor is shown in Fig. 4.8. The high-level functional blocks of this neural network processor include a paralleled vector quantizer chip and a digital pipelined codebook generator chip.

For the paralleled vector quantizer, a mixed-signal VLSI design technique is used. The analog circuitry performs massively paralleled neural computation and digital circuitry processes multiple-bit address information. This neural-based vector quantizer realizes a full-search vector quantization process for each input vector at a time complexity *O(1)*. It consists of the input neurons, programmable synapses, summing neurons, winner-take-all cells, and an index encoder. The programmable synapse matrix is composed of $M \times N$ synapse cells, which correspond to N M-dimensional codevectors. The output neuron array is composed of N summing neurons, which perform paralleled summation of the distortions between the input vectors and codevectors. The winner-take-all block consists of N competitive circuit cells which perform paralleled comparison among N *inverted* distortion values and choose a single winner. This block also provides a sufficiently high output level for the winning neuron against the rest. The index encoder circuit is an N-to-n decoder that uses binary codes to encode N classes.

For the digital pipelined codebook generator, the digital-signal-processing (DSP) circuit can be used. The digital codebook generator is a

coprocessor to support a high-speed neural-network learning algorithm. The generator consists of an interface and timing control block, a DSP-based FSO trainer, a dual-port vector memory, a vector address handler, and a digital-to-analog (D-to-A) converter array. The vector address handler uses the digital n-bit index of the winning neural unit generated by the paralleled vector quantizer chip to access the corresponding winning codevector and frequency. The DSP-based FSO trainer counts the winning frequency and then updates the codevector of the winning neural unit. The updated codevector is written to both the digital codebook memory and the analog synapse matrix. The digital codebook memory is built with two-port dynamic memory and organized as N-word by $8xM$-bit to reduce the I/O communication traffic. An incremental adaptation of the codebook is performed in a read-modify-write cycle only when a winning codevector is chosen. The vector address handler is also shared by the paralleled vector quantizer to address the corresponding synapses for codevector loading, modification, and refreshing. The image data is accessed from the host computer through the interface and timing control block. The image data is divided into subimage blocks and can be stored in the dual-port vector memory. A subimage block is handled as a digital input vector. The digital input vector is converted into the analog value using the D-to-A converter array and fed to input neurons of the paralleled vector quantizer.

4.4 Detailed Circuit Implementation

Advanced studies to improve the circuit performance and to reduce the area/power of the electronic building blocks are essential to implement the highly complex neural systems in VLSI technologies [4.21, 4.22, 4.23]. Computer simulation and laboratory experiments on these neural circuits have been conducted.

4.4.1 Input Neuron

In the input layer, each input neuron consists of a unity-gain buffer. The input signal is composed of M input lines and each input line is applied to one row of N synapse cells. The load capacitance for the input neuron is quite significant. It is around 5 pF for the N = 64 case. Thus, the input voltage needs to be buffered before it is distributed to the synapse cells. The input neuron is a conventional operational amplifier in a unity-gain configuration. The experimental input neuron has a DC gain of 95.82 dB. The settling time to within 0.1% accuracy is 80 nsec for the 3 *Vp-p* input pulse.

4.4.2 Programmable Synapse

The programmable synapse design is a modified wide-range Gilbert multiplier [4.22], which can perform real-valued multiplications in four quadrants and achieve 8-bit precision. In order to realize the function specified in Eq. (4.2), the synapse cell calculates the square of the difference between the input voltage and the synapse weight value. Figure 4.9 shows the circuit schematic of the synapse cell and the size of each transistor. The layout of one synapse occupies 112 λ x 82 λ in the MOSIS-scalable CMOS design [4.24].

In order to achieve a wide operation range, the components of the differential pair for $(V_1\text{-}V_2)$ and $(V_3\text{-}V_4)$ are separated using the current mirror circuitry. The output current is obtained from the cascade-current mirror stage, consisting of transistors M_{21} through M_{24}. It can be approximated by

$$I_{out} = \sqrt{\frac{k\beta_1\beta_{11}}{2}}\,(V_1\text{-}V_2)(V_3\text{-}V_4), \qquad (4.8)$$

where k is the current gain from transistor $M_{3(4)}$ to transistor $M_{13(16)}$, and β_1 and β_{11} are the transconductance coefficients of transistors M_1 and M_{11}, respectively. Figure 4.10 shows the measured DC characteristics of the synapse cell. The linearity error is less than 2% for an input voltage range of -1.5 V to 1.5 V. In order to calculate $(X_j\text{-}W_{ij})^2$, the inputs are rearranged. Here X_j is applied to V_1, V_3 and W_{ij} is applied to V_2, V_4 for the (i,j)th synapse cell. Then Eq. (4.8) can be simplified to

$$I_{ij} = \sqrt{\frac{k\beta_1\beta_{11}}{2}}\,(X_j\text{-}W_{ij})^2, \qquad for\ 1 \le i \le N,\ and\ 1 \le j \le M. \qquad (4.9)$$

In Fig. 4.11, the simulated output characteristics of the square function are shown. The center of the parabolic curves is shifted by the weight values. The input voltage X_j is fed into the synapse cell through the input neuron. The synapse weight value is dynamically stored on the capacitance of the MOS transistors. It must be refreshed periodically since a parasitic leakage exists at the diffusion-to-substrate junction.

4.4.3 Output Summing Neuron

The output summing neuron converts the summed current into an analog voltage, which is sent to the winner-take-all (WTA) circuit. The output

of each summing neuron is the distortion measure. The minimum among the distortion measures is chosen as the winner in the competitive layer. Since the WTA circuit only selects the maximum input, the sign of the output of each neuron needs to be reversed. The inverting operation of the output neuron converts the summed current into the voltage with the sign reversed as shown in Fig. 4.12(a). The circuit diagram and the transistor sizes for the amplifier used in the output neuron are shown in Fig. 4.12(b). Current summation occurs at every column of the synapse matrix. To ensure the linear current-to-voltage conversion for a wide operation range, the output neuron is designed to support a summing current of more than 1 mA. It has a large output buffer to handle a large amount of summed current. The settling time to within 0.5% accuracy is 25 nsec for a 0.8-mA summed current and 2-pF load capacitance. Linear resistance is used to convert the current into the voltage. Since the accuracy of an untrimmed passive resistor is very low (possibly up to 20% error), multiple MOS transistors biased in the triode region are used to synthesize the linear resistance [4.25]. The layouts of the current summing neuron and the linear floating resistor occupy 116 λ x 228 λ and 116 λ x 64 λ, respectively.

4.4.4 Winner-Take-All Cell

The performance of the WTA circuit built with transistors biased in the subthreshold region [4.26] is moderately limited due to the inherently low-speed operation and a small noise immunity. Our modified WTA circuit operates in a strong inversion region and can provide fully binary output values that are easily interfaced with digital circuitry for network learning.

Our analog WTA circuit can determine the winning cell at one cycle, instead of $\log_2 N$ clock cycles using MAXNET [4.27].

The WTA circuit schematic and the size of each transistor are shown in Fig. 4.13. Each cell occupies 58 λ x 96 λ in the scalable CMOS design. One WTA cell consists of two portions. The first portion converts input voltage into the current which is compared and redistributed in the common signal line. In the second portion, the current is converted into the output voltage. All transistors operate in the saturation region. V_{CM} is the common-node voltage to which all source terminals of input transistors $M1$ are connected. As the number of inputs increases, the circuit can be extended by abutting this common signal node from the cells. Through this node, the total bias current is contributed by every cell. Since the source terminal is at the same voltage for all the cells, the current flowing through each cell is proportional to V_i^2. Thus, the largest input can fetch the largest current out of the total bias current. This largest current can make the corresponding output saturated at the positive supply voltage value. On the other hand, the other outputs will be pushed toward the negative power supply value. The total bias current is provided by the transistor M_5 of the cells. Instead of using a fixed amount of the total bias current, each cell provides its own share of the bias current. Since the bias current increases in proportion to the number of inputs, the circuit response time is quite independent of the number of inputs.

4.4.5 Analysis of Large Number of Winner-Take-All Cells

Let the inputs with the same input voltage level be assigned into groups 1 through L. The group i has n_i elements. The current flowing through each cell in group i is

$$I_i = \frac{\beta_1}{2}(V_i - V_{CM} - V_{th})^2 \tag{4.10}$$

The total bias current is distributed in the following way:

$$\sum_{i=1}^{L} n_i I_i = N I_B \tag{4.11}$$

and

$$N = \sum_{i=1}^{L} n_i \tag{4.12}$$

where N is the number of competitive inputs, and I_B is the bias current flowing in transistor M_5 of each cell. When the input voltage to the j-th group is the largest, the number of cells in the j-th group should be one and the current flowing in this cell, I_j , should be larger than the current flowing through a single cell in any other group to ensure the winner-take-all operation,

$$I_j > \max \{I_i , i = 1, 2, ..., L, \text{ and } i \neq j \} \tag{4.13}$$

or equivalently

$$V_j > \max \{V_i , i = 1, 2, ..., L, \text{ and } i \neq j \}. \tag{4.14}$$

To facilitate the analysis of this WTA circuit in a large network, the following conditions are assumed. There are three groups of input voltages: the winning voltage V_W, the second largest input voltage V_L, and the smallest one V_S. The numbers of cells in these groups are 1, L, and N-L-1, respectively. From Eq. (4.10), the current flowing through a single cell in each group can be expressed as

$$I_W = \frac{\beta_1}{2}(V_W - V_{CM} - V_{th})^2,$$
$$I_L = \frac{\beta_1}{2}(V_L - V_{CM} - V_{th})^2, \qquad (4.15)$$

and

$$I_S = \frac{\beta_1}{2}(V_S - V_{CM} - V_{th})^2.$$

The total current is

$$I_{total} = 1 \times I_W + L \times I_L + (N-L-1) \times I_S. \qquad (4.16)$$

In Fig. 4.14, calculated results for a 1000-input WTA circuit are shown. V_W, V_L, and V_S are set to 2.51, 2.50, and 2.49 V, respectively. As the number of the second largest input increases, the current flowing into the winning cell decreases monotonically. This results from the fact that more current is consumed by the cells in the second group, while the total bias current is constant. In Fig. 4.14(a), calculated results of the output levels are shown. In Fig. 4.14(b), the response time of the winning output voltage is shown. The output level of the winning cell decreased due to the reduced available current. Similarly, the response time of the circuit increased, because the amount of charging current was reduced.

In the case where the differences between competitive inputs are small, the performance of the WTA circuit can be degraded severely. The resultant input current in the winning cell is not large enough to be completely differentiated from all losing cells. Thus, the output voltage difference between the winning and losing cells is not large enough to be directly interfaced with the digital index encoder, because the winning cell output has intermediate value between 0 and 5 V. Although the winning output is still larger than any other output, the response time for the above

condition is quite long. More charging current is consumed by the transistor $M3$ (Fig. 4.13), which is now biased in the linear region. To circumvent these problems, a cascaded version of the winner-take-all circuit can be used.

One of the main sources to restrict the number of the WTA cells to be connected side by side is the resistance along the common signal line. An analysis of the parasitic-resistance effects on the number of the WTA cells is given in Appendix 4.A.

4.4.6 FSO Network

Figure 4.15 shows the functional block diagram and physical layout of one slice of the FSO network consisting of key circuit blocks. The simulated processing time for one network iteration is less than 500 nsec. Each iteration cycle includes input buffering, synapse multiplication, neuron summing, winner-take-all operation, and index encoding. In Fig. 4.16, SPICE simulation results of the circuit are shown for one typical operation. The load capacitances for each block are effectively included. The major delay in the network is at the input neuron due to the large load capacitance associated with the long signal wire. Transistor sizes of the input neurons can be increased to reduce the delay time.

In the neural-based vector quantizer chip design, the number of 8-bit synapse cells for each output summing neuron is 25. Careful transistor sizing and layout generation are performed. The common-centroid layout technique can greatly alleviate the device mismatch effect [4.25]. Use of large devices can also reduce sensitivity to mobility variation and channel-length modulation variation [4.28]. The carefully chosen device sizes of Fig. 4.9 help

to keep the variation of the transconductance constant β below 1% [4.29]. In addition, the operational dynamic range of the synapse cell can be increased with larger devices, which can ensure the 8-bit accuracy for the analog circuitry. Notice that the analog-computing accuracy requirements for image compression are much more relaxed than those for image recognition [4.30]. It is probable to have the second closest codevector represent the input vector if these two codevectors are extremely close. The induced degradation is negligible from a lossy image-compression view. In general, the distances among codevectors are much larger than that can be differentiated by one least significant bit. This situation is supported by our FSO algorithm. However, the high-accuracy vector matching may be required for other applications such as pattern recognition with a large vector dimension. For high-accuracy vector matching applications, an analysis of device-variation effects on the FSO network dimensionality is given in Appendix 4.A.

4.5 Testing of the Neural Network Chip

Testing of VLSI neural network chips is an important task in constructing artificial neural systems. The experimental results for the neural-based vector quantizer chip and its associated circuit cells are presented.

The testing of the vector quantizer chip was performed by using a dedicated test bed as shown in Fig. 4.17. It consists of an IBM PC/AT computer, an digital signal processing coprocessor, an interface board, an HP 1650A logic analyzer, and an HP 4145B semiconductor parameter analyzer.

The HP 4145B analyzer is used to measure the characteristics of the neural circuits. The logic analyzer is used for diagnosing and debugging digital functions. An interface board is used to accept the image data from the host computer and to convert them into the analog signal format. The analog outputs of the 25 D-to-A converters are sent to the neural-based vector quantizer chip via 25 sample-and-hold LF398A amplifiers from Texas Instruments Inc. The Motorola DSP56000 boards are used for updating the synapse values. During the learning phase, the DSP board receives the codebook index and calculates the new synapse values. These output data and new synapse values are stored in the digital memory. The updated synapse values are converted into the analog signal and sent to synapse cells.

To investigate the characteristics of large WTA circuit, a separate 200-cell WTA circuit was tested. The measured results are shown in Fig. 4.18. The input of cell-101 is increased linearly. The second largest input is set to 2.52 V, and the others to 2.50 V. Figure 4.18(a) shows five curves representing the output voltages of cell-101 that correspond to the cases where the numbers of the second largest inputs are 2, 50, 100, 150, and 199, respectively. A higher input voltage is needed for the winning cell since more current flows to the group of cells having the second largest input as its number increases. Figure 4.18(b) shows the variation of the threshold for a winning cell across the test structure. The five curves correspond to the outputs of cell-1 with different positions of the second largest input at cell-2, cell-50, cell-100, cell-150, and cell-200, respectively. From the measurement results, it is clear that a cell can be a winner if its input is greater than those of the other cells by 15 mV. The measured response time of the WTA circuit is around 60

nsec at 1-pF load capacitance for cases ranging from 50 cells to 200 cells. As the number of the cells increases, the charging current available for the winner is increased. On the other hand, the parasitic capacitance through the common signal line is also increased. The combined effects of these two factors determine the response time.

The prototype neural network processor chip for a 25-dimensional adaptive vector quantizer of 64 codevectors was fabricated in a silicon area of 4.6 mm x 6.8 mm using the 2-μm CMOS technology from the MOSIS Service of the USC/Information Sciences Institute at Marina del Rey, CA [4.24]. This prototype chip includes 25 input neurons, 25 x 64 synapse cells, 64 distortion-computing neurons, a winner-take-all circuit block, and a digital index encoder. The die photo of this neural-based vector quantizer chip is shown in Fig. 4.19. The power lines from analog and digital blocks are separated to avoid noise coupling from digital parts to the highly sensitive analog parts. This chip can also be extended to implement vector quantization of a larger codebook. An adaptive vector quantizer of 1024 codevectors can be implemented by cascading 16 such prototype chips or by using a larger design in a submicron fabrication technology [4.31]. The performance of the neural-based vector quantizer chip is summarized in Table 2.

In Fig. 4.20, the functional testing results of the vector quantizer chip is shown. The original image is a 100x100-pixel House image. The reconstructed image is created by using the codebook generated by the neural computing test bed.

4.6 Conclusion

A frequency-sensitive self-organization network has been described and shown to be effective for adaptive vector quantization. The efficiency of this FSO network is measured by its compression ability, the resulting distortion, error tolerance, and the suitability for VLSI implementation. Based upon this frequency-sensitive self-organization method, a neural-based adaptive vector quantizer has been developed. By using a mixed analog-digital design approach in the massively paralleled computation blocks, the advantages of small silicon area, low power consumption, and reduced I/O requirement can be achieved. A VLSI chip for 25-dimensional vector quantizer of 64 codevectors has been fabricated and tested. Its throughput rate is 2 million vectors per second and its equivalent computation power is 3.2 billion connections per second. It achieved an intrinsic compression ratio of 33.

Appendix 4.A: Analysis of Non-ideal Effects

Fundamental limitations of analog neural computing are caused by non-ideal factors such as the process variation, transistor mismatches, and offset voltage. Several effects that determine the dimensionality of the network with the 8-bit accuracy are analyzed.

4.A.1 Effects of Device Variations on the FSO Network Dimensionality

The operation of the network is restricted by the variations of device parameters such as process non-uniformity and transistor mismatches in synapse matrix. To consider the effect of the parameter variations of devices on the processing accuracy, two columns are considered among the synapse matrix. In two columns, all pairs of two synapse values are assumed to be the same except for one pair of synapse values as follows:

$$W_{1i} = W_{2i} = W_i \quad \textit{for all } i \textit{ and } i \neq k. \tag{A1}$$

The summed output current in each column is expressed from Eq. (9) as

$$I_1 = I_{1k} + \sum_{i=1,\, i\neq k}^{M} I_{1i}\,, \tag{A2}$$

and

$$I_2 = I_{2k} + \sum_{i=1,\, i\neq k}^{M} I_{2i}\,. \tag{A3}$$

The difference of these two currents is

$$I_{out} \equiv I_1 - I_2 = I_{lsb} + I_\Delta\,, \tag{A4}$$

where

$$I_{lsb} = \sqrt{\alpha_{1k}}(X_k - W_{1k})^2 - \sqrt{\alpha_{2k}}(X_k - W_{2k})^2, \tag{A5}$$

and

$$I_\Delta = \sum_{i=1,\, i\neq k}^{M} (\sqrt{\alpha_{1i}} - \sqrt{\alpha_{2i}})(X_i - W_i)^2. \tag{A6}$$

Here, I_{lsb} is the current determined from the desired synapse values, while I_Δ is the current due to the variations of the device parameters. Ideally, if there are no parameter variations in the devices, then I_Δ is zero and the difference current can be determined only from I_{lsb}. In practice, the absolute current value determined from the desired synapse values must be larger than that due to the device parameter variations. That is,

$$|I_{lsb}| > |I_{\Delta}|. \tag{A7}$$

Considering the following conditions:

$$\alpha_{1k} = \alpha_{2k} = \alpha_0,\ \alpha_{1i} = \alpha_0\ (1 + \Delta),\ \alpha_{2i} = \alpha_0 (1 - \Delta),$$

then Eqs. (A5) and (A6) become

$$|I_{lsb}| = |\sqrt{\alpha_0}\,[(X_k - W_{1k}) + (X_k - W_{2k})]\,(W_{1k} - W_{2k})| \tag{A8}$$

and

$$|I_{\Delta}| = \left| \sum_{i=1,\, i \neq k}^{M} \sqrt{\alpha_0}\left(\sqrt{1 + \Delta} - \sqrt{1 - \Delta}\right)(X_i - W_i)^2 \right|, \tag{A9}$$

respectively. For illustration purposes, assume that all inputs are to be matched to their weight values in the post-training process so that

$$X_i - W_i = V_m \text{ for all } i. \tag{A10}$$

Then the number of the possible inputs is determined as

$$M < 1 + \frac{2\,|W_{1k} - W_{2k}|}{V_m\left(\sqrt{1 + \Delta} - \sqrt{1 - \Delta}\right)}. \tag{A11}$$

In the case of 8-bit accuracy computation, the minimum difference of the synapse value is $V_{FS} / 2^8$. V_{FS} is the full-scale dynamic range. The number of possible inputs in Eq. (A11) is shown in Fig. 4.A.1 with respect to V_m. If the variation is within 1.5%, then the 5x5 input can be applied with the matching between input and synapse values of 1% of the full dynamic range.

4.A.2. Effects of Parasitic Resistance on the Number of the WTA Cells

One of the main sources to restrict the number of the WTA cells to be connected side by side is the resistance along the common signal line. An

analysis of the parasitic-resistance effects on the number of the WTA cells is given in this section.

Through the common signal line, the input currents are redistributed and compared one another. In general, the common signal line is made of the metal, of which sheet resistance is very small. However, the length of this line is so long that the resistance value cannot be ignored when the connected number of cells are large and comparison occurs between two far ends of this line. The voltage drop across the common signal line causes the gate-to-source voltage of each input transistor to be different although the applied input voltage is the same.

To analyze the effect of this finite resistance value along the common signal line on the number of the cells, simple model is introduced in Figure A.2, where each cell is represented by the equivalent current source. The current flowing through each cell is IB for the state of equilibrium. When an input voltage is applied to all cells, the current is represented by

$$I_j = I_B + \Delta_j \tag{A12}$$

with the condition of

$$\sum_{j=1}^{N} \Delta_j = 0, \tag{A13}$$

where N is the number of the competing cell. If the largest input voltage is applied to cell-1 as the winning input, then the difference of the input voltages between this cell and cell-i is expresses as,

$$V_{in}^1 - V_{in}^i = \sqrt{\frac{2}{\beta}} \left(\sqrt{I_B + \Delta_1} - \sqrt{I_B + \Delta_i}\right) + R \sum_{j=1}^{i} (i\text{-}j)\, \Delta_j, \tag{A14}$$

where $\beta = \mu\, Cox\, (W/L)$ and R is the unit resistor value of the common signal line between two adjacent cells. The first term in (23) is the voltage difference

for the different current assuming the perfect match of two cells. The second term in (23) is the voltage drop along the common line from cell-1 to cell-i, which is zero for the ideal case. Thus, for the proper WTA operation, the magnitude of the first term must be larger than that of the second term. In Figure 9, the above two terms are shown for the two ends of cell-1 and cell-N given the following conditions:

$$I_1 = I_B + \left(\frac{N-1}{2}\right)\Delta, \ldots \quad \text{(A15)}$$

$$I_j = I_B + \left(\frac{N-1}{2} - (i-1)\right)\Delta, \ldots \quad \text{(A16)}$$

$$I_{\frac{N+1}{2}} = I_B, \ldots \quad \text{(A17)}$$

and

$$I_N = I_B - \left(\frac{N-1}{2}\right)\Delta, \quad \text{(A18)}$$

where Δ is 80 nA. From the process parameter, the sheet resistance is known to be 0.026 Ω per square and there are 14 squares in the common signal line of each cell from the layout. Figure 9 shows that about 600 cells can be connected in series and Figure 10 shows the schematic block diagram to increase the number of the competing cells.

References

[4.1] A. K. Jain, "Image data compression: a review," *Proc. of IEEE*, vol. 69, no. 3, pp. 349-389, Mar. 1981.

[4.2] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 379-423 and 623-656, 1948.

[4.3] R. M. Gray, *Source Coding Theory*, Kluwer Academic Publishers: Boston, MA, 1990.

[4.4] A. Gersho, "On the structure of vector quantizers," *IEEE Trans. on Inform. Theory*, vol. 28, no. 2, pp. 157-162, Mar. 1982.

[4.5] R. M. Gray, "Vector quantization," *IEEE Acoustics, Speech, and Signal Processing Magazine*, pp. 4-29, Apr. 1984.

[4.6] Y. Linde, A. Buzo, R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Comm.*, vol. COM-28, pp. 84-95, Jan. 1980.

[4.7] S. Grossberg, "Competitive learning: from interactive activation to adaptive resonance," *Cognitive Sci.*, vol. 11, pp. 23-63, 1987.

[4.8] S. Grossberg, "Adaptive pattern classification and universal recording: I. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, vol. 23, pp. 121-134, 1976.

[4.9] S. Grossberg, "Adaptive pattern classification and universal recording: II. Feedback, expectation, olfaction, illusions," *Biological Cybernetics*, vol. 23, pp. 187-202, 1976.

[4.10] T. Kohonen, *Self-Organization and Associative Memory*, 2nd Ed., Spring-Verlag: New York, NY, 1988.

[4.11] T. Kohonen, "An introduction to neural computing," *Neural Networks*, International Neural Network Society, vol. 1, pp. 3-16, 1988.

[4.12] N. M. Nasrabadi, Y. Feng, "Vector quantization of images based upon the Kohonen self-organizing feature maps," *Proc. of International Joint Conference on Neural Networks*, vol. I, pp. 101-108, San Diego, CA, June 1988.

[4.13] J. Naylor, K. P. Li, "Analysis of a neural network algorithm for vector quantization for speech parameters," *Proc. of 1st Ann. INNS Meet.*, pp. 310-314, Boston, MA, 1988.

[4.14] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, D. Melton, "Competitive learning algorithm for vector quantization," *Neural Networks*, International Neural Network Society, vol. 3, pp. 277-290, 1990.

[4.15] W.-C. Fang, B. J. Sheu, O. T.-C. Chen, "A neural network based VLSI vector quantizer for real-time image compression," J. A. Storer, J. H. Reif (Eds.), *Proc. of Data Compression Conference*, IEEE Computer Society Press: Los Alamitos, CA, 1991.

[4.16] R. Hecht-Nielsen, "Application of counterpropagation networks," *Neural Networks*, International Neural Network Society, vol. 1 , pp. 131-141, 1988.

[4.17] B. Kosko, "Stochastic competitive learning," *Proc. of IEEE International Joint Conference on Neural Networks*, vol. II, pp. 215-226, San Diego, CA, June 1990.

[4.18] D. E. Rumelhart, D. Zipser, "Feature discovery by competitive learning," *Cognitive Sci.*, vol. 9, pp. 75-112, 1985.

[4.19] D. E. Rumelhart, J. L. McClelland, *Parallel Distributed Processing*, MIT Press: Cambridge, MA, 1986.

[4.20] D. DeSieno, "Adding a conscience to competitive learning," *Proc. of IEEE International Joint Conference on Neural Networks*, vol. I, pp. 117-124, San Diego, CA, June 1988.

[4.21] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, Addison Wesley: Reading, MA, 1985.

[4.22] C. A. Mead, *Analog VLSI and Neural Systems*, Addison Wesley: New York, NY, 1989.

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

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

[4.25] R. Gregorian, G. C. Temes, *Analog MOS Integrated Circuits for Signal Processing*, Wiley-Interscience: New York, NY, 1986.

[4.26] M. A. Mahowald, T. Delbruck, "Cooperative stereo matching using static and dynamic image features," *Analog VLSI Implementation of Neural Systems*, C. A. Mead, M. Ismail (Eds.), Kluwer Academic Publishers: Boston, MA, pp. 213-238, 1989.

[4.27] R. R. Lippmann, "An introduction to computing with neural nets," *IEEE Acoustics, Speech, and Signal Processing Magazine*, vol. 4, no. 2, pp. 4-22, Mar. 1987.

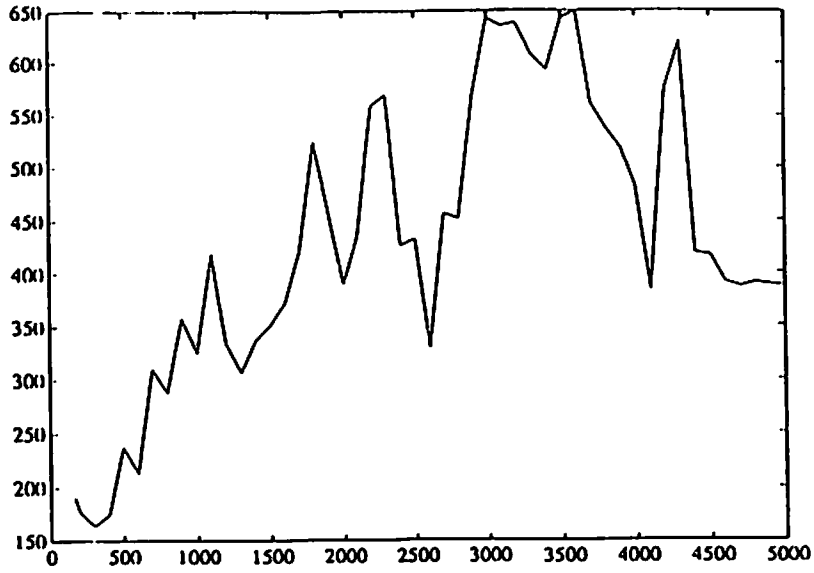
[4.28] P. R. Gray, R. G. Meyer, *Analysis and Design of Analog Integrated Circuits*, John Wiley & Son: New York, NY, 1984.

[4.29] K . R. Lakshmikumar, R. A. Hadaway, M. A. Copeland, " Characterization and modeling of mismatch in MOS transistors for precision analog design," *IEEE J. Solid-State Circuits*, vol. SC-21, no. 6, pp. 1057-1066, Dec. 1986.

[4.30] W. K. Pratt, *Digital Image Processing*, 2nd Edition, John Wiley & Sons: New York, NY, 1991.

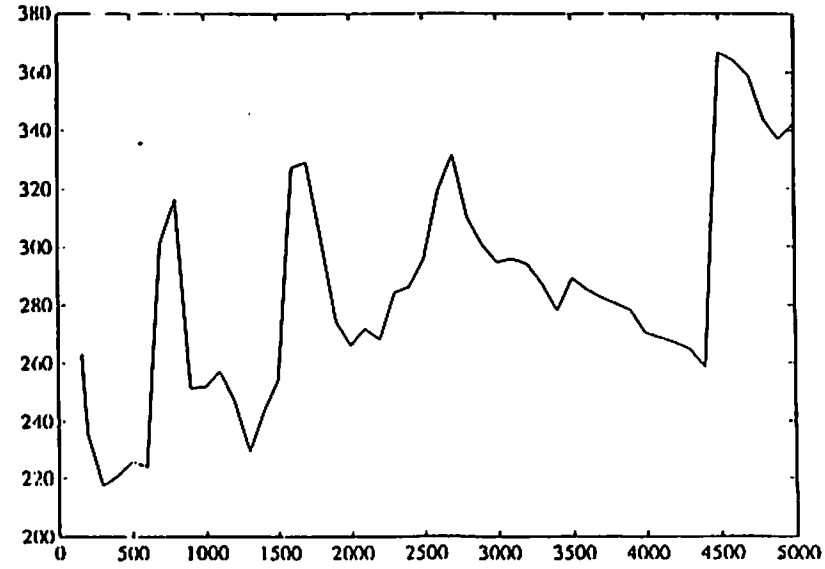
[4.31] G. Lewicki, "Foresight: A fast turn-around and low cost ASIC prototyping alternative," *Proc. of IEEE ASIC Seminar and Exhibit*, pp. 6- 8, Rochester, NY, Sept. 1990.

MSE



F_{th}

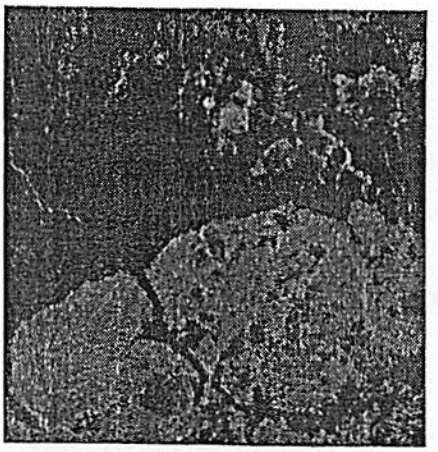
MSE



F_{th}

Fig. 4.1 Plots of mean-squared errors of image compression versus the upper-threshold frequency. The one-iteration FSO method is used for a 6-bit codebook on 5x5 subimage blocks. (a) SAR Ice image of Beaufort Sea, Alaska; 512 x 512 pixels. (b) Girl image; 512 x 512 pixels.

(a)



(b)

(c)

Fig. 4.2 Image compression using the FSO method on 5x5 subimage blocks.
(a) Original SAR Ice image of Fig. 4.1(a).
(b) Reconstructed image using 10-bit one-iteration FSO codebook; MSE=86.31.
(c) Reconstructed image using 10-bit two-iteration FSO codebook; MSE=82.35.

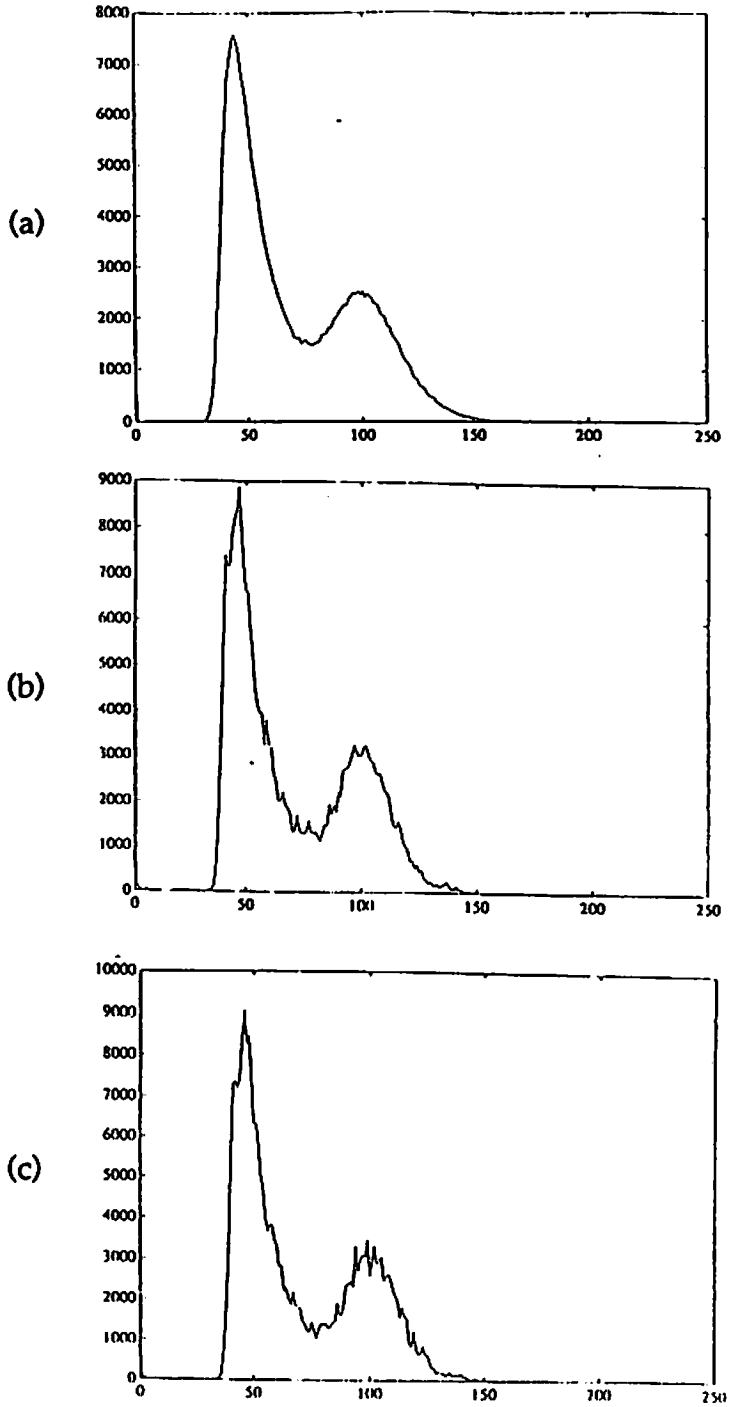


Fig. 4.3 Histogram of the 512 x 512-pixel SAR Ice image.
(a) Original image.
(b) Reproduced image using one-iteration 10-bit FSO method.
(c) Reproduced image using two-iteration 10-bit FSO method.

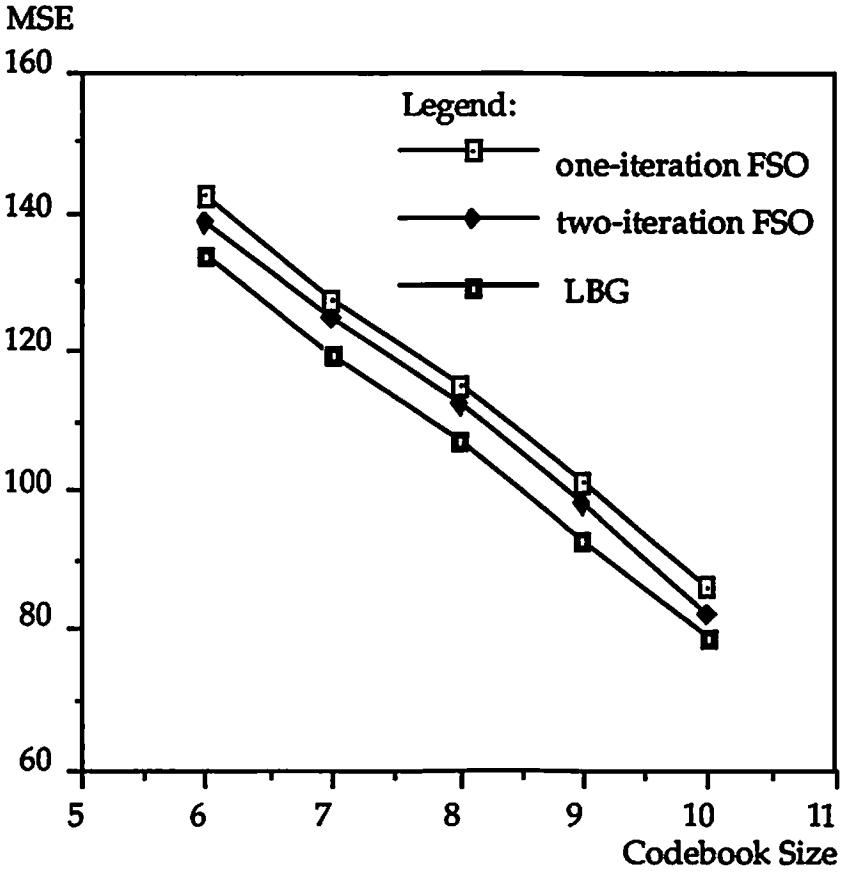


Fig. 4.4 Mean-squared errors of image compression using the FSO method and the LBG method on 5x5 subimage blocks of the 512 x 512-pixel SAR Ice image.

Table 4.1 Performance comparison of three VQ methods

Algorithms / Codebook Size	one-iteration FSO			two-iteration FSO			LBG	
	MSE	SDR	*Fth*	MSE	SDR	*Fth*	MSE	SDR
10-bit	86.31	18.16	20	82.35	18.37	20	78.64	18.61
9-bit	101.36	17.47	40	97.89	17.62	40	92.84	17.89
8-bit	115.04	16.92	80	112.08	17.03	80	106.95	17.28
7-bit	127.51	16.47	160	124.41	16.58	160	119.80	16.78
6-bit	142.27	16.00	320	138.56	16.11	320	133.78	16.30
Note: MSE: Mean-Squared Error; SDR: Signal-to-Distortion Ratio; *Fth*: Upper-Threshold Frequency.								

(a)

(b)

Fig. 4.5 Adaptive image compression using the FSO method.
(a) Girl image reconstructed by using a 10-bit FSO codebook from SAR image; MSE = 1062.59.
(b) Girl image reconstructed by using a 10-bit FSO codebook with adaptive training; MSE = 59.84.

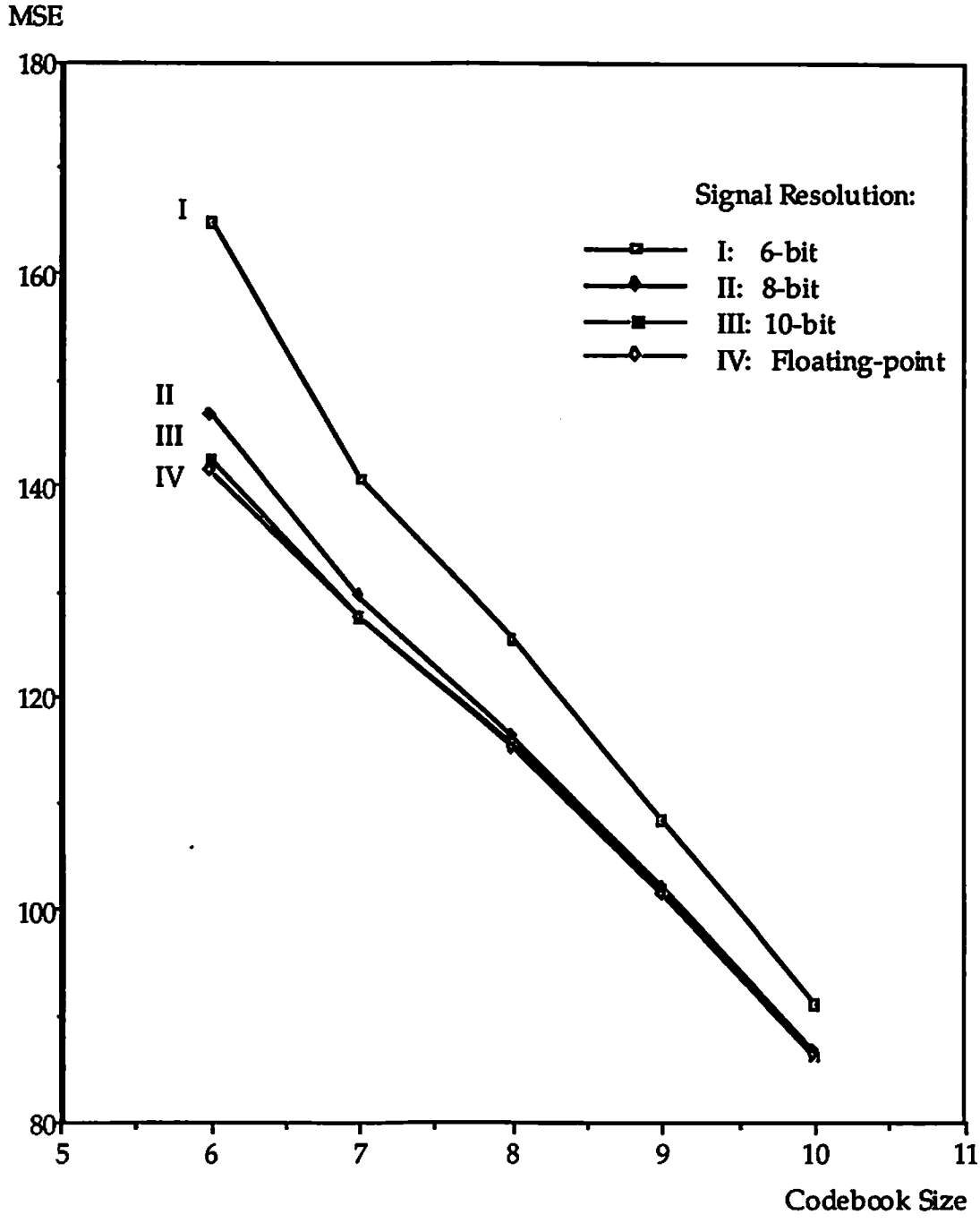


Fig. 4.6 Mean-squared errors of image compression using the one-iteration FSO method for 6-bit, 8-bit, 10-bit, and floating-point resolutions on 5x5-pixel subimage blocks of the 512x512-pixel SAR Ice image.

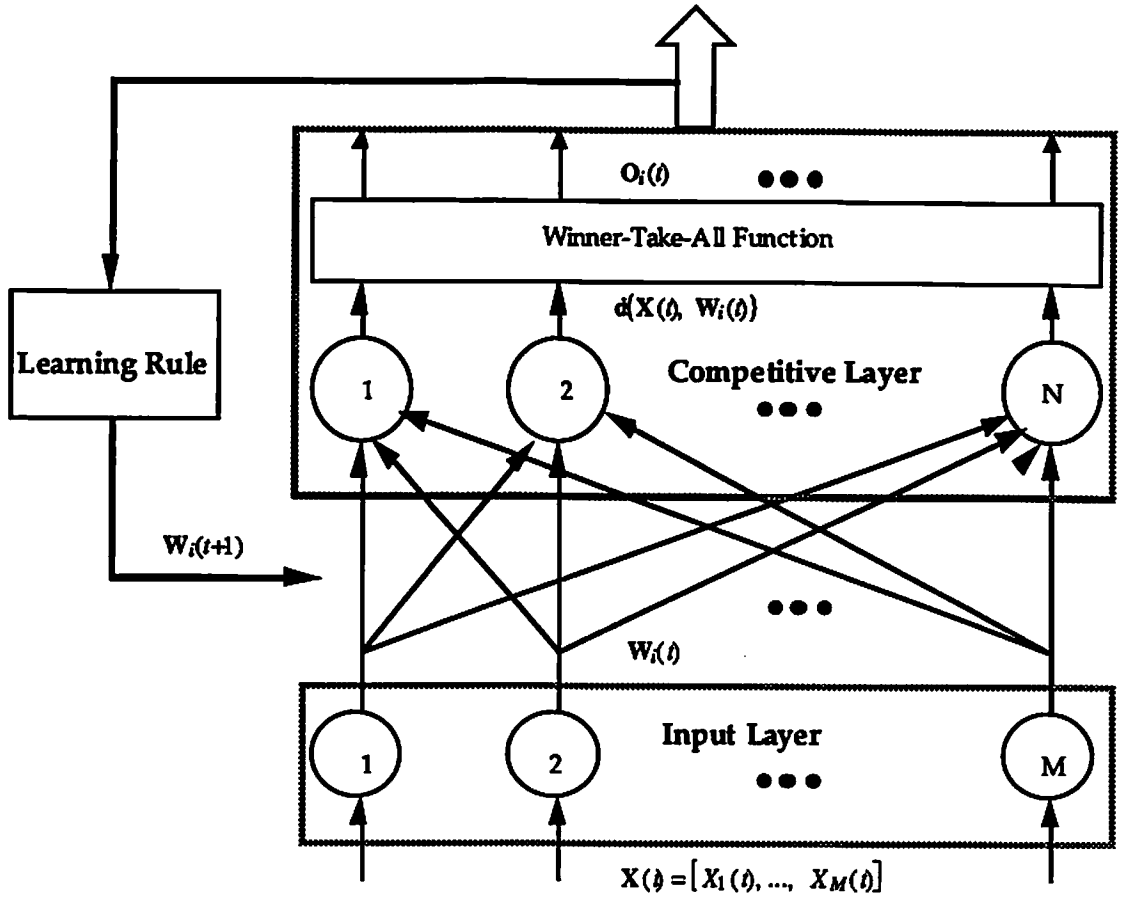


Fig. 4.7 The architecture of the FSO neural network.

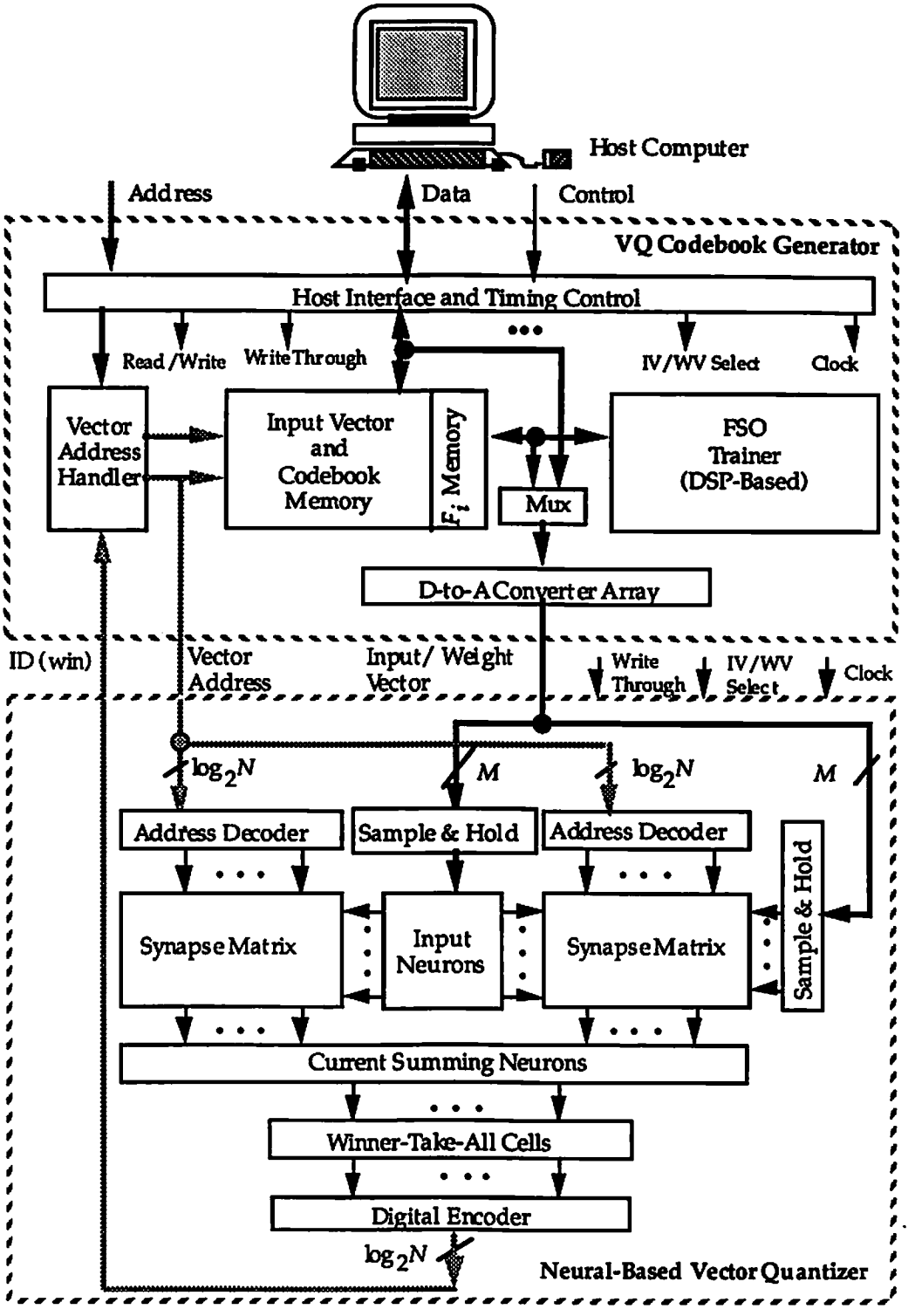
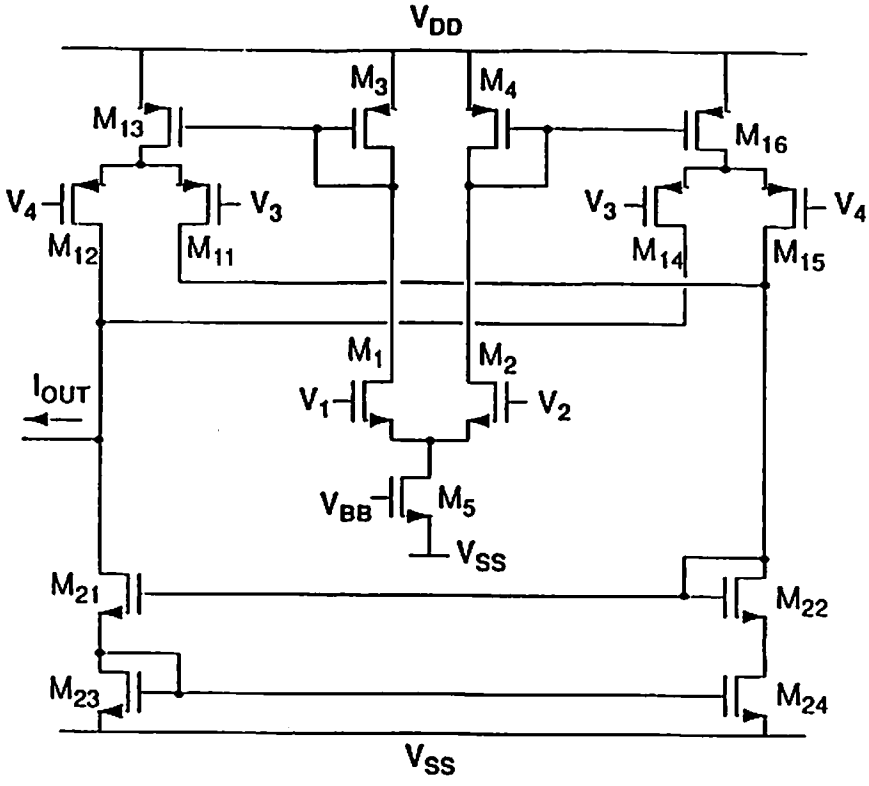


Fig. 4. 8 The VLSI neural processor based on the FSO method.



Transistor	$M_{1,2}$	$M_{3,4}$	M_5	$M_{11,12,13,15}$	$M_{13,16}$	$M_{21,22}$	$M_{23,24}$
Size (μm/μm)	4/32	10/3	12/4	4/22	21/3	4/8	4/12

Fig. 4.9 Circuit schematic and transistor sizes for the programmable synapse based on Gilbert multiplier.

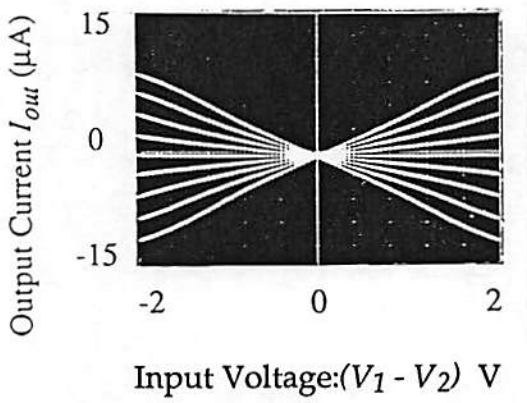


Fig. 4.10 Measured synapse output current versus neuron input voltage (V_1-V_2) with different weight voltage values(V_3-V_4) : -2.00, -1.00, -0.50, -0.25, 0.00, 0.25, 0.50, 1.00, and 2.00 V, from bottom to top.

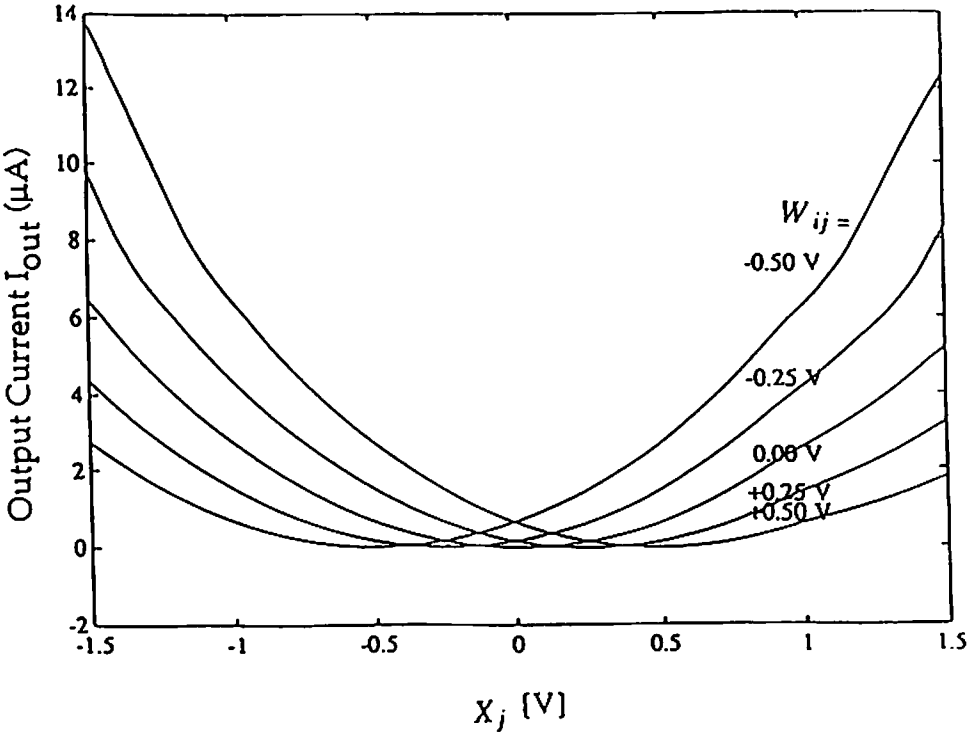
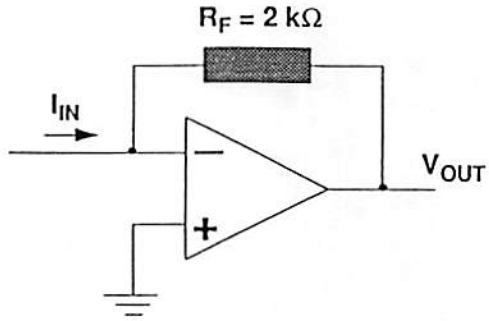
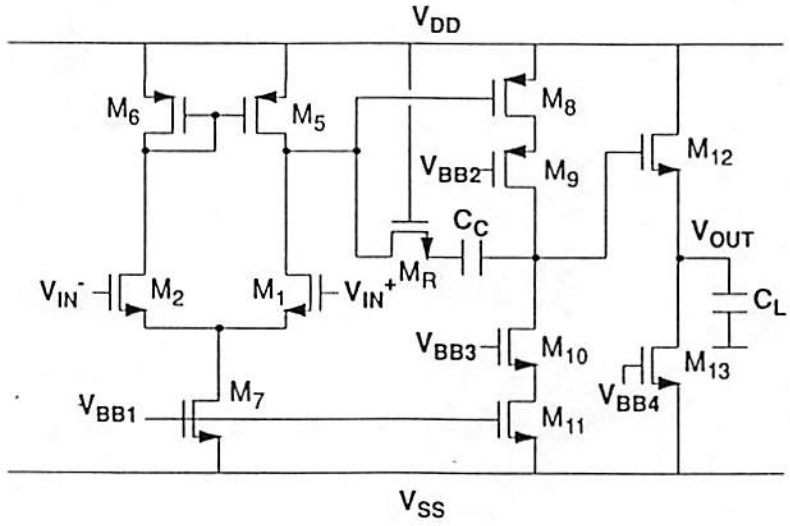


Fig. 4.11 Simulated characteristics of the synapse cell for $(X_j-W_{ij})^2$ computing. Here X_j is applied to V_1, V_3 and W_{ij} is applied to V_2, V_4 for the (i,j)th synapse cell.



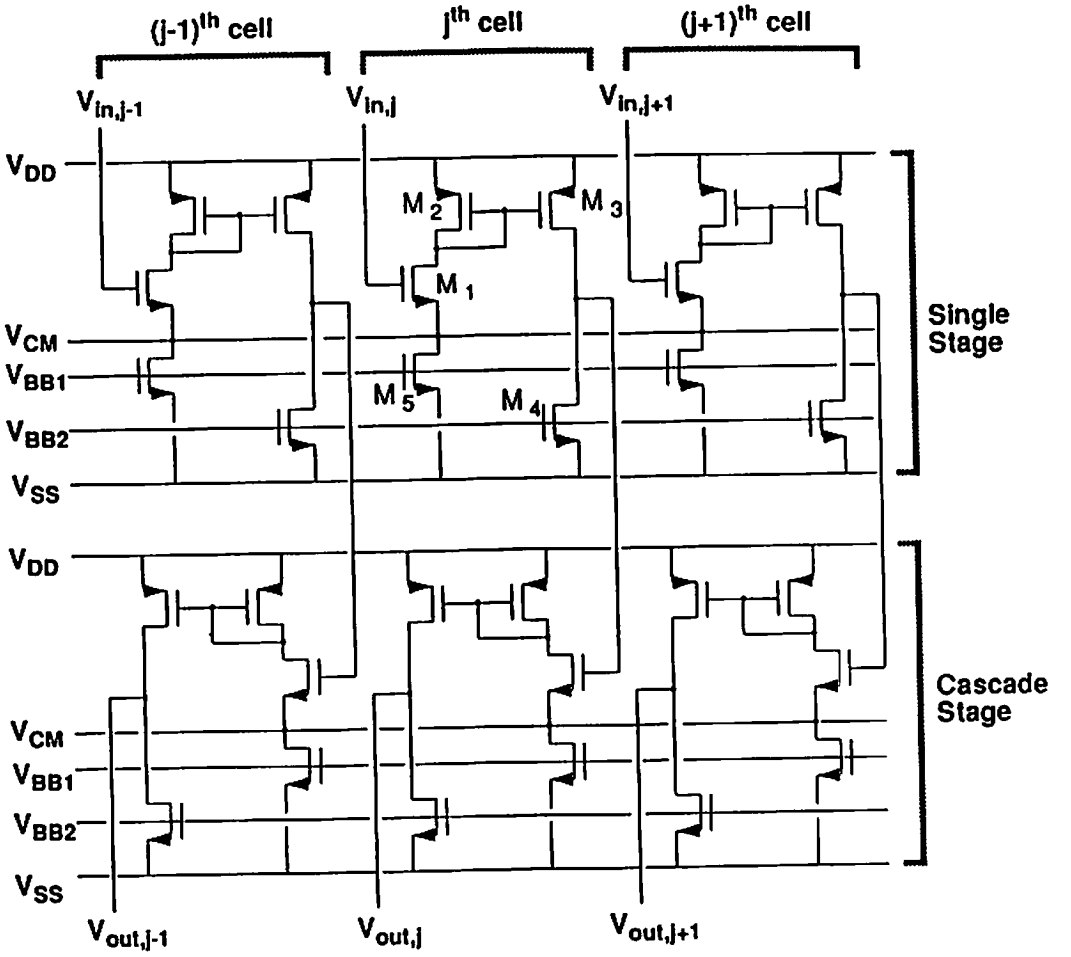
(a)



Transistor	M_1	M_2	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_R	C_C
Size (μm/μm)	16/2	16/2	20/4	20/4	40/4	46/4	90/2	26/2	52/4	440/2	150/4	50/2	1.0pF

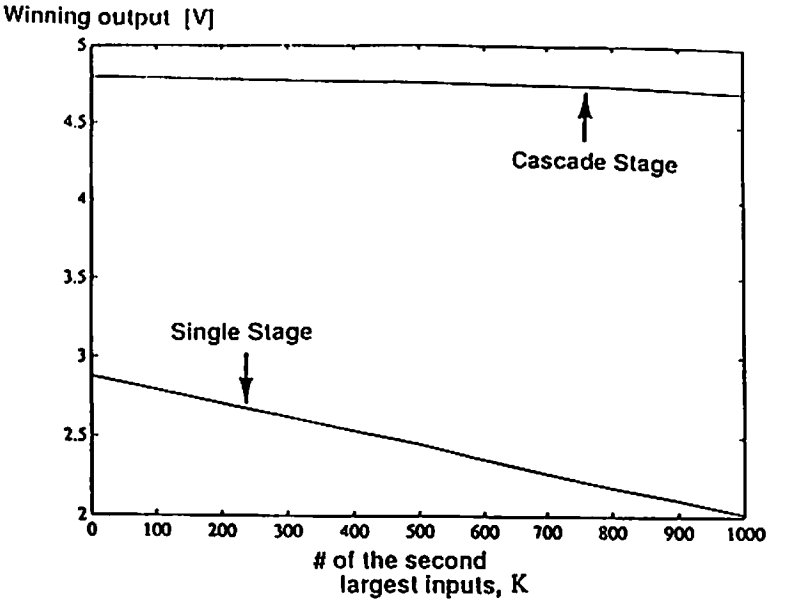
(b)

Fig. 4.12 The output neuron.
(a) Functional diagram.
(b) Detailed circuit schematic.

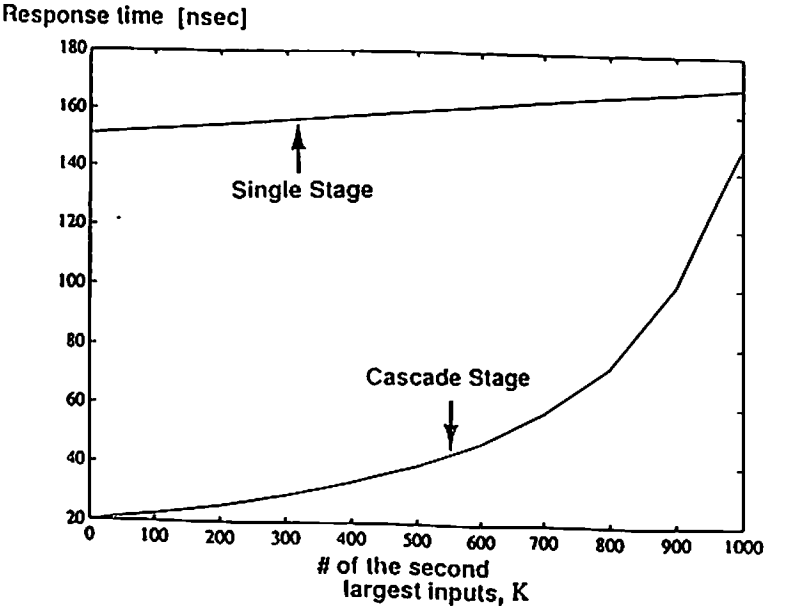


Transistors	M_1	M_2	M_3	M_4	M_5
Size (μm / μm)	8 / 2	8 / 4	16 / 4	8 / 4	16 / 4

Fig. 4.13 Detailed circuit schematic of the winner-take-all cells.

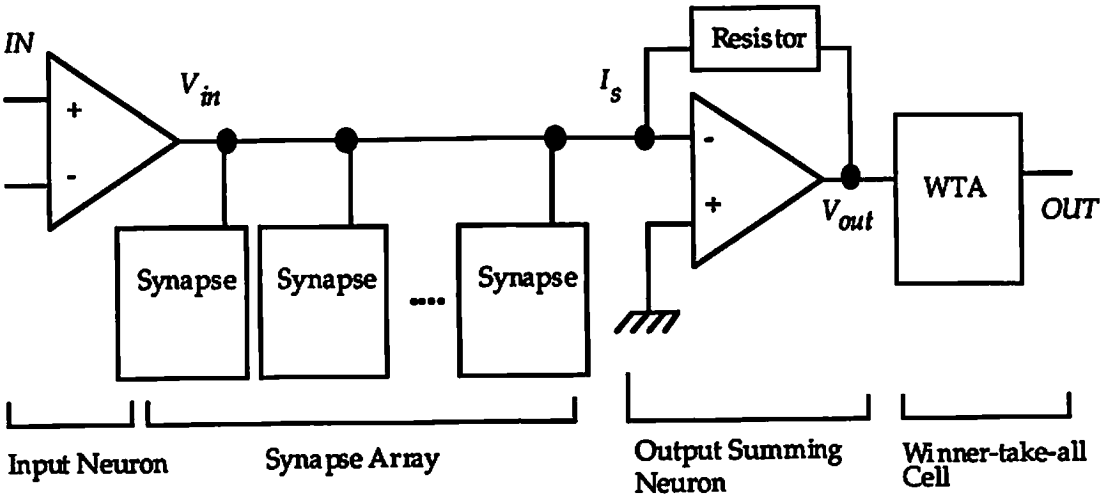


(a)

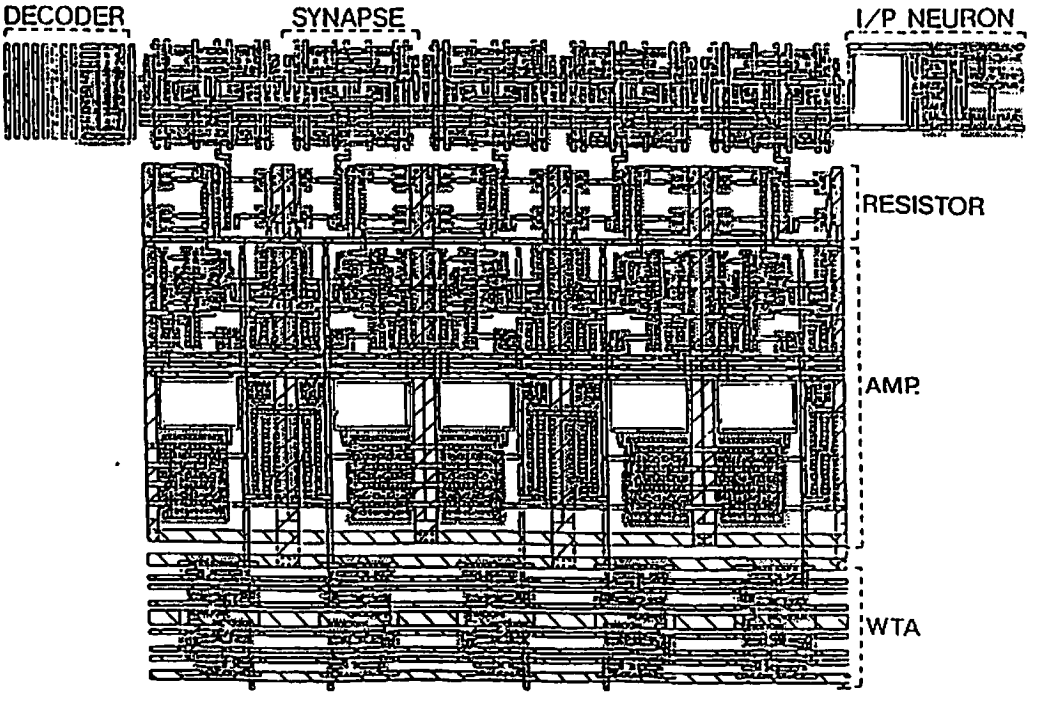


(b)

Fig. 4.14 Calculation results on a 1000-input WTA with different numbers of cells having the second largest input voltage value.
(a) DC level of the winning output.
(b) Response time of the winning output.



(a)



(b)

Fig. 4.15 The FSO network slice.
(a) Functional block diagram.
(b) Physical Layout.

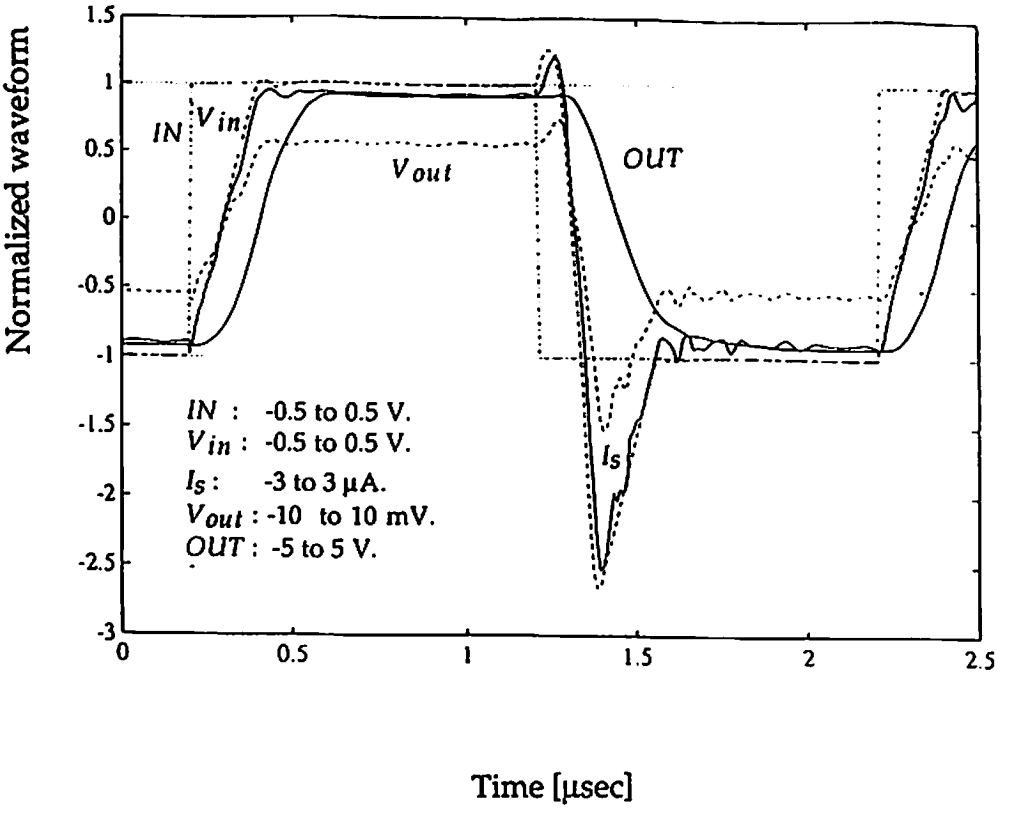


Fig. 4.16 The characteristics of the FSO network slice.
IN = signal to the input neuron.
V_{in} = the buffered input fed to the synapse cell.
I_S = output current of the synapse cell.
V_{out} = output voltage of the output neuron.
OUT = output voltage of the WTA cell.

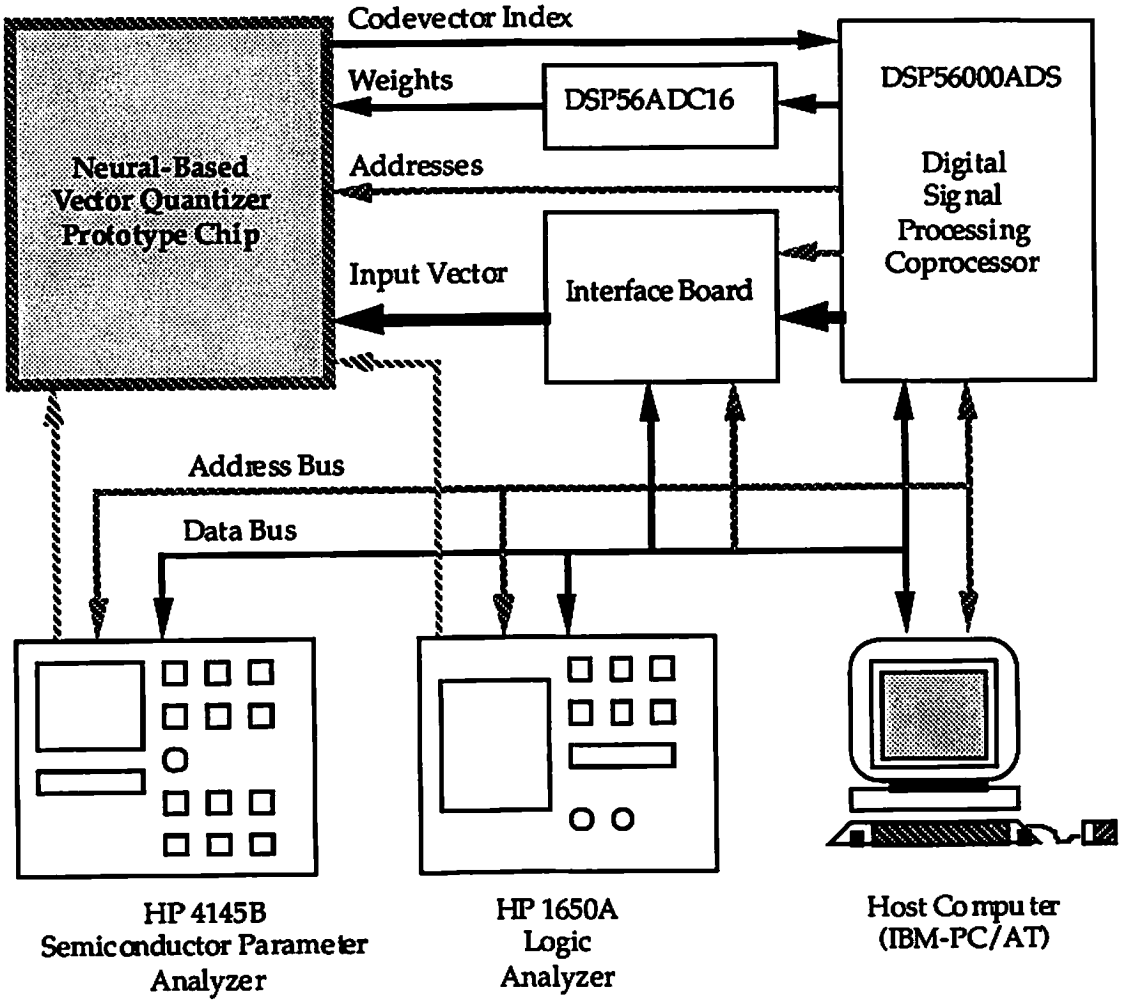
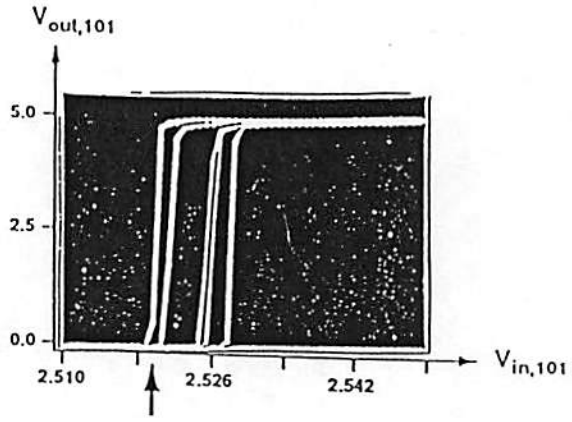
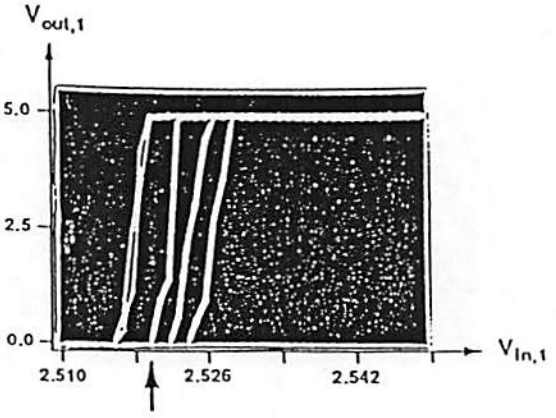


Fig. 4.17 Schematic of a dedicated testbed for the neural-based vector quantizer chip.



(a)



(b)

Fig. 4.18 Measurement results of a 200-input WTA test structure.
(a) Output voltages of the winner with different numbers of cells having the second largest input: 2, 50, 100, 150, and 199, from left to right.
(b) Output voltages of the winner with different positions of the cell having the second largest input: 2, 50, 100, 150, and 200, from left to right.

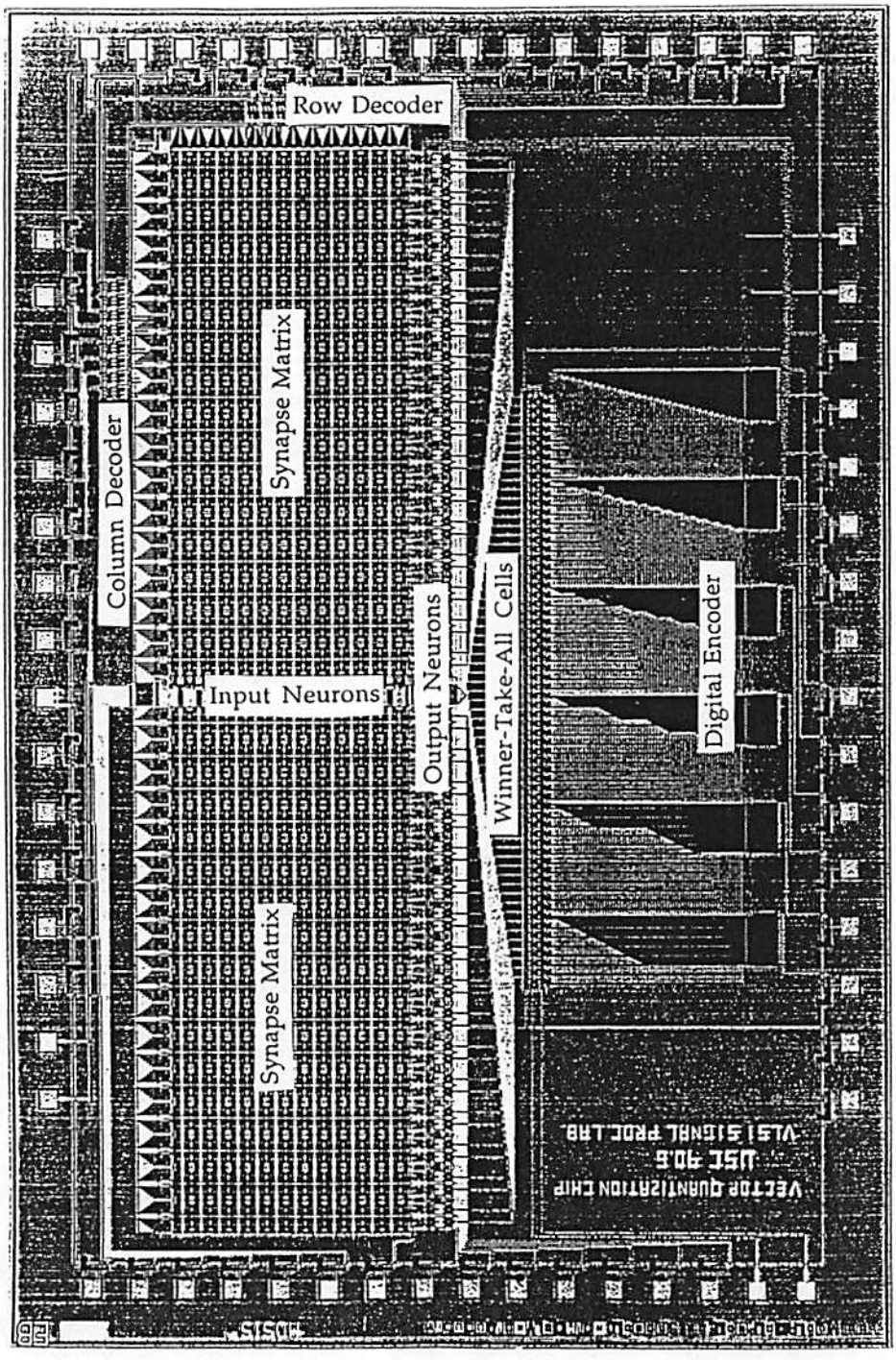


Fig. 4.19 A die photo of the neural-based vector quantizer prototype chip.

(a) (b)

Fig. 4.20 Functional testing results for the neural-based vector quantizer chip.
(a) Original image.
(b) Reconstructed image.

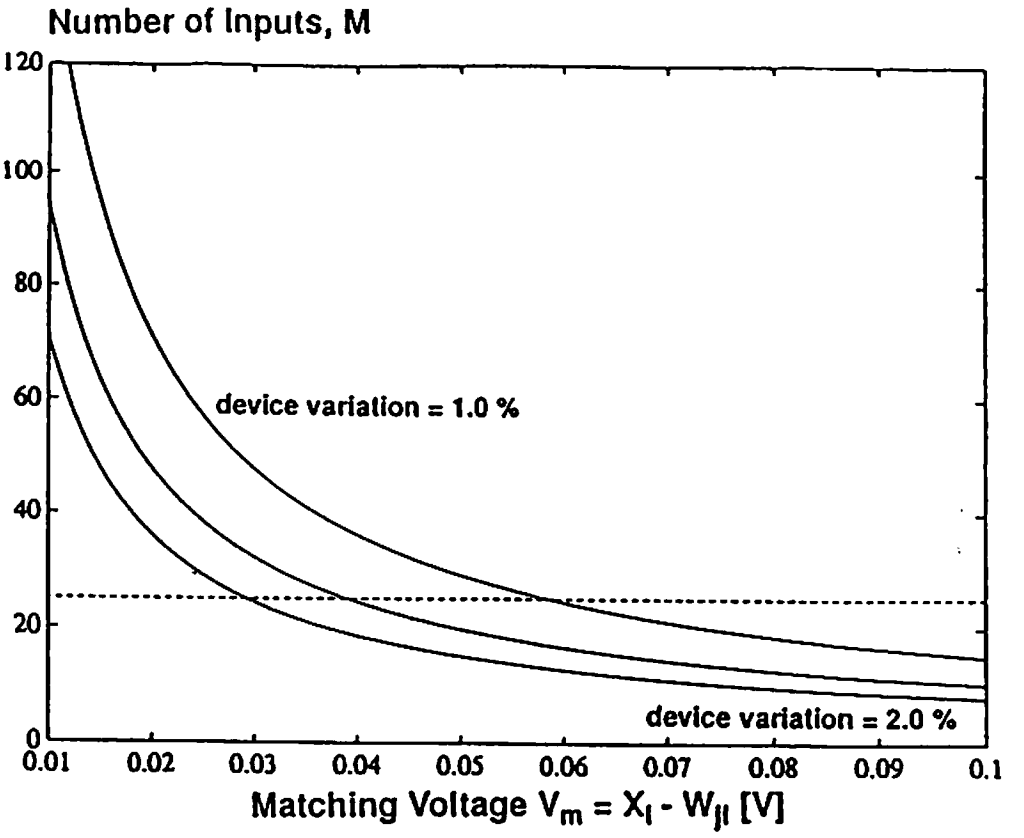


Fig. 4.A.1 The number of the possible dimensionality versus the matching voltage with different device variations.

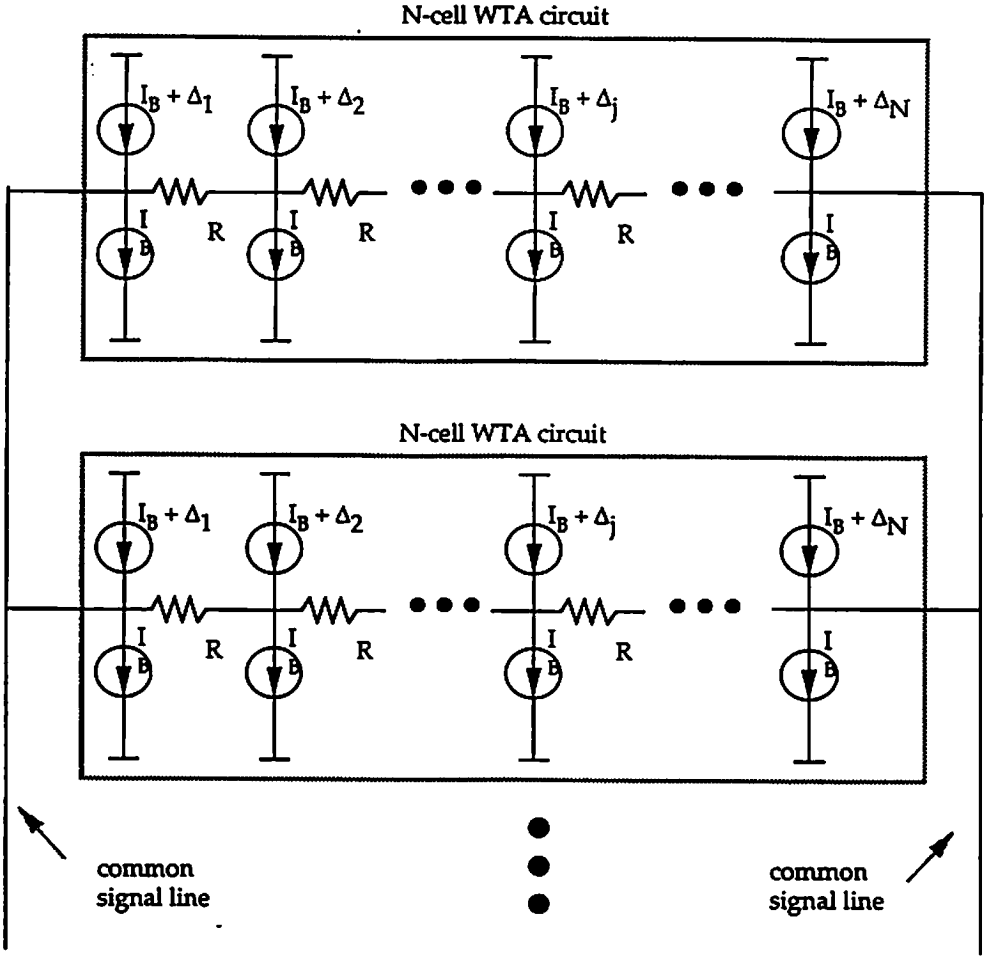


Fig. 4A.2 Simple model of the winner-take-all circuit.

Chapter 5

VLSI Neural Network Processor for Optical-Flow Based Motion Compression

In this chapter, we describe a neural network based motion compression algorithm and its associated VLSI neuroprocessor design. The neuroprocessor design is based on a locally connected multiple competitive neural network developed for high performance optical flow computing systems. The proposed VLSI neural processor design can achieve a high-speed wide-range motion estimation and thus an efficient image sequence compression by taking advantage of the massively parallel neural computing architecture and VLSI technology. An extendible VLSI *neuroprocessor* has been designed with a silicon area of 2,482 x 5,636 λ^2 in MOSIS scalable CMOS process. The mixed analog-digital design techniques are utilized to achieve compact and programmable synapses with gain-adjustable neurons and winner-take-all cells for massively parallel neural computation. *Hardware annealing* through the control of neurons' gain helps to efficiently search the optimal solutions. Measured results of the programmable synapse, summing neuron, and winner-take-all circuitry are presented. A 1.25 x 1.17 cm^2 chip in a submicron CMOS technology can accommodate 128 velocity-selective neuroprocessors and achieve 166.4 Giga connections per second. Computing of optical flow using one neural chip can be accelerated by a factor of 379 than a Sun-4/260 workstation. Real-time motion estimation on industrial

video images is practical using an extended array of VLSI neuroprocessors. Actual examples on moving vehicles are presented.

5.1 Introduction

5.1.1 Motion Estimation and Video Compression

The goal of video compression is to eliminate the temporal and spatial redundancy of video image sequences and thus reduce the bandwidth and storage required for the transmission and recording of the video signal. Fig. 5.1 shows a generic motion-compensated predictive video compression system. Delta modulation performed pointwise between successive frames can reduce temporal redundancy over regions where there is no motion. However, the motion-compensated predictor makes the delta modulation more generally effective since the motion of regions in the scene is determined so that image points between which the modulation differences are derived can be more accurately chosen.

Motion estimation has been studied extensively by several researchers in the context of video compression [5.1] - [5.11]. Much of the work in video compression is closely related to work in optical flow determination. The pel-recursive motion estimation algorithm for video compression developed by Netravali and Robbins [5.7] is similar to the optical flow estimation derived by Horn and Schunck [5.3], except for two important differences: (1) Honr and Schunck use smoothed velocity estimates and (2) the pel-recursive equations use a fixed parameter rather than the factor varies with the magnitude of the image gradient. The algorithm of

Netravali and Robbins has problems on the converge and performance accuracy [5.8].

The apparent difference between the work on image compression and the work on optical flow estimation is due to the different motivation for the investigation. Although video compression does not require neither high-accurate estimation of the motion nor the sharp detection of motion boundaries, a better motion estimation algorithm could reduce the video bandwidth further. But any such improvement must be justified against the complexity of the hardware required to implement the video compression technique.

5.1.2 Optical Flow Estimation

Optical flow is the apparent motion of the brightness patterns in an image. Generally, the optical flow corresponds to the motion field [5.3]. Optical flow provides information about the spatial arrangement of the objects, the rate of change of these arrangements in a given scene, and also the perceiver's own movements. The important parameters about the three dimensional structure and motion of the scene can be obtained by analyzing the optical flow field [5.12]. For video compression application, optical flow provides motion vectors to choose the difference-modulated image points that correspond to the displaced versions of the same patch of image frame.

Optical flow is estimated from the time varying imagery and commonly represented by a dense *motion field* that assigns a two-dimensional velocity vector to every point on the visual field. According to the nature of the measured primitives, existing approaches to optical flow computing can

be divided into two type: the image intensity based approach and the token based approach.

intensity Based Approach The intensity based approach relies on the assumption that changes in intensity are strictly due to the motion of the object. The image intensity values and their spatial and temporal derivatives are used to compute optical flow. By expanding the intensity function into a first-order Taylor series, Horn and Schunck [5.3] derived an optical flow equation using the brightness constancy assumption and spatial smoothness constraints. An iterative algorithm for solving the resulting equation was also developed.

Token Base Approach 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.

5.1.3 Neural Network Applications to Optical Flow Computing

The optical flow computing is very computation intensity and difficult to perform for high-speed applications on general-purpose computers. This problem can be alleviate by using a novel VLSI neuroprocessor design for high-speed optical-flow estimation.

Recently, several researchers used neural networks to conduct optical flow computing [5.13-5.15]. To prevent the smoothness constraint

from taking effect across strong velocity gradients, a line process has been incorporated into the optical flow equation. The resulting equation is non-convex 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 obtained a near-optimum solution. Convergence of such a network was obtained within a few iteration cycles. An analog and binary resistive neural networks was used in [5.13] for computing the optical flow using Horn's method [5.3]. Basically, this 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. No attempt was made to detect large displacements.

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 derivatives 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 [5.14] 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 on rotation invariant measured primitives extracted from successive image frames. A deterministic decision rule was used for the

updating of neurons states. The deterministic annealing process achieves convergence within a few iteration cycles. This neural network can detect large displacements.

This chapter presents a *locally connected multiple competitive neural network* and its associated *VLSI array neuroprocessors* for high performance optical flow computing using Zhou's method [5.14]. The proposed optical flow neuroprocessor is applicable for high-speed motion compression due to its massively parallel processing architecture, VLSI implementation, and good estimation performance. An extendible VLSI *neuroprocessor* has been designed with a silicon area of 2,482 x 5,636 λ^2 in MOSIS scalable CMOS process. The mixed analog-digital design techniques are utilized to achieve compact and programmable synapses with gain-adjustable neurons and winner-take-all cells for massively parallel neural computation. Measured results of the programmable synapse, summing neuron, and winner-take-all circuitry are presented. A 1.25 x 1.17 cm^2 chip in a submicron CMOS technology can accommodate 128 velocity-selective neuroprocessors and achieve 166.4 Giga connections per second. Computing of optical flow using one neural chip can be accelerated by a factor of 379 than a Sun-4/260 workstation. Real-time motion estimation on industrial video images is practical using an extended array of VLSI neuroprocessors. Actual examples on moving vehicles are presented.

5.2 Optical Flow Computing Using Neural Networks

A neural network for optical flow computing based on work by Zhou and Chellappa [5.14] is defined below. Let $k_{11}(i,j)$ and $k_{12}(i\oplus k, j\oplus l)$ are the principal curvatures of the first image, $k_{21}(i, j)$ and $k_{22}(i\oplus k, j\oplus l)$ are the principal curvature of the second image, $g_1(i,j)$ and $g_2(i\oplus k, j\oplus l)$ are the intensity values of the first and second images, respectively, while vi,j,k,l is the output state of pixel (i,j) with velocity (k,l), *where* $1 \le i \le Nr$, $1 \le j \le Nc$, $-Dx \le k \le Dx$, $-Dy \le l \le Dy$. The symbol $\oplus$ denotes that

$$f_{a\oplus b} = \begin{cases} f_{a+b} & \text{if } 0 \le a+b \le N_c, N_r \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

The principal curvatures are defined as

$$k_1(i,j) = M + \left(M^2 - G\right)^{\frac{1}{2}} \tag{5.2}$$

and

$$k_2(i,j) = M - \left(M^2 - G\right)^{\frac{1}{2}} \tag{5.3}$$

where G and M are the Gausian and mean curvatures given by

$$G = \frac{\partial^2 g(i,j)}{\partial i^2} \cdot \frac{\partial^2 g(i,j)}{\partial j^2} - \left(\frac{\partial^2 g(i,j)}{\partial i \partial j}\right)^2, \tag{5.4}$$

and

$$M = \frac{1}{2}\left[\frac{\partial^2 g(i,j)}{\partial i^2} + \frac{\partial^2 g(i,j)}{\partial j^2}\right]. \tag{5.5}$$

A polynomial fitting technique is used to estimate the derivatives. A smoothness constraint is used for obtaining a smooth optical flow field and a line process is employed for detecting motion discontinuities. S_0 is an index set for all pixels in a $2Wr+1$ x $2Wc+1$ smoothing window centered at point (i,j). The line process consists of vertical and horizontal line, 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 corresponding error function for computing the optical flow from a pair of image frames can be expressed as

$$
\begin{aligned}
E = & \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} \sum_{k=-D_x}^{D_x} \sum_{l=-D_y}^{D_y} \qquad (5.6) \\
& \Big[\{ A \big[k_{11}(i,j) - k_{21}(i \oplus k, j \oplus l) \big]^2 + A \big[k_{12}(i,j) - k_{22}(i \oplus k, j \oplus l) \big]^2 + \\
& + \big[g_1(i,j) - g_2(i \oplus k, j \oplus l) \big]^2 \} v_{i,j,k,l} \\
& + \{ \frac{B}{2} \sum_{s \in S} (v_{i,j,k,l} - v_{(i,j) \oplus s,k,l})^2 \\
& + \{ \frac{C}{2} \Big[(v_{i,j,k,l} - v_{i \oplus 1,j,k,l})^2 (1 - L^h_{i,j,k,l}) + (v_{i,j,k,l} - v_{i,j \oplus 1,k,l})^2 (1 - L^v_{i,j,k,l}) \Big] \\
& + D [L^h_{i,j,k,l} + L^v_{i,j,k,l}] \} \Big]
\end{aligned}
$$

where A, B, C, and D are empirical constants, and $S = S_0 - (0,0)$ is an index set excluding (0,0). The first term of Eq. (5.6) 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 is the smoothness constraints on the solution. The third term is a line process to weaken the smoothness constraint and to detect motion discontinuities. Parameters A, B, C, and D are used for weighting the relative importance of each term. 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 $B/2$. If all lines are off, the weights at the four nearest neighbors of the center point are increased by $C/2$. These parameter play an important role in the algorithm behavior.

Let $T_{i,j,k,l;m,n,k,l}$ be the synaptic interconnection weight from neuron (m,n,k,l) to neuron (i,j,k,l), $I_{i,j,k,l}$ be the bias input, $V = \{ v_{i,j,k,l},\ 1 \le i \le N_r,\ 1 \le j \le N_c,\ -Dx \le k \le D_x,\ -Dy \le l \le D_y$ be a binary neuron state of neural network with $v_{i,j,k,l}$ denoting the state of the (i,j,k,l)-th neuron, N_r and N_c be the maximum dimensions of neuron layer in i and j axes, respectively, and Dx and Dy be the maximum values of velocity components in k and l directions, respectively. The interconnection weights and bias inputs are defined as

$$\begin{aligned} T_{i,j,k,l;\,m,n,k,l} &= 2B \sum_{s \in S} \delta_{(i,j),\,(m,n)\oplus s} \\ &+ C\,[(1-L^{h}_{i,j,k,l})\delta_{i,m}\delta_{j\oplus 1,n} + (1-L^{h}_{i,j\oplus(-1),k,l})\delta_{i,m}\delta_{j\oplus(-1),n} \\ &\quad + (1-L^{v}_{i,j,k,l})\delta_{i\oplus 1,m}\delta_{j,n} + (1-L^{v}_{i\oplus(-1),j,k,l})\delta_{i\oplus(-1),m}\delta_{j,n} \\ &- [\,48B + C\,(\,4 - L^{h}_{i,j,k,l} - L^{h}_{i,j\oplus(-1),k,l} - L^{v}_{i,j,k,l} - L^{v}_{i\oplus(-1),j,k,l}\,)\,]\,\delta_{i,m}\delta_{j,n} \end{aligned} \tag{5.7}$$

and

$$\begin{aligned} I_{i,j,k,l} = -\{ & A\left[k_{11}(i,j) - k_{21}(i\oplus k,\, j\oplus l)\right]^2 + A\left[k_{12}(i,j) - k_{22}(i\oplus k,\, j\oplus l)\right]^2 + \\ & + \left[g_1(i,j) - g_2(i\oplus k,\, j\oplus l)\right]^2 \} \end{aligned} \tag{5.8}$$

where $\delta_{a,b}$ is the Dirac delta function. The interconnection weights $T_{i,j,k,l;m,n,k,l}$ consist of the *smoothness constraints* and the *line process* only. The bias input $I_{i,j,k,l}$ contain all information from the images. The term $D\,[L^{h}_{i,j,k,l} + L^{v}_{i,j,k,l}]$ is ignored since it does not contain neurons states. And thus the error function for computing the optical flow is transformed into the *energy function* of the neural network that is defined by

$$\begin{aligned} E = -\frac{1}{2} & \sum_{i=1}^{N_r} \sum_{j=1}^{N_c} \sum_{k=-D_x}^{D_x} \sum_{l=-D_y}^{D_y} \\ & \Big(\sum_{(m-i,\,n-j) \in S_0} T_{i,j,k,l\,;\,m,n,k,l}\, v_{m,n,k,l}\, v_{i,j,k,l} + I_{i,j,k,l}\, v_{i,j,k,l} \Big), \end{aligned} \tag{5.9}$$

A *locally connected multi-layer neural network* for optical flow computation is shown in Fig. 5.2. A set of $(2Dx + 1)(2Dy + 1)$ layers of neurons are used to represent the optical flow field. Each layer corresponds to a different velocity and contains $N_r \times N_c$ neurons if the images are of size $N_r \times N_c$. All neurons in the same layer are self-connected and locally interconnected with other neighboring neurons in a smoothing window of size $(2Wr + 1)(2Wc + 1)$. There are no interconnections between neurons in the different layers except those in the same hypercolumns. Every image pixel is represented by $(2Dx + 1)(2Dy + 1)$ mutually exclusive neurons which form a *hyperneuron* for velocity selection. When the (i, j, k, l)-th neuron at the point (i, j) in the (k, l)-th layer is active high, the velocity of pixel (i, j) is kB and lB in the k and l direction, respectively. Here, B is the sampling bin size of the velocity component range.

Optical flow computation is performed by the neuron evaluation using a massively parallel updating scheme which is based on the minimal mapping theory [5.17]. The initial states of the neurons are set as

$$v_{i,j,k,l} = \begin{cases} 1 & \text{if } I_{i,j,k,l} = max\,(I_{i,j,p,q}; -D_k \le p \le D_k, -D_l \le q \le D_l) \\ 0 & \text{otherwise.} \end{cases} \tag{5.10}$$

which 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. In the updating scheme, the minimal mapping theory is also used to handle the case of two neurons having the same largest inputs. Since the first and second terms in Eq. (5.6) do not contain the line process, the updating of the line process is prior to the updating of neuron

states. Let $L_{i,j,k,l}^{v;\,new}$ and $L_{i,j,k,l}^{v;\,old}$ denote the new and old states of the vertical line $L_{i,j,k,l}^{v}$, respectively. Let $\psi_{i,j,k,l}$ be the potential of vertical line $L_{i,j,k,l}^{v}$, given by

$$\psi_{i,j,k,l} = \frac{C}{2}(v_{i,j,k,l} - v_{i+1,j,k,l})^2 . \tag{5.11}$$

Then, the new state is determined by

$$L_{i,j,k,l}^{v;\,new} = \begin{cases} 1 & if\ \psi_{i,j,k,l} > 0 \\ 0 & otherwise. \end{cases} \tag{5.12}$$

Whenever the states of neurons $v_{i,j,k,l}$ and $v_{i+1,j,k,l}$ are different, the vertical line will be active in the case that C is greater than zero. If C is zero, then all lines are inactive, which mean that no line process exists in the network operation. A similar updating scheme is also used for the horizontal lines.

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

$$u_{i,j,k,l} = \sum_{(m-i,\, n-j) \in S_0} T_{i,j,k,l\,;\,m,n,k,l} v_{m,n,k,l} + I_{i,j,k,l} . \tag{5.13}$$

The $u_{i,j,k,l}$ is then processed by the maximum *evolution* function $O(.)$ to determine the velocity of the pixel,

$$v_{i,j,k,l} = O(u_{i,j,k,l}) = \begin{cases} 1 & if\ u_{i,j,k,l} = max\,(u_{i,j,p,q}\,;\, -D_x \le p \le D_x ,\, -D_y \le q \le D_y) \\ 0 & otherwise. \end{cases} \tag{5.14}$$

Notice that only the winning neuron is active high and the other neurons of the same hypercolumn are turned off. The network operation will be terminated whenever the energy function of the network reaches a minimum. Energy function minimization of the optical-flow neural network proceeds similar to Hopfield networks [5.18]. When the network reached a stable state, the optical flow field is determined by the neuron states.

The neural network algorithm for the optical flow computing can then be summarized as follows:

1. Set the initial state of the neurons.
2. Update the state of all lines synchronously.
3. Update the state of all neurons synchronously.
4. Check the energy function. If energy does not change anymore, stop; otherwise, go back to step 2.

In view of high parallelism and local conductivity, this network is well suited for VLSI implementation and thus real-time applications.

5.3 VLSI Optical-flow Neuroprocessors Design

A special purpose VLSI neuroprocessor has been designed to fully exploit algorithm parallelism of the neural-based optical flow computing and to enable real-time applications. A VLSI architecture for effective implementation of the multi-layer stochastic neural networks is obtained by projecting the 3-dimensional neural networks along the *velocity profile* direction into a 2-dimensional plane. As shown in Fig. 5.3, each small frame represents one *velocity-selective hyperneuron* which contains $(2D_x+1)(2D_y+1)$ *velocity-sensitive neurons* to specify $(2D_x+1)(2D_y+1)$ velocity selections. Each hyperneuron is interconnected with its $(2W_r+1)(2W_c+1)-1$ neighboring hyperneurons through a point-to-point interconnection which contains one output port and $[(2W_r+1)(2W_c+1)-1]$ input ports. The hyperneuron is designed as a neuroprocessor within which the velocity selectivity of an image pixel can be conducted. A functional diagram of the velocity-selective

hyperneuron is shown in Fig. 5.4. Its building blocks include a velocity-sensitive neuron array and a neighbor interconnection block.

5.3.1 Velocity-sensitive neurons array

The *velocity-sensitive neurons array* has *(2Dx+1)(2Dy+1)* velocity-sensitive neurons which are laterally connected through winner-take-all circuits. The velocity selectivity is generated by a competition mechanism which is implemented with the winner-take-all circuits. Only the velocity-sensitive neuron which corresponds to the maximum excitation will be the winner to represent the velocity for the given pixel. As shown in Fig. 5.5, the *velocity-sensitive neuron* is built with one synapses array, one summing neuron, and one winner-take-all cell. The *synapses array* contains $(2W_r + 1)(2W_c + 1)+1$ programmable synapses of s-bit accuracy. The synapse weights $T_{ijkl;mnkl}$ are stored in the synapse capacitors and dynamically refreshed by the digital synapse memory block. The binary outputs V_{mnkl} of its neighboring neurons are routed to the corresponding mask ports to perform the network evaluation. The *summing neuron* is functioning as a parallel current adder. Each summing neuron with its associated programmable synapses array perform a parallel *inner-product* computation as specified in Eq. (5.13). The *winner-take-all cell* contributes to a *maximum evolution* function on the analog outputs of summing neurons as specified in Eq. (5.14). The binary output of winner-take-all cell represents the velocity status.

5.3.2 Neighbors Interconnection

To solve the three dimensional interconnection problem for optical flow computing, the analog point-to-point interconnection for local communication and the digital common bus for global communication are used. Since each pixel is affected by its near neighbors , each hyperneuron has to receive/send information from/to the neighboring hyperneurons during the network operation. Data communication between these locally interconnected hyperneurons become one key factor on the overall system performance. There are three methods accomplish the local data communication with trade-offs between the operation speed and silicon area.

The first method is to use the digital *bit-serial* point-to-point interconnection. The $[(2Dx + 1)+ 2Dy + 1)]$-bit $vi_{jkl,mnpq}$'s are transmitted in a bit-serial order by using a time-multiplexing technique. The total number of the interconnection lines is reduced by a factor of $[(2D_x + 1)\ (2D_y + 1)]$. However, the time required for data transfer increases with the same factor. Besides, the required hardware overhead includes multiplexing control signals, the associated decoding circuit, and one-bit latch for each synapse.

The second method is to use *bit-parallel* point-to-point interconnection scheme. The $v_{ijkl,mnpq}$'s are transmitted by using the word-wide point-to-point interconnections. A high data rate is achieved. However, the total number of interconnection lines for each hyperneuron is as large as $[(2D_x+1)(2D_y+1)][(2W_r+1)+(2W_c+1)]$. The silicon area for the interconnection routing is large. The required large pin count become a major constraint for hardware implementation.

The third method is to use a analog bit-parallel point-to-point interconnection. The $vi_{jkl,mnpq}$'s are converted to an analog value and then

transmitted to the neighboring hyperneurons. The analog $vi_{jkl,mnpq}$'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 practical for multi-hyperneuron hardware implementation.

Table 5.1 compares these three above-mentioned neighbor interconnection methods. Recent studies on electro-optic computer have shown that the limitation of conventional electrical interconnection can be overcome by using electro-optical interconnect through the free space. The electro-optic devices, such as photo detectors and modulators can be integrated on a chip to circumvent the communication bottlenecks [5.19]. This prospective approach is under investigation.

5.3.3 Digital Co-processor

The digital co-processor calculates and stores the synapse weights and bias inputs in a static-RAM which has $[(2W_r+1)(2W_c+1)+1](2D_x+1)(2D_y+1)$ s-bit words for each hyperneuron. The s-bit D-to-A converter transforms the digital representation of the synapse weights into analog values for charging the weight-storage capacitance of the synapse matrix. While doing the retrieving process, the write-through circuits allow a concurrent writing of the converted data to the corresponding synapse capacitor. A two-port static-RAM and differential amplifier-based synapse design allow network retrieving and learning processes to occur

concurrently. The digital synapse memory can be shared by a cluster of hyperneurons.

5.4 Detailed Circuit Implementation

Advanced studies to improve the circuit performance and to reduce the area/power of the neural building blocks are essential to implement the highly complex neural systems in VLSI technologies [5.16]. Computer simulation and laboratory experiments on these neural circuits have been conducted.

5.4.1 Programmable Synapse

Figure 5.6 shows a circuit design of a programmable synapse cell. It is a modified transconductance amplifier that can perform real-valued masking operations and achieve 8-bit precision. In order to realize the function specified in Eq. (5.13), the synapse cell produces output current $I^s_{i,j}$ according to mask voltage V_{mask} and weight voltage $V^s_{i,j}$. The mask voltage of each synapse cell is directly dependent on the binary value of the $v_{m,n,k,l}$ which is the velocity status of the neighboring pixels. When the V_{mask} is at logic 1, the V_{bias} is connected to V_{on} and provides the amplifier with a specific bias current I_{max}. While the V_{mask} is at logic 0, the V_{bias} is connected to the negative power supply V_{SS} so that there is no synapse output current is produced. Therefore, the V_{mask} perform a masking operation on the synapse weight voltage $V^s_{i,j}$. The layout of one synapse occupies 112 λ x 82 λ in the MOSIS-scalable CMOS design.

Figure 5.7 shows the measured DC characteristics of the synapse cell. The bias voltage controls the dynamic range of the synapse cells by adjusting the bias current in the transconductance amplifiers. The maximum synapse conductance is decided by devices of the differential pair and the bias current, while the minimum synapse conductance is determined by the resolution of the synapse weight value. The synapse weight value is dynamically stored on the capacitance of the MOS transistors. It must be refreshed periodically since a parasitic leakage exists in the diffusion-to-substrate junction. An 8-bit resolution can be supported in this DRAM-like synapse cell [5.16].

5.4.2 Output Summing Neuron

The output summing neuron sums up the synapse output currents and converts the summed current into the output voltage, which is sent to the winner-take-all (WTA) circuit. The output of each neuron is the selectivity measure and the maximum among them is chosen as the winner in the competitive layer. The output neuron has its summed current converted into the voltage with the sign preserved. In Fig. 5.8, the circuit diagram and the transistor sizes for the amplifier used in the output neuron are shown. To ensure the linear current-to-voltage conversion for a wide operation range, the output neuron can support the summing current up to 1 mA. It has a large output buffer to allow a large amount of current. Linear resistance is required to convert the current into the voltage. Since the accuracy of a passive resistor is very low (possibly up to 20% error) without the additional trimming technique, multiple MOS transistors biased in the triode region can

be used to synthesize the linear resistance. The layouts of the current summing neuron and the linear floating resistor occupy 116 λ x 228 λ and 116 λ x 64 λ, respectively.

5.4.3 Winner-Take-All Cell

The winner-take-all circuit functions as a multiple-input parallel comparator. Only the WTA cell with the maximum input voltage will have the logic-1 output value, while the other cells have the logic-0 output value. The WTA circuit schematic and the size of each transistor are shown in Fig. 5.9. Each cell occupies 58 λ x 96 λ in the scalable CMOS design. One WTA cell consists of two portions. The first portion converts input voltage into the current which is compared and redistributed in the common signal line. In the second portion, the current is converted into the output voltage. All transistors operate in the saturation region. V_{CM} is the *common node voltage* to which all source terminals of input transistors *M1* are connected. As the number of inputs increases, the circuit can be extended by abutting this common signal node from the cells. Through this node, the total bias current is contributed by every cell. Since the source terminal is at a common voltage for all the cells, the current flowing through each cell is proportional to Vi^2. Thus, the largest input can fetch the largest current out of the total bias current. This largest current can make the corresponding output saturated at the positive supply voltage value. On the other hand, the other outputs will be saturated at the negative power supply value. The total bias current is provided by the transistor M_5 of the cells. Instead of using a fixed amount of the total bias current, each cell provides its own share of the bias current.

Since the bias current increases in proportion to the number of inputs, the circuit response time is independent of the number of inputs.

5.4.4 Velocity Status Register

The velocity status of image pixel is represent by the $(2D_k+1)(2D_l+1)$ binary output of winner-take-all circuits. As show in Fig. 5.10 , the velocity status can be encoded and stored in a register which is accessible by the digital co-processor through the digital system bus.

5.4.5 Neighbor Interconnection Sender

The neighbor interconnection sender is used to convert the encoded binary code to the analog value and send it to the neighboring neuronprocessors. It is implemented by using a voltage-scaling digital-to-analog converter as shown in Fig. 5.11. The voltage-scaling converter uses a series of resistors connected between V_{ref} and V_{-ref} to provide intermediate voltages. For an N-bit converter, the resistor string would have $2^N + 1$ resistor segments. Unity-gain followers are used to buffer the resistor string from conductive loading. Each tap is connected to a switch that is controlled by the digital word.

5.4.6 Neighbor Interconnection Receiver

When the analog velocity information from the neighboring neuroprocessors is received, the neighbor interconnection receivers are used to convert the analog values back to the $(2D_k+1)(2D_l+1)$ -bit binary codes. Notice that only one of these bits is logic-1 and the others are logic-0. Each

neuroprocessor has a total of $(2W_i+1)(2W_j+1)-1$ neighbor interconnection receivers. A high-speed receiver is implemented by using a voltage-scaling analog-to-digital converter as shown in Fig. 5.12. One voltage-scaling resistor is used to generate reference voltages. Each reference voltage is shared by comparators in the same velocity-sensitive neural unit. A comparator and its combinational circuit are associated with each synapse cell. The combinational circuits are used to make sure that only one of the $(2D_k+1)(2D_l+1)$ binary outputs is logic-1.

5.5 Experimental Results

5.5.1 Prototype Chip

Figure 5.13 shows the layout design for a 25-velocity selective hyperneuron. A size of 5x5 smoothing window *and* $Dx = Dy = 2$ are used. The 25-velocity selective hyperneuron for one image pixel is implemented with a silicon area of 2,482 x 5,636 λ^2 and contains 25 neurons, 25 x 27 synapse cells. Two raw of synapses are used to increase the resolution of the synapse weights updated by the bias inputs. The extra raw of synapses are also used to enhance the fault tolerance of the network. With an advanced submicron CMOS technology, 125 neuroprocessor can be accommodated into one VLSI neural chip of 12.5 mm x 11.7 mm in size. The chip layout is shown in Fig. 5.14. It requires a 240-pin PGA package. The analog interprocessor data communication requires 192 pins. Th interconnection routing area occupies 23% of the chip area. In the digital bit-parallel method, 960 pins are required for the interprocessor data communication. Only 24 neuroprocessors can be

accommodated in the same chip area and 85% of the chip area is used for the interconnection routing. Use of compact and electrically programmable mixed-signal neural units is the key to achieve high-density hardware design. The point-to-point interconnections among hyperneurons are effectively designed using the analog bit-parallel method.

5.5.2 Chip Performance Measurement

Table 5.2 shows that the measured processing time for one network iteration is about 500 nsec. Each iteration cycle includes input buffering, synapse masking, neuron summing, winner-take-all operation, velocity status encoding, digital-to-analog conversion, and analog-to-digital conversion. The computation power of the 128-neuroprocessor chip is about 166.4 Giga connections per second. Computing of optical flow using one 128-neuroprocessor chip can be accelerated by a factor of 379 than a Sun-4/260 (a 16 MHz SPARC processor) with 16 MB of RAM. The performance is based on a 2 MHz system clock rate for the 2-μm neural chip implementation. Use of the 0.5-μm VLSI technology will approximately improve the performance by a factor of 4.

Figure 5.15 shows the measurement results of the two-stage 9-cell winner-take-all circuits. The winner-take-all circuits achieve a resolution of 10 mV. In the mixed-signal neural chip design, the common-centroid layout technique can greatly alleviate the device mismatch effect. Use of large devices can also reduce sensitivity to mobility variation and channel-length modulation variation. The carefully chosen device sizes of help to keep the variation of the transconductance constant β below 1%. In addition, the

operational dynamic range of the synapse cell can be increased with larger devices, which can ensure the 8-bit accuracy for the analog circuitry.

5.6 Optical-flow Neuroprocessor Based System

5.6.1 System Implementation and Operation

Real-time optical flow computation on industrial images is practical using an extended array of hyperneurons. Figure 5.16 shows an overall system configuration for a real-time optical flow system using velocity-selective hyperneurons array. The system contains a host computer, an array controller, a parallel up/down load interface, and a hyperneurons array.

The operation for computing the optical flow on the high-speed neural system can then be summarized as follows:

1. The host system set the initial state of the optical-flow neuroprocessors array by down-loading the corespondent sub-image pixels, initial velocity maps, and synapse weights to the velocity sensitive neurons.
2. All velocity-sensitive neurons are parallel operated to synchronously update all the velocity maps of the sub-image.
3. Apply electronic deterministic annealing mechanism to search the optimal solution. Repeat Step 1 and step 2 till the whole image is done.
4. Detect occluding elements and update bias inputs accordingly. Compute optical flow with new bias inputs.
5. Determine the optical flow field according to the stable velocity maps.

The optical flow field can then be used for the motion-compensated video image compression.

5.6.2 Hardware Constrained System-Level Analysis

Incorporating the application and hardware implementation constrains, system-level analysis on different images have been conducted to illustrate the performance of the optical flow estimation chip. Since the electronic neural processor is designed for video coding, computing for the terms which are weighted by the parameter C is not included to simplify the neuron-state updating of the network. The parameter C is set to be 0 in the system-level simulation to virtually emulate the hardware performance. In additions, the mismatch effect of neural circuits is included in the system-level analysis.

Two sets of successive image frames directly produced by a CCD camera were used. To estimate the principal curvatures and the intensity values, a 5 x 5 window was chosen and a third order polynomial was used for all frames. Figure 5.17(a) 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 x 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, Dx = 5, and Dy = l, the velocity field was obtained after 36 iterations. Figure 5.17(b) shows the final result with using synapse weights obtained by including the effects of process variation. Comparing with the result in Fig. 5.17(c), which the effects of process variation are not included, the motion information of the moving object still can be successful detected.

Figure 5.18(a) 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 x 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, Dx = 7, and Dy = l, 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. 5.18(b) and Fig. 5.18(c), 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.

5.7 Conclusion

In this chapter we presented an effective VLSI neuroprocessors design for high-speed high-accuracy optical flow based image sequence compression by using the massively parallel neural computing architecture and mixed-signal VLSI design techniques. The work includes: (1) Algorithm improvement and simulation for the optical flow computing; (2) Application-specific multi-layer stochastic neural network architecture development; (3) Detailed circuit and chip design to fully explore the mixed analog-digital neural network capability in high speed optical flow computing. (4) Integration of this neuroprocessor chip into an advanced optical flow computing system. The proposed VLSI optical flow computing system is also essential for other potential optical-flow based applications such as motion perception, scene analysis, and motion estimation.

References

[5.1] C. Cafforio, F. Rocca, "Methods for measuring small displacements of television images," *IEEE Trans. on Information Theory*, vol. IT-22, pp.573-579, 1976.

[5.2] Haskell, "Frame-to-frame coding of television pictures using two-dimensional Fourier transforms," *IEEE Trans. on Information Theory*, vol. IT-20, pp. 119-120, 1974.

[5.3] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, pp. 185-203, Aug. 1981.

[5.4] R. A. Jones, H. Rashid, "Residual recursive displacement estimation," *Proc. Conf. Pattern Recognition and Image Processing*, pp. 508-509, 1981.

[5.5] J. O. Limb, J. A. Murphy, "Measuring the speed of moving objects from television signals," IEEE Trans. on Communication, vol. COM-23, pp. 474-478, 1975.

[5.6] J. O. Limb, J. A. Murphy, "Estimating the velocity of moving images in television signals," IEEE Trans. on Communication, vol. COM-23, pp. 474-478, 1975.

[5.7] A. N. Netravali, J.D. Robbins, "Motion-compensated television coding: Part I," Bell System Tech. J., vol. 58, pp. 631-670, 1979.

[5.8] B. G. Schunck, "Motion segmentation and estimation," Department of Electrical Engineering and Computer Science, MIT., Ph. D. Dissertation, 1983.

[5.9] B. G. Schunck, "The motion constrain equation for optical flow," Proc. Int. Joint Conf. on Pattern Recognition, Montreal, pp. 20-22, 1983.

[5.10] J. A. Stuller, A.N. Netravali, "Transform domain motion estimation," Bell System Tech. J., vol. 58, pp. 1673-1702, 1979.

[5.11] J. A. Stuller, A.N. Netravali, J.D. Robbins "Interframe television coding using gain and displacement compensation," Bell System Tech. J., vol. 59, pp. 1227-1240, 1980.

[5.12] K. Prazdny, "On the information in optical flows," Computer Vision, Graphics and Image Processing, 22:239-259, 1983.

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

[5.14] Y. Zhou and R. Chellappa, "Computation of optical flow using a neural network," IEEE Int. Conf. on Neural Networks, Vol. 2, pp. 71-78, San Diego, CA, July, 1988

[5.15] W.-C. Fang, B. J. Sheu, "Real-time computing of optical flow using adaptive VLSI neuroprocessors," *IEEE International Conference of Computer Design*, pp. 122-125, Cambridge, MA, Oct. 1990.

[5.16] B. W. Lee, Bing J. Sheu, "Hardware simulated annealing in electronic neural networks," IEEE Trans. on Circuits and Systems. Vol. 37 , no. 6, June 1990.

[5.17] S. Ullman, *The Interpretation of Visual Motion*, M.I.T. Press, Cambridge, MA, 1979.

[5.18] J. J. Hopfield, D. W. Tank, "Neural computation of decision in optimization problems," Biol. Cybern., vol. 52, pp. 141-152, 1985.

[5.19] A. Dickinson, M. E. Prise, "An integrated free space optical bus," Proc. of IEEE Int. Conf. on Computer Design, pp. 62-65, Cambridge, MA, 1989.

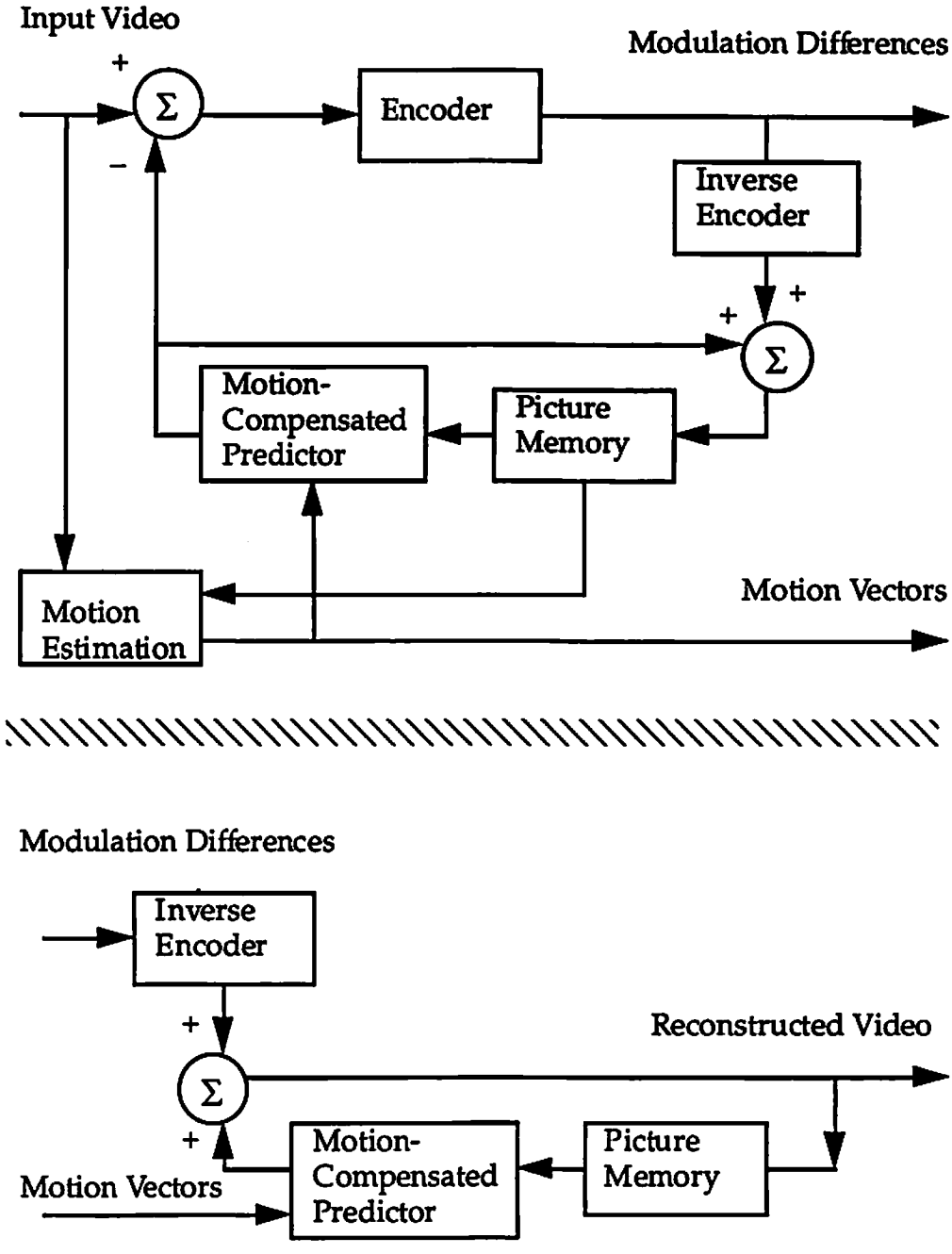


Fig. 5.1 A generic motion-compensated predictive video compression encoder.

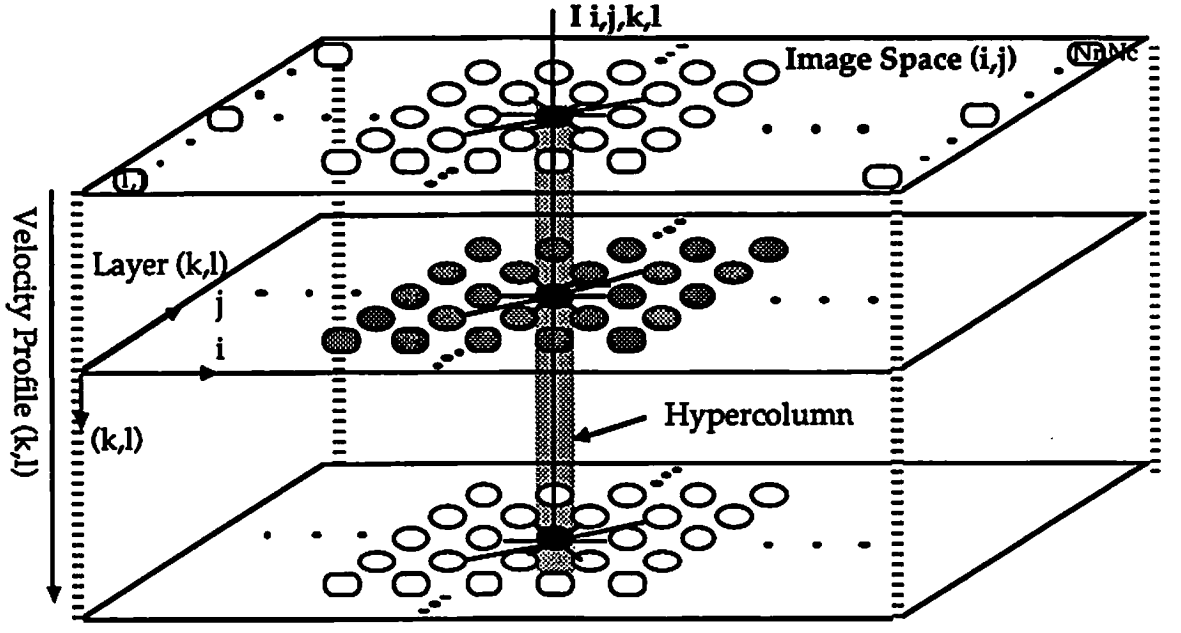


Fig. 5.2 A locally connected multi-layer neural network.

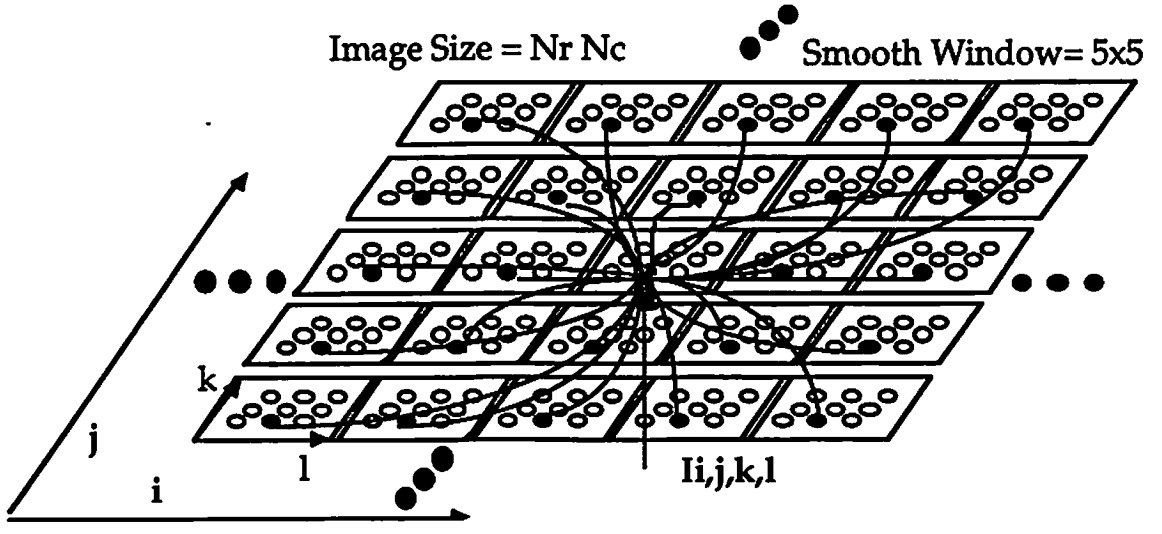


Fig. 5.3 A 2-D array neuroprocessors for optical flow.

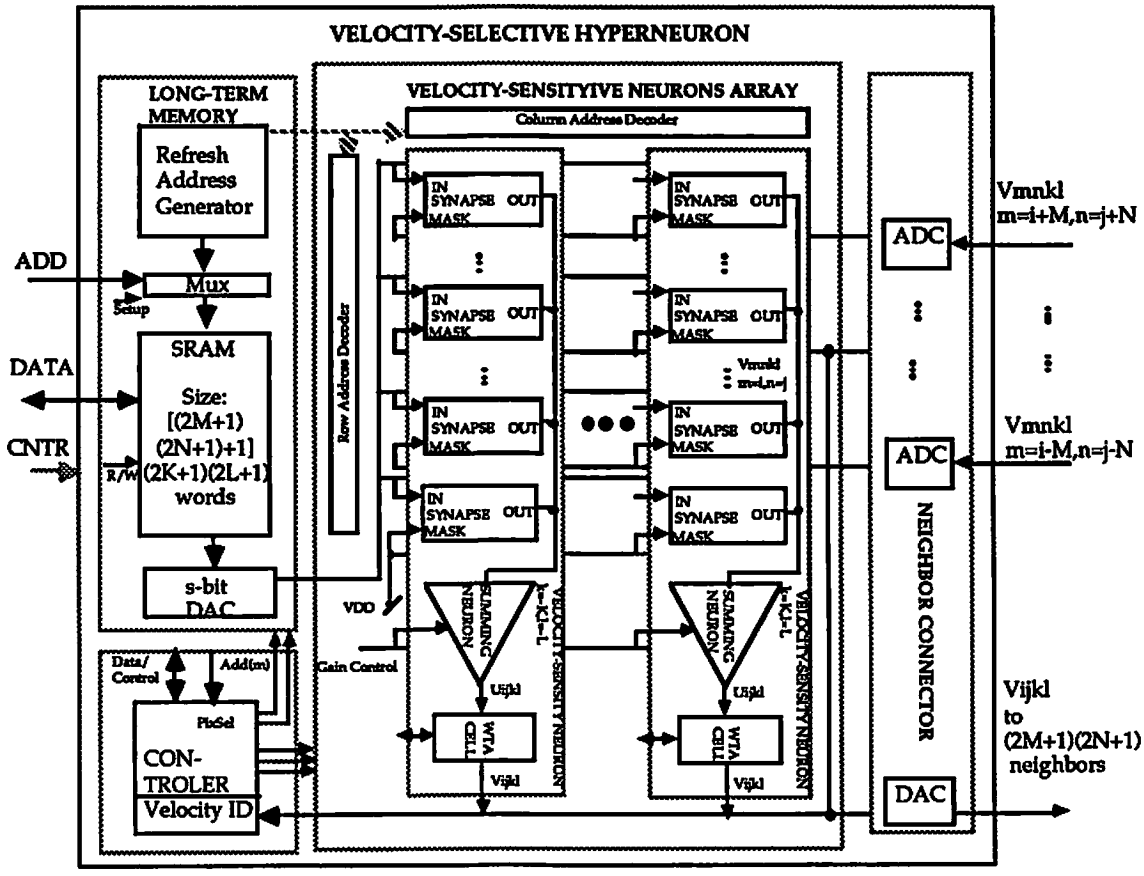


Fig. 5. 4 Functional diagram of the velocity-selective hyperneuron.

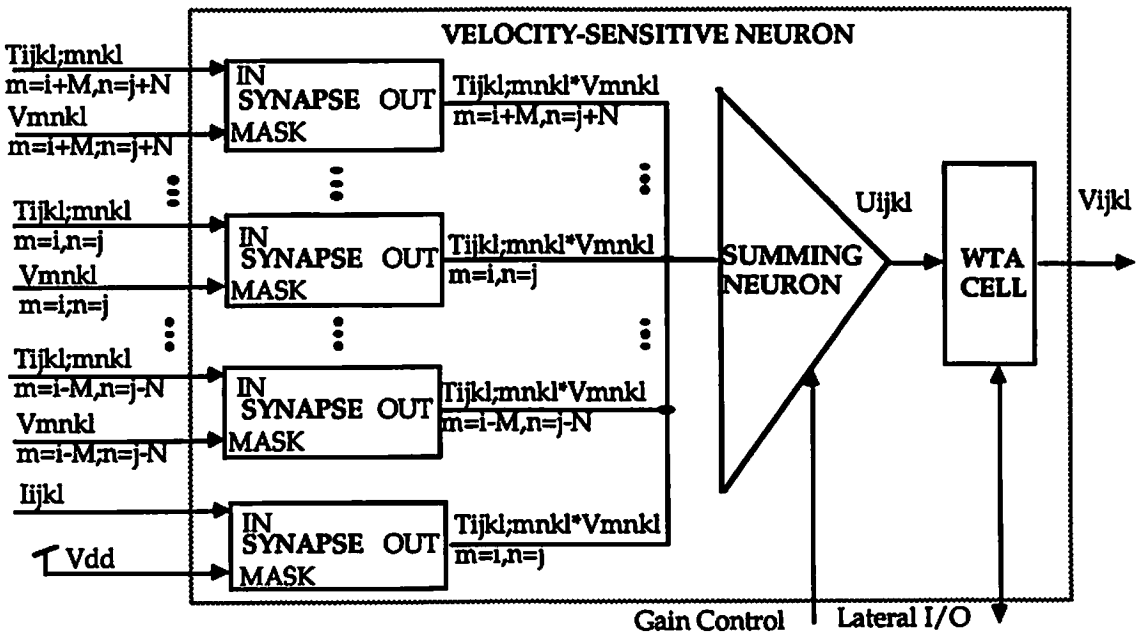
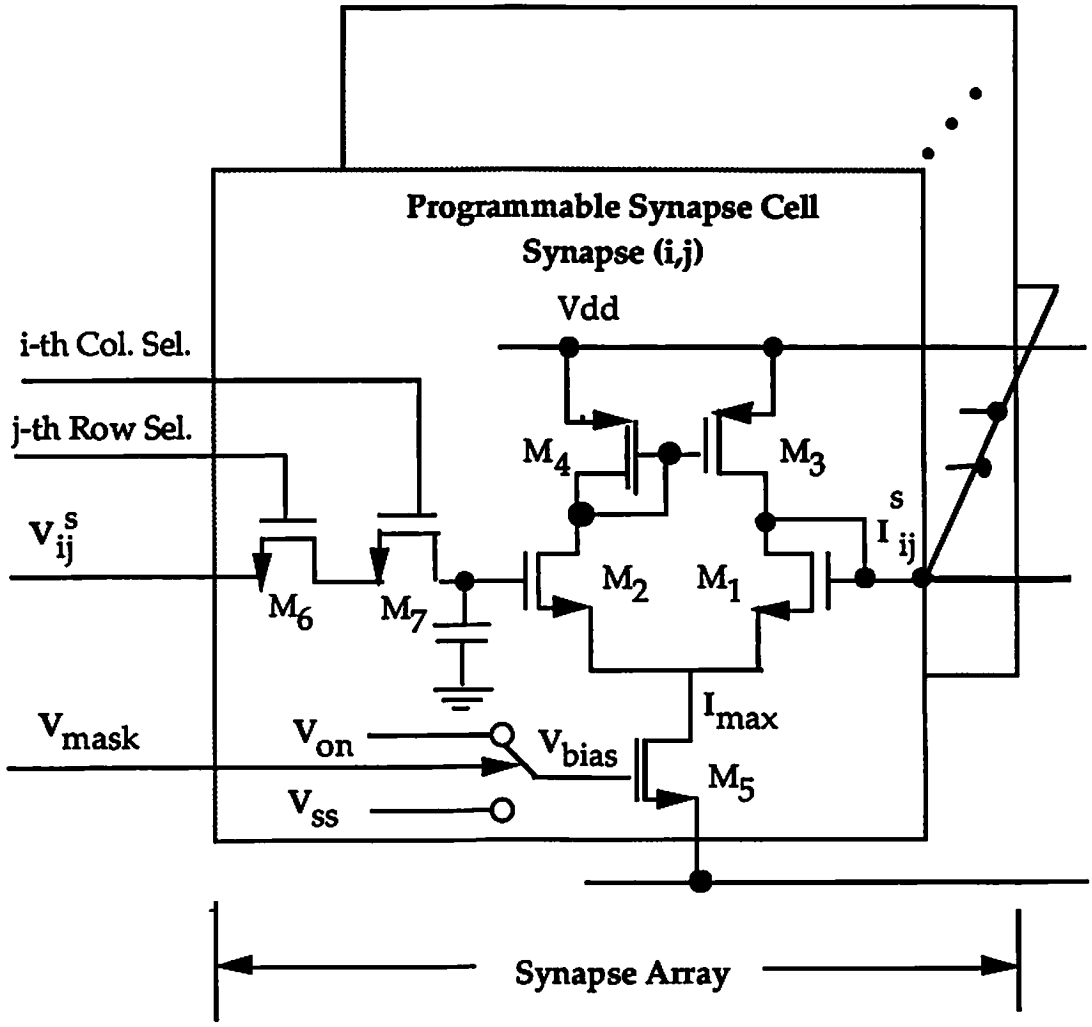


Fig. 5.5 Block diagram of the velocity-sensitive neuron.



Transistor	M_1	M_2	M_3	M_4	M_5	M_6	M_7
Size (μm)	3/39	3/39	4/4	4/4	5/3	4/2	4/2

Fig. 5.6 Circuit schematic of the programmable synapse cell.

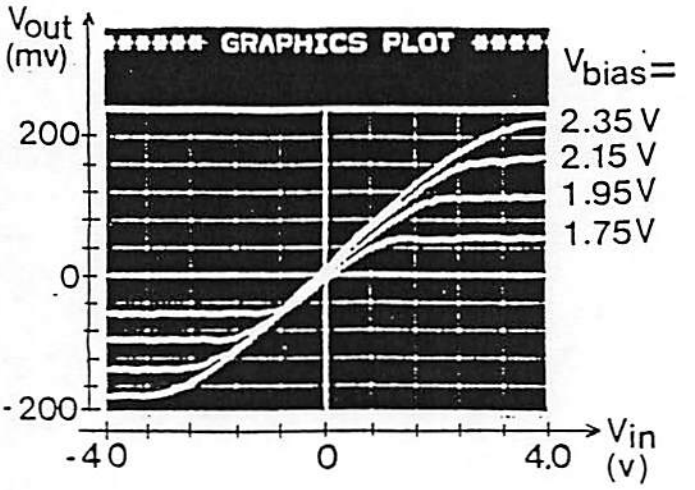
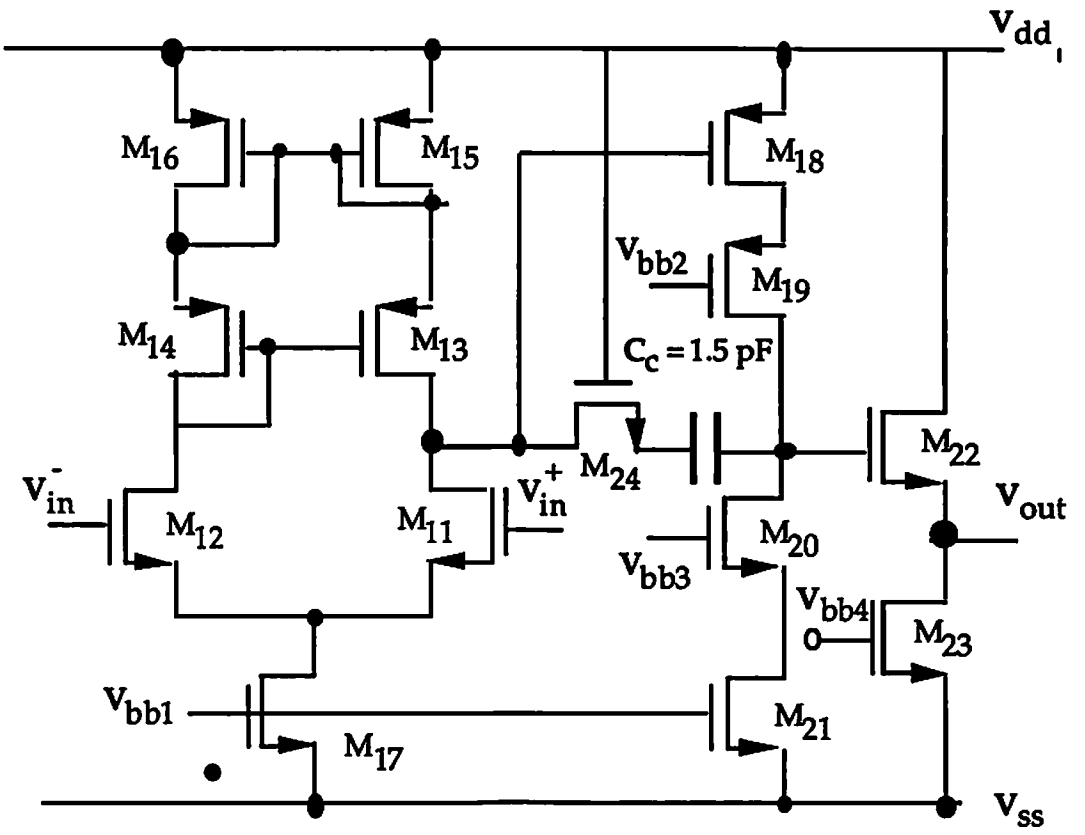
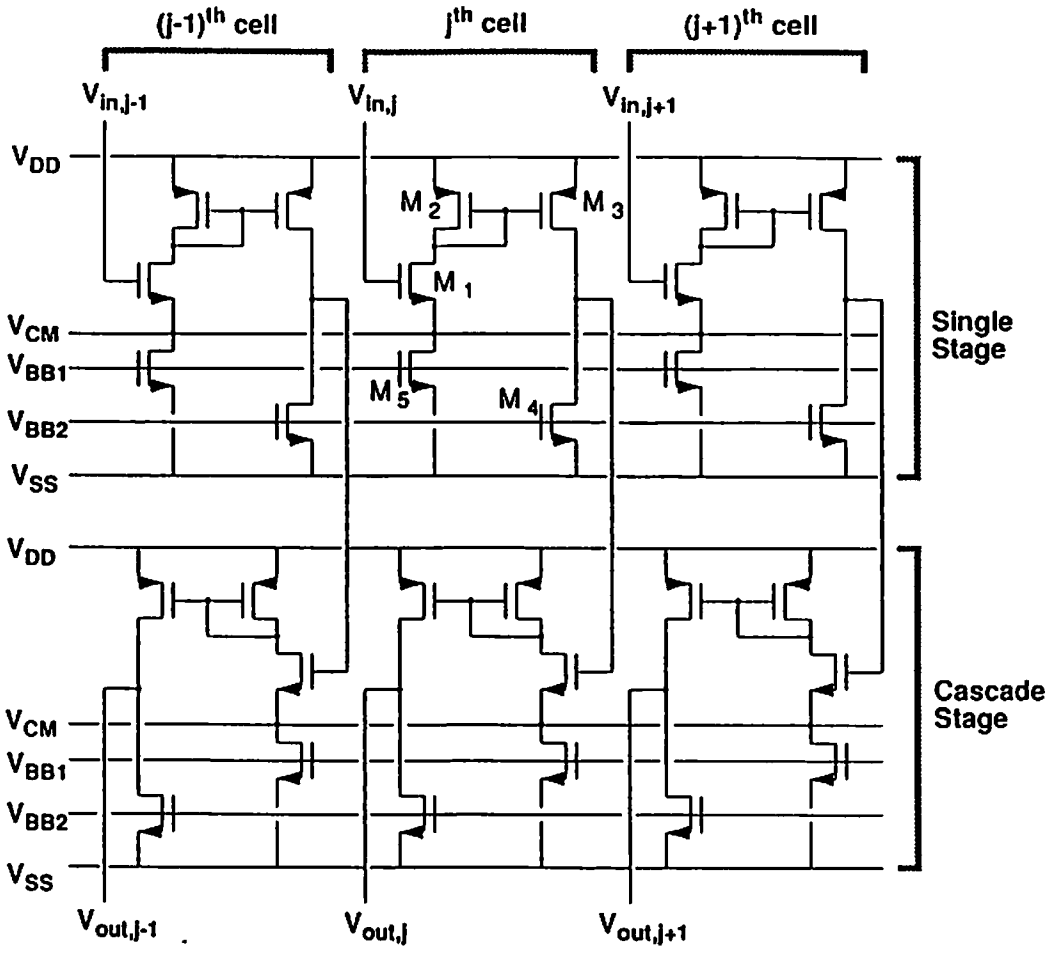


Fig. 5.7 A measured DC characteristics of the synapse cell.



Transistor	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}
Size (μm)	16/2	16/2	6/2	6/2	20/4	20/4	40/4
Transistor	M_{18}	M_{19}	M_{20}	M_{21}	M_{22}	M_{23}	M_{24}
Size (μm)	46/4	90/2	26/2	52/4	600/2	400/4	50/2

Fig. 5.8 A circuit diagram and its associated transistor size of the output summing neuron.



Transistors	M_1	M_2	M_3	M_4	M_5
Size (μm / μm)	8 / 2	8 / 4	16 / 4	8 / 4	16 / 4

Fig. 5.9 A circuit diagram and its associated transistor size of the winner-take-all cell.

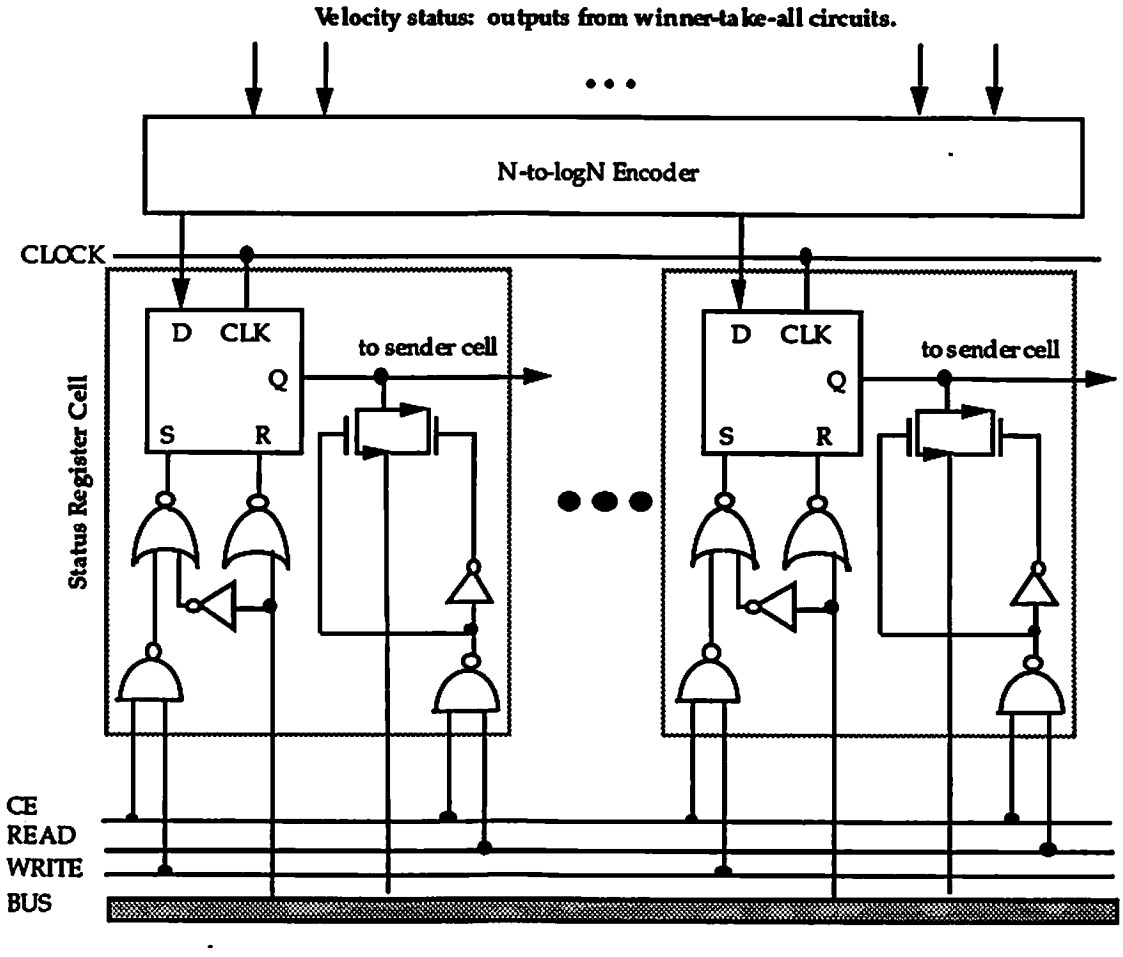


Fig. 5.10 Velocity status register of a hyperneuron that is accessible by the digital co-processor through the digital system bus.

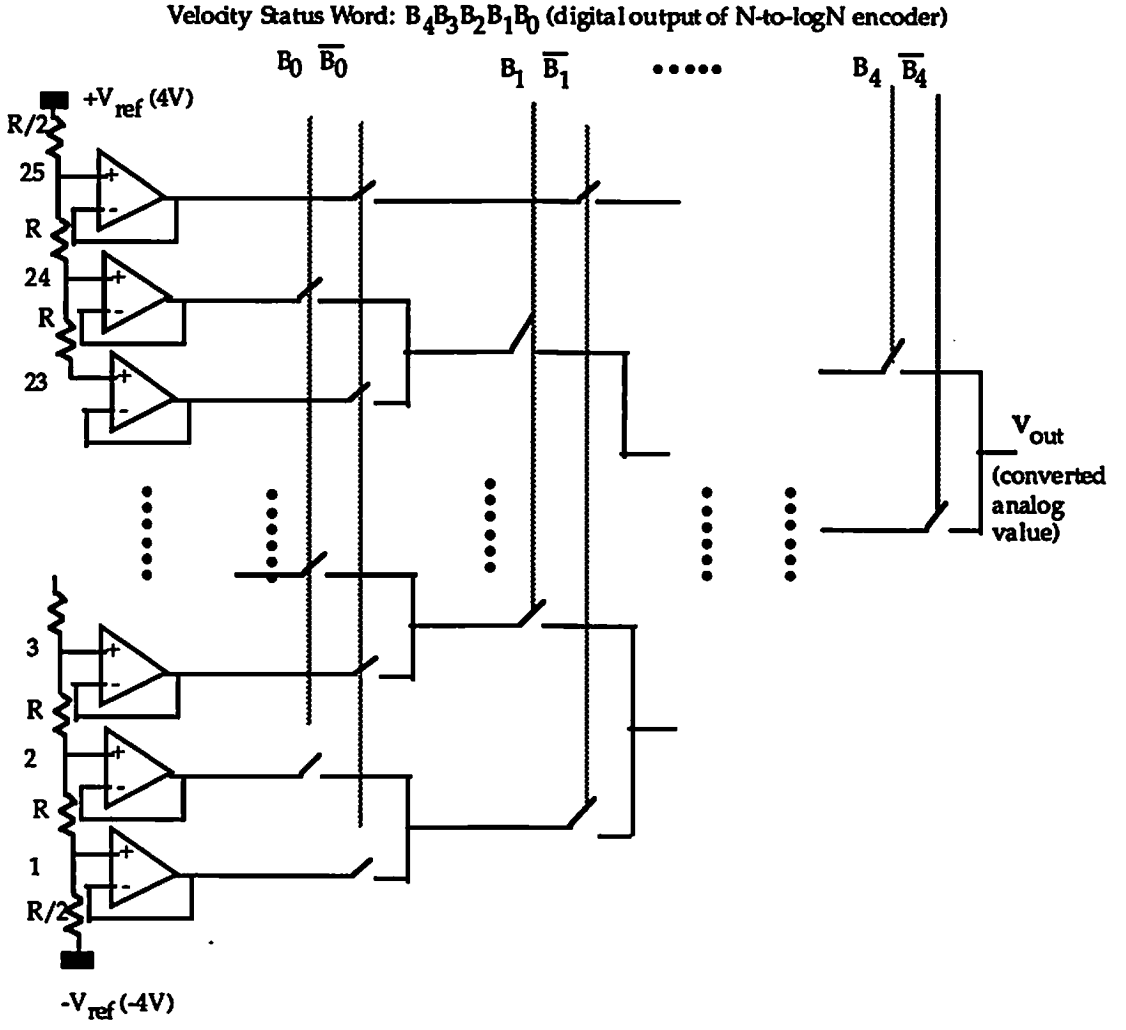


Fig. 5.11 Neighbor interconnection sender based on a voltage-scaling digital-to-analog converter .

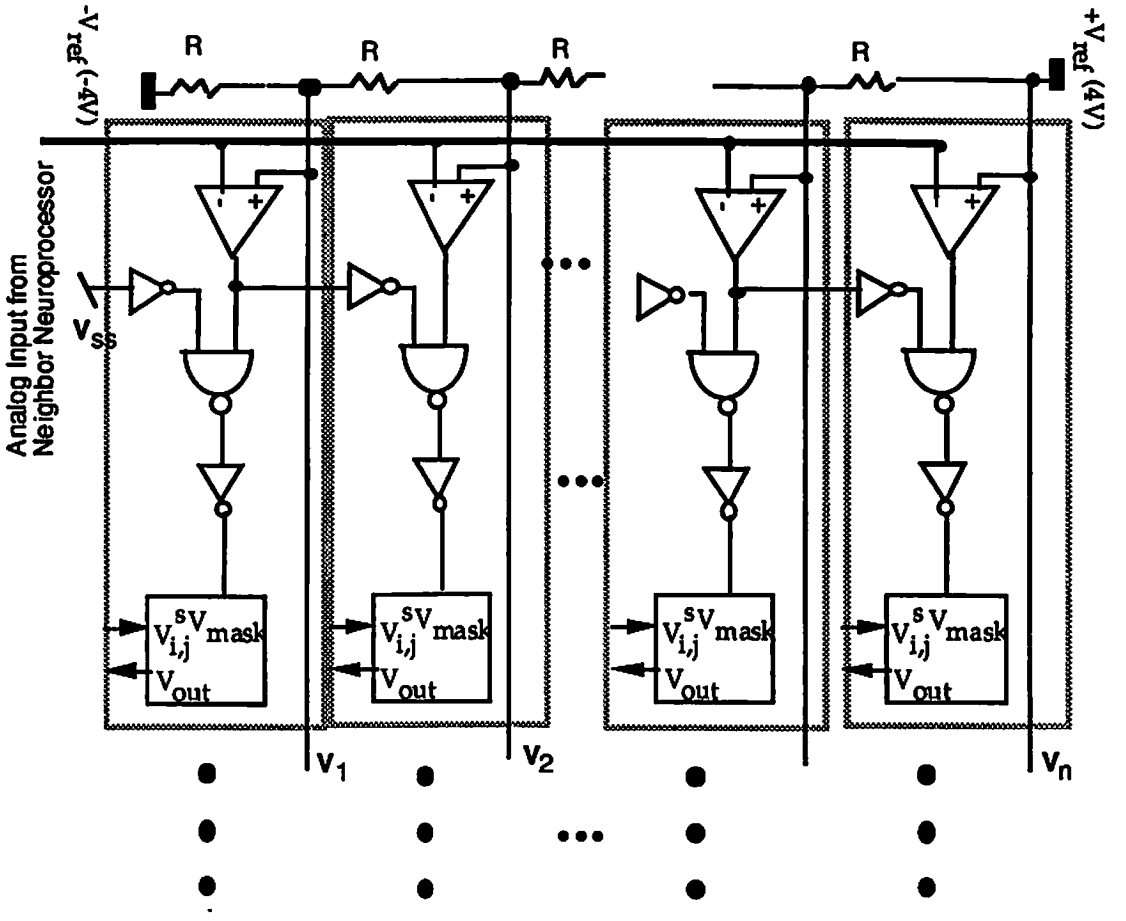


Fig. 5.12 Neighbor interconnection sender based on a voltage-scaling analog-to-digital converter .

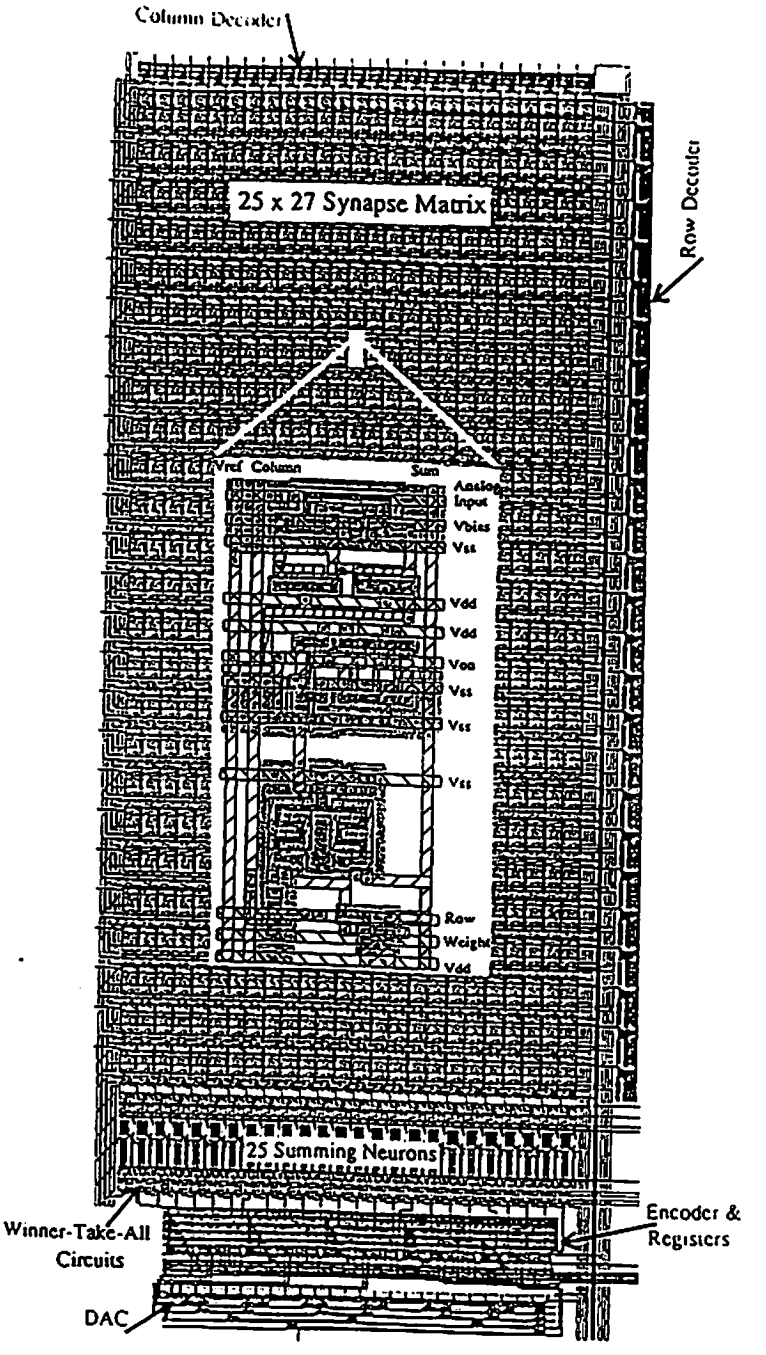


Fig. 5.13 The layout of one velocity-selective neuroprocessor. The 25-velocity selective hyperneuron for one image pixel is implemented with a silicon area of 2,482 x 5,636 λ^2 and contains 25 neurons, 25 x 27 synapse cells.

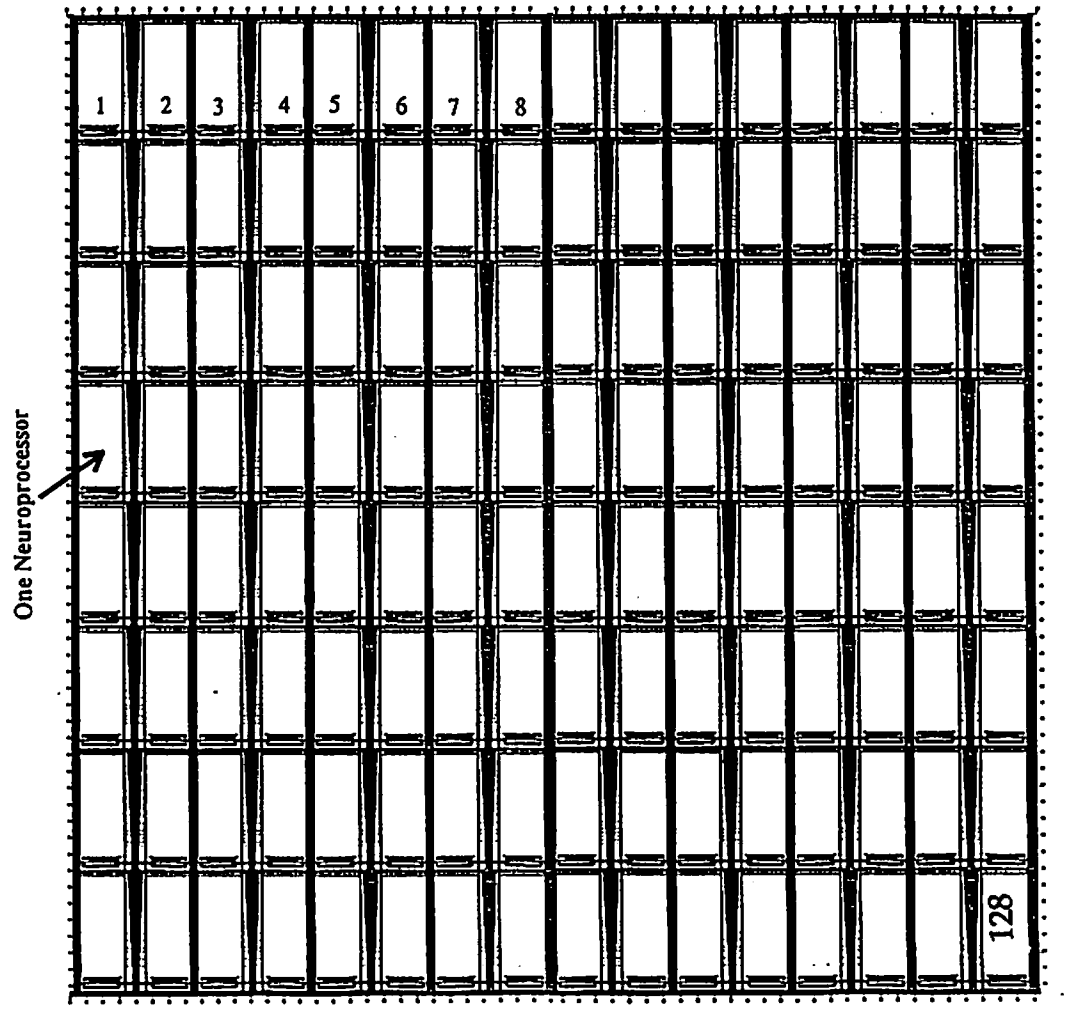


Fig. 5.14 The layout of a 128-neuroprocessor chip that occupies 12.5x11.7 mm^2 silicon area in a submicron CMOS technology.

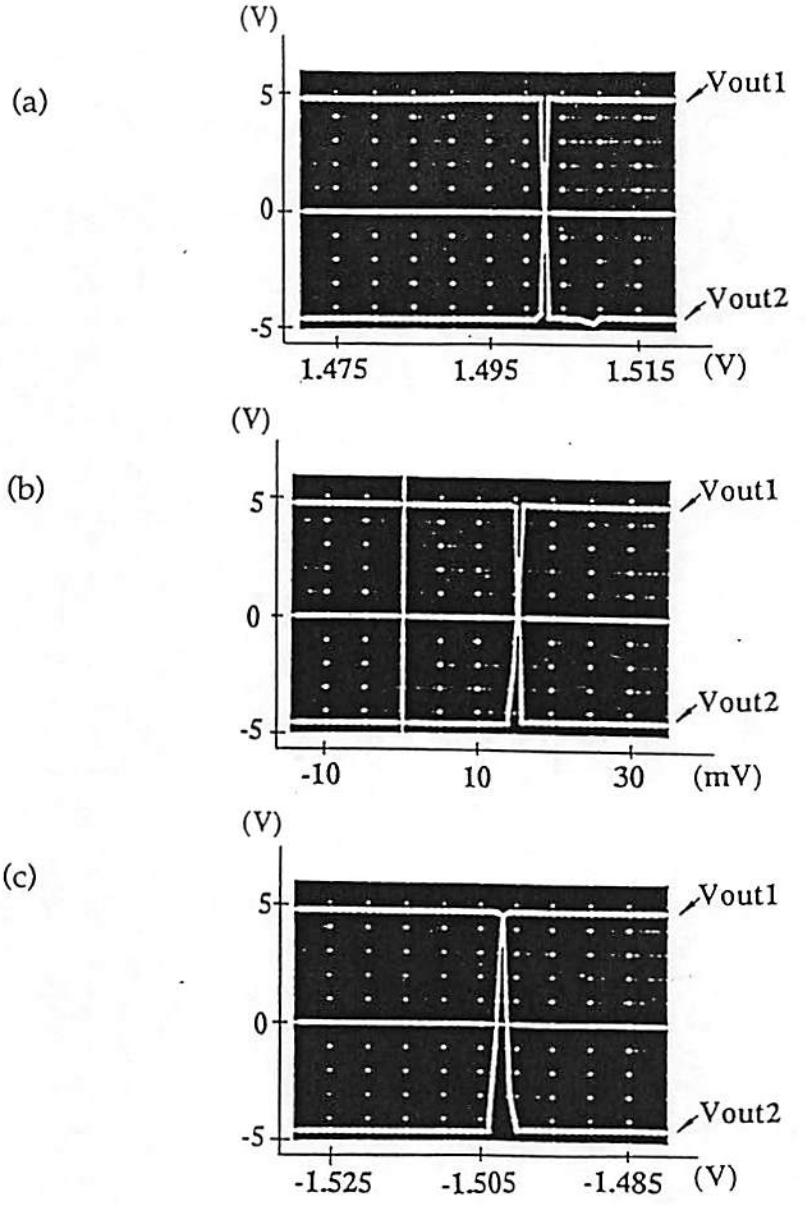


Fig. 5.15 The measurement results of the two-stage winner-take-all circuits: (a) Output of the 9-cell WTA test structure with one input sweeps from 1.47V to 1.52V, the second input is connected to 1.5V. (b) Output of the 9-cell WTA test structure with one input sweeps from -0.015V to 0.035V, the second input is connected to 0.015V. (c) Output of the 9-cell WTA test structure with one input sweeps from -1.53V to -1.48V, the second input is connected to -1.5V.

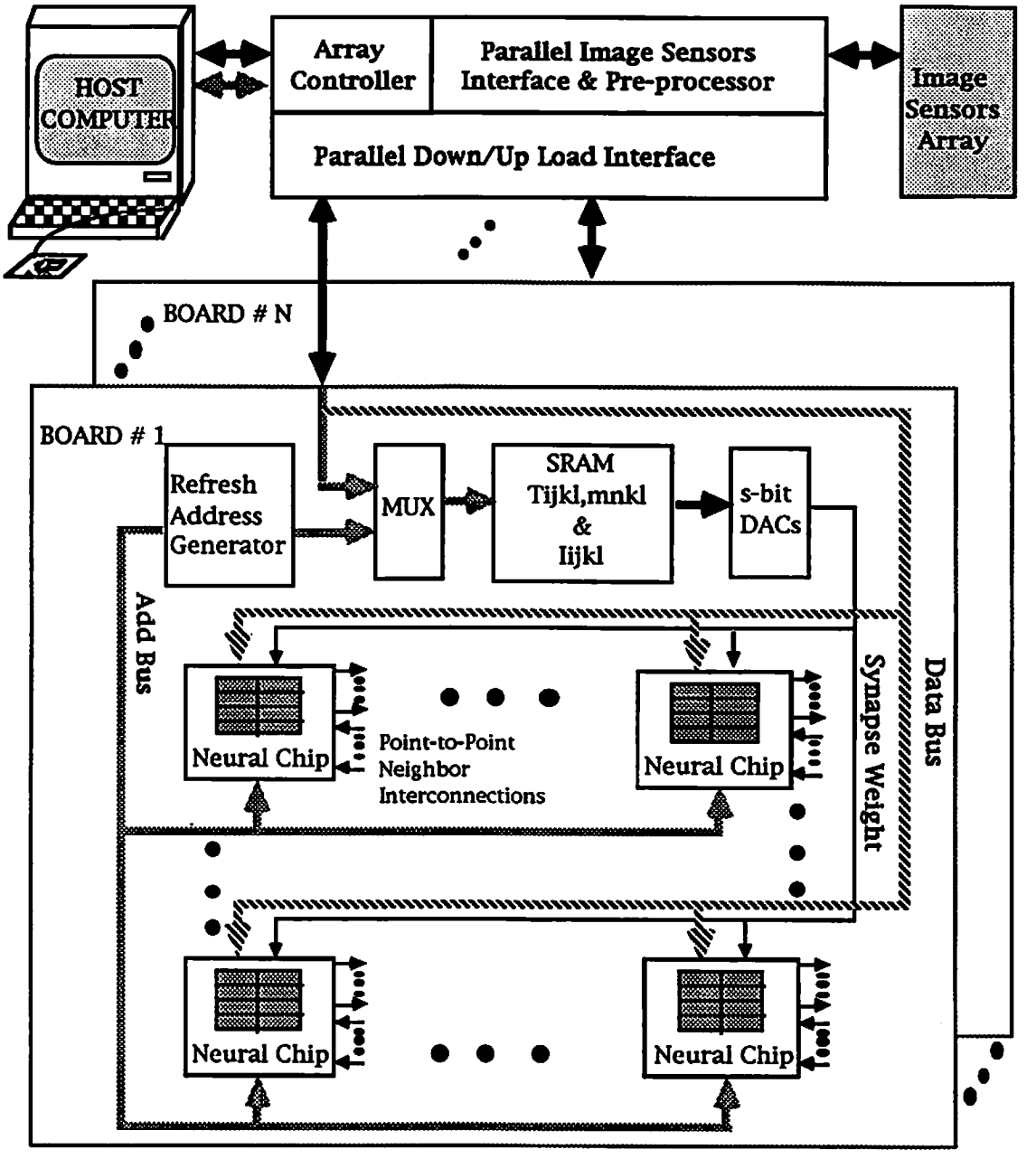
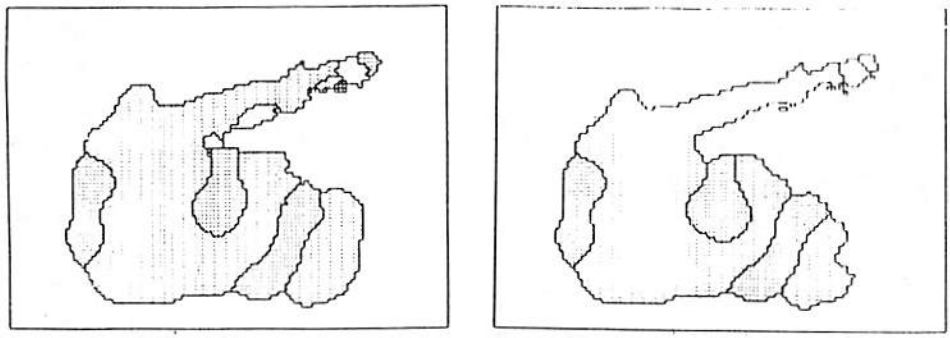


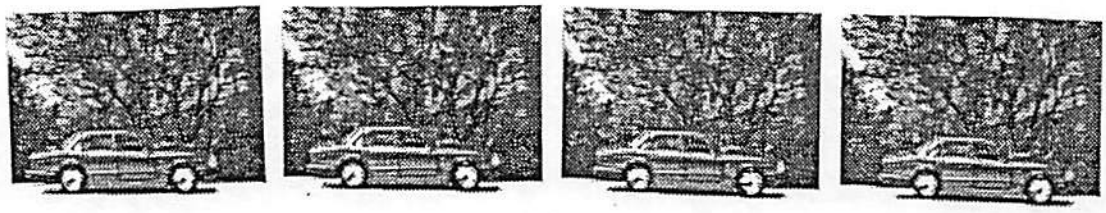
Fig. 5.16 System diagram for real-time optical flow computing using VLSI array neuroprocessors.

(a)

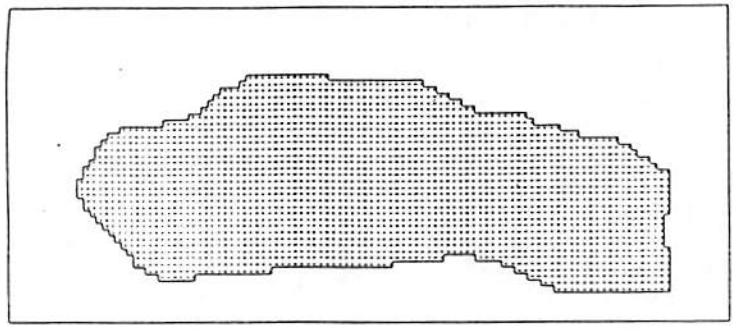


(b) (c)

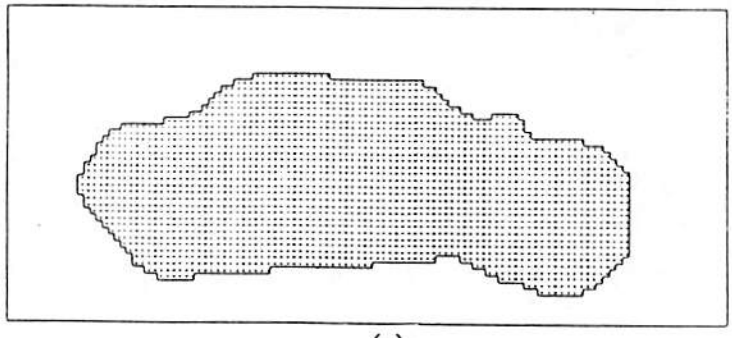
Fig. 5.17 System-level analysis on a sequence of four sedan images.
(a) The four temporal images of a moving sedan.
(b) By setting A = 4, B = 250, C = 0, Dx = 5, and Dy = 1, and using device mismatch effect, the velocity field was obtained after 36 iterations.
(c) Using same conditions as (b) except the device mismatch effect has not been included.



(a)



(b)



(c)

Fig. 5.18 System-level analysis on a sequence of four sedan images.
(a) The four temporal images of a moving missile launcher.
(b) By setting A = 4, B = 850, C = 0, Dx = 7, and Dy = 1, and using device mismatch effect, the velocity field was obtained after 36 iterations.
(c) Using same conditions as (b) except the device mismatch effect has not been included.

Table 5.1 Comparison of Neighbor Interconnection Schemes

Performance \ Method		Analog Bit-Parallel Interconnection	Digital Bit-Parallel Interconnection	Digital Bit-Serial Interconnection
Interrconnection time		1	1/5	X/5 (ns) X = vector dimension of velocity vector.
Interrconnection area		1	X	1
Pin count		X	X W	X
Real Design Example	Chip Size	1.5 x 2.8 cm^2	1.5 x 2.8 cm^2	
	Number of Processor	64	12	
	Interconnection Routing Area	23% of chip	85% of the chip	
	Pin Count	178	3250	
	Network Iteration	522	250	
	Speed Performance	8.03×10^{10}	2.08×10^{10}	

Table 5.2 Measured Network Iteration and Operation Speed.

		Measured Results
	Input Neuron Buffering (parallel with weight loading)	50 ns
one network iteration	Synapse Masking (Multiplication)	120 ns
	Output Neuron Summing	20 ns
	Winner-take-all Operation	38 ns
	Encoding and Data Latch	50 ns
	Velocity vector to value conversion (i.e. digital-to-analog conversion) & Velocity value to vector conversion (i.e. analog-todigital conversion)	84 ns (loading 5 pF) 263 ns (loading 50 pF)
	Velocity State Readout	50 ns
Network iterations time with an output loading of 5 pF.		263 ns
Network iterations time with an output loading of 50 pF.		522 ns
Totoal Processing Time for M weights and N iterations with an output loading of 5 pF.		343N + 100 M (ns)
Totoal Processing Time for M weights and N iterations with an output loading of 50 pF.		522 N + 100 M (ns)

Chapter 6

High-Speed VLSI Pipelined Processor Design for Lossless Image Data Compression

Abstract

An efficient VLSI pipelined processor design for high-speed lossless compression based on "Rice algorithm" has been developed to meet the increasing strong demands on high-volume/high-speed image data communication and storage. The Rice algorithm is an adaptive lossless coding scheme that provides near-optimal performance over a broad range of data entropies. The Rice algorithm is also an efficiently implementable scheme for VLSI realization. A VLSI pipelined architecture was developed to allow compact implementation of a single-chip VLSI compressor. This lossless compressor is named UNC-PSI14,K+ since it implements an advanced version of the Rice 's universal noiseless coding method called PSI14,K+. The chip layout was generated for a 1.0-micron CMOS technology. It occupies a compact chip area of 5.1 x 5.3 mm^2, with 49,000 transistors, 57 input/output pads, and 6 power/ground pads. The total power dissipation is 0.4 watts at the 40 MHz system clock with a 50% switching duty cycle. This compressor chip is mounted in a 84-pin pin-grid-array package. It can operate up to 40 Mpixels/sec. The potential applications of the proposed lossless compressor include database management systems, scientific instruments, CAE workstations, desktop computing machines, and the data systems that require high-speed compression without fidelity loss.

6.1 Introduction

Most real-world applications produce varied source symbol distributions. The optimality of variable length codes such as Huffman code is only efficient for a given source symbol distribution over a narrow range of data entropies. One of the earliest approaches of the adaptive variable length coding to alleviate this limitation was that developed by Robert F. Rice of Jet Propulsion Laboratory (JPL) and subsequently called the "Rice algorithm." Rice's universal noiseless coding technique appeared in its early form in [6.1] and was generalized in [6.2-6.4]. References [6.1-6.4] provide the development and analysis of some practical adaptive techniques for effective entropy coding of a broad class of data sources. Extensions and modifications to the original algorithms have been applied for a diverse set of applications in various forms [6.5-6.9]. The Rice algorithm is an adaptive scheme that provides near-optimal performance over a broad range of data entropies. Unlike Huffman coding, it does not require prior statistics of data. The Rice algorithm is also an efficiently implementable scheme for VLSI design. High-speed VLSI architectures of the Rice machine were presented in [6.10] and [6.11] to meet real-time satellite image compression requirements of the Earth Observation System satellite. Recent work at JPL has delivered a working VLSI lossless compressor that implement a special version of Rice machine called PSIss+ for space imaging applications [6.12]. A VLSI coder/encoder chip set based on the PSIss+ algorithm has been also produced at the University of Idaho in conjunction with the Goddard Space Flight Center

[6.13]. The algorithm definitions and performance characteristics of the PSIss+ was provided in [6.14].

This paper presents a high-speed VLSI processor design for the UNC-PSI14,K+ algorithm that is a more general and improved version of the Rice machine [6.4]. Section II describes the technique and performance of the UNC-PSI14,K algorithm. Section III is devoted to a detailed VLSI architecture and circuit design. Section IV describes a high-speed VLSI chip implementation of the UNC-PSI14,K encoder.

6.2 The Rice Algorithm and Its Performance

6.2.1 Rice's Universal Noiseless Coding Technique

Rice's universal noiseless coding is an adaptive lossless data compression technique that is employed to compress the data without inducing any distortion in the reconstructed image. A brief description of the Rice coding technique is given first to relate the Rice algorithm to an effective VLSI design. Interested readers are referred to [6.1-6.4].

Standard source transform

Since the Rice's universal noiseless coding assumes that the data are in probability decreasing order, the source data must be preprocessed (e.g., by taking difference of adjacent pixels) so that the algorithm can apply. The prediction and relabelling operations are used to map the original source data into *standard source data*. The prediction operation reduces the data entropy if there is a high correlation among adjacent pixels. The prediction operation involves the operation of taking differences between consecutive samples:

$$\Delta = x(i) - x(i\text{-}1) \tag{6.1}$$

where x(i) is the i th sample and x(i-1) is the i-1 th sample.

The relabelling step transforms an (n+1)-bit difference into an n-bit integer representation:

$$\delta = \begin{cases} 2\Delta - 1 & , \ 0 < \Delta \leq \theta \\ -2\Delta & , -\theta \leq \Delta \leq 0 \\ \theta + |\Delta| & , \ |\Delta| > \theta \end{cases} \tag{6.2}$$

where θ = MIN[6.x(i-1), 2^{n-1} - x(i-1)] and n is the number of bits per input pixel.

Fundamental Sequence

Let $\{s_i\}$ be the source symbols, n be the length of s_i in bits, P_i be the probability of occurrence, $\{x_i\}$ be the symbol used to represent $\{s_i\}$, and $\{L_i\}$ be the length of x_i in bits , where $0 \leq i \leq N$-1 and $N = 2^n$-1. The noiseless coding minimizes the data volume required to represent the source data without inducing any distortion. It minimizes

$$L = \sum_i P_i L_i \tag{6.3}$$

This is achieved by using less data to represent the source symbols which occur more frequently, i.e.,

$$L_i \leq L_j \quad \text{if} \quad P_i \geq P_j \ . \tag{6.4}$$

The noiseless coding algorithms include the Huffman coding and universal noiseless coding. The Huffman coding algorithm derives the optimal set of $\{L_i\}$, which requires the knowledge of the exact probability distribution [6.16,17]. On the other hand, the universal noiseless coding algorithm derives a suboptimal set of $\{L_i\}$ which only requires the knowledge of the probability ordering, i.e., it is assumed that (after relabling if necessary)

$$P_i \geq P_{i+1} \quad \text{for} \quad 0 \leq i \leq N\text{-}2 \ . \tag{6.5}$$

In [6.4], x_i is chosen as

$$x_i = (0...0)1 \quad , \tag{6.6}$$

where there are i 0's inside the parenthesis. Hence,

$$L_i = i+1 \tag{6.7}$$

and

$$L = \sum_{i=0}^{N-1} (i+1) P_i \ . \tag{6.8}$$

The mapping of the source symbols resulting from Eq. (6.6) is called the fundamental sequence. In [6.15], the "fundamental sequence" is shown to be equivalent to a class of Huffman code under the Humblet condition [6.18] , for source symbol sets having a Laplacian distribution. The performance of the fundamental sequence method depends on the source entropy. It is shown that its rate is close to the theoretical limit while the entropy is between 1.5 and 2.5 bits/pixel [6.4]. Its performance degrades while the entropy increases.

Split-Sample Scheme

An approach to improve the performance of the fundamental sequence method is achieved by the split-sample coding scheme [6.4]. This approach splits the source symbols into two parts: most significant bits and least significant bits. The least significant bits, which are considered to have higher activity are passed to the receiving end without coding, and the most significant bits, which are considered to have lower activity are passed to the noiseless coding to reduce the redundancy.

The splitting method is described as follows:

(a) Split each input sample into two parts: most significant bits and least significant bits.

(b)Transmit the least significant bits without coding.

(c) Apply the fundamental sequence coding to the most significant bits.

Let k be the number of least significant bits and n-k be the number of most significant bits. The k-bit splitting method is equivalent to divide $\{s_i\}$ into 2^{n-k} blocks of 2^k symbols each. Let Q_i represent the probability of the i^{th} block, then

$$Q_i = \sum_{j=(i-1)2^k}^{i2^k-1} P_j \ , \qquad (6.9)$$

and (i+1) bits are assigned to represent the symbols in the i^{th} block. Hence,

$$L = \sum_{i=0}^{2^{n-k}-1} (k+i+1)Q_i \ . \qquad (6.10)$$

Note that when k = 0, (10) becomes (8). The optimal choice of k depends on the shape of $\{P_i\}$. If P_i decreases substantially as i increases, smaller k will yield better results. Conversely, if P_i decreases slowly as i increases, larger k will yield better results.

The assumption on the knowledge of the probability ordering is quite realistic since the source data can be preprocessed . The preprocessed data typically assumes some degree of probability ordering . For this reason, the universal noiseless coding technique is more practical due to no need of any prior information on the source data statistics.

Features of the Rice algorithm are summarized as the following:

Advantages

- No distortion induced in the reconstructed data by the coding algorithm.

- Does not require the complete knowledge about the source statistics. Only probability ordering is required. More robust to mismatch in the statistics.
- Prefix code. No extra information is required for time coding.
- Simple for implementation.

Disadvantages

- Require an extra buffer for generating synchronous output samples. Buffer overflow and underflow may occur, which causes loss of data.
- Sensitive to channel bit errors. Any bit error causes loss of the remaining data. Require a reference sample for each line to prevent an error propagation.

6.2.2 UNC-PSI14,K+ Algorithm

This section briefly summarize a practical configuration of the UNC-PSI14,K+ algorithm. As shown in Fig. 6.1, the PSI14,K+ coder consists of two separated functional parts: The front-end pre-processor is a predictor followed by a symbol mapper, while the second part is an adaptive variable length coder PSI14,K that performs the actual adaptive symbol coding. Figure 2 shows an algorithmic structure of the PSI14,K+ coding in more detail.

Preprocessing

The "+" in PSI14,K+ refers to those processings that precede the actual coding by adaptive variable length coding. The UNC-PSI14,K+ design presumes that preprocessing can be done by using control signal *E1* to select an external preprocessor or a build-in preprocessor. The function of an external preprocessor can be arbitrary. The build-in preprocessor is defined as a simple one-dimensional predictor and mapper as specified in Eq. (6.1) and (6.2). In addition, the predicted sample can be prepared by using control signal *E2* to select an external predicted sample or a internal predicted sample.

Mid-step K-Sample Splitting

The "K" in PSI14,K+ refers to the K-Sample Splitting operation that is a mid-step between the preprocessing and AVLC coding. The parameter K gives an external control of the coder PSI14,K. Incrementing K by one will shift the dynamic range upward by one bit/sample.

PSI14,K Code Options

With K =0, PSI14,0 is equivalent to PSI14. With K ≠ 0, the net effect of a mid-step split of K bits is to transform each PSIi,k in the internal PSI14 into another Split-Sample option with K additional splits. The set of 12 code operators of the internal PSI14 are elaborated below by using the notations from [6.4].

Code Operator PSI0 Code operator PSI0,0 is defined as

$$PSI\,0[\tilde{\delta}^n] = cfs[\beta_1] * cfs[\beta_2] * \ldots cfs[\beta_a] \qquad (6.11)$$

The variable-length code cfs[•] is defined in [6.4]. The 3-tuple β_i's are derived by the following operations:

$$\widetilde{FS} = PSI\,1[\tilde{\delta}^n] = \zeta_1\zeta_2\ldots\zeta_{F_0} \qquad (6.12)$$

$$\widetilde{\overline{FS}} = \bar{\zeta}_1\bar{\zeta}_2\ldots\bar{\zeta}_{F_0}$$

$$Ext^3[\widetilde{\overline{FS}}] = (\bar{\zeta}_1\bar{\zeta}_2\bar{\zeta}_3)^* (\bar{\zeta}_4\bar{\zeta}_5\bar{\zeta}_6)^* \ldots (\bar{\zeta}_{F_0}00)$$

$$Ext^3[\widetilde{\overline{FS}}] = \beta_1^*\beta_2^*\ldots\beta_a$$

where $a = L\,(Ext^3[\widetilde{\overline{FS}}]) = \left\lceil \frac{F_0}{3} \right\rceil$ 3-*tuples*. .

Fundamental Sequence, $PSI_{1,0}$ The Fundamental Sequence (FS) operator forms the backbone for all reversible coding operators used in the UNC-PSI14,K except for the Backup operator. It generates a codeword for each entry value of $\tilde{\delta}^n$. For each unsigned integer i of $\tilde{\delta}^n$, its FS codeword is an (i+1)-bit binary string consisting of i zeroes followed by a single one. The non-zero one makes the code reversible and serves as a marker for the decoding process. This operator is also called $PSI_{1,0}[\bullet]$ or $PSI_1[\bullet]$.

Code Operator, $PSI_{1,k}[\bullet]$ This code operator combines the $PSI_{1,0}$ with a k-sample splitting operation, where $1 \le k \le 9$. For each n-bit sample value of $\tilde{\delta}^n$, the sample splitting operation splits it into two sub-samples. One is the decimal equivalent of the last k *LSB* bits and the other is the decimal

equivalent of the first $(n\text{-}k)$ *MSB* bits. Let $SS_{n,k}[6.\tilde{\delta}^n]$ denotes an operator which performs the *SS* operation on each n-bit unsigned value of a sample sequence $\tilde{\delta}^n$. Each sample is splitted into two sub-sample values and thus two sequences are generated. The first, denoted as $\underline{\tilde{M}}^{n-k}$ is obtained from grouping all sub-samples with $(n\text{-}k)$ *MSB* bits from the original sequence. The second, denoted as $\tilde{L}_k$ is obtained from grouping all sub-samples with the last k *LSB* bits. By using an asterisk (*) symbol to define concatenation, an application of the $PSI_{1,k}$ code operator on $\tilde{\delta}^n$ is then defined as follows:

$$PSI\ 1,k[\tilde{\delta}^n] = PSI\ 1,0[\underline{\tilde{M}}^{n-k}] * \tilde{L}_k \qquad (6.13)$$

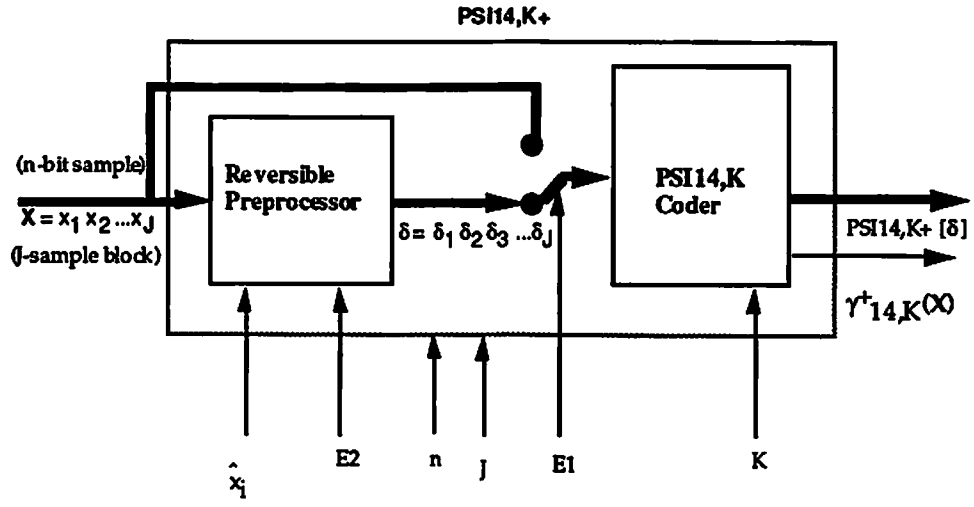
Code Operator, $PSI_3[\bullet]$ (Backup) The Backup operator takes the input block of n-bit sample data before entering the AVLC as its output. This code operator is defined as $PSI_3[\bullet]$.

Optimal Code Selection

The optimum criteria for selecting the best option to represent is to choose the winning code that produces the shortest coded sequence. A clear-cut decision regions based on F_0 can be used to effectively choose the winner. The equations lead to the decision regions for the internal PSI14 code is specified in [6.4].

Other Algorithm Requirements

The number of bits of data quantization lies in the range of $2 \leq n \leq 16$. The desired block size is over the range $1 \leq J \leq 16$. The fixed block size J= 16 is practical for the majority of applications. The UNC-PSI14,K processes input data on a line-by-line basis. The line starts when the ENABLE signal is turned on and ends when it is turned off. Therefore, a line can be any length of samples. Each sample consists of n bits in parallel. The final output from



Module PSI14,K+ High-Level Functional Block Diagram

Fig. 6.1 High-level functional block diagram of the PSI14,K+ coder.

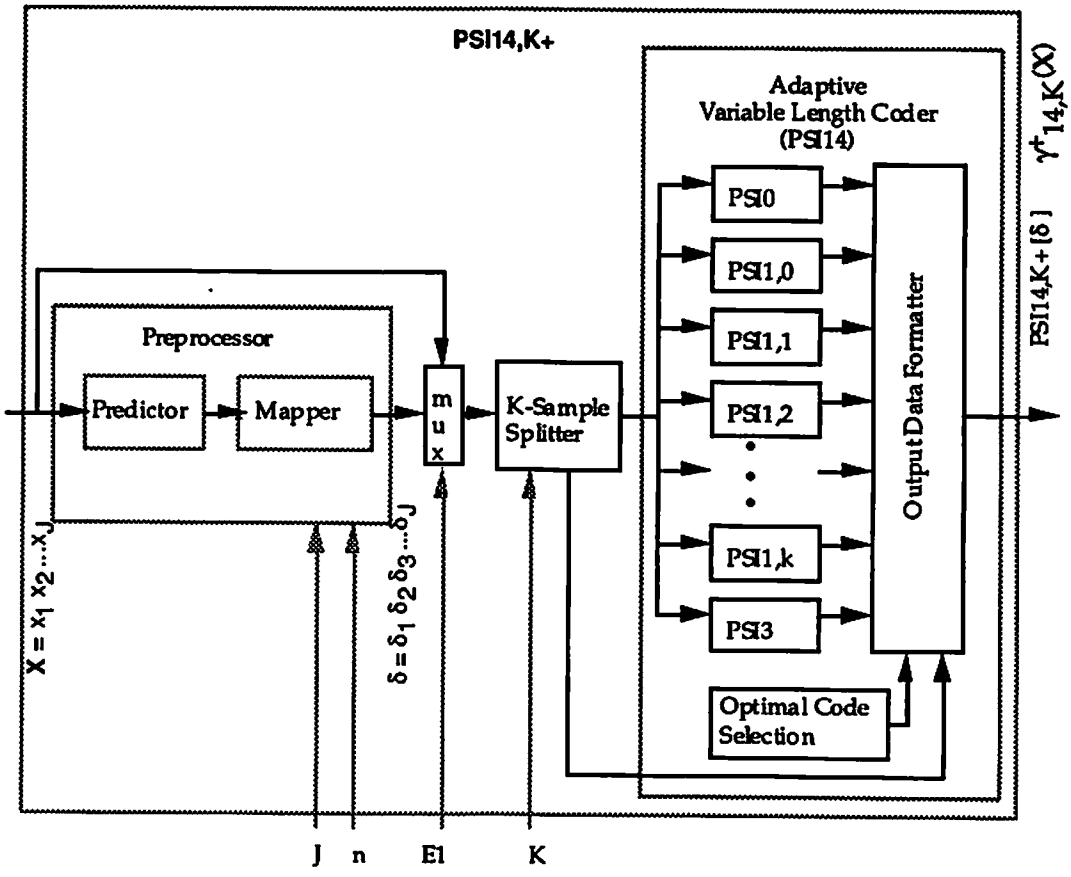


Fig. 6.2 An algorithmic structure of the UNC-PSI14,K+ coding.

the UNC-PSI14,K after processing an input line is a compressed bit string of data which is completely packed and there are no inter-block gaps between compressed bit strings for the blocks. The compressed bit string are read out by groups of 16 bits in parallel.

Figure 3 depicts the format for the output bit string from the UNC-PSI14,K+ coder. It consists mainly of a reference sample and a collection of compressed data blocks. The first n-bits of the compressed output bit string is the first n-bit sample value of the input data line and can be used as the reference sample. The compressed bit string for each data block consists of a 4-bit *ID code* and followed by either a backup block of J n-bit mapped differences or a *FS compressed bit string* appended with J (k-+K)-bit sub-strings, where k is determined by the selected code operator $PSI_{1,k}$. The first 4-bit field is a code ID which is transmitted as a header along with the compressed bit string for each J-sample input data block. The output of split sample *LSB* bits begins with the highest-order split bits and then followed by the lower-order split bits. If the Backup operator is selected, the output is a block of $J\,xn$ bits from the block of J mapped differences.

6.2.3 Performance Measurement

Entropy is a quantitative measure defined to represent the amount of information for a data set [6.19]. The less likely an event, the larger its information. Conversely, the more likely an event, the smaller its information. For a single event i with a probability of occurrence P_i, its entropy is defined to be

$$H_i = - \log_2 P_i \quad \text{(bits)}. \tag{6.14}$$

For a discrete memoryless source with k alphabets, the entropy is defined to be

$$H = \sum_{i=1}^{k} P_i H_i = - \sum_{i=1}^{k} P_i \log_2 P_i \quad \text{(bits)}. \tag{6.15}$$

Here P_i is the probability of symbol i occurring out of k alphabets. When logarithms to the base 2 are used in the entropy calculation, the resulting unit of information is a binary bit.

In source coding, the above-defined entropy is the lower bound on the average code length given memoryless source symbol statistics. Performance of lossless coding is measured by using the following features. Let P_i be the probability of the ith sample and L_i be the code length of the ith sample, where $0 \leq i \leq N\text{-}1$. The average code length is

$$L = \sum_{i} P_i L_i \tag{6.16}$$

The code efficiency is

$$E = \frac{H}{L} \tag{6.17}$$

The compression ratio is

$$CR = \frac{\log N}{L} \tag{6.18}$$

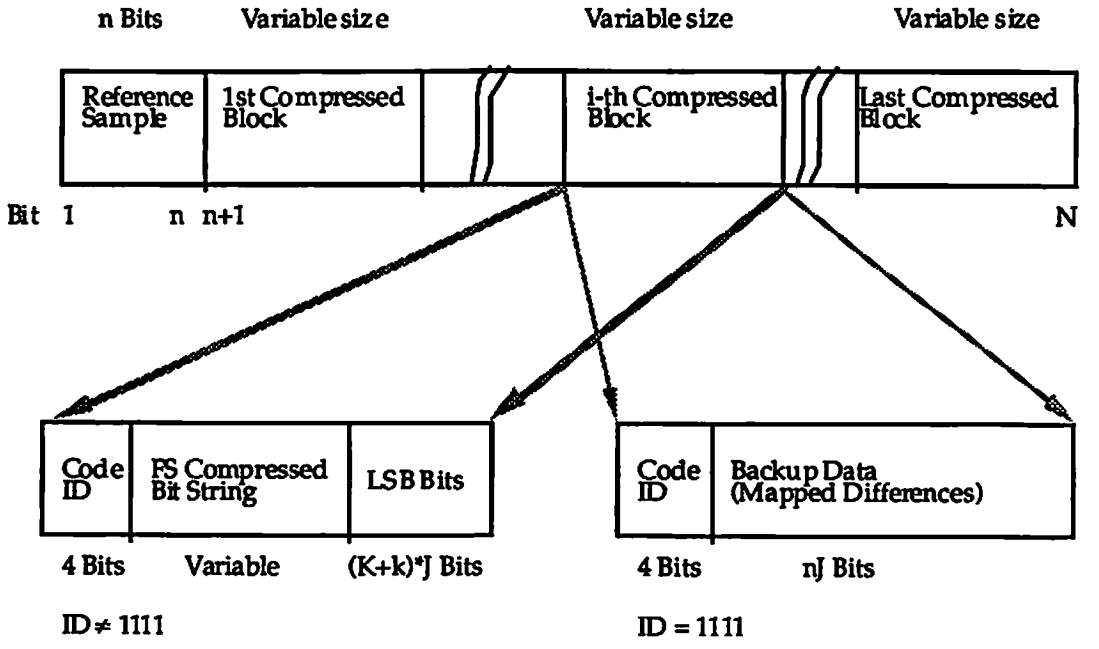


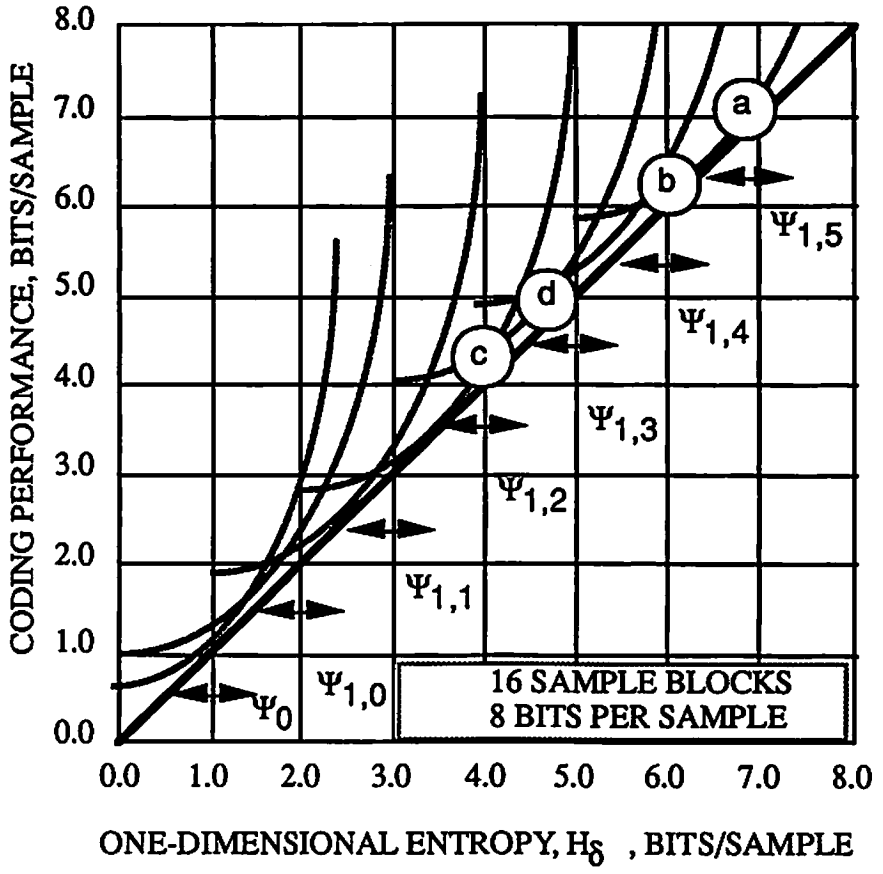
Fig. 6.3 The PSI14,K+ compression output format.

The maximum compression ratio for any lossless coding is

$$CR_{max} = \frac{\log N}{H}. \tag{6.19}$$

6.2.4 Experimental Results

The entropy and the bit rate of various images was calculated. The results are shown in Fig. 6.4. Depending on the types of image, the entropy appears in the range of 0 to bits/pixel. The differential entropy (the entropy of the difference image) was also calculated. Note that in the UNC-PSI14,K coding, the differences between consecutive data are coded instead of the original data values. Unlike the optical image (e.g. pepper), little reduction in the entropy was found for the synthetic aperture radar image (e.g. sea ice) The reason is due to the speckle noise. It is concluded that the Rice algorithm is an adaptive lossless coding scheme that provides near-optimal performance over a broad range of data. In addition, the universal noiseless coding algorithm would appear more advantageous for the smooth data but not for the noisy data such as the SAR imagery.



(a) SAR sea ice image

H=6.0, H_δ=6.8, L=6.9, CR =1.16

(b) SAR LAX image

H=6.5, H_δ=6.0, L=6.1, CR =1.31

(c) Pepper image

H=4.9, H_δ=4.0, L=4.1, CR =1.95

(d) Elaine image

H=6.1, H_δ=4.7, L=4.8, CR =1.67

Fig. 6.4 Performance of the UNC-PSI14,K+ coder.

6.3 Design and Implementation of the VLSI PSI14,K+ Encoder

6.3.1 System and Chip Partition

Figure 6.5 shows an end-to-end high rate data system diagram. To achieve a reliable high-speed communication/storage, the source and channel codec become an essential portions of the advanced high rate data system [6.20]. To design efficient hardware for the high rate data system, the PSI14,K+ encoder chip and its system-level hardware co-processor design have been developed by adopting a balanced software and hardware implementation. Figure 6.6 shows a system-level hardware architecture for the PSI14,K+ and its interfaces with other portions of the high rate compression system. A FIFO-based post-end interface buffer is used to alleviate the variable output rate issue of the adaptive lossless coding. The source encoder outputs the compressed samples whenever the compressed block is available. The internal word packer packs the encoded data and the auxiliary data into words. Then the co-processor collects the compressed data into the standard source packet for transmission or storage. Details of the VLSI coder design will be described as follows.

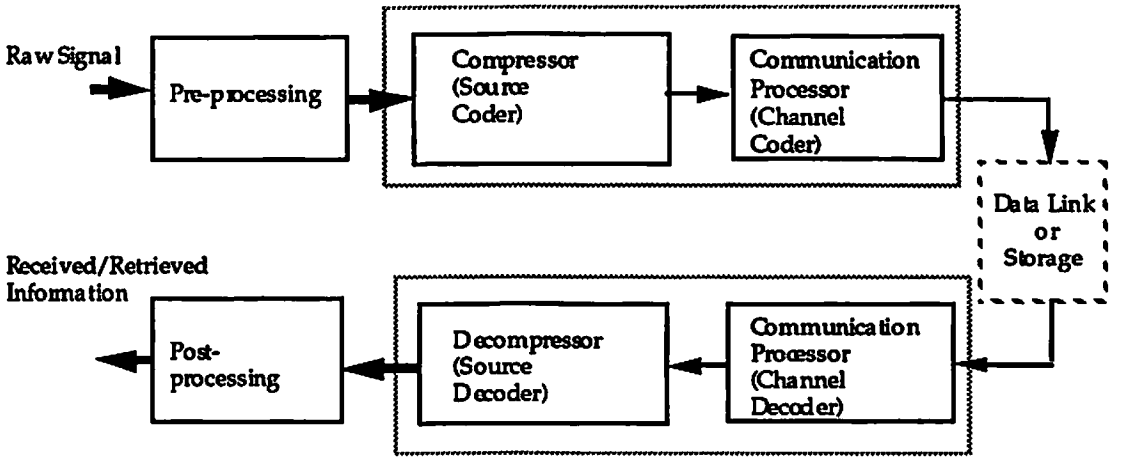


Fig. 6.5 An end-to-end high rate communication/storage system with source/channel coding .

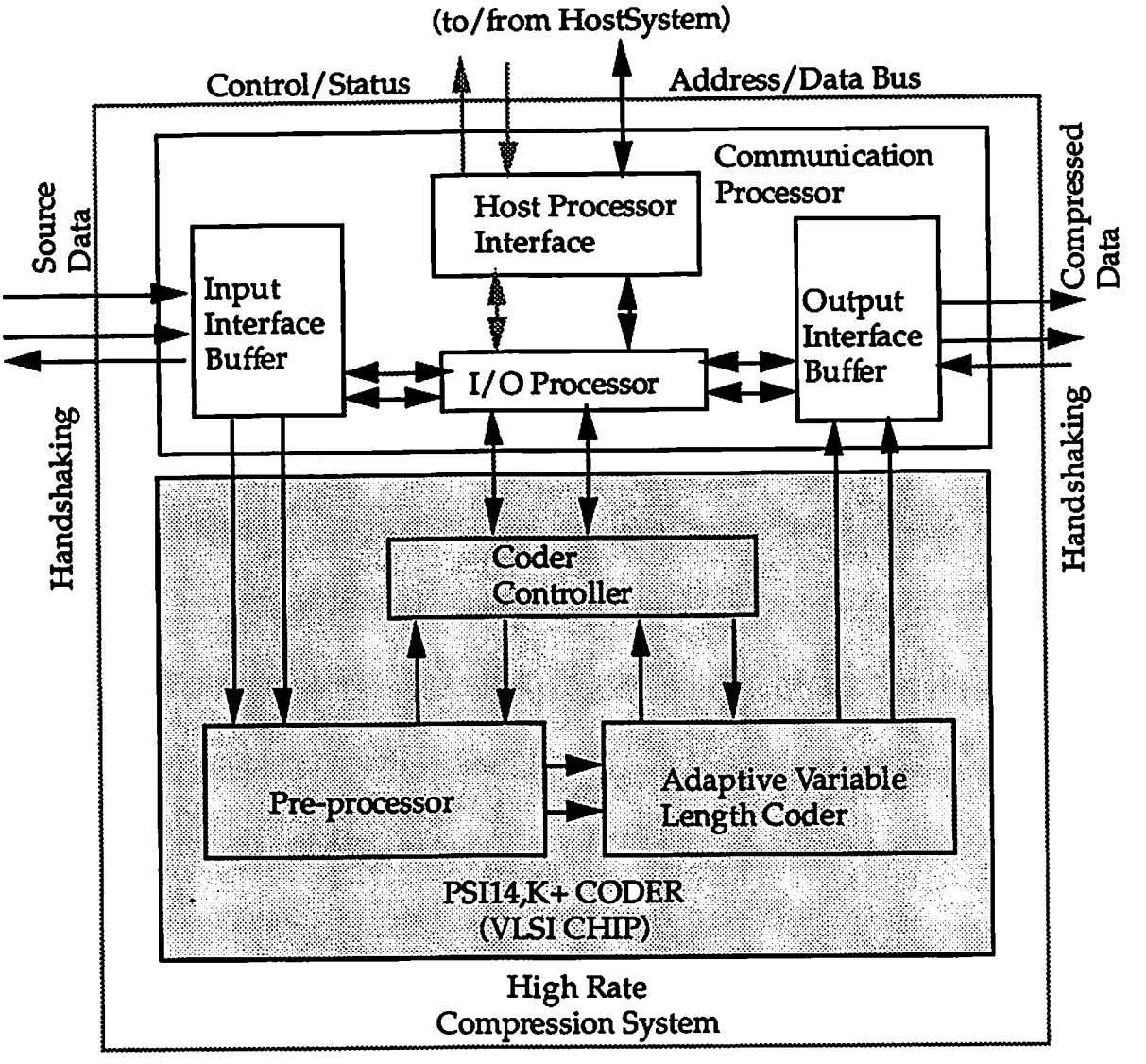


Fig. 6.6 A high rate compression system design.

6.3.2 Preprocessor Module

A functional design of the Preprocessor module is shown in Fig. 6.7. The sample predictor module performs the de-correlation function by taking differences between consecutive samples through the use of one delay element and one subtracter. The prior sample value is used as the predicted value. The prediction error is then defined as the difference between the current sample value and the predicted value. Since each sample value has n bits, the resulting difference is a $(n+1)$-bit number. For the first sample of the line, its prediction error is set to zero by using DIFZ to reset the Preprocessed Data Register. The Delta Mapper transforms the mapping function described in Eq. (6.2) by using a 12-bit data path design that is composed of four multiplexers, two adders, and one comparator. A physical layout design of the Preprocessor module is shown in Fig. 6.8. The silicon area is 1321 x 822 μm^2. The power dissipation is 17.7 mW at a 10 MHz clock rate.

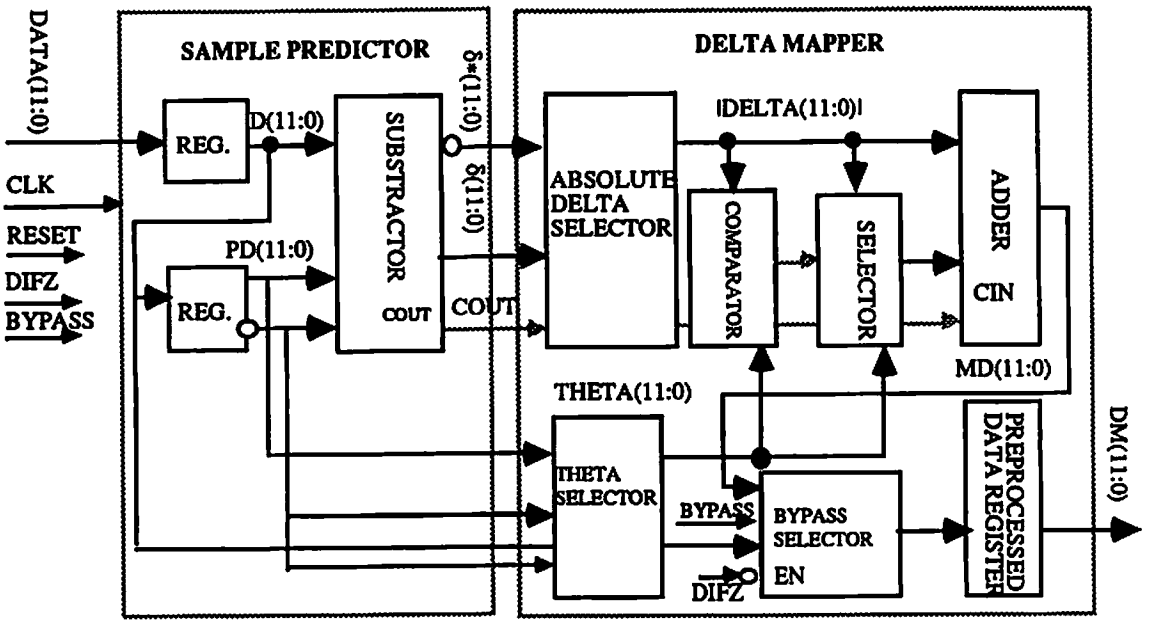


Fig. 6.7 Block diagrams for the Preprocessor module.

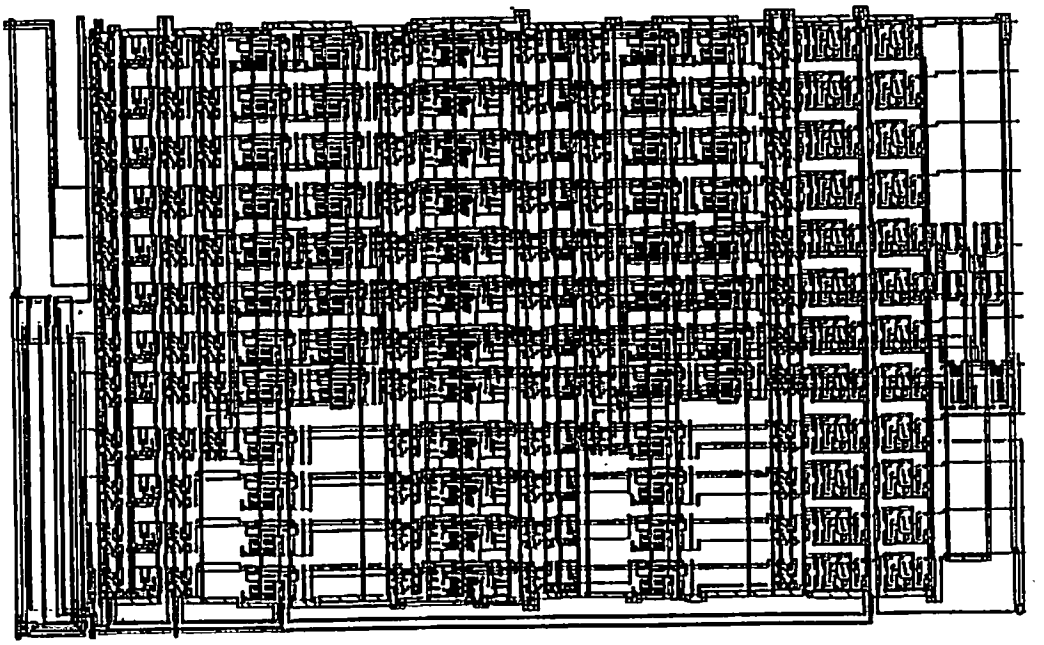


Fig. 6.8 A GDSII layout plot of the Preprocessor module. (1321 x 822 μm^2)

6.3.3 Adaptive Variable Length Coder

A functional design of the AVLC is shown in Fig. 6.9. The AVLC receives the string of unsigned integers from the preprocessor and performs coding to reduce its data rate on a block-by-block basis. Each block is made up of 16 unsigned integers. For each input block, an option is chosen out of 12 available code operators, which include the Backup operator, the Fundamental Sequence (FS) operator, and the FS with k-bit sample splitting (SS), where $1 \leq k \leq 9$. With the chosen option, the AVLC proceeds to perform coding on the input block. The resulting compressed bit string with its code ID are then sent to the data packer module for assembly.

A brief description of each building block is as follows:

Block Buffer. The Block Buffer module temporarily stores the current 16-sample block to enable block-pipelined operation of the AVLC. At the end of an input line, if less than 16 values remain, zeros will be appended to fill the block. The resultant block will then be processed like other blocks. It is built with 16 12-bit shift registers. The silicon area is 1554 x 828 μm^2. The power dissipation is 8.7 mW at a 10 MHz clock rate.

Code Selector. The *Fo summer* is used to sum up all the unsigned integers in $\tilde{\delta}^n$. The set of decision regions as described in [6.4] is realized with a random logic decoder. The silicon area is 728 x 1033 μm^2. The power dissipation is 3.5 mW.

Code Operators. The Fundamental Sequence (FS) Generator generates a codeword for each entry value of $\tilde{\delta}^n$ and records it by using a biased accumulator to track the "single one" position of each entry in a FS sequence

memory. The k-bit Sample Splitting and the mid-step K-bit Sample Splitting are implemented with *right-logical barrel shifters.* The bit length of each block is calculated by the same biased accumulator for generating FS sequence. The PSI0 is implemented in a ROM table-look-up. Note that the twelve code options are fully implemented with only three hardware coding blocks. The silicon area of the Code Operator is 1097 x 1553 μm^2. The power dissipation is 8.8 mW at a 10 MHz clock rate.

FS Packer. The FS Packer prepares 16-bit words for Formatter to pick up from FS sequence with ID bits. Its silicon area is 1030 x 929 μm^2. The power dissipation is 6.1 mW.

SS Packer. The SS Packer buffers and packs k lowest-order bits of the 16 split samples into k 16-bit words for the Formatter module. It can be implemented in a silicon area of 2177 x 2747 μm^2 with a power dissipation of 24.8 mW.

Backup Packer. If the backup code operator option is selected, Backup Packer module buffers and packs the 16 mapped differences into 16-bit words for the Formatter. The Backup Packer can be implemented in a silicon area of 1650 x 2964 μm^2 with a power dissipation of 2.2 mW.

Formatter. The final output from the UNC chip after processing an input line is a compressed bit string of data . The Formatter completely concatenates and reads out the packed words from FS Packer, SS Packer, or Backup Packer modules to ensure that there are no inter-block gaps between compressed bit strings. It can be built with barrel shifters and control logic gates. The silicon area is 955 x 1908 μm^2. The power dissipation is 6.3 mW.

Controller. The on-chip UNC controller is a simple finite state machine that is implemented with a couple of D-type or JK-type flip-flops and

combinatorial gates. The silicon area is 962 x 920 μm^2. The power dissipation is 4 mW.

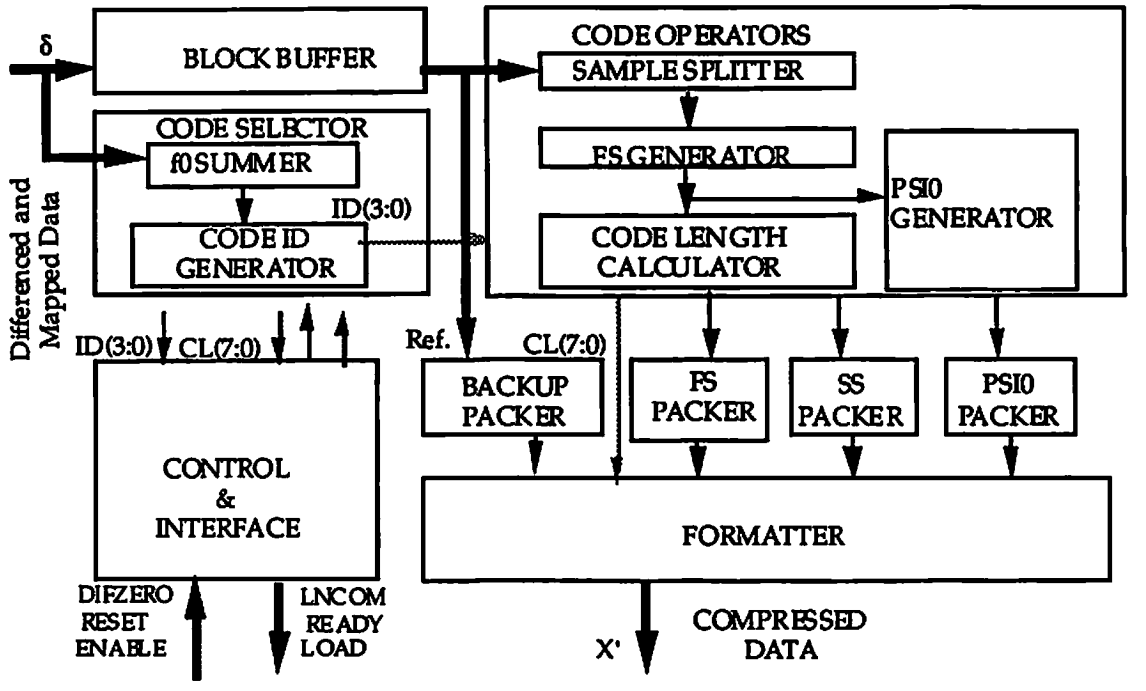


Fig. 6.9 Functional Block Design of the PSI14 adaptive variable length coder.

6.4 The Prototype Chip

A VLSI layout design of the UNC-PSI14,K chip has been done by using a 1-μm CMOS technology. It occupies a compact chip area of 5.1 x 5.3 mm^2, with 49,000 transistors, 57 input/output pads, and 6 power/ground pads. It is designed to provide a high throughput rate up to 480 MBit/sec at 40 MHz system clock. The chip information is summarized in Table 6.1. A GDS-II layout of the 1-μm UNC-PSI14,K chip is shown in Fig. 6.10.

Salient features of the VLSI UNC-PSI14,K compressor design are:

• High data rate and low power: A single UNC-PSI14,K chip provides a throughput rate of 480 MBit/second with a worst case power dissipation of less than 0.4 watt by using a 1-micron CMOS technology.

• Algorithm-specific VLSI design: The algorithm-specific VLSI design method is used to achieve an efficient system performance. The highly pipelined data path and modular coder options can be effective implemented in VLSI.

• Compatibility with lossy compressors: It can be cascaded with any high-ratio lossy compressor to further increase the compression ratio.

• Expandability to array compression system: Due to the inherent nature of UNC-PSI14,K line-based compression algorithm, an array compressor system offers ultra high-speed data processing through a parallel array architecture using multiple UNC-PSI14,K compressor chips.

• Self-adaptive scheme: It uses an adaptive approach to compress different data contents by allowing a subset of code operators to be selected.

• Near-optimized compression ratio: The compressed data entropy is very close to the entropy of the data source.

• Fixed-line error containment: UNC-PSI14,K has a fixed line format and thus limiting error propagation.

6.5 Conclusion

An efficient VLSI pipelined processor design for high-speed lossless compression based on the Rice's UNC-PSI14,K+ algorithm has been developed for high performance data communication and storage

applications. The Rice algorithm is an adaptive lossless coding scheme that provides near-optimal performance over a broad range of data entropies. The Rice algorithm is also an efficiently implementable scheme for VLSI systems. The VLSI pipelined architecture was designed to allow compact implementation in a single-chip VLSI compressor. The chip layout was generated for a 1.0-micron CMOS technology. It occupies a compact chip area of 5.1 x 5.3 mm^2, and operates at a 40 MHz system clock. The potential applications of the proposed lossless compressor include database management systems, scientific instruments, CAE workstations, desktop computing machines, and the data systems that require high-speed compression without fidelity loss.

Table 6.1. Characteristics of the UNC- PSI14,K+ Chip

Table 1. Chip information

Chip Name	UNC compressor chip
Design Method	Custom design using compiled cell
Process Technology	1.0 micron CMOS
Die Size	5.1 x 5.3 microns
Total # of Device	50,000
Total # of Cells	3224
No. of Pads	63
Package	68-pin PGA

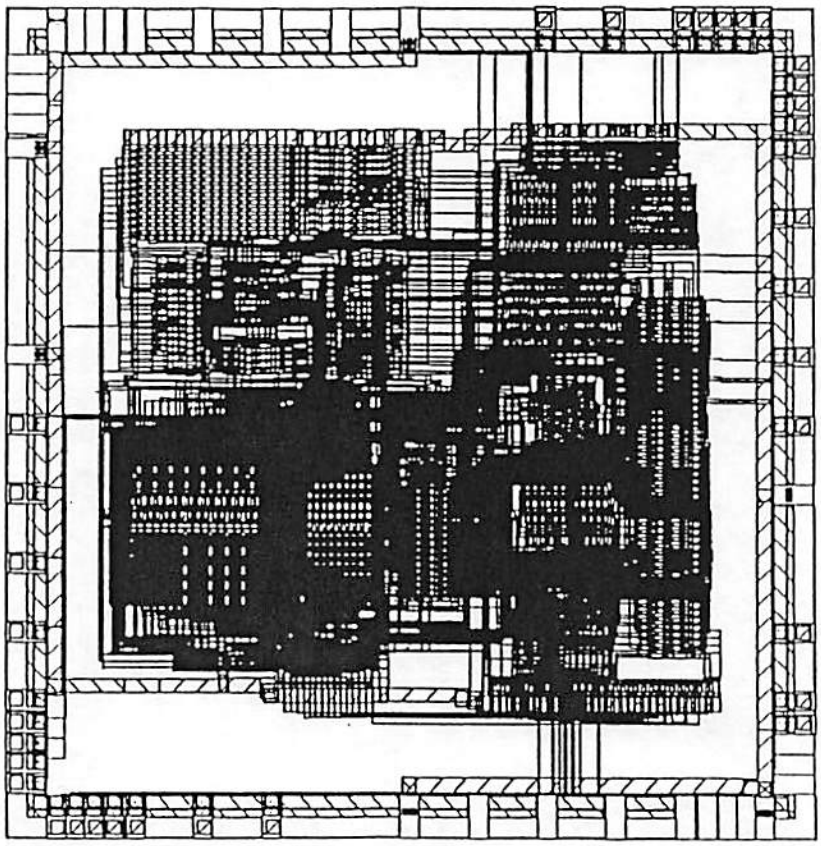


Fig. 6.10 A GDS-II layout plot of the 1-mm UNC-PSI14,K+ design.

References

[6.1] R. F. Rice, J. R. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data," IEEE Trans. on Comm. Tech. vol. COM-19, no. 6, pp. 889-897, Dec. 1971.

[6.2] R. F. Rice, " Some Practical Universal Noiseless Coding Techniques," JPL Publication 79-22, Jet Propulsion Laboratory, Pasadena, California, Mar. 15, 1979.

[6.3] R. F. Rice, J.-J Lee, " Some Practical Universal Noiseless Coding Techniques, Part II," JPL Publication 83-17, Jet Propulsion Laboratory, Pasadena, California, Mar. 1, 1983.

[6.4] R. F. Rice, "Practical Universal Noiseless Coding Techniques," Part III, JPL Publication 91-3, Jet Propulsion Laboratory, Pasadena, California, Nov. 15, 1991.

[6.5] R. F. Rice, E. Hilbert, J.-J Lee, A. Schlutsmeyer, "Block Adaptive Rate Controlled Image Data Compression," Proceeding of the 1979 National Telecommunications Conference, Washington, D. C., Nov. 1979

[6.6] R. F. Rice and A. Schlutsmeyer., "Data compression for NOAA weather satellite systems," SPIE Proc: Advances in Image Transmission II, vol. 249 1980.,

[6.7] R. F. Rice and J.-J. Lee, "Noiseless coding for the Gamma ray spectrometer," JPL Publication 85-53, Jet Propulsion Laboratory Pasadena, California, 1986.

[6.8] R. F. Rice and J.-J. Lee, "Noiseless coding for the magnetometer," JPL Publication, 87-19, Jet Propulsion Laboratory, Pasadena, California, 1987.

[6.9] R. F. Rice, J.-J Lee, W.-C. Fang, "Data Compression for an Unmanned Aerial Vehicle," JPL Internal Document D-4397, Jet Propulsion Laboratory, Pasadena, California, April 1987.

[6.10] J.-J. Lee, W.-C. Fang, R. F. Rice, "Real-time Data Compressor for Eos-class Missions," Proceeding of the 1987 IEEE Aerospace EASCON Conference, Washington, D. C., Oct. 1987.

[6.11] J. -J. Lee, W.-C. Fang, R. F. Rice, "VLSI Universal Noiseless Compressor for Eos-class Missions," Proceeding of the 1988 SPIE Conference, Los Angeles, California, Feb. 1988.

[6.12] J.-J. Lee, W.-C. Fang, et al "A Very High Speed Noiseless Data Compression Chip for Space Imaging Applications", Proceedings IEEE Data Compression Conference, Snowbird, Utah, April 1991.

[6.13] J. Venbrux and N. Liu, "A Very High Speed Lossless Compression/ Decompression Chip Set," Proceedings IEEE Data Compression Conference, Snowbird, Utah, April 1991.

[6.14] R. F. Rice, P.-S. Yeh, and W. Miller "Algorithms for a Very High Speed Universal Noiseless Coding Module", JPL Publication 91-1, Jet Propulsion Laboratory, Pasadena, California, February 15, 1991.

[6.15] P.-S. Yeh, et al., "On the Optimality of Code Options for a Universal Noiseless Coder," JPL Publication 91-2, Jet Propulsion Laboratory, Pasadena, California, February 15, 1991.

[6.16] D. A. Huffman, "A method for the construction of minimum redundancy codes," Poc., IRE vol. 40, pp. 1098-1101,1952.

[6.17] R. C. Gailager "Variations on a theme by Huffman," IEEE Trans. on Information, Theory, Vol. 1T-24, No. 6, pp. 668-674, 1978.

[6.18] P. A. Humblet "optimal source coding for a class of integer alphabets," IEEE Trans., of Inf. Theory, Vol. 1T-24, No. 1, pp., 110-112, 1978.

[6.19] C. Shannon, "A Mathematical Theory of Communication, "Bell Syst. Tech. J 21, pp. . 397 423, July 1948.

[6.20] R. B. Miller, D. A. Nichols, "High Rate Data Systems," AIAA/NASA International Symposium on Space Information Systems in the Space Station Era, Washington, D. C., June 1987.

Chapter 7

Future Work

Future research directions includes the following recommendations:

• The application of FSO network to speech coding, pattern recognition, sequence classification, multi-media transmission or storage systems, etc. Among various neural network paradigms, the self-organization network has the desirable property of effectively structured presentation of features of the source signals. Self organization is required in several image and vision processing applications such as image compression, pattern recognition, pattern mapping, motion estimation, sequence classification, etc. Although software simulations of these neural network paradigms can be performed. The software implementation usually take too long to be practical for real applications. Advances of effective hardware implementation of the neural network can overcome this problem. The development of FSO neuroprocessor is a significant success in the VLSI artificial neural network research. Work should be done to apply the existing or modified FSO neuroprocessor to other applications which are based on the same self-organization neural network paradigm.

• The implementation of the proposed adaptive image compression system by using 0.5 μm VLSI technology.

The standard IC technology will be ultra-dense sub-micron multilevel-interconnection CMOS in the mid 1990's. The power dissipation density will be much higher due to the multi-level interconnection and the vertically built circuitry. Ultra low power dissipation density is required for the high density of circuit elements. The traditional analog circuitry requires relatively large area and power due to the relatively large voltage supply and swings. Low voltage operation on the order of 1 volt or less will be required for the sub-micron digital circuitry due to the geometries and will be desired for the sub-micro analog circuitry due to the severely limited breakdown voltages. A analog-digital mixed circuitry are essential to interface with an analog world and also to integrate maximum amount of circuitry onto a single chip for an efficient system implementation. Computation efficiency of the analog circuit functions is desirable. For example, the multiplication can be perform in a high density low power analog multiplier instead of a complicate digital multiplier. The accuracy of the analog circuitry is often sufficient for the system applications of the analog world.

• The use of VQ in combination with analog sensor.

The current approach using a uniform analog-to-digital converter followed by lossy data compression is suboptimal. An advanced approach would be convert the analog signal into vector codes by employing vector quantization design techniques to tailor a better source code to represent the characteristic of source data. Flexible and adaptable signal representation schemes will become more and more important since the nature of the signals to be represented for computing includes more and more physical-world signals

such as sound, image, etc. which tend to be more hard to characterize analytically.

• The new data compression approaches by combining source and channel encoding.

In general, compressed data has increased sensitivity to noise or transmission errors. The sample values within a block of compressed data may be all invalidated by the error introduced by the communication channel. More stringent communication quality and continuity requirements for transported data must be applied. For planetary missions, typically the end-to-end bit error rate requirement on compressed data is 1×10^{-6} whereas for uncompressed data it is 1×10^{-3}. It is essential to use Reed-Solomon encoding on the downlink channel to minimize the effect of noise on the quality of compressed data.

• The optimal integration of error containment and error correction with data compression.

Vector quantization sends compressed data in fixed sized blocks, thus limiting error propagation. Rice algorithm has a fixed line format that provides error containment. Work should be done to look for ways in which data compression could improve error containment behavior and aid error detection and correction.

Appendix A

Journal and Conference Publications Out of the Dissertation Work

A.1 Journal Publications

[1] W.-C Fang, B. J. Sheu, O. T.-C. Chen, J. Choi "A VLSI neural processor for image data compression using self-organization networks," *IEEE Trans. on Neural Networks: Special Issue on Neural Network Hardware,* vol. 3, no. 3, May 1992.

[2] J.-C. Lee, B. J. Sheu, W.-C. Fang, " VLSI neuroprocessor for video motion detection" *IEEE Trans. on Neural Network, 1992* (accepted, to be published).

[3] O. T.-C. Chen, B. J. Sheu, W.-C. Fang, " Image compression on a VLSI neural-based vector quantizer," *Information Processing and Management,,* 1992. (accepted, to be published).

[4] W.-C. Fang, C.-Y. Chang, B. J. Sheu, "Systolic tree-searched vector quantizer for real-time image compression," *VLSI Signal Processing, IV,* H. S. Moscovitz, Kung Yao, R. Jain (Eds.), pp. 352-361 (Chap. 34), IEEE PRESS: Piscataway, NJ, 1991.

A.2 Conference and Workshop Publications

[1] W.-C. Fang, B. J. Sheu, "Real-time computing of optical flow using adaptive VLSI neuroprocessors," *IEEE International Conference of Computer Design*, pp. 122-125, Cambridge, MA, Oct. 1990.

[2] W.-C. Fang, B. J. Sheu, " VLSI adaptive image compression," *Advanced Research in VLSI: Proceedings of the 1991 University of California/Santa Cruz Conferences*, Carlo H. Sequin (Ed.), pp. 371-386, MIT Press, Cambridge, MA, 1991,

[3] W.-C. Fang, B. J. Sheu, O. T.-C. Chen "A neural network based VLSI vector quantizer for real-time image compression," *IEEE Proceedings on the first Data Compression Conference*, Snowbird Utah, pp. 342-351, IEEE Computer Society Press: Los Alamitos, Apr., 1991.

[4] B. J. Sheu, W.-C. Fang "Real-time high-ratio image compression using adaptive VLSI neuroprocessor," *IEEE Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. Toronto, Ontario, Canada, May, 1991.

[5] W.-C. Fang, B. J. Sheu, J.-C. Lee, "A Neuroprocessor for real-time image flow computation," *IEEE Proceedings of International Conference on Acoustics, Speech and Signal Processing*, pp. 2413-2416, Toronto, Ontario, Canada, May, 1991.

[6] O. T.-C. Chen, B. J. Sheu, W.-C. Fang, " Adaptive codebook construction and real-time hardware implementation for binary tree-searched vector quantizer," *IEEE Workshop on Visual Signal Processing and Communications*, pp. 35-38, Taiwan, Republic of China, June 1991.

[7] W.-C. Fang, B. J. Sheu, O. T.-C. Chen, " A real-time VLSI neuroprocessor for adaptive image compression based upon frequency-sensitive competitive learning," *International Joint Conference on Neural Networks*, pp. 429-435, Seattle, WA, July, 1991.