USC–SIPI REPORT #292

Architecture and Simulation of Selected Fine-Grained VLSI Array Processors

by

Tony H.-Y. Wu

November 1995

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

To My Parents,

Wen-Jyh Wu

and

Su-Chin Hung,

for their love and support.

Acknowledgments

I would like to express my deepest gratitude to my research advisor, Professor Bing Sheu, for his generous guidance, encouragement, and support throughout these years of graduate study. I am also grateful to Professor Theodore W. Berger and Professor Edward K. Blum for serving on my dissertation committee. I would like to thank Professor George A. Bekey and Prof. Vasilis Z. Marmarelis to serve as the other two members on my Ph.D. Qualifying Examination Committee.

I am very grateful to Professor Leonard Silverman, Dean of the Engineering School; Professor Hans H. Kuehl, Chairman of the Electrical Engineering - Electrophysics Department; Professor Robert A. Scholtz, Chairman of the Electrical Engineering - Systems Department; Ms. Ramona Gordon, Ms. Anna Fong, Ms. Evelyn Jamora, and Ms. Gloria Halfacre in the Electrical Engineering Program, for providing such a great research environment for my Ph.D. study at the University of Southern California. Support from several research organizations including the Signal and Image Processing Institute (SIPI), the Center for Neural Engineering (CNE), the Integrated Media Systems Center (IMSC), and the MOSIS Service of Information Science Institute (ISI) is also highly appreciated.

Valuable discussions with graduated doctoral colleagues from VLSI Signal Processing Laboratory were truly stimulating, including Dr. Robert C. Chang and Dr. Joongho Choi on VLSI array processing chips design, Dr. Sa H. Bang for the cellular neural network annealing theory, Dr. Josephine C.-F. Chang on digital VLSI design,

and Dr. Oscal T.-C. Chen on image and information processing. Many thanks to Eric Y. Chou, Steve H. Jen, Richard H. Tsai, David C. Chen, Michelle Y. Wang for friendly interaction.

After studying in the SIPI for one and half year, I am very glad to have the chance to serve as the teaching assistant for the courses related to the VLSI signal/image/video processing and VLSI neural networks. The experience really helps me a lot, such as the opportunity of internship in Summer 1995 at AT&T Bell Laboratory, Holmdel, New Jersey. I would like to thank all the fellow colleagues in the VLSI System Research Department, especially Dr. Horng-Dar Lin.

Finally, I would like to dedicate this work to my parents, Wen-Jyh Wu and Su-Chin Hung, my sister, Wang-Ting Wu, my brother, Liang-Wei Wu, for their love, understanding, patience and support during my doctoral studies.

The research was partially supported by ONR under contracts N00014-94-I-0568 and AT&T Bell Laboratory's University Relationship Program.

Contents

List Of Tables

List Of Figures

Abstract

With rapid advances of semiconductor manufacturing technologies and the mature computer-aided design tools, the trend of moving from serial signal processing to parallel processing follows a natural progress. The constraint on VLSI implementation in circuit complexity can be alleviated by the use of regular, repetitive architectural structures. Scalability and massive parallelism provide the enormous throughput rate and processing capability that conventional sequential processors cannot achieve.

An analog compact neural network model is reviewed. It is a powerful parallel processing paradigm consisting of densely-connected analog computing cells. Various applications, such as edge detection, hole-filling, and connected-component detection, can be accomplished by changing the local interconnection strengths, which are programmed through the coefficient templates.

The behavioral simulator *cnna*, which reads in the configuration information and simulates the dynamic behavior of the network, is presented. An unique feature of this simulator is the hardware annealing capability which provides an efficient method of finding globally optimal solutions. Effects of hardware annealing and the different bias setting are presented. Performance comparisons with respect to image sizes, and template sizes have been summarized. A comparison with simulators developed by other researchers has also been included.

The parallel computation architecture is defined. A compiler and its associated programming language greatly facilitate the simulation of compact neural networks in order to optimize the design parameters. The framework of the whole simulation environment is also presented.

An innovative architectural mapping based on the multiplication-accumulation techniques which is similar to distributed arithmetic is described. The array processor is most suitable for high performance requirement of 2-dimensional forward and inverse discrete cosine transformation tasks. Each computing cell contains two adders: one for 2-bit multiplication and the other one for partial-sum accumulation. The computation power could reach 10 giga-operations-per-second at a system clock 100 MHz by using a $0.5\mu m$ CMOS manufacturing technology. The accuracy tests of the inverse discrete cosine transformation have been verified and comparison with other researchers' work is summarized. These accomplished results lay a critical foundation for the increasingly important multimedia applications, especially in image, video, and speech processing.

Chapter 1

Introduction

1.1 Signal Processing and VLSI

During the past decade, there has been a dramatic efforts on mapping various signal/image processing applications onto very large-scale integration (VLSI) architectures. Many researchers have been contributing their efforts and making significant impacts from different levels including algorithms, architectures, and detailed VLSI implementations. Rapid advance of semiconductor technologies and mature computer-aided design (CAD) tools keep pushing this fast-growing field to a higher level.

However there are applications which demand such tremendous computation power that the conventional methods can not reliably provide. In 1992, National Science Foundation published a booklet describing the *Grand Challenging: High-Performance Computing and Communications* [1]. It calls for a thousand-fold improvement in computing capability, and a hundred-fold improvement in computer communication capability. Figure 1.1 shows a plot of many mission needs mentioned in *Grand Challenge*. The goal is to achieve one tera operations per second on a wide range of applications, and a communication network capable of one billion bits per second.

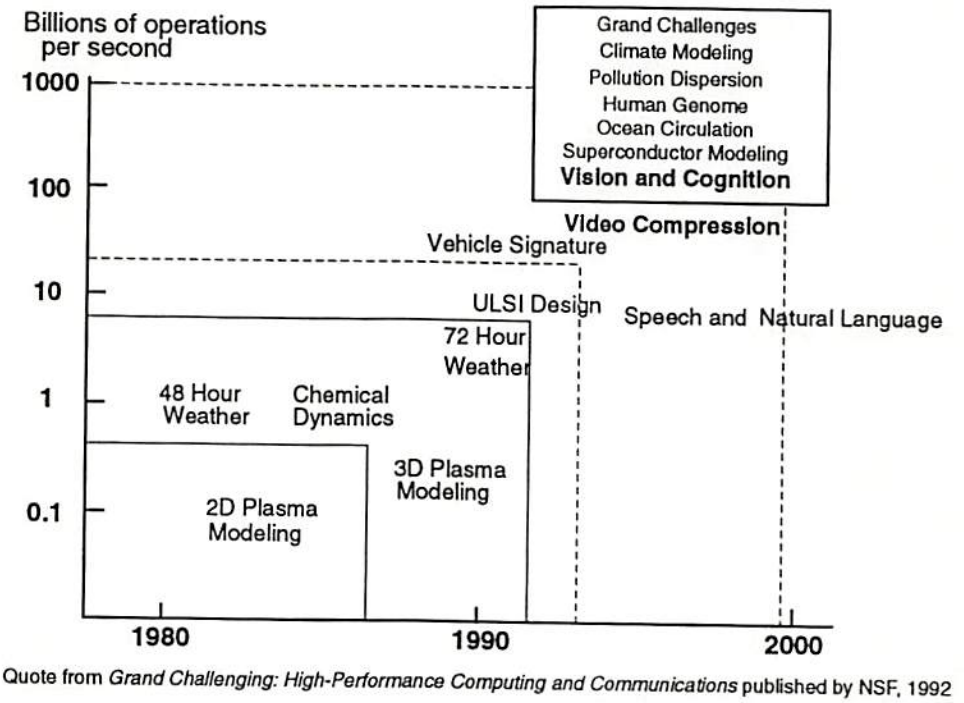


Figure 1.1: Grand challenge which calls for improvement for computing and communication capability [1].

For example, the neural network research attempts to learn from the nature's success and to mimic some of the nature's tricks in order to accomplish intelligent information processing tasks not easily performed by the conventional methods. Neural networks are composed of massively parallel architectures that process large quantity of information in continuous values, and solve varieties of ill-defined and/or computation-intensive signal processing tasks.

When embedded in microelectronic hardware implementation, neural networks exhibit high degree of fault tolerance to system damage and also high data throughput rate due to paralleled data processing. Microelectronic and optical implementations of neural network chips will make it possible to insert low-cost modules into

existing and newly developed systems, and facilitate improved performance in applications such as pattern recognition, noise filtering, cluster detection, process control and adaptive control.

Another evident example comes from video signal processing. Multimedia is one of the most powerful forms of communication ideas which incorporate every type of media ever developed [2]. It continues to creep into our lives as entertainment and information become available in digital formats. But the huge amount of information of uncompressed digital video are not only far more beyond what the state-of-art microprocessor or digital signal processor can handle but also waste the precious bandwidth even it is feasible in the future.

There are certain spatio redundancy in each video frame and temporal redundancy among the video sequences. Various standards, such as JPEG [3], MPEG standards [4] or CCITT H. 261 [5], which help to reduce those redundancies were introduced to pave a clear way for the multimedia applications. However the enormous demands on the processing power and memory access continue to push the vast improvement in both architectural innovation and VLSI implementation for a cost-effective implementation.

For the signal processing applications which require the range of tera-flop operations computing capability, even the general-purpose parallel computers can not offer satisfactory processing speed or fail the real-time requirement. At present, special-purpose array processors will become the only appealing alternative [6].

1.2 Why Array Processing?

The trend of moving from serial processing to massively parallel processing follows a natural progress. The brain cells work cooperatively so that an animal can run

and catch, see and hear at the same time. Although the detailed information of brain operation still remains a puzzle to be solved by the scientists, the knowledge that has been accumulated through the biological neural network research does give good clues toward the construction of a new-generation parallel processing machine in the near future. And the attempt to duplicate the functions help us toward the way of developing future intelligent machines.

The constraint on VLSI implementation in circuit complexity can be alleviated by the use of the regular, repetitive architectural structures. Therefore the array processing, which is a special format of parallel processing, has the advantages on the modularity, regularity, efficient communication, scalability, massive parallelism, and minimized I/O [6]. The small number of building blocks can ease the burden on the circuits design, simulation and verification. Uniform communication will keep the layout simple and reduce the overhead with the saving of wiring. Scalability and massive parallelism provide the enormous throughput rate and processing capability that conventional sequential processors cannot achieve.

Array processing is not without any limitation. First, the global synchronization driven by a global clock is the simplest way for small size of arrays. But it will be affected by clock skew and cause unnecessary slowdown in the clock rate for a large system. Another concern is that each processing element will possess limited functions or programmability for compactness purpose. Not every algorithm can be effectively mapped onto the array structure, which is related to the complexity of each processing element (PE). In this dissertation, both array processors described will share the SIMD (single-instruction-multiple-data) [7] style of design, which requires a global clock to synchronize the operation and requests each processing element working for the same function.

1.3 VLSI System Trend

Rapid progresses in VLSI electronic and micro-mechanical technologies have made possible the implementation of complex data and signal processing functions on a single silicon chip, which now contain multi-million transistors. The use of VLSI circuits can greatly reduce the physical size and enhance the performance and reliability of microelectronic systems.

The trend of VLSI is judged according to the feature size, number of transistors per chip, silicon area per chip and the clock rate. The trends are plotted in Figure 1.2-1.5. The solid-line is the best fit of all data points. The dashed line represents the best fit to the smallest feature size and gives us the most optimistic prediction. The time period shown is between 1985 and 2005.

Feature size is related to the minimum channel length, which is an important indicator showing how small each transistor can be. As the size shrinks, more and more transistors can be placed on a single chip with the same area and provide higher performance and more complex functions. At the same time, the clock speed could have been increased at almost 5 times every decade according to the scaling effect [8]. Beyond the year 2000, with the help of new packaging technologies such as multi-chip modules (MCM) and optical interconnection, operation and transmission can reach the rate near or above 1 GHz.

Power consumption has become a problem when the sizes of the chips increase. Higher clocking rate, which also means higher switching rate, will also cause heat problems. Since we often rely on many processing elements in the array processor, a highly compact cell with low power consumption is very much necessary in the design requirement. Another issue to be taken into consideration is the interconnection

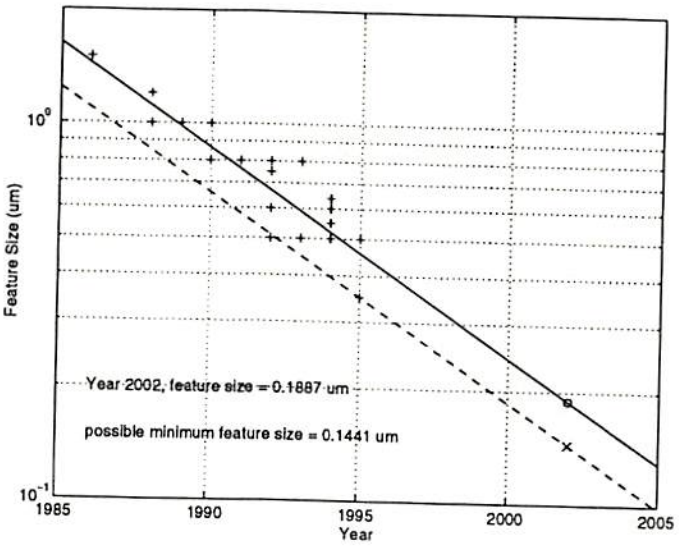


Figure 1.2: VLSI trend: Feature size.

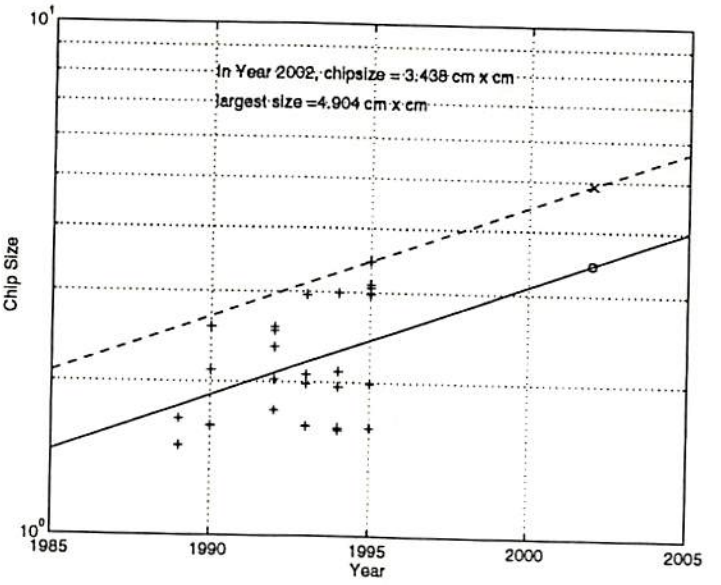


Figure 1.3: VLSI trend: Chip size.

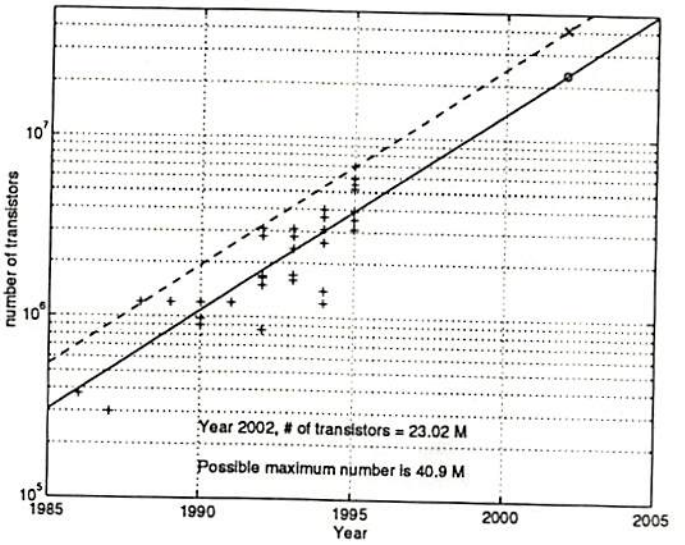


Figure 1.4: VLSI trend: Integration level.

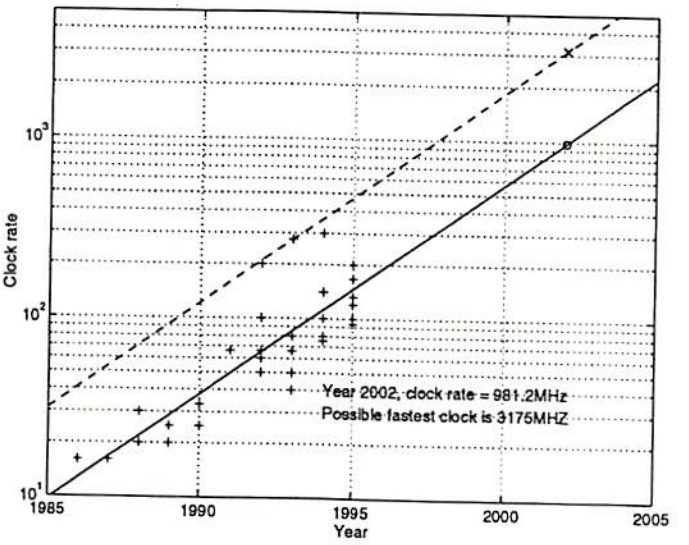


Figure 1.5: VLSI trend: Clock rate.

among the chips. Electronic connection has its 2-D plane limitation and optical 3-D transmission/receiving would be a promising choice in the future.

The data are collected from H. B. Bakoglu's book [9] and results reported in IEEE ISSCC conference, IEEE Micro magazines and press releases from the chip vendors over the years. A complete table is listed in Appendix A.

1.4 Design Methodology of VLSI Systems

Before we identify an application and start to build a VLSI system, computer software simulation plays a very important role. At the beginning high-level programming language, such as general-purpose C language or mathematics-oriented MATLAB programs can be used to simulate the behavior of the entire system. Various algorithm implementations under different architectures and dataflows are the design factors to be reviewed and compared at this stage. The design flow of VLSI system is shown in Figure 1.6.

Nowadays the hardware description languages, such as VHDL or Verilog, emerge as the bridge between the high-level simulation and the detailed circuit simulation by providing different architecture structures. Designers can change the architecture descriptions according to the design schedule, which could be either high-level behavioral models or detailed low-level circuits. It helps the designers to choose an appropriate and effective solution before the final architecture is set and detailed circuit design starts.

In actual electronic implementation, the choices lie within either digital, analog, or hybrid approaches. Digital design is based on well-established and reliable techniques. It can easily realize cost-effective ASIC chip with high performance as compared with software simulation on sequential computers with the help of

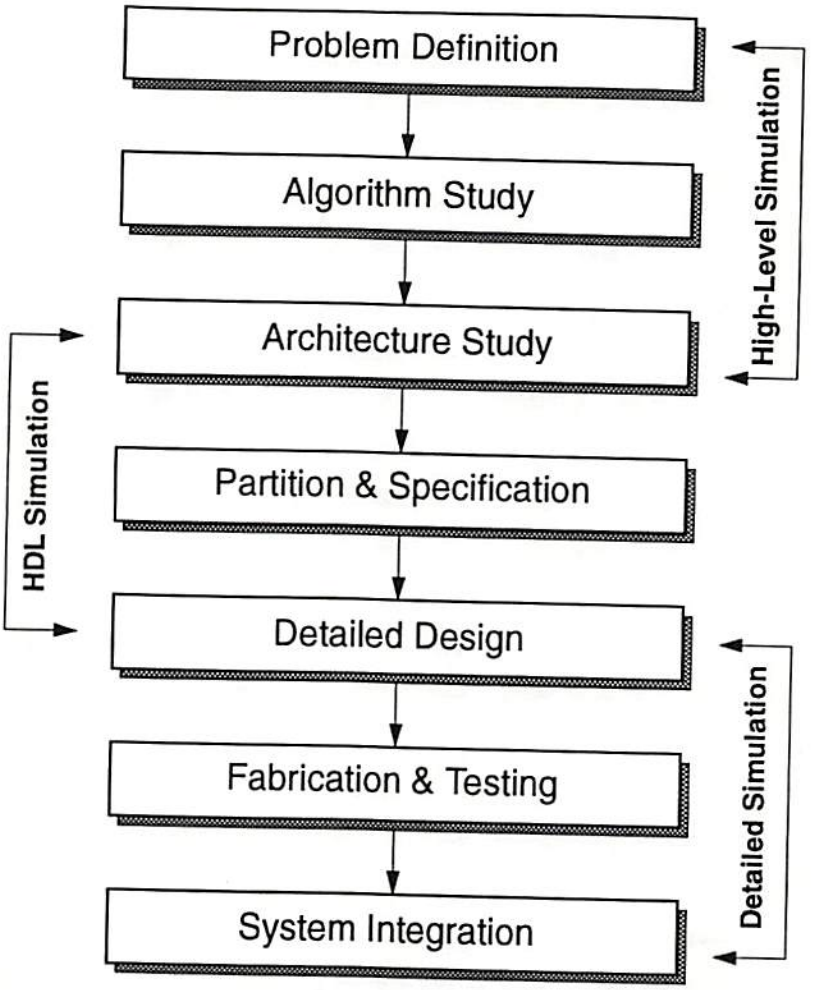


Figure 1.6: The design flow of a VLSI system.

sophisticated CAD tools. But this approach still has its own constraints, too. Interconnections represent one of the main factor of using silicon area. Higher precision also comes with the price of larger silicon area and higher pin-count requirement. Examples of digital implementation include Intel's Pentium and P6 CISC microprocessors, IBM's PowerPC 604 RISC processors and TI's TM32C030/40/80 digital signal processors. Other digital neural network chips include HNC-100 chip from

HNC Inc. [10, 11], CNAPS chip from Adpative Solution Inc. [12, 13], and MA16 chip from Siemens Corporation [14].

On the other hand, analog design can provide a compact, dense array which is very much required in some biologically-inspired applications. For example, the sigmoid function of a neuron can be simply emulated by the combination of an operational amplifier and the synapse cells can be modeled by using resistors or analog multipliers. At present, the precision of weights does not seem to have easy reproduction for long-term storage by using a purely analog approach. Approaches such as dynamic capacitors [15, 16], logical digital memory with D/A converter [17], floating-gate analog memory [18], charge-couple device (CCD) [19] were all proposed to solve this problem.

Hybrid design, which takes the advantages of both digital and analog designs, would provide a better choice in implementing the neural network hardware. Digital storage preserves the signal strength during the operation and helps to cross the chip boundary. Analog cells such as multiplier, comparator or current minor, can be used as basic compact building blocks.

1.5 Organization of This Dissertation

The rest of this dissertation is organized as follows.

Chapter 2 described the paradigm of the compact neural network, which can be described as a two-dimensional mixed-signal processor array and its hardware-annealing capability. Hardware annealing method provides a quick way to find the globally optimal solution, which is an important topic in the engineering discipline.

Chapter 3 covers the simulation techniques and partitioning issues for the behavioral simulator. The behavioral simulator could help designers not only gain

insight on the system operations, but also optimize the hardware-software co-design characteristics. Selective simulation results are also presented.

Chapter 4 describes the overview of the universal machine and our supporting programming language, compiler and simulation environment. A high-level language is defined to facilitate the parallel programming.

In Chapter 5, an innovative mapping of 2-D DCT algorithm to the compact array processors is presented. The array processor at first is used as a motion estimation search engine. With minimum additional hardware, it can efficiently compute the 2-D DCT coefficients which meet the IEEE recommendation of DCT implementation. The overall throughput rate can be higher than 10 GOPS.

Chapter 6 summarizes the results of this dissertation.

In Appendix A, a collected list of the microprocessor information is provided in a table, which is used to predict the VLSI trend. Appendix B gives detailed description of simulating the compact neural network by using SPICE circuit simulator. Appendix C gives a short description of the discrete-time time-delayed network which can memorize some patterns, which could be related to the biological memory system. The user's guide of the behavioral simulator *cnna* is listed in Appendix D.

Chapter 2

Overview of Compact Mixed-Signal Array Processors

2.1 Role of the Compact Neural Network

The computer vision or image processing tasks can be classified into three main categories and as shown in Figure 2.1. First, an image capture device is used at the front end, which can be either a CCD camera, an image scanner or an optical sensor. Those devices capture the images in different formats, such as color images, black and white images, or images represented in different levels of grey levels.

At the other end, the post-processing tasks, including image processing, pattern recognition, are performed either in software by using general-purpose computers or in application-specific ICs (ASICs) to achieve maximum performance. Certain knowledge could be extracted or some patterns could be recognized.

The extraction or recognition tasks could be greatly facilitated if image pre-processing steps are effectively performed. Some information obtained from the capture devices are not required for post-processing. For example, the edge of an image is usually a very important feature. Thus transmitting the edge information only could greatly reduce the required bandwidth instead of transmitting the

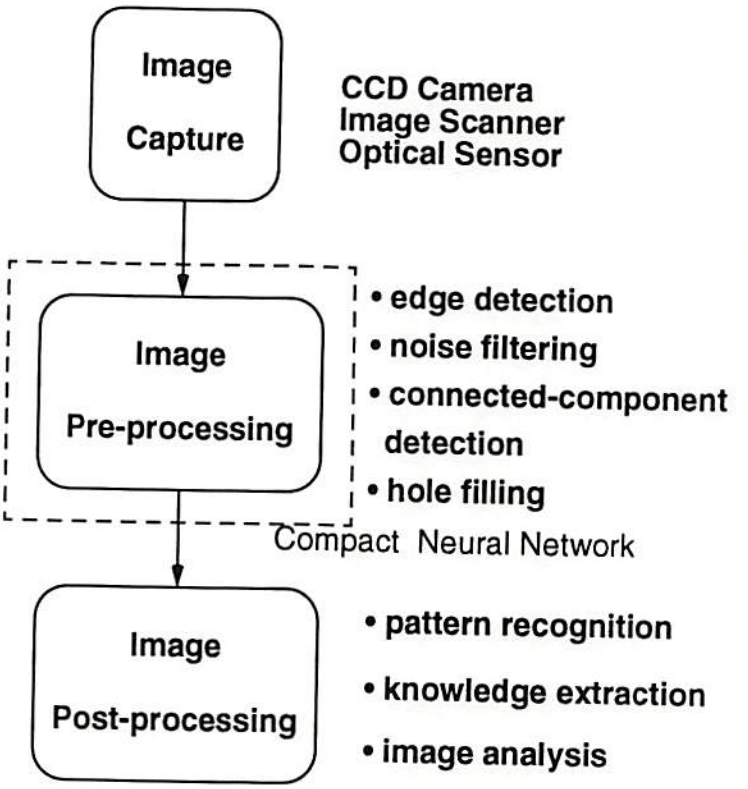


Figure 2.1: The role of the compact neural network: to act as a image pre-processing processor.

whole image in full grey-scale representation. The compact neural network consists of mixed-signal array processors which can accomplish the above-mentioned pre-processing tasks.

The paradigm of the compact mixed-signal array processor is related to the cellular neural network (CNN) paradigm, first proposed by Chua, et al. in 1988 [20, 21, 22]. The two most fundamental ingredients of the paradigm are: **the use of analog processing cells with continuous signal values**, and **local interaction within a finite radius**. Many results on the algorithmic development, VLSI implementations of the compact neural network systems were reported in the first three IEEE International Workshops on Cellular Neural Networks and Their Applications (Budapest, Hungary, 1990; Munich, Germany, 1992; Rome, Italy, 1994); the book

entitled: *Cellular Neural Networks*, which was edited by T. Roska, J. Vandewalle; and papers published in IEEE Trans. on Circuits and Systems, and other IEEE journals and conference proceedings.

The compact neural networks can be viewed as a 1- or n-dimensional array of many identical dynamical cells, which satisfies two properties:

- most *interactions are local* within a finite radius r, and
- all *state variables are continuous-valued signals.*

A coefficient template specifies the interaction between each cell and its neighboring cells in terms of their input, state, and output variables. The coefficient template may be a linear or a nonlinear function of the state, input, and output variables of each cell. It could contain time-delay or time-varying values. The dynamic systems may be perturbed by some noise sources of known statistics. In this dissertation, only the cases of linear output functions and without time-delay effects are discussed.

2.2 Overview of Basic Theory and Computation Paradigm

The compact neural network[1] can be either a continuous- or discrete-time neural network that features a multiple-dimensional array of neuron cells with local interconnection among the cells. The basic paradigm proposed by Chua and Yang [20, 21] is a continuous-time network in the form of an n-by-m rectangular-grid array where n and m are the numbers of rows and columns, respectively. Each cell corresponds to an element of the array. However, the geometry of the array needs not to be rectangular and can be such shapes as triangle or hexagon [24]. A multiple of arrays can

[1]The basic theory was well-described in Dr. Sa Hyun Bang's Ph.D. dissertation [23], and is included in this chapter to facilitate the understanding of compact neural network operation.

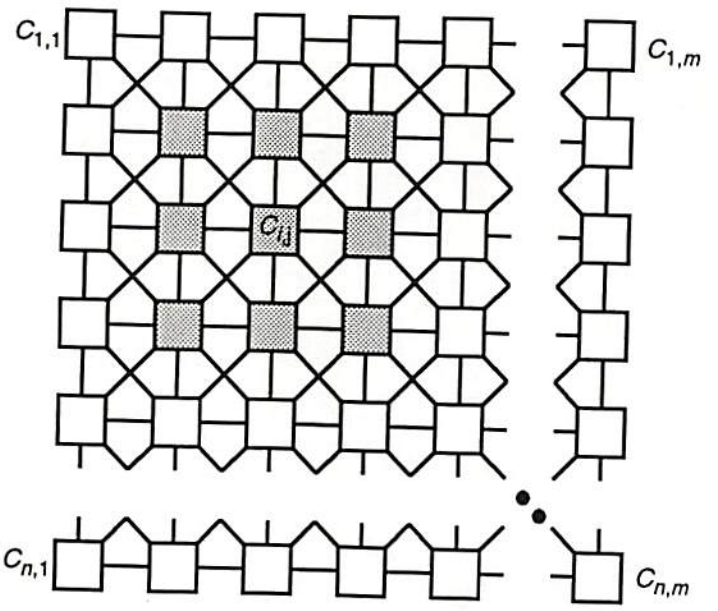


Figure 2.2: An n-by-m 2-dimensional compact neural network on rectangular grid (shaded squares are the neighborhood cells of $C(i,j)$).

be cascaded with an appropriate interconnect structure to construct a multi-layered network.

The r-th neighborhood cells $N_r(i,j)$ of a cell $C(i,j)$, $1 \leq i \leq n$,$1 \leq j \leq m$, are defined as those cells $C(k,l)$,$1 \leq k \leq n$,$1 \leq l \leq m$, for which $|k-i| \leq r$ and $|l-j| \leq r$. Figure 2.2 shows the scheme of an n-by-m compact neural network with neighborhood range $r = 1$. The cells filled with dashed lines represent the neighborhood cells $N_1(i,j)$ of $C(i,j)$, including $C(i,j)$ itself.

The cell $C(i,j)$ has the direct interconnection with $N_r(i,j)$ through two kinds of weights, i.e., the feedback weights $A(k,l;i,j)$ & $A(i,j;k,l)$ and feedforward weights $B(k,l;i,j)$ & $B(i,j;k,l)$, where the index pair $(k,l;i,j)$ represents the direction of signal from $C(i,j)$ to $C(k,l)$. Hereafter T_A and T_B will be used to represent these two synapse weights for short, respectively.

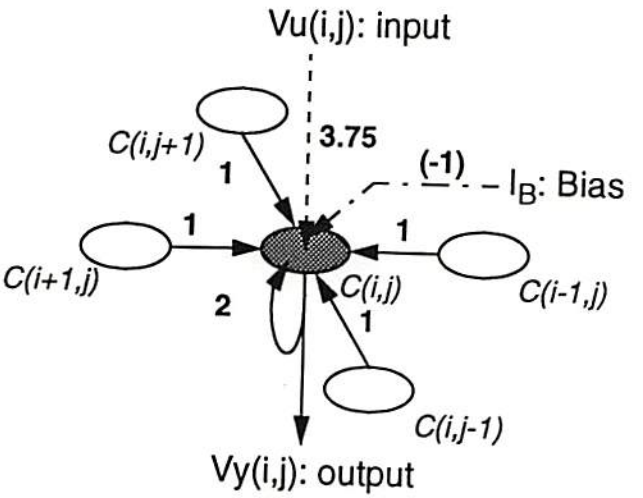


Figure 2.3: Interconnection diagram for the hole-filling template.

Each cell $C(i,j)$ communicates directly with its neighborhood cells $C(k,l) \in N_r(i,j)$. Since the cells $C(k,l)$ have their own neighborhood cells, it also communicates with all other cells indirectly. The term **template** is used to describe the data format which stores the connection-strength information, such as the feedback synapse weight T_A, feedforward synapse weight T_B and bias value I_B.

For example, a hole-filling template is described as

$$\begin{aligned} T_A &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \\ T_B &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3.75 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \\ I_B &= -1. \end{aligned} \tag{2.1}$$

Figure 2.3 illustrates the interconnection information of cell $C(i,j)$ for the hole-filling template.

The block diagram of each cell $C(i,j)$ in the network is further illustrated in Figure 2.4. The external input to the cell is denoted by $v_{uij}(t)$ and typically assumed

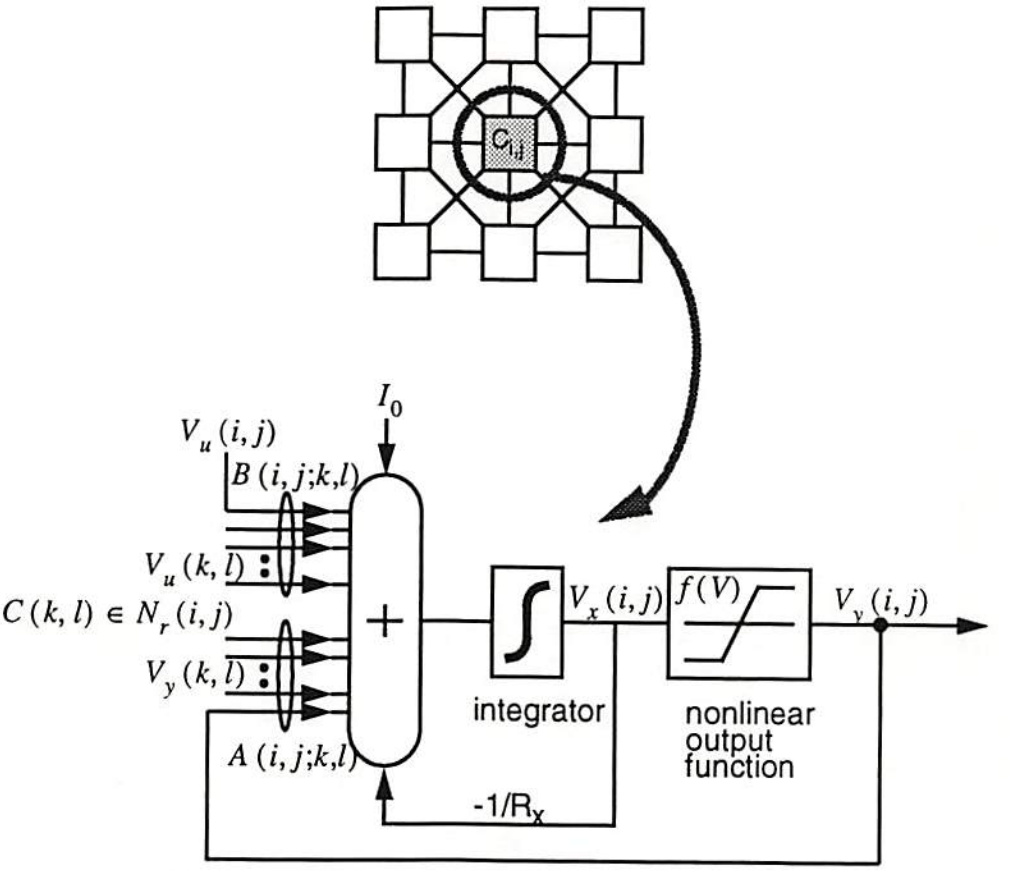


Figure 2.4: Functional block diagram of neuron cell in the 2-dimensional compact neural network.

to be constant $v_{uij}(t) = v_{uij}$ over an operation interval $0 \leq t < T$. The input is connected to $N_r(i,j)$ through the feedforward weights $B(i,j;k,l)$'s. The output of the cell, denoted by v_{yij}, is coupled to the neighborhood cells $C(k,l) \in N_r(i,j)$ through the feedback weights $A(i,j;k,l)$'s.

Therefore, the input signals consist of the weighted sum of feedforward inputs and weighted sum of feedback inputs. In addition, a constant bias term is added to the cell. If the weights represent the transconductance values among the cells, the total input current i_{xij} to the cell is given by

$$i_{xij}(t) = \sum_{C(k,l) \in N_r(i,j)} A(i,j;k,l) v_{ykl}(t) + \sum_{C(k,l) \in N_r(i,j)} B(i,j;k,l) v_{ukl}(t) + I_b, \quad (2.2)$$

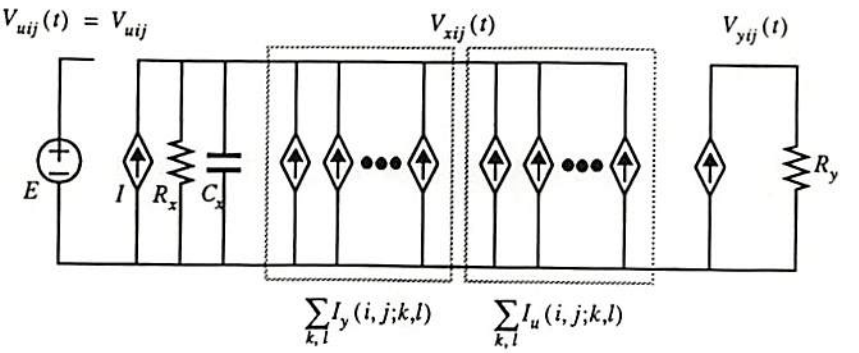


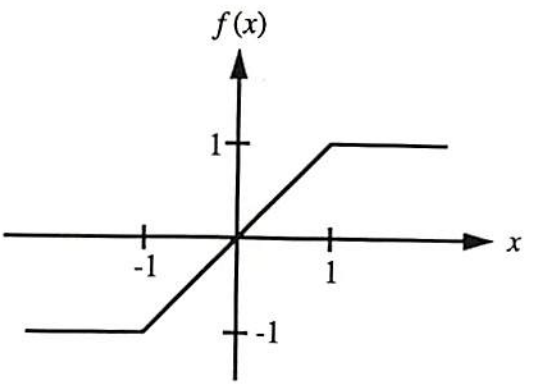
Figure 2.5: Equivalent circuit diagram of one cell [23].

where I_b is the bias current. The equivalent circuit diagram of a cell is shown in Figure 2.5, where R_x and C_x are the equivalent resistance and capacitance of the cell, respectively.

Here, I_b, R_x, and C_x are assumed to be the same for all cells throughout the network. All inputs are represented by dependent current sources and summed at the state node. Due to the capacitance C_x and resistance R_x, the state voltage v_{xij} is established at the summing node and satisfies a set of differential equations

$$\begin{aligned} C_x \frac{dv_{xij}(t)}{dt} &= -\frac{1}{R_x} v_{xij}(t) + i_{xij}(t) \\ &= -\frac{1}{R_x} v_{xij}(t) + \sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l) v_{ykl}(t) \\ &+ \sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l) v_{xkl}(t) + I_b;\ 1 \le i \le n, 1 \le j \le m. \end{aligned} \tag{2.3}$$

The cell contains a nonlinearity between the state node and the output and its input-output relationship is represented by $v_{yij}(t) = f(v_{xij}(t))$. The nonlinear function used in a compact neural network can be any differentiable, non-decreasing function $y = f(x)$, provided that $f(0) = 0$, $df(x)/dx \ge 0$, $f(+\infty) \to +1$ and



$$f(x) = \frac{1}{2}(|x+1| - |x-1|)$$

Figure 2.6: The piecewise-linear output function.

$f(-\infty) \to -1$. Two widely used nonlinearities are the piecewise-linear and sigmoid functions as given by

$$y = f(x) = \begin{cases} \frac{1}{2}(|x+1| - |x-1|) & \text{piecewise-linear function,} \\ (1 - e^{-\lambda x})/(1 + e^{-\lambda x}) & \text{sigmoid function.} \end{cases} \tag{2.4}$$

Here, the parameter λ is proportional to the gain of the sigmoid function. For a unity neuron gain at $x = 0$, $\lambda = 2$ may be used for the sigmoid function. However, if the positive feedback in the cell is so strong that the feedback factor greater than one, the gain of the cell needs not to be large for guaranteed binary output in the steady state. Typically, a unity gain $df(x)/dx|_{x=0} = 1$ is used. The transfer characteristic of the piecewise-linear function is shown in Figure 2.6.

The piecewise-linear function provides a mathematical tractability in the analysis, while the sigmoid-like nonlinearity can be easily obtained from electronic circuits such as an operational amplifier. The shift-invariant network has the interconnection that do not depend on the position of cells in the array except at the edges and is the most desirable feature when implementing a large-size electronic network such as a

VLSI chip. The weights of a shift-invariant array neural network can be represented by the $(2r+1) \times (2r+1)$ feedforward and feedback coefficients templates

$$\begin{aligned} \mathbf{T_A} &= [a_{p,q}, -r \le p, q \le r], \\ \mathbf{T_B} &= [b_{p,q}, -r \le p, q \le r]. \end{aligned} \tag{2.5}$$

Let $N = n \times m$ be the number of cells in the array. By using the vector and matrix notations, (2.3) can be re-written as

$$C_x \frac{d\mathbf{x}}{dt} = -\frac{1}{R_x}\mathbf{x} + \mathbf{A}\mathbf{y} + \mathbf{B}\mathbf{u} + I_b\mathbf{w}, \tag{2.6}$$

where

$$\begin{aligned} \mathbf{x} &= [x_1\ x_2\ \cdots\ x_N]^T = [v_{x1}(t)|v_{x2}(t)|\cdots|v_{xn}(t)]^T, && N \times 1 \\ \mathbf{y} &= [y_1\ y_2\ \cdots\ y_N]^T = [v_{y1}(t)|v_{y2}(t)|\cdots|v_{yn}(t)]^T, && N \times 1 \\ \mathbf{u} &= [u_1\ u_2\ \cdots\ u_N]^T = [v_{u1}|v_{u2}|\cdots|v_{un}]^T, && N \times 1 \\ \mathbf{A} &= toeplitz((\mathbf{A_0}|\mathbf{A_1}|\cdots|\mathbf{A_r}|0|\cdots), (\mathbf{A_0}|\mathbf{A_{-1}}|\cdots|\mathbf{A_{-r}}|0|\cdots)), && N \times N \\ \mathbf{B} &= toeplitz((\mathbf{B_0}|\mathbf{B_1}|\cdots|\mathbf{B_r}|0|\cdots), (\mathbf{B_0}|\mathbf{B_{-1}}|\cdots|\mathbf{B_{-r}}|0|\cdots)), && N \times N \\ \mathbf{w} &= [1\ 1 \cdots 1]^T. && N \times 1 \end{aligned}$$

Here,

$$\begin{aligned} \mathbf{v_{xk}} &= [v_{xk1}(t)\ v_{xk2}(t) \cdots v_{xkm}(t)], && 1 \times m \\ \mathbf{v_{yk}} &= [v_{yk1}(t)\ v_{yk2}(t) \cdots v_{ykm}(t)], && 1 \times m \\ \mathbf{v_{uk}} &= [v_{uk1}\ v_{uk2} \cdots v_{ukm}], && 1 \times m \\ \mathbf{A_k} &= toeplitz((a_{k,0}\ a_{k,1} \cdots a_{k,r}0\cdots), ((a_{k,0}\ a_{k,-1} \cdots a_{k,-r}0\cdots)), && m \times m \\ \mathbf{B_k} &= toeplitz((b_{k,0}\ b_{k,1} \cdots b_{k,r}0\cdots), ((b_{k,0}\ b_{k,-1} \cdots b_{k,-r}0\cdots)), && m \times m \end{aligned}$$

and $toeplitz(\mathbf{a}, \mathbf{b})$ is defined as the Toeplitz matrix with $\mathbf{a}$ in the first row and $\mathbf{b}$ in the first column. Note that the sub-matrices $\mathbf{A_k}$ and $\mathbf{B_k}$ are Toeplitz, but $\mathbf{A}$ and $\mathbf{B}$ are not. The elements of $\mathbf{T_A}$ and $\mathbf{T_B}$ are often normalized to the scale of T_x, e.g., 10^{-3}. The notations of voltages $v_x(t)/v_y(t)$ and the state variables $\mathbf{x}/\mathbf{y}$ will be used interchangeably hereafter. Because $-1 \le y_k \le +1$, $\forall k$, the output variable $\mathbf{y}$ is confined within the N-dimensional hypercube so that $\mathbf{y} \in \mathbf{D^N} = \{\mathbf{y} \in$

$\mathbf{R^N} : -1 \leq y_k \leq 1; k = 1, 2, \cdots, N\}$. The coefficient templates are symmetric if $A(i,j;k,l) = A(k,l;i,j)$ and $B(i,j;k,l) = B(k,l;i,j)$. In this case, $\mathbf{A}$ and $\mathbf{B}$ are symmetric matrices and the stability of the network is guaranteed. In fact, the symmetry of $\mathbf{A}$ is a sufficient condition for stability. Under the constraint conditions $|v_{xij}(0)| \leq 1$ and $|v_{uij}| \leq 1$, $\forall i, j$, the shift-invariant compact neural network always produces a stable output in the steady state. Moreover, if $A(i,j;i,j) > 1/R_x$, then the saturated binary outputs are guaranteed.

In any compact neural network, all states $v_{xij}(t)$,$\forall t \geq 0$, are bounded and the bound $v_{x,max}$ can be determined by [20]

$$\begin{aligned} v_{x,max} &= 1 + R_x|I_b| \\ &+ R_x \max_{\substack{1 \leq i \leq n, \\ 1 \leq j \leq m}} \left(\sum_{C(k,l) \in N_r(i,j)} (|A(i,j;k,l)| + |B(i,j;k,l)|) \right). \end{aligned} \tag{2.7}$$

The terms in (2.7) account for the initial value, bias, feedback, and feedforward interactions, respectively. Therefore, the operating range of the circuits for summing and integration in Figure 2.2 (b) must be at least $-v_{x,max} \leq v_{xij}(t) \leq v_{x.max}$.

2.3 Hardware Annealing on Compact Neural Networks

Optimization[2] is an important subject in solving scientific and engineering problems. The most common searching technique for finding the global minimum is to use gradient descent, which finds the direction for the next iteration from the gradient of the objective function. For complicated problems, the gradient descent technique often gets stuck at a local minimum where the objective function has surrounding

[2]Description of hardware annealing was included in Dr. Bang Lee's dissertation [25]. This section provides convenient overview of the related materials.

barriers. In addition, the complexity of most combinatorial optimization problems increases dramatically with the problem size and makes it very difficult to obtain the globally optimal solution in a reasonably computational time. Simulated annealing is one of the popular approaches which is widely applicable to the combinatorial optimization problem [26, 27].

The parallel hardware annealing technique borrows the concept from the metallurgical annealing and is not just an implementation of the popular software simulated annealing. In order to find the lowest energy level of a piece of metal, the best way is to melt it and to reduce the temperature slowly in order to allow atoms to fit into the lattice sites. Since the number of iterations at a given temperature and the cooling rate of the temperature should be compromised in order to speed up the convergence process, a very large computation time is usually required in software simulation of the hardware annealing. However, recent advances in microelectronic technologies make possible the design of compact electronic neural networks with hundreds of neurons. Many analog electronic neural network processors are equipped with gain-adjustable output neurons which allow the execution of hardware annealing [28, 29].

The hardware annealing is performed by the neuron-gain control $g(t)$, which is assumed to be the same for all neurons throughout the network for simplicity of analysis and illustration [23]. The initial gain at time $t = 0$ can be set to an arbitrarily small, positive value such that $0 \leq g(0) << 1$, and after the annealing process for t_A seconds the final gain $g(t_A) = 1$ is maintained until the next operation.

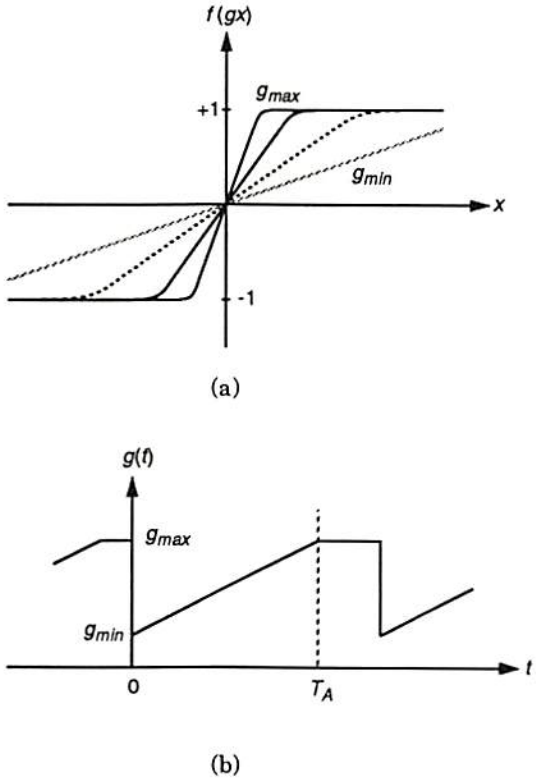


Figure 2.7: Modified neuron cell for hardware annealing in the compact neural network. (a) Transfer characteristics of nonlinearity for several gain control parameters. (b) Gain control function $g(t)$.

When the hardware annealing is applied to a compact neural network by increasing the neuron gain $g(t)$, the transfer function can be described by

$$v_y = f(gv_x) = \begin{cases} +1, & \text{if } v_x > +1/g \\ gv_x, & \text{if } -1/g \leq v_x \leq +1/g \\ -1, & \text{if } v_x < -1/g \end{cases} . \tag{2.8}$$

Figure 2.7(a) shows the transfer characteristic of the piecewise nonlinearity for several gain control parameters g.

Note that the saturation level is still $y = \pm 1$ and only the slope of $f(x)$ around $x = 0$ varies. In Fig. 2.7(b), the gain control function $g(t)$ with constant slope is plotted. In each annealed operation, $g(t)$ increases linearly from $g_{min} = g(0)$ to $g_{max} = 1$ for $0 \leq t \leq T_A$. Then, the maximum gain is maintained for $T_A < t \leq T$,

during which the network is stabilized and the initialization operation $\mathbf{x} = \mathbf{x}(0)$ may take place.

The dynamic of a compact neural network is described by a set of the first-order nonlinear differential equations as defined in (2.3), where $v_{yij} = f(g \cdot v_{xij})$ at this moment. In a compact form,

$$c\mathbf{x} = -T_x\mathbf{x} + \mathbf{A}\mathbf{y} + \mathbf{b}, \tag{2.9}$$

where $\mathbf{y} = f(g\mathbf{x})$ and $\mathbf{b} = \mathbf{B}\mathbf{u} + I\mathbf{w}$. When the neuron gain is not equal to one, the Lyapunov energy function of the network can be modified as,

$$\begin{aligned} E &= -\frac{1}{2}\sum_{i,j}\sum_{C(k,l)\in N_r(i,j)} A(i,j;k,l)v_{yij}(t)v_{ykl}(t) + \frac{1}{2gR_x}\sum_{i,j}(v_{yij}(t))^2 \\ &- \sum_{i,j}\sum_{C(k,l)\in N_r(i,j)} B(i,j;k,l)v_{yij}(t)v_{ukl} - \sum_{i,j} I_b v_{yij}(t) \\ &= -\frac{1}{2}\mathbf{y}^T\mathbf{A}\mathbf{y} + \frac{T_x}{2g}\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{b} \\ &= -\frac{1}{2}\mathbf{y}^{\mathbf{T}}\left(\mathbf{A} - \frac{T_x}{g}\mathbf{I}\right)\mathbf{y} - \mathbf{y}^{\mathbf{T}}\mathbf{b} = -\frac{1}{2}\mathbf{y}^{\mathbf{T}}\mathbf{M}_{\mathbf{g}}\mathbf{y} - \mathbf{y}^{\mathbf{T}}\mathbf{b}. \end{aligned} \tag{2.10}$$

where the factor $1/g$ in the second term stems from the energy associated with the piecewise-linear function with a neuron gain other than unity. Since $\mathbf{M_g}$ is also a real symmetric matrix, it can be diagonalized as $\mathbf{M_g} = \mathbf{A} - (T_x/g)\mathbf{I} = \mathbf{Q}\mathbf{\Lambda_g}\mathbf{Q^T}$, where $\mathbf{\Lambda_g}$ is the diagonal matrix of eigenvalues λ_k, $k = 1, 2, \cdots, N$, and $\mathbf{Q}$ is an $N \times N$ matrix whose columns are made of orthonormal set of eigenvectors $\mathbf{e_k}$'s. In an annealed neural network, the elements of $\mathbf{\Lambda_g}$ are time-varying. However, $\mathbf{Q}$ is independent of the neuron gain because $\mathbf{M}$ and $\mathbf{M_g}$ commute. By noting that $\mathbf{M_g} = \mathbf{A} - T_x\mathbf{I} - ((1-g)Tx/g)\mathbf{I} = \mathbf{M} - ((1-g)Tx/g)\mathbf{I}$, the relationship between the eigenvalues of unannealed and annealed network can be easily shown to be

$$\lambda_k = \lambda'_k - \frac{(1-g)T_x}{g}, \qquad \mathrm{k} = 1, 2\cdots, N \tag{2.11}$$

where λ_k's are the eigenvalues of $\mathbf{M}$. In the hardware annealing, the eigenvalues λ_k's are changed from all negative initial values to the final values λ_k's by increasing the neuron gain g, such that the energy function (2.10) which is initially a convex function of $\mathbf{y}$, is transformed gradually into a concave function. The initial neuron gain g_0 must be chosen such that $\lambda_k(g_0) < 0$, $\forall k$.

Detailed theoretical proof and an example to demonstrate the superiority of hardware annealing technique had been found in [23, 30]. It had been successfully applied to Hopfield networks and the cellular neural networks. But the application of the hardware annealing is not limited to only neural networks, global optimal solutions can be achieved in both image edge detectors and in wireless communication receivers.

Chapter 3

The Behavioral Simulator

3.1 Simulation Techniques

In order to solve the system of 2-dimensional $n \times m$ networks using a digital computer, the governing equation (2.3) is best represented in the matrix form as

$$C_x \frac{d\mathbf{x}}{dt} = -\frac{1}{R_x}\mathbf{x} + \mathbf{T_A y} + \mathbf{T_B u} + I_b \mathbf{w}. \tag{3.1}$$

The C_x is used for integration purpose and R_x is used to provide a leakage path. For behavioral simulation, these two variables will be normalized to 1 and the time scale used hereafter will be in the normalized time unit. Since the inputs are kept as constants during each operation, the last two terms can also be lumped together as another constant term. Once the output transfer function is included, the overall systems can be expressed as a set of differential equations to be solved as

$$\frac{d\mathbf{x}}{dt} = -\mathbf{x} + f(\mathbf{T_A x}) + \mathbf{L_c}, \tag{3.2}$$

where $f(\cdot)$ is the output transfer function and $\mathbf{L_c}$ is the lumped constant vector whose values are equal to $\mathbf{T_B u} + I_b \mathbf{w}$. By using differential equation solver subroutines provided by many software vendors [31, 32], the whole system dynamics can be simulated and analyzed.

One severe drawback for this simulation method is that, for an $n \times m$ network, the feedback matrix T_A and control matrix T_B will have the dimensions of $mn \times mn$, which increases on the order of $O(n^4)$. When a large system is to be simulated, very large storage resources will be required to hold the data for these two matrices. Only a small portion of the entities are non-zero and others will be zero. Precious storage space and computing resource are not efficiently utilized. Besides, it will be quite challenging to partition the computing jobs for multi-processors or multi-computer systems because of the synchronization requirement, non-regularity of the network and the inefficiency in the routing and use the communication bandwidth.

An alternative solution is to apply the "convolution" idea popularly used in the digital image processing. Because the cells are locally connected and space-invariant, the basic approach for the integration is to consider a sub-image block whose size is the same as the feedback matrix T_A (or feedforward matrix T_B). It moves from left to the right, from top to the bottom of the image as shown in Figure 3.1. The output state will only be updated after the whole image has been processed at each iteration. Only new state information on the border has to be passed to the neighboring block. The simulation will stop after all the outputs saturated to either 1 or -1 and no longer changed their values.

By applying this "convolution" concept, it will not only save the storage space required but also increase the data hit rate during the software simulation. That's mainly because most of the time the data are in the cache for local access.

The updating equation in each cell can be written in the form as

$$v_{xij}(t_{n+1}) = v_{xij}(t_n) + \int_{t_n}^{t_{n+1}} f'(v_{xij})dt. \tag{3.3}$$

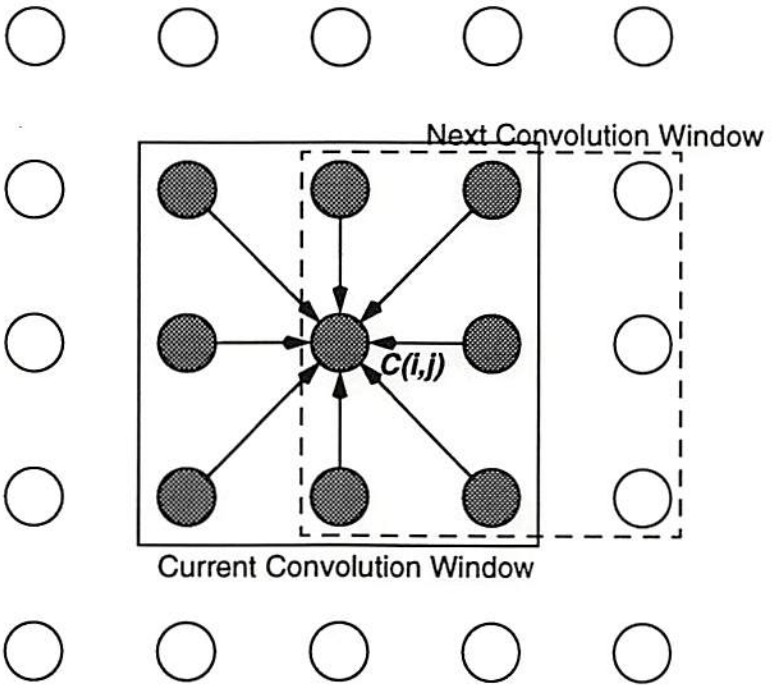


Figure 3.1: Operation windows used in the convolution-type integration.

Now the problem can be treated as solving the set of differential equations with initial values $V_x(0)$ in each sub-image block. The one-step method, such as the simplest Euler's method or the more elaborated fourth-order Runge-Kutta method, can be used for the integration. The latter will cost more in terms of computation time because it evaluates four derivatives per iteration. However its high cost is compensated by its accuracy in transient behavior analysis and thus is usually favored. The fourth-order Runge-Kutta for one-step integration is given by [33]

$$V_x[t_{k+1}] = V_x[t_k] + \frac{\delta t}{6}(F_1 + 2F_2 + 2F_3 + F_4), \tag{3.4}$$

where δt is the integration time step and F_1, F_2, F_3, F_4 are four intermediate terms.

One advantage to solve the compact network in this scheme is that the whole system can be easily partitioned for parallel computation using multi-processor or multi-computer systems. As shown in Figure 3.2, assuming that the 2-dimensional

compact network is partitioned into $P \times Q$ sub-blocks for parallel computation, the overall execution time T_e will be equal to

$$T_e = \frac{nm}{PQ}T_u + \frac{2}{B_c}(\frac{n}{P} + \frac{m}{Q})T_i, \tag{3.5}$$

where T_u is the computing time for each cell and T_i represents the time for inter-block communication. B_c is the communication bandwidth factor between adjacent sub-blocks. The first term decides how much computation time is needed within each sub-block and the second term is related to the interval required for block-to-block communication. The more sub-blocks divided, the less the computing time will be needed. The penalty will be the larger number of computing resource required, massive synchronization among the blocks, and the communication burden among the blocks.

The total execution time versus the number of blocks divided per dimension is plotted in Figure 3.3 with different T_i/T_u ratios. In the experiment the input is assumed to have the same size in both dimensions (256×256) and the number of blocks divided in each dimension is assumed to be equal. The communication bandwidth factor B_c is set to be 1 to facilitate the analysis. When a multi-computer system is used to simulate the network, the T_i/T_u ratio tends to be large due to the extra memory access time and bus arbitration time. When the number of divided sub-blocks increases, the execution time difference among different T_i/T_u ratios will also increase. This implies that a fine-grained system will need better communication to improve its system performance.

A compact analog VLSI implementation will be an extremely powerful approach for $P \to n, Q \to m$. Here T_u is equal to $k \cdot RC$, where k is a scaling factor depending on the coefficient template and was proved to have an upper-bound limit for each template [20]. Therefore saturated binary results will be achieved after a certain

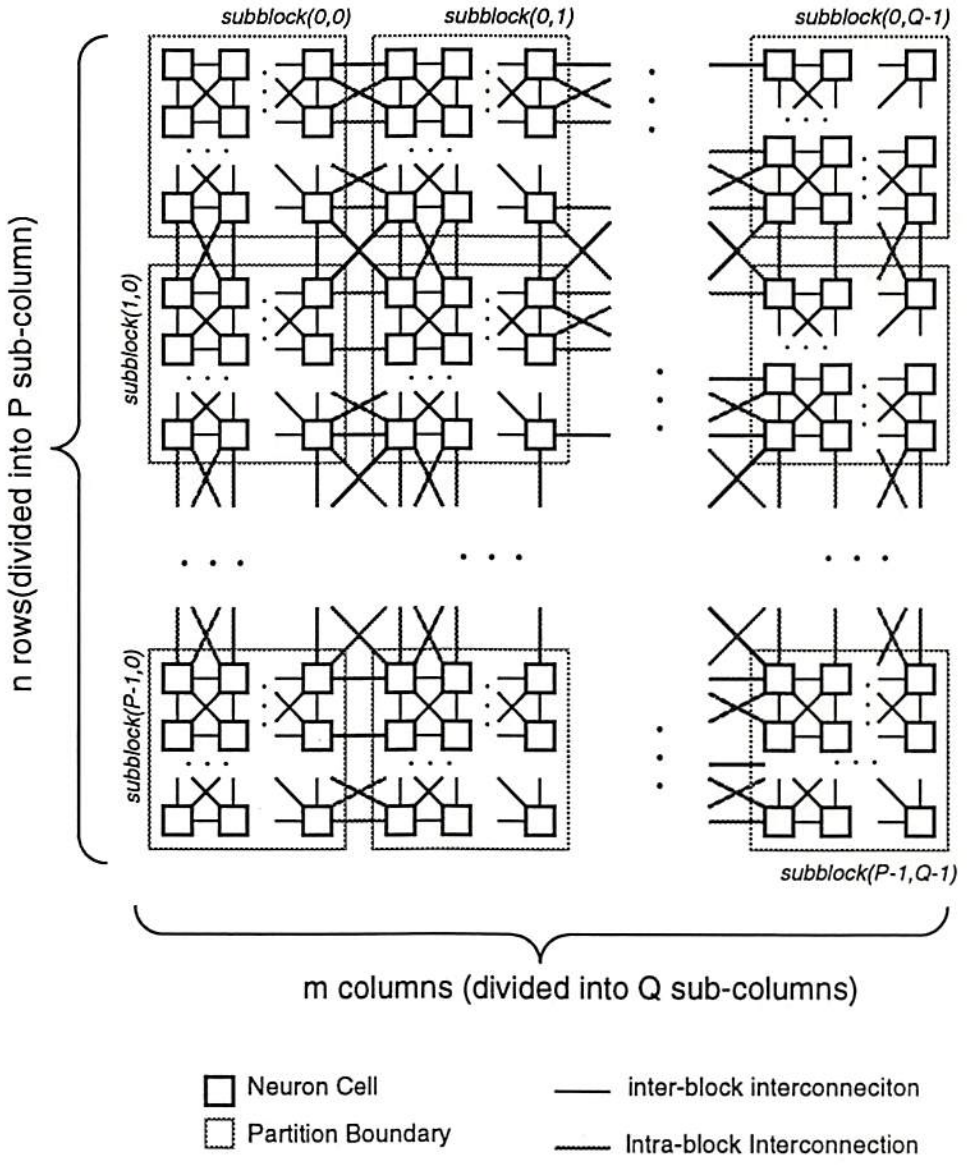


Figure 3.2: Partition of an $n \times m$ network into $P \times Q$ sub-blocks, where the numbers shown are different T_i/T_u values.

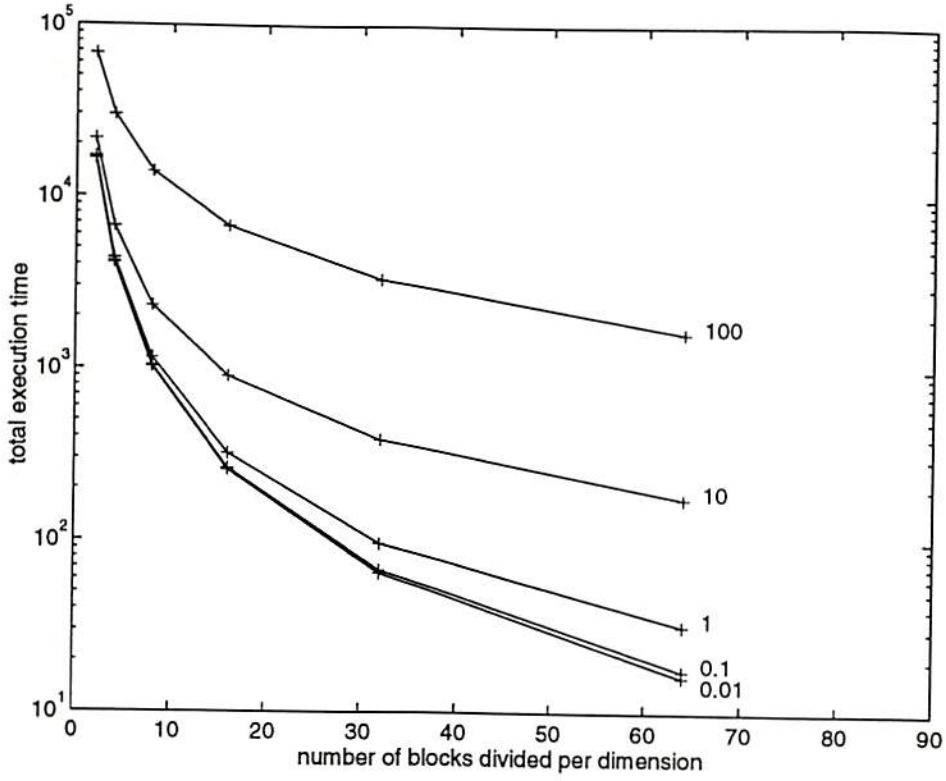


Figure 3.3: Total execution time versus number of blocks divided in each dimension.

amount of execution time. If the network can be realized on one silicon micro-chip, the updated interconnection will occur simultaneously when the state changes. T_i will become negligibly small and the second term in equation (3.5) can be dropped. The overall execution time will be equal to just $k \cdot RC$. By using modern VLSI fabrication techniques, the RC constant will be in the range of 10 ns to 1 μs. The achieved speed is enormously fast when dedicated microelectronic hardware is built.

3.2 Related Work on the Simulators

Several research versions of simulators have been announced. The CNN Workstation [34], the XCNN simulator [35], the SIRENA environment [36] or the Neurobasic

Table 3.1: Comparison of different neural network simulation programs.

Simulator	cnna	XCNN[35]	NeuroBasic[37]	CNNM[34]
Graphical User Interface	Yes (cnng)	Yes	No	Text-Menu
Annealing Capability	Yes	No	No	No
Multiple-step Simulation Language Support	C-like	BNF style	Basic-Like	No
Image Size	unlimited	unlimited	unlimited	47 x 47
Image Display	Use companion software	Built-in	Use companion software	Built-in (only a few different colors available)
Dynamic Analysis	With companion software	Animation during execution	Animation at certain prespecified time	No

simulator [37], are representative examples. A comparison of the simulator programs is summarized in Table 3.1.

The CNN Workstation, developed by the Dual and Neural Computing Systems Laboratory, in Budapest, Hungary, provides a simple experimental tool for studying cellular neural networks. Transients of compact neural networks with linear, nonlinear, and delay-type templates can be monitored graphically. A basic menu-driven user interface provides the control mechanism of the system.

Another software package XCNN simulator from Texas A&M University in College Station, Texas, focuses on a multi-layered neural network structure performing color image processing applications. Additional post processor is used to perform

pixel-wise logical operations among different layers. The commands are based on a specialized BNF-like language.

Researchers in the Universidad de Sevilla, Spain, developed the SIRENA environment which is a general framework for artificial neural networks, with emphasis on cellular neural networks. The focus is on the simulation and modeling of the non-ideal effects in VLSI implementations, with efficiency comparable to SPICE circuit simulator. Graphics interface is provided for simulation supervision and image visualization.

The Neurobasic simulator from Swiss Federal Institute of Technology, in Zurich, Switzerland, is another simulation environment for neural networks which uses the Basic programming language as the development tool. It is also designed to execute on the MUSIC parallel computer. The neuron function can be evaluated very fast because of the massive parallelism of computer hardware.

The network can also be simulated by using SPICE circuit simulator, and the appropriate circuit elements. More detailed explanations are described in Appendix B. Using SPICE can provide excellent transient analysis capability but it usually takes much long time to simulate the network. The output can be nicely displayed as the way results from other simulations are handled.

3.3 The Behavioral Simulation Environment

The framework of the behavioral simulation environment is shown in Figure 3.4. The simulator accepts the configuration command files and the images files as input and generates the result output images after the simulation. Those images can be visualized using companion software or via the graphical user interface.

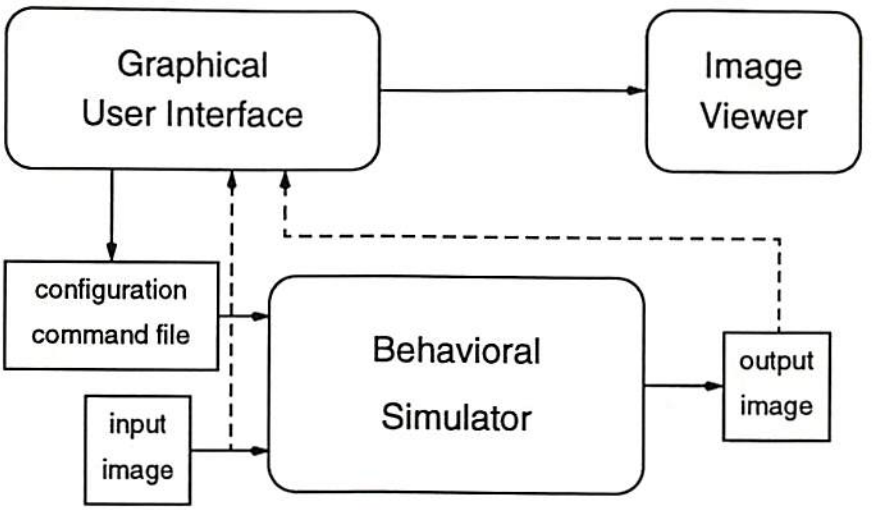


Figure 3.4: The flow of the compact neural network simulation environment.

The compact neural network annealing simulator *cnna* was constructed based on the integration method described above. It is developed using the portable C language and consists of more than 3,500 lines of code. It runs under either Unix operating system with a suitable C compiler. The behavioral simulator can provide valuable information and is a good tool to characterize the behavior of the system.

An X-window graphical user interface based on XView widget is built on top of the text-mode simulation which provides a more intuitive way for the user to execute the simulator. A snapshot of the graphics interface is shown in Figure 3.5.

The strength of the compact networks lies in the programmability by changing the coefficient templates. That is to say, the content of the templates can be viewed as instruction sets used in the conventional digital microprocessors. The simulator can either read the commands from a text file or accept the input from the graphic user interface and properly established the configurations for the network operation. The input images and initial states can be handled as the operands. The system will only need information of new templates for different applications. Therefore a general-purpose simulation environment can be built.

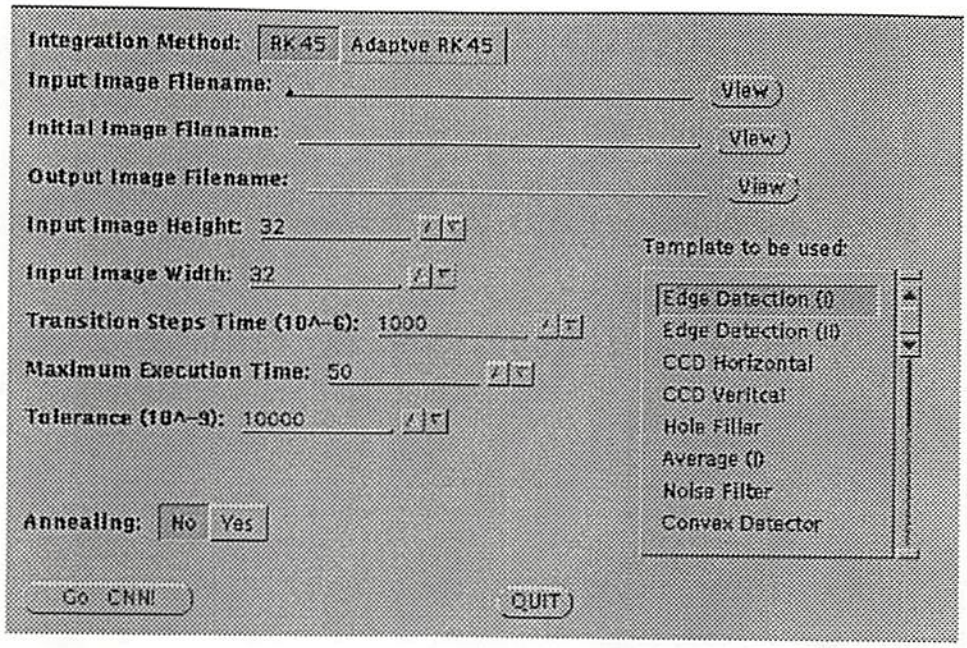


Figure 3.5: A snapshot of the graphics user interface of the behavioral simulator.

The user can program the command files for different applications. A sample of the command file that is used to simulate the vertical connected-components detection operation described in the next section is listed in the following:

```
METHOD RK4
TRAN 0.01 10.0
ANNEAL N
TOL 0.0001

ISIZE 20 20
INPUT ccd.20
INITIAL ccd.20
OUTPUT ccd20ns.out

TSIZE 3 1
TEMPA 0 1 0 0 2 0 0 -1 0
TEMPB ALL 0
BIAS 0.0
BOUNDARY 0
```

The commands can be summarized into 3 groups. The first category contains **simulation commands** which specify what kind of integration method is used for simulation, what is the integration interval, and whether annealing capability is applied. The second category consists of **I/O commands** where the input/output image information is provided. The last category contains the template **configuration commands** which can be provided individually or with the help from the template library.

Template library is supported which accommodates more than 50 useful templates. A list of selective known templates for cellular nonlinear networks can be found in [38]. Additional coefficients templates were also reported in [39, 40]. Several examples are used in the next section for demonstration.

3.4 Features of the CNNA

There are several features that could be incorporated in this behavior simulator to help us study the system behavior before the detailed design of the hardware. It provides us valuable information for effects of the non-linear network and non-ideal microelectronic fabrication. Those effects can be summarized as:

Internal state limitation Although the internal states will be bounded to a certain value as proved in [20], it is not desirable to have such a large dynamic range for the actual circuit simulation. With limited swing voltage (or current) range, a large dynamic will sacrifice the resolution. To decide what range is appropriate, the simulator can help.

Non-ideal output function The output function described in section 2.4 has the characteristics such as passing the origin, skew-symmetric with respect to the origin point, and saturating at fixed output when the input is large. However,

the output function when implemented in hardware will not be so perfect. Symmetric characteristic is not always achieved and the output might keep growing even when the saturation point is reached. To simulate this effect, a look-up table for output function can be used to study the effect of the desired output function.

Crosstalk in the interconnection There are heavy communication activities among the computing cells during the execution which might cause the crosstalk. Especially for the analog implementation, the burst noise might randomly strengthen or weaken the templates weights and thus lead to a different solution. By imposing randomly generated noise with a pre-specified strength, the simulator could provide useful information.

Limited resolution For digital simulation, the resolution can be as high as desired with the penalty of longer execution time. For analog implementation, the resolution supported by the analog circuits is typically limited to about 7 to 8 bits. Use of integer data type instead of floating point data type can accurately simulate the effect.

An unique feature of this simulator is the annealing capability. Hardware annealing [23, 41], which is effectively the paralleled version of the popular mean-field annealing used in analog arrays, provides an efficient method of finding globally optimal solutions. It is performed by changing the gain value of the input-output transfer function $f(\cdot)$, which is described by (2.8) for the piecewise linear function. At the beginning of the annealing process, the initial gain can be set to a very small, positive value. During the annealing process the gain keeps increasing and the final gain $g_t = 1$ for the piecewise linear function is maintained until the next operation. Notice that the new current-mode circuit scheme is used and the maximum gain

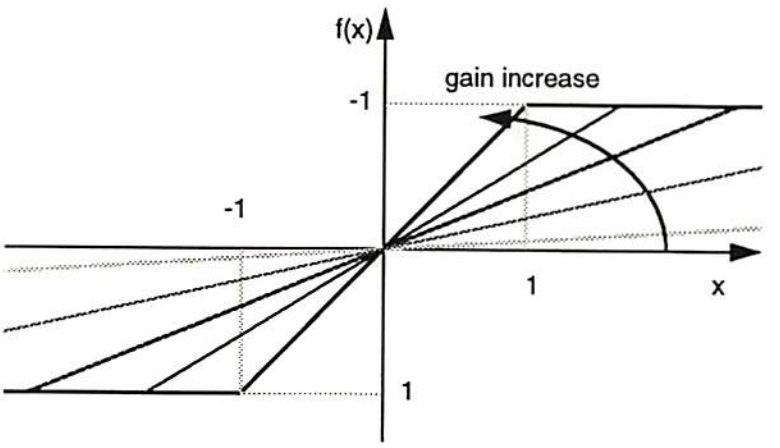


Figure 3.6: Annealing processing with changing gain.

value in the cellular network is only 1. Figure 3.6 shows the transfer characteristics of the nonlinearity for several values of the gain control parameter g_t.

The simulator takes much longer CPU time for the annealing process because the gain g_t needs to be changed during the simulation. This is due to the use of low neuron-gain at the beginning, in contrast to the constant high gain for simulation without the annealing. The states will change in order to deterministically search for the optimal solution in the solution space.

3.5 Simulation Results

The connected-component detection operation [42] can count the number of connected components in each row (or column) of the input image. This operation is performed by using the appropriate template for the row operation,

$$A(i,j;k,l) = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad B(i,j;k,l) = \underline{0}, \qquad I_b = 0. \tag{3.6}$$

The input image to be processed is entered as the initial state values $V_x(0)$. The output will be saturated to either 1 or -1 and the numbers of cells which have

positive outputs in each row is the number of connected components. These positive output values will be separated by a -1. Figure 3.7 shows the final results using this template. The execution time for this specific application grows linearly with the image size because the 1-dimensional dependency of the data.

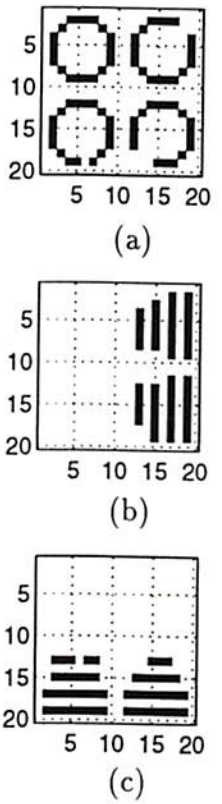


Figure 3.7: (a) Input artificial image. (b) Output result along the rows after connected-component detection operations. (c) Output result along the columns.

Another example is the hole-filling operation [43], which can fill the holes within the edge of objects in the input image. The edge of the hole has to be at least eight-connected for proper filling. A pixel X is said to be eight-connected to the neighbors if it is a logical one and at least one of its east, west, north, south, northeast,

northwest, southeast or southwest neighbors is also logical one. The operation can be realized by the template

$$A(i,j;k,l) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad B(i,j;k,l) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad I_b = -1. \qquad (3.7)$$

The image to be processed is entered as the input, and the initial states are all set to 1. As time evolves, the pixels to be filled will stay at 1 and the those pixels which will not be filled will keep decreasing their values and finally saturated at -1. Figure 3.8 shows the results of hole-filling operation on two simple spiral images.

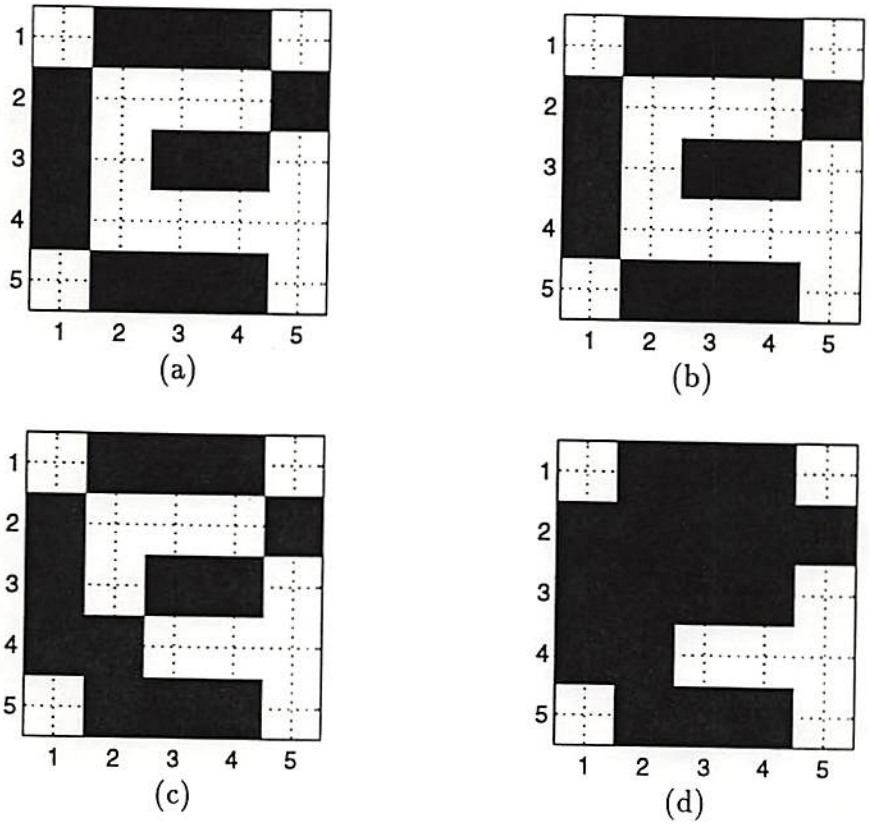


Figure 3.8: (a) Input image without an 8-connected pattern. (b) Output image which has no change. (c) Input image with a 4-connected object. (d) Hole-filling output image where the enclosed pixels are filled.

The applications described above both use the binary input images. For some applications, gray-level images are used. In the edge-detection operation, the required templates are

$$A(i,j;k,l) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$B(i,j;k,l) = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}, \quad I_b = -0.3. \tag{3.8}$$

An example that shows the superiority of hardware annealing for finding the global optimal is shown in Figure 3.9. The original *Mickey* image is plotted in Figure 3.9 (a) and it is a gray-level 187 $\times$ 294 image. After we applied the conventional digital image processing operation with

$$M = \begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}, \tag{3.9}$$

and threshold set to 0, an edge image is generated in Figure 3.9 (b). The simulation result shown in the Figure 3.9 (c) is produced by (3.8) without hardware annealing. In Figure 3.9(d), an improved edge result is shown when the annealing is applied. The annealing speed and the threshold I_b are the parameters for the annealing operation.

The compact neural network can be enhanced by incorporating chaotic neurons into the array to explore the rich spatio-temporal relationship. Such complex networks are important models for physical systems and biological signal processing with many degrees of freedom [44]. The Chua's circuits can be used as chaotic cells [45]. Due to the high dimensionality of the complex chaotic networks, accurate simulation will be a challenging task and is not addressed in this dissertation. Our

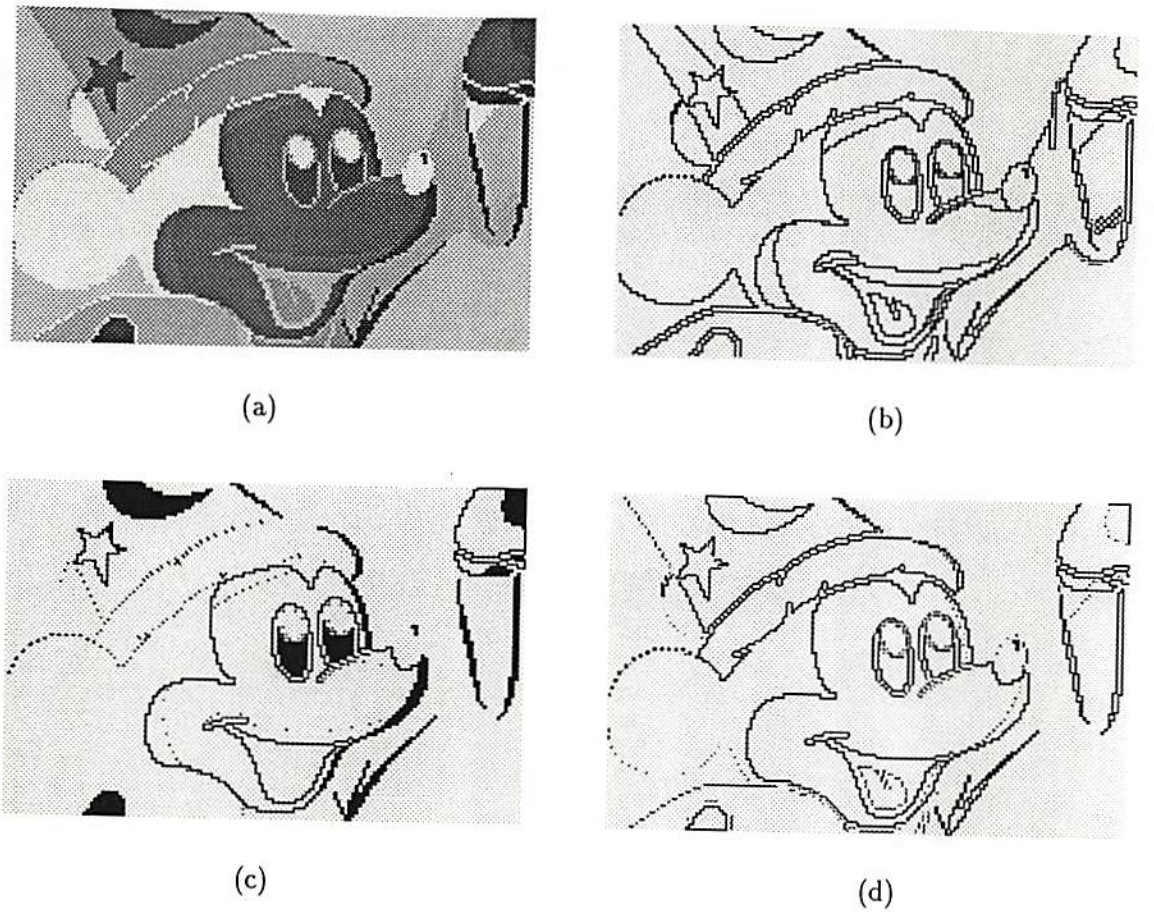


Figure 3.9: Demonstration of the hardware annealing method. (a) Original Mickey image. (b) The result obtained by using the traditional digital image processing. (c) The result after applying the edge detection function template. (d) The result when both edge detection template and hardware annealing are applied.

behavioral simulator can be used in conjunction with the mixed-mode circuit simulator iSPLICE [46] for development of application-specific array processing VLSI chips in pattern recognition.

3.6 Discussion on the Simulation Results

3.6.1 Performance Evaluation

Since the convolution concept is used, the required computation time is linearly dependent on the number of total cells, number of synapse weights, and the number of iterations executed. Table 3.2 summaries the simulation time for different cases.

Table 3.2: Performance comparison of different operations by using **cnna** simulator.

Function	Image and Size	Execution CPU Time	Iteration Steps
Edge Detection	lenna (32x32)	6.53 sec	1,000
	lenna (64x64)	26.33 sec	1,000
	lenna (128x128)	104.62 sec	1,000
	lenna (256x256)	418.30 sec	1,000
Horizontal Connected-Component-Detection	letter A (32 x 32)	31.13 sec	2,000
	letter A (64x64)	185.78 sec	3,000
	letter A (128x128)	1476.50 sec	6,000
	letter A (256x256)	19799.44	20,000
Image Halftoning	lenna (32x32)	54.89 sec	2,000
	lenna (64x64)	329.48 sec	3,000
	lenna (128x128)	1755.75 sec	4,000
	lenna (256x256)	6664.56 sec	5,000

All the simulations are done on a SUN Sparc-20 workstation running Solaris 2.3. In the first case, the edge detection template, which is describe in (3.8), is used where most of the non-zero coefficients terms are in T_B and can be lumped together. Therefore the simulation time is much less than the other two cases. The time required is approximately proportional to the number of cells.

The second case is the connected-component detection which is described in (3.6). The task takes longer time to finish as compared with the edge detection case because there are only feedback synapse weights (T_A). Besides, the behavior which moves the positive pixels (value +1) along a certain direction also contributes to a larger number of integration steps before the output saturates. This leads to a longer execution time, given the same image size. The last testing template is a 5-by-5 halftoning template, which is larger than the 3-by-3 template in the other two cases. Since more interconnections are involved, it takes longer execution time.

Table 3.3 lists the number of iteration steps for the outputs to saturate. For the edge detection operation, the numbers are not linearly proportional to the image size, mainly because it is a locally-interconnected operation. In contrast, the connected-component detection operation involves the pixel value movement along a certain direction. Therefore it almost takes twice the number of steps needed whenever the image is enlarged by a factor of 2.

3.6.2 Performance Comparison with Other Simulation Programs

Performance comparison with different simulation programs is listed in Table 3.4. Edge detection and connected-component detection operations are used as the benchmarks with two test images sizes of 32-by-32 and 64-by-64.

Table 3.3: Convergence analysis of different operations by using **cnna** simulator.

Function	Image and Size	Steps Needed for Saturation
Edge Detection	lenna (32x32)	344
	lenna (64x64)	430
	lenna (128x128)	471
	lenna (256x256)	472
Horizontal Connected-Component-Detection	letter A (32 x 32)	1185
	letter A (64x64)	2393
	letter A (128x128)	4773
	letter A (256x256)	9627

NeuroBasic can complete both four tests at a speed comparable to **cnna**. CNNM simulator does well in the speed test at the price of accuracy. That causes an incorrect answer for the connected-component detection operation. Besides, it can only support the simulation up to 47-by-47 cell size due to the DOS memory limitation.

The programs are executed on different platforms. It would be unfair to compare the speed directly. According to the benchmark program used in the MATLAB, a speed index is used as a reference. The speed index is approximately proportional to the computing power of the platform used. The higher the index is, the higher computation capability the machine can have. The execution time multiplied by the speed index would be a good measurement for the performance comparison.

Table 3.4: Performance comparison of different compact neural network simulation programs.

Simulator	cnna	NeuroBasic	CNNM	H-SPICE
edge detection (32x32)	6.53 sec	7.56 sec	1 sec for 20 iterations*	1860.8 sec
edge detection (64 x 64)	26.33 sec	28.69 sec	not supporting this size	fail because of insufficient memory
CCD (32x32)	31.13 sec	17.64 sec	11.0 sec	3786.6 sec
CCD (64x64)	185.78 sec	101.47 sec	not supporting this size	fail because of insufficient memory
Simulatoin Platform	SUN SPARC 20	SUN SPARC 20	IBM PC Pentium 90 MHZ	SUN SPARC 2
Speed Index	2.73	2.73	2.46	1.0

* The simulation is done by applying the edge detection template on a 32x32 image consisting of a solid filled square with size of 20-by-20. The simulation on the lenna image didn't succeed because the simulator didn't stop after the output saturates

3.6.3 Impact of the Bias Value

For many applications that mainly worked on the binary input images, the bias value will not change the state of each neuron cell which is either 1 or -1. That means the bias value is not an important factor. But for the grey-level (continuous-level) input images, the bias will actually affect its overall behavior.

Take the edge detection operation as an example, where the template information is listed in (3.8). The original image is an eye portion of the Lenna image. If the bias value is varied from -1 to 0, different edge results will be achieved as shown in Figure 3.10. When the bias is set lower than -1, too much information can be filtered out.

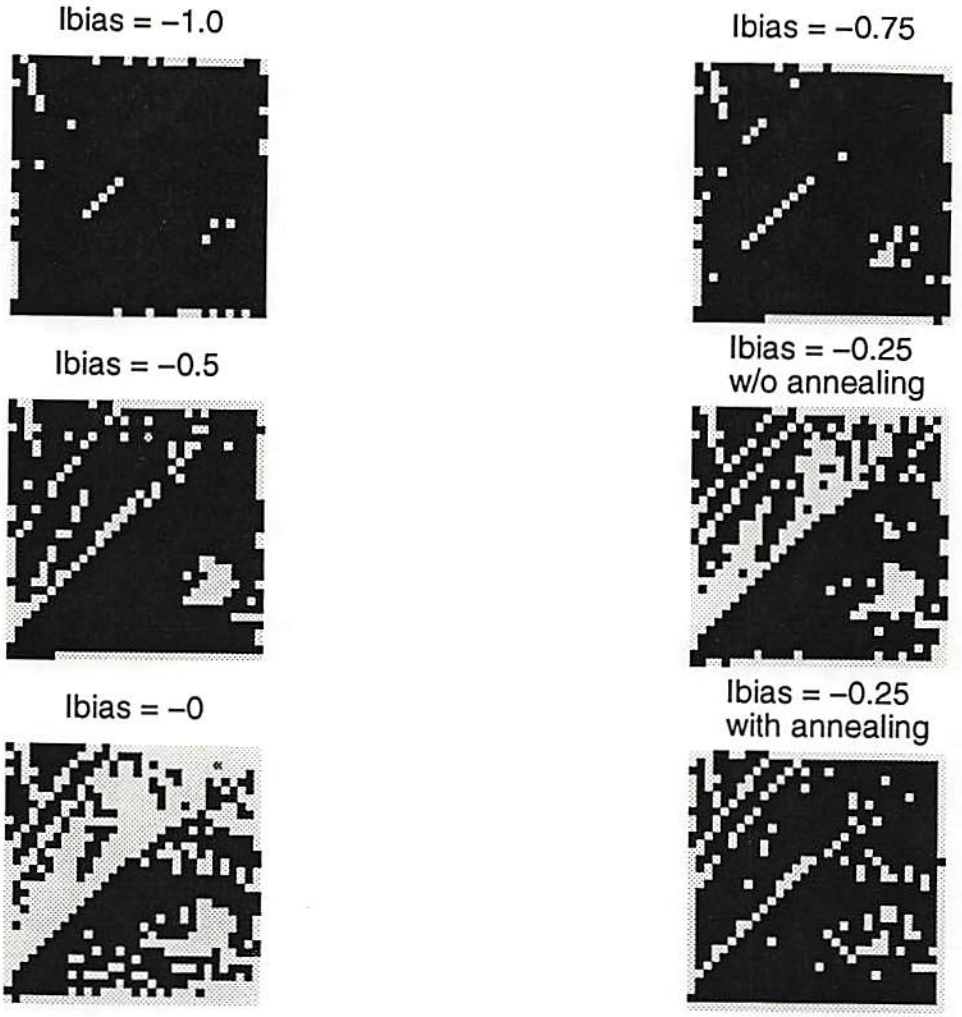


Figure 3.10: The output image after edge detection operation with different bias values.

This phenomenon also occurs with the digital Sobel edge detector [47]. After the Sobel operator is applied, a threshold value has to be set to decide whether the pixel represents an edge or not. With different threshold values, the quality of the edge result will not be the same, as shown in Figure 3.11. Therefore, the choice of the bias or the threshold value is very critical for the edge detection operation.

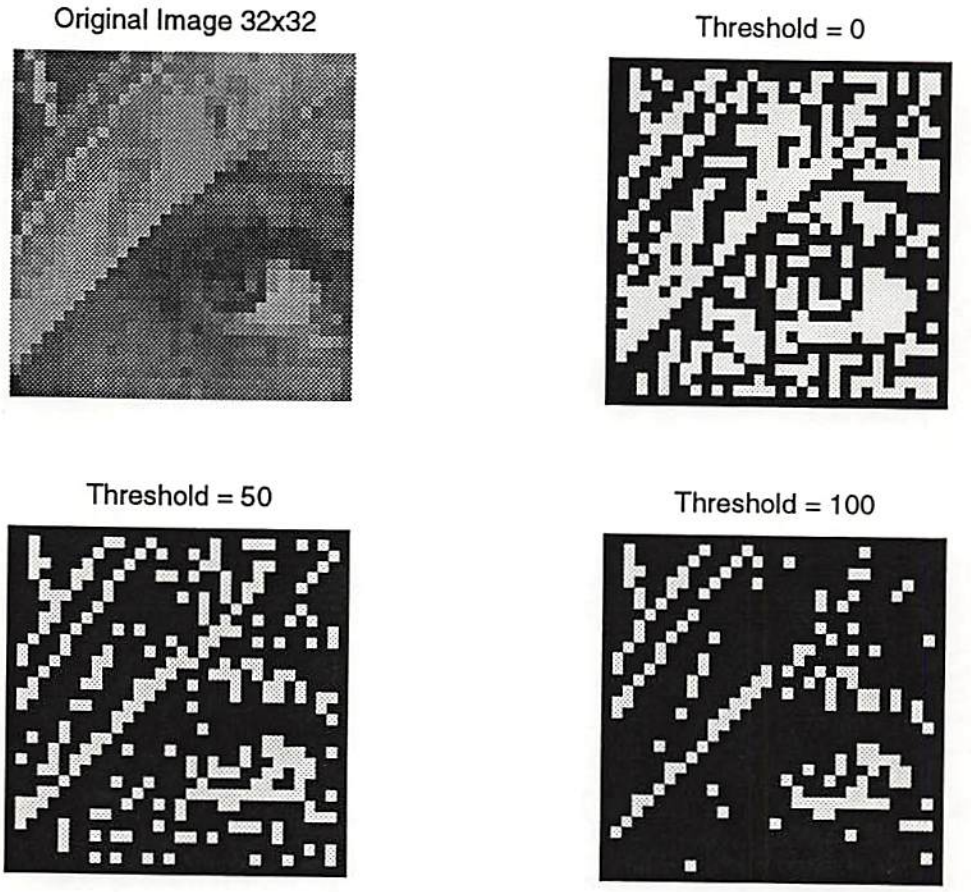


Figure 3.11: The output image after digital edge detection with different threshold values.

Chapter 4

The Parallel Architecture: Array Computing

For more than 40 years, the concept of the *von Neumann machine* has dominated the design of the computation architectures and systems [48]. Traditionally a computer consists of a single or multiple central processing units (CPUs), the memory and the controller circuits.

In contrary to the digital CPU design, a hybrid analog-digital scheme [49] can be used to construct a smart and compact computing machine. It is an architecture for an analog stored-program 2-dimensional array computer with distributed local memory, analog-and-logic computing modules, as well as local and global communication and control units. The global stored-program and the massively parallel execution are the key features of the array computing for implementing the processing algorithms.

A comparison among CISC (Complex Instruction-Set Computing) CPU, RISC (Reduced Instruction-Set Computing) CPU and the paralleled architecture is summarized in Table 4.1.

Table 4.1: Comparison among paralleled architecture, RISC, and CISC.

	CISC	RISC	Paralleled Architecture
Implmentation	Pure digital	Pure digital	Hybird (digital+analog)
Cycle per instruction (CPI)	2-15	<1.5	varied for logic instruciton and analog instruction
Addressing modes	12-24	3-5	2
Memory type	Digital memory	Digital memory	Digital memory + analog memory
I/O	Limited numbers	Limited numbers	large I/O numbers required
Generall-purpose registers number	8-24	32-192	depend on local cell number
Operation speed	20-100 MHz (gate-level)	100-250MHz (gate-level)	1-10 MHz (module level)
Data rate (connections/sec)	10^8	10^9-10^{10}	10^{14}-10^{15}

4.1 Review of the System Architecture

The global architecture[1] of the paralleled computing machine [49] is shown in Fig. 4.1. The computing cells are arranged on a regular grid. Each cell is controlled by the *global analog-logic programming unit* (GAPU). The GAPU receives the processing program and executes them in a specified order with the given template information. Due to the huge amount of inputs and outputs, the I/O lines can to be multiplexed and cells on the same row can share the same bus line. Therefore the overall input/output requirement can be minimized.

[1]The information described in this section was quoted from the publication by Roska and Chua [49].

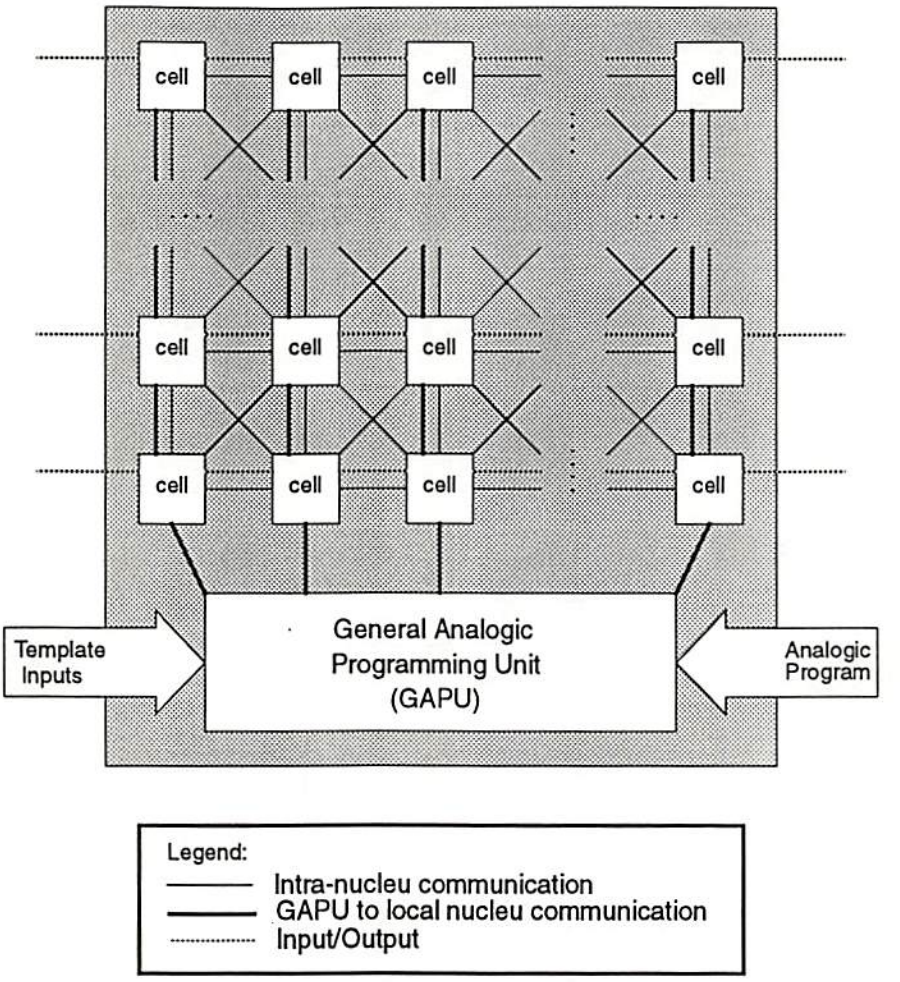


Figure 4.1: The overall view of the paralleled computing architecture [49].

There are several analog and digital circuit modules in the GAPU and the block diagram of a GAPU is shown in Fig. 4.2. Those modules that need to be processed in the analog form are marked with thick black lines. Control signals can be decoded to leverage the inter-chip wiring constraints. The functions of the modules in GAPU are described here.

- The *Analog program register* (APR) stores several analog processing instructions, i.e., the coefficient template elements. The information will determine what kinds of operation the chip would perform.

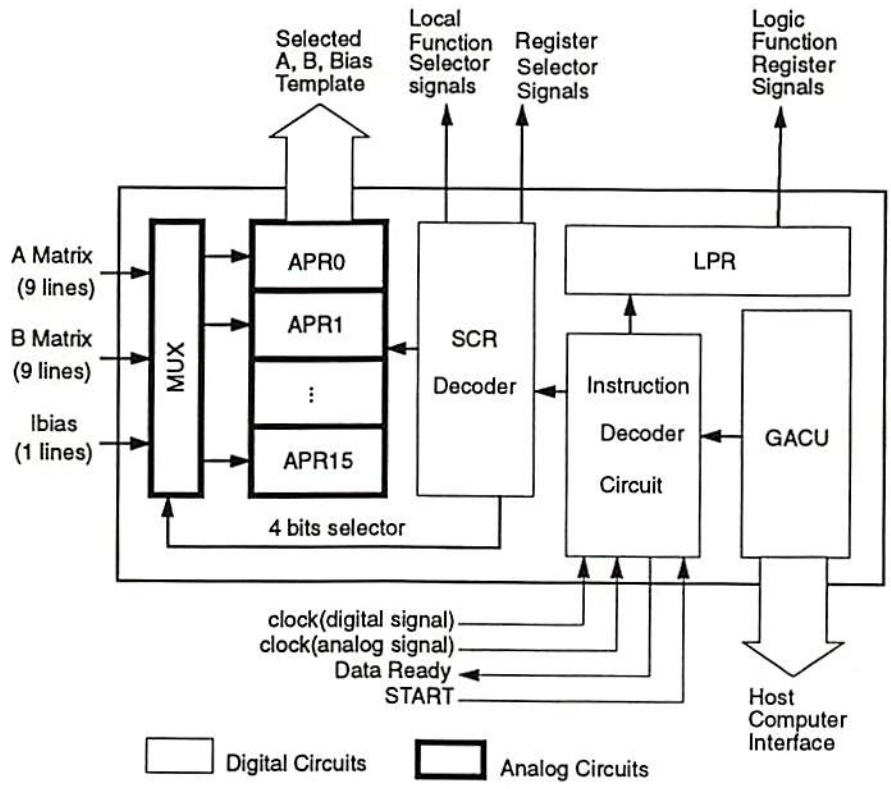


Figure 4.2: Block diagram of the GAPU [49].

- The *Logic program register* (LPR) stores the local logic-function selection, i.e., AND or OR operation. These can be implemented in a digital form although the compact neural network can do them in an "analog" form.

- The cell-configuration information is stored in the *Switch configuration register* (SCR). By switching to the appropriate configuration, each local cell can perform different kinds of tasks.

- The analog-digital programs are stored in the *Global analog-digital control unit* (GACU). Instructions are fetched and executed in sequence.

- The *Instruction decoder circuit* (IDC) provides the required circuits for host communication, signal synchronization for analog and logic operations, and instruction decoding.

Besides the analog circuit, several analog and digital components are needed to complete a core cell. The block diagram of each core cell is shown in Fig. 4.3. **LAM1, LAM2, and LAM3** are used for the input, initial state, and bias, respectively. There are four other general-purpose analog data registers which can be adjusted for different applications.

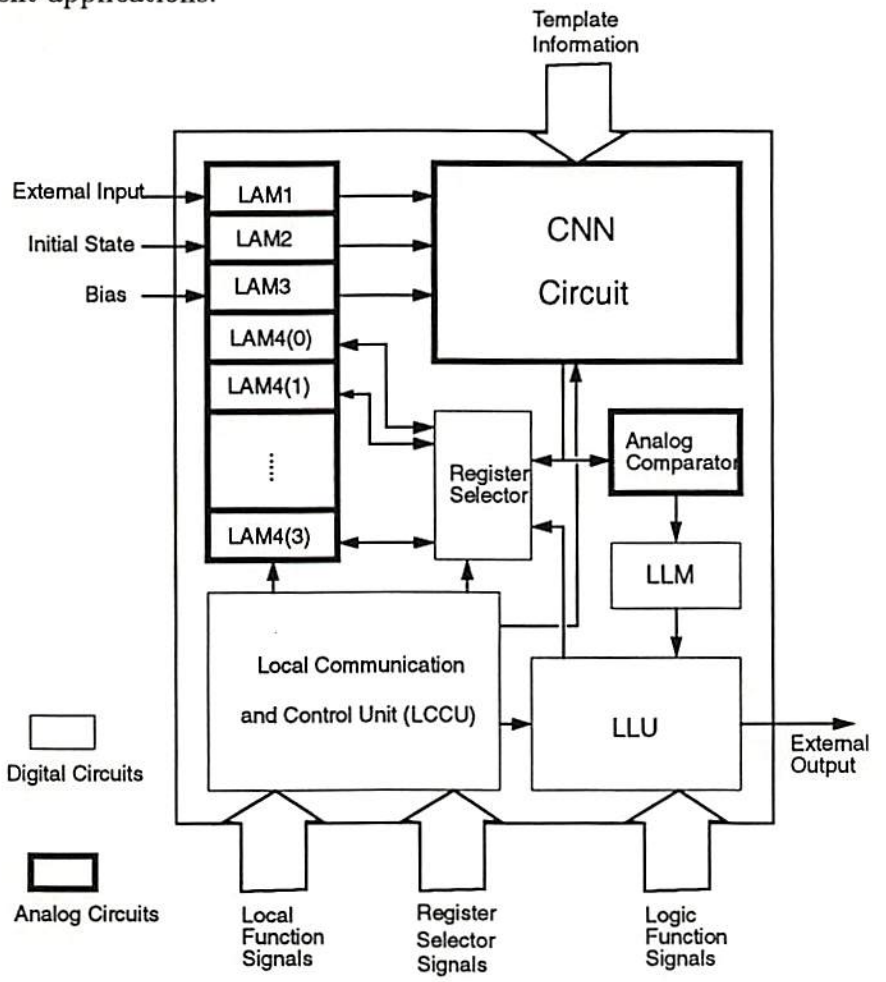


Figure 4.3: Block diagram of a core cell [49].

- The *Local analog memory* (LAM) and *Local logic memory* (LLM) greatly facilitate the implementation of several algorithmic steps on the cell array by using the local copy of the data. LAMs also act as buffers and they save the driving loading of the global signal outputs.

- The *Local communication and control unit* (LCCU) receives the code of the switch configuration from SCR and re-configures the proper switches in the local cells.

- The *Local logic unit* (LLU) reads the control signals from LPR and executes the logic functions.

- The *Local analog output unit* (LAOU) converts an analog output to the logic value for further processing.

4.2 The TW Programming Language

In order to make the paralleled computing architecture easier for users, a programming language **TW** is created which contains a small subset of the popular C programming language. It hides the information needed for the simulation and the functional definitions are the only necessary items to know. The compiler *cnnc* will translate the program into executable pseudo code. Therefore the users can focus on the high-level issues such as application development.

The **TW** program consists of three parts: program declaration, variable declaration, and the main execution statements. Variables can be integers (INT), floating-point numbers (FLOAT), grey-level images (AIMG) which contains analog (or continuous) values, or binary images (LIMG) which contains logic (0 or 1) values. The variables have to be declared before they can be used. Several predefined variables are **TIME**, **STEP**, **TOL**, **WIDTH** and **HEIGHT**. Their meanings are listed in table 4.2.

Table 4.2: The built-in variables used in the **TW** languages.

Variable Names	Meanings	Preset Value
WIDTH	width of the processing image	no preset value
HEIGHT	height of the processing image	no preset value
TOL	tolerance of the error	10000 (in $1e^{-9}$ unit)
STEP	stepsize of simulation	1000 (in $1e^{-6}$ unit)
TIME	max simulation time	100

The main execution statement portion starts with keyword **BEGIN** and stops with keyword **END**. Only signal assignments and the function assignments capabilities are implemented and all the statements will be executed sequentially. No loop statements are supported. A sample program is provided in the following.

```
PROGRAM TEST:

INT TIME,STEP,TOL,WIDTH,HEIGHT;
AIMG M1, M2, M3, M4;
LIMG M5;

BEGIN
TIME = 50;
        STEP = 10000;
        TOL = 1000;
        WIDTH = 24;
        HEIGHT = 24;
        M1 = LOAD(''wa.24'');
        M2 = EDGE(M1,M1);
        M3 = CCD_V(M2,M2);
        M4 = CCD_H(M2,M2);
        M5 = OR(M3,M4);
        OUT(M5,''wa.out'');
END
```

The first 5 statements override the system default values of the simulation settings. Then the file *wa.24* is loaded into the image register M1. After the edge

detection operation, the result will be saved in register M2. The horizontal and vertical connected component detection operations will be applied on the same image M2 and the results written to registers M3 and M4, respectively. The output is sent out after a logic-OR operation.

As seen in the example, The **TW** language is a function-oriented one and its capabilities rely on the functions supported in the template library, which is listed in section 4.5. Each function simulates the behavior that the controller GAPU configures a new connection and execute a certain task.

The **TW** language provides a very general solution for the behavioral simulation. Therefore, it can not only execute the single-layer simulation but also act for a multi-layer application [50, 51], where multiple steps of execution are needed. If further extended grammar is supported, it can also support the simulation for the time-multiplexing cellular neural network [52] simulation. However, this option was not implemented. It processes the input image block-by-block and the number of each block is the same as the number of PE's within the network.

4.3 Compiler for the Paralleled Computing Architecture

There are two parts for the compilation process: analysis and synthesis [53]. The analysis part breaks up the source program into constituent pieces and creates an intermediate representation of the source program, called "token". The synthesis part constructs the desired target program from the tokens. The flow of the process is illustrated in Figure 4.4.

In the analysis step, the source program is divided according to the grammar which has a hierarchical structure. The grammar is a subset of the C language

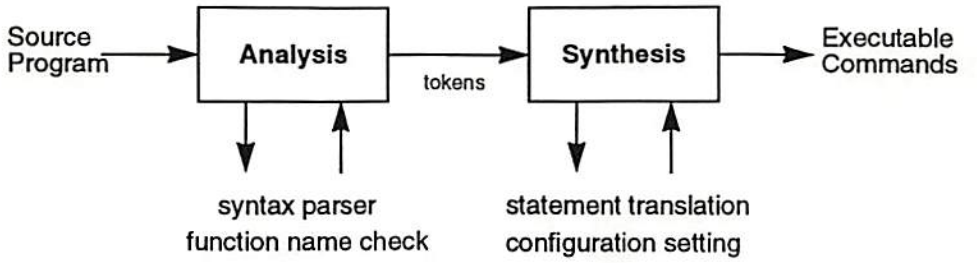


Figure 4.4: Flow of the compilation process.

grammar, which is defined in the Appendix A of the book, *The C Programming Language* [54]. In our implementation, this step is assisted by using the parser generator program *bison* which converts the grammar description of the **TW** language into a C program to parse that grammar. The parser will also build the variable table, check the syntax, and report any semantic error.

The synthesis step requires more specialized techniques. In order to achieve simplicity, only a few data types and assignment statements are supported. The compiler translate the statements into separate command files and the actual actions are done by invoking the behavioral simulator mentioned in the previous chapter.

In fact the **TW** language is a parallel programming tool where the natural of the paralleled computing architecture has strong influence on the compiler design. For example, the global behavior of the whole system is more important than the individual one. Therefore the global conditional-branch instructions might not be necessary. Saturation for each processing element has to be guaranteed before the execution, and the synchronization among the cells are important.

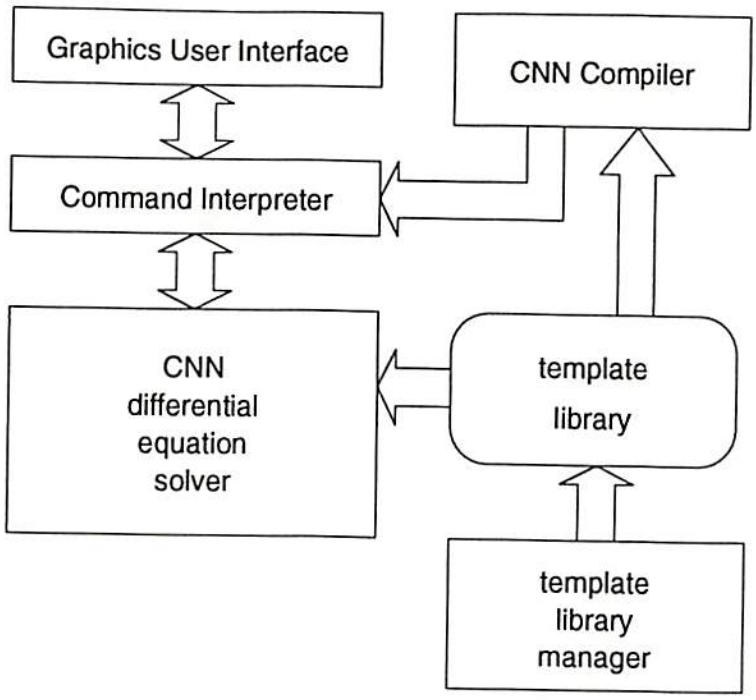


Figure 4.5: Framework of the paralleled architecture simulation environment.

4.4 Framework of the Paralleled Architecture Simulation Environment

The simulation environment consists of several components and is illustrated in Figure 4.5 The behavioral simulator *cnna* which provides the command interpreter and the differential solver engine is described in Chapter 3. The graphical user interface program *cnng* give the user a more interface to supply the simulation information. The choice of template to be used is selected by the scroll-bar items so the users do not have to memorize the function.

The compiler *cnng* will interpret the source code written in **TW** language and generate commands for the simulation. The template library plays a key role supporting various functions. The template manager helps to keep track of the library.

Table 4.3: Characteristics of simulation environment components.

Component Name	Command Name	Lines of C Codes
Behavioral Simulator	cnna	3,500
Graphics User Interface	cnng	1,500
Compiler	cnnc	5,000
Library Manager	libmgr	500

In our implementation, all the components are individual programs. Table 4.3 lists the characteristics of various components.

A suitable software-hardware co-design scheme is shown in Fig. 4.6. The **TW** language is written and translated into configuration information using the compiler, which retrieves the detailed built-in template information via the template manager or user-supplied information. Then those information can be sent to simulator for behavioral simulation or be translated into binary machine codes. A host, either PC or workstation, can send the binary codes to the universal machine chip for execution.

4.5 Appendix: Known Templates

A compilation of selected known templates for compact neural networks is presented in the appendix of this chapter. Many of the items were taken from the previous efforts described in [39, 40]. The detailed information of the selected templates is listed in Table 4.4.

Table 4.4: Function templates that can be used. M: Matrix of Input Image pixels; X: Don't care term; x_s and u_s are state and input of border cells [38].

Application	T_A	T_B	d	x(0)	u	x_s	u_s
Noise Filtering	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	M	X	0	X
Hole Filling	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-1	1	M	-1	X
Convex Corners Detector	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}$	-3	M	M	X	-1
Edge Detector	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}$	-2	M	M	X	-1
Laplacian Operator for Edge Detection	$\begin{bmatrix} 0 & -0.5 & 0 \\ -0.5 & 2 & -0.5 \\ 0 & -0.5 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-2.5	M	M	X	X
Connected Component Detection	$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & -1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	M	X	-1	X
Average	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	M	X	-1	X

Table 4.4 (cont'd)

Application	T_A	T_B	d	x(0)	u	x_s	u_s
Vertical Line Detector	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & 1 & -0.25 \\ -2 & 2 & -2 \\ -0.25 & 1 & -0.25 \end{bmatrix}$	-6.25	M	M	-1	-1
Horizontal Line Detector	$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	M	X	0	X
Diagonal Line Detector	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & -0.25 & -2 \\ -0.25 & 2 & -0.25 \\ -2 & -0.25 & 1 \end{bmatrix}$	-6.25	M	M	-1	-1
AND	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-1	M1	M2	X	X
OR	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	2	M1	M2	X	X
NOT	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	M	M	X	X
Isolated Pixels Detection	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 2 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}$	-4	M	M	X	-1

Table 4.4 (cont'd)

Application	T_A	T_B	d	x(0)	u	x_s	u_s
Isolated Pixels Elimination	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & -0.25 & -0.25 \\ -0.25 & 0.5 & -0.25 \\ -0.25 & -0.25 & -0.25 \end{bmatrix}$	0	M	M	X	-1
Delete one-pixel wide diagonal lines in the input images	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -0.25 & 0 & -0.25 \\ 0 & 0 & 0 \\ -0.25 & 0 & -0.25 \end{bmatrix}$	-2	M	M	X	-1
Delete one-pixel wide vertical lines in the input images	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -0.25 & 0 \\ 0 & 0 & 0 \\ 0 & -0.25 & 0 \end{bmatrix}$	-1.5	M	M	X	-1
Thresholding	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-1	M	X	X	X
Detect SW-NW Diagonals	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -2 & 0 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & -2 \end{bmatrix}$	-9	M	M	X	-1
Preserve Marked Objects	$\begin{bmatrix} 0.25 & 0.25 & 0.25 \\ 0.25 & 1 & 0.25 \\ 0.25 & 0.25 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1.75 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-5.5	M	M	X	-1
Thinning	$\begin{bmatrix} 0 & 0.4 & 0 \\ 0.4 & 1.4 & 0.4 \\ 0 & 0.4 & 0 \end{bmatrix}$	$\begin{bmatrix} 4.6 & -2.8 & 4.6 \\ -2.8 & 1 & -2.8 \\ 4.6 & -2.8 & 4.6 \end{bmatrix}$	-7.2	M	M	-1	-1

Table 4.4 (cont'd)

Application	T_A	T_B	d	x(0)	u	x_s	u_s
Peel from the left side	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-5	M	M	X	-1
Right Border Detector	$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 2 & -1 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & -3 \\ 0 & 0 & 0 \end{bmatrix}$	-2	M	M	-1	-1
Shadow Creation	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 2 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	0	1	M	-1	X
Vertical Shadows	$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	2	M	X	-1	X
Delete Two-pixel Wide Vertical Line	$\begin{bmatrix} -0.1 & 0.4 & -0.1 \\ 0.4 & 0 & 0.4 \\ -0.1 & 0.4 & -0.1 \end{bmatrix}$	$\begin{bmatrix} 0.2 & 0 & 0.2 \\ -3 & 2.5 & -3 \\ 0.2 & 0 & 0.2 \end{bmatrix}$	-5	M	M	-1	-1
Black Output Image	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	4	M	X	X	X
White Output Image	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	-4	M	X	X	X

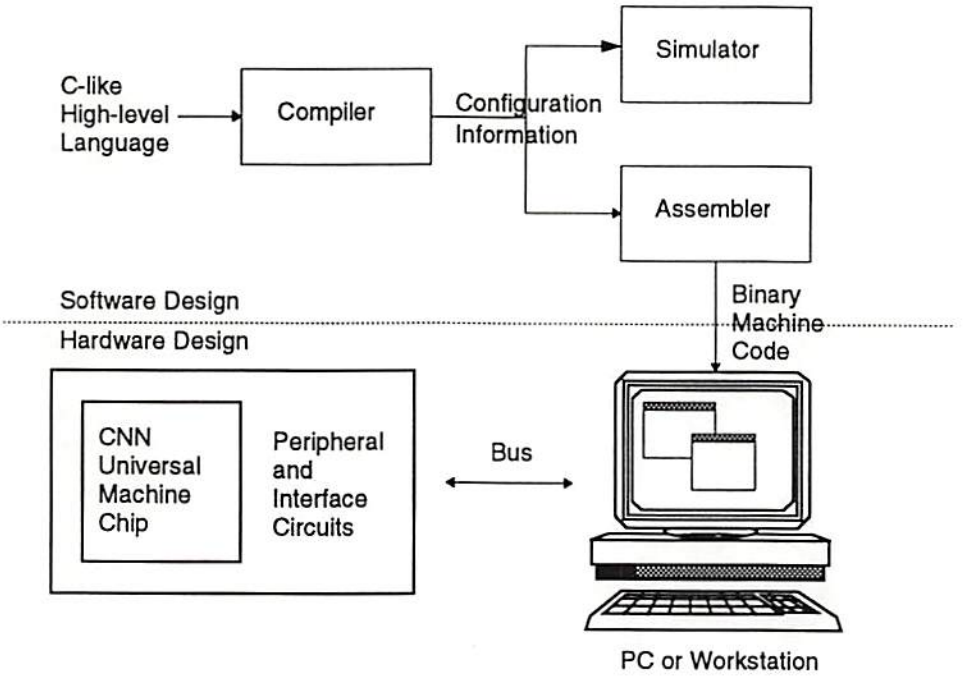


Figure 4.6: A possible software-hardware co-design scheme.

Chapter 5

A Versatile Video Array-Coprocessor

5.1 Introduction

There are certain spatio redundancy in each image frame and temporal redundancy among the video sequences. The Motion Picture Experts Group (MPEG) standards [4] which help to decrease the redundancies were established to pave a clear way for the multimedia applications. However the huge demands on the processing power and memory access keep pushing the rapid advances in both architectural innovation and VLSI design for a cost-effective implementation.

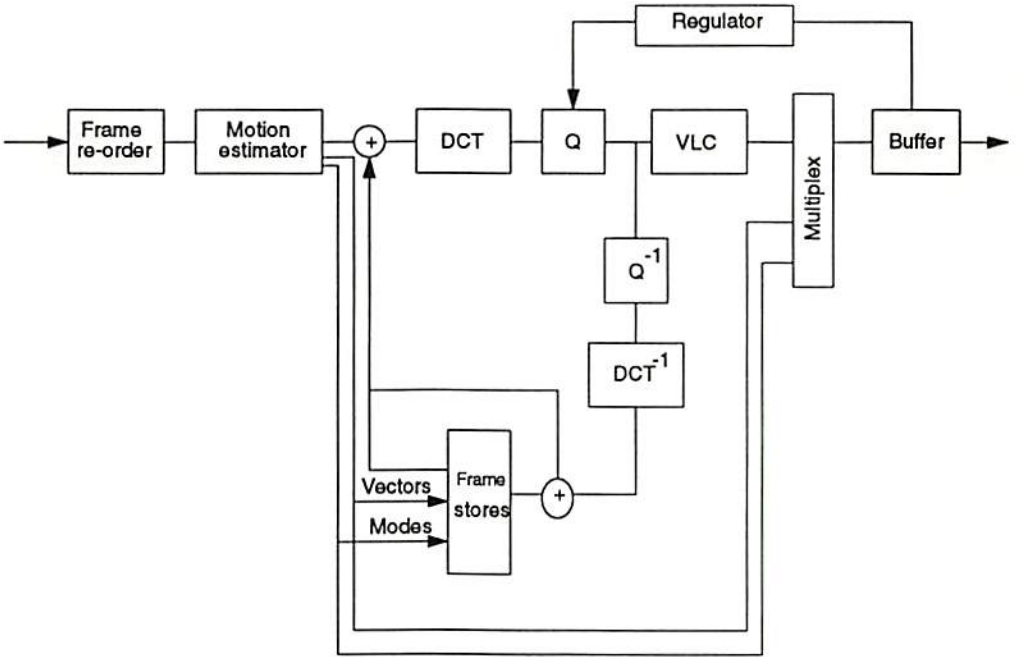
Transform coding, especially the Discrete Cosine Transform (DCT) is widely used in JPEG [3], MPEG [4], CCITT H.261 [5] standards to reduce the spatial redundancy. Direct computation of the 2-D $N \times N$-point DCT takes $O(N^4)$ multiplications [55]. With the help of fast algorithm and the row-column decomposition techniques, the computation requirement is reduced substantially and becomes more economic from the VLSI point of view [8].

In motion compensation, the previous frame is searched to find a block of data that is the best match of the block of the current frame. The offset is represented as a motion vector. A subtraction is performed and the residual prediction error is

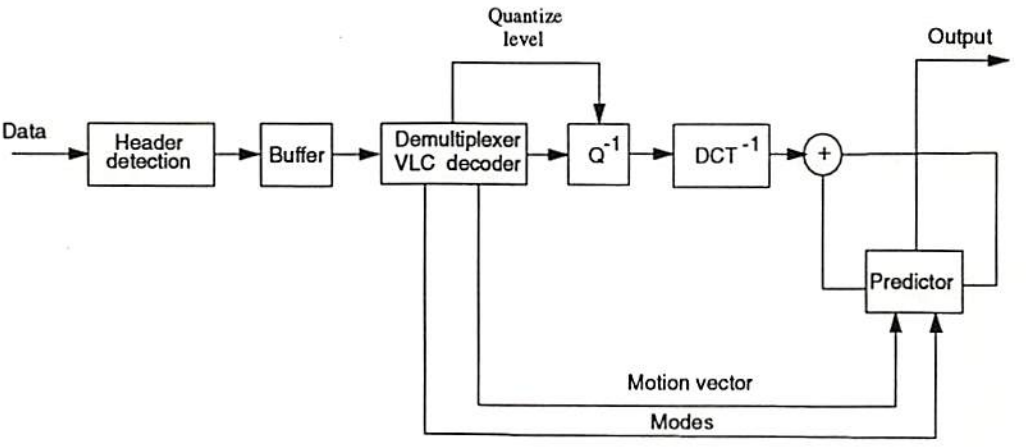
quantized and encoded. The data-representing motion vectors form the prediction of the current frame.

Figure 5.1 shows the MPEG encoding and decoding functional blocks. Among the real-time video encoding tasks, the above two functions, motion estimation, and 2-D DCT/IDCT are the two most computation-intensive steps. For a much simpler decoder, the major speed limitation comes from the IDCT computation. Even with the rapid advances in VLSI technologies, the state-of-the-art general-purpose microprocessor or digital signal processor still cannot handle the jobs efficiently. Some specific motion estimator [56, 57, 58, 59] or DCT [60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73] processors were reported but they are either too slow for real-time MPEG-2 [74, 75] or HDTV applications [76], or occupy too large area to be cost-effective.

In implementing the AVP-III video codec, an array-structure processor had been designed to efficiently work as the search engine [77]. By applying a new 2-step searching scheme instead of the full search scheme, it is shown that only less than 25% of the computing resource is used and the rest of the time the processor simply stays idle. That is a waste of the computing resource. In this chapter, an innovative mapping based on the multiplication-accumulation technique using the concept similar to distributed arithmetic [78, 79] is described. It helps to fully utilize the array processor for the high performance 2-dimensional discrete cosine transform or inverse discrete cosine transform (2-D DCT/IDCT). The array structure has been maintained and minimum additional hardware is added, which leads to compact and cost-effective VLSI design. Pipelining and parallel execution helps to achieve high throughput rate while the latency is still kept low. According to the analysis, it takes about another 50% of the computing resource for the 2-D DCT operation for the encoding process.



(a)



(b)

Figure 5.1: The functional block diagram of the MPEG video compression standards [4]. (a) The encoding process. (b) The decoding process.

Since the 2-D inverse discrete cosine transform (2-D IDCT) can be applied with the same structure except different coefficients, the array processor can also be used for IDCT operation in decoder. The design is not limited to DCT operations. The same concept can be further extended to 1-D or 2-D filters with additional hardware which also provides certain programmability. The whole design can be used as a video coprocessor to off-load the computation from the host CPU or as a key component in the real-time video codec set-top box.

5.2 Basics of 2-Dimensional Discrete Cosine Transform

The 2-D Discrete Cosine Transform (2-D DCT) has excellent energy compaction for highly correlated data. Thus it is widely used in the image compression scheme and the N × N 2-D DCT is defined as [80]

$$X(u,v) = \frac{2}{N}C(u)C(v)\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x(i,j)cos(\frac{(2i+1)u\pi}{2N})cos(\frac{(2j+1)v\pi}{2N}), \quad (5.1)$$

where $x(i,j)$ $(i,j = 0, 1, ..., N-1)$ is the image pixel value and $X(u,v)$ $(u,v = 0,1,..., N-1)$ is the transformed coefficient. $C(0) = 1/\sqrt{2}$, $C(i)=1$ if $i \neq 0$.

The direct implementation of the 2-D DCT is not practical since it involves multiplication with the computing complexity of $O(N^4)$. Because the 2-D DCT kernel is known to be orthogonal and separable [80], the original formula can be split-up into two 1-D DCT's by the row-column decomposition technique.

Usually the 1-D fast algorithm [81, 82, 83] is applied to further reduce the total number of multiplications needed by lumping terms earlier as done in the DFT case. But the butterfly structure is not suitable or efficient to map into regular array

structure. Therefore, the formula of the fast algorithm is restored and the 1-D DCT (N=8) can be described in matrix form as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} A & A & A & A & A & A & A & A \\ D & E & F & G & -G & -F & -E & -D \\ B & C & -C & -B & -B & -C & C & B \\ E & -G & -D & -F & F & D & G & -E \\ A & -A & -A & A & A & -A & -A & A \\ F & -D & -G & E & -E & G & D & -F \\ C & -B & B & -C & -C & B & -B & C \\ G & -F & E & -D & D & -E & F & -G \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \tag{5.2}$$

where A = $cos(\pi/4)$, B = $cos(\pi/8)$, C = $sin(\pi/8)$, D = $cos(\pi/16)$, E = $cos(3\pi/16)$, F = $sin(3\pi/16)$ and G = $sin(\pi/16)$. x_i (i=0, 1, .., 7) is the pixel data and X_u (u=0,1,.. 7) is the transformed coefficients. The term $\sqrt{2}$ had been absorbed into the matrix as term A and additional division by 2 can be easily implemented by hardware. Later it will be demonstrated that the DCT formula in this form is more suitable for an 8×8 array structure implementation.

Similarly, 2-D inverse DCT (IDCT) is defined as

$$x(i,j) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)x(i,j)cos(\frac{(2i+1)u\pi}{2N})cos(\frac{(2j+1)v\pi}{2N}), \tag{5.3}$$

where $C(u)$, $C(v)$ have the same definitions as in DCT. And the 8-point 1-D fast IDCT can be described as

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} A & A & A & A & A & A & A & A \\ D & E & F & G & -G & -F & -E & -D \\ B & C & -C & -B & -B & -C & C & B \\ E & -G & -D & -F & F & D & G & -E \\ A & -A & -A & A & A & -A & -A & A \\ F & -D & -G & E & -E & G & D & -F \\ C & -B & B & -C & -C & B & -B & C \\ G & -F & E & -D & D & -E & F & -G \end{bmatrix}' \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}. \tag{5.4}$$

The constant A, B, C, D, E, F, G are same as used in (5.2).

5.3 Multiplications Using Adders

As seen in (5.2), the operations needed are multiplications and additions. Since the area of each PE has to be kept small, a dedicated multiplier inside each PE is not appropriate. One way to get around of it is to make full use of the two adders already existing in the PE. In the following, the multiplier-accumulator scheme constructed by adders only is introduced where the original idea is very similar to the distributed arithmetic (DA) [78, 79]. DA is a very efficient means for computation that is dominated by inner products. The original architecture can be maintained with a few extra components.

Taking the operation of an 8-bit unsigned number (A) multiplied by another 8-bit unsigned number (B) as an example,

$$\begin{aligned} Sum &= A \cdot B = \sum_{k=0}^{3} \left[A \cdot B_{2k}^{2k+1}\right] \cdot 2^{2k} \\ &= \left[\left[A \cdot B_6^7 \cdot 2^2 + A \cdot B_4^5\right] 2^2 + A \cdot B_2^3\right] 2^2 + A \cdot B_0^1, \end{aligned} \tag{5.5}$$

where B_l^k means the $k-th$ bit to $l-th$ bit in B. The second operand B is decomposed into four 2-bit numbers, which will be used as multiplicand in the successive four stages. The decomposed stages of the multiplications are shown in Figure. 5.2. In stage one the two most significant bits B(7,6) is multiplied by A and the result sum0 propagates to the second (next) stage. In the second stage B(5,4) is multiplied with A and the result is then accumulated with sum0. If the process is continued, and the multiplication is finished after four stages.

Let us take a closer look at the 2-bit multiplication. The operation can be achieved with just one adder. Assume that each of the 2-bit numbers contains the most significant bit (MSB) and the least significant bit (LSB). The MSB will decide whether a left-shift by 1 bit is necessary or not, whereas the shift-left operation

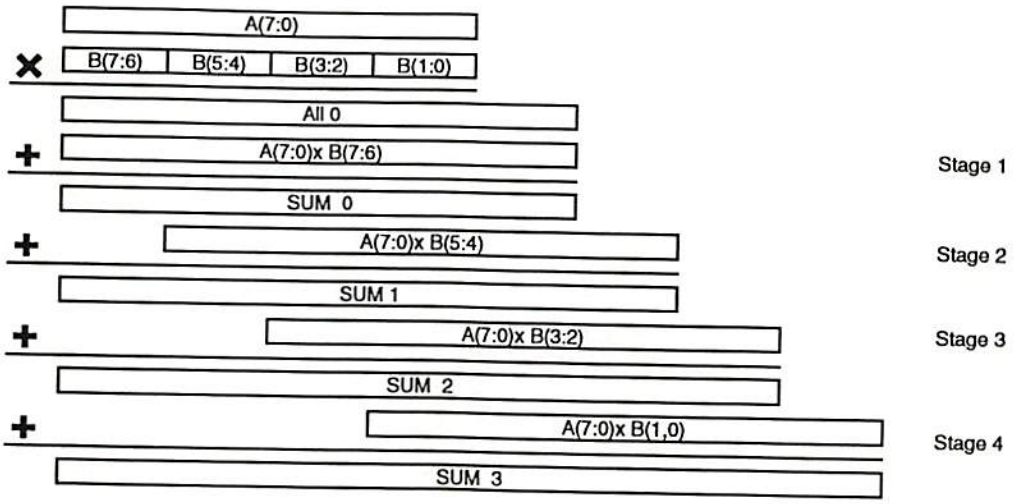


Figure 5.2: Multiplication using multiple adders.

simulates a x2 multiplication. The LSB decides whether a copy or zero will be sent to the adder. The pseudo-code expression that helps to explain the operation is described as

```
if (MSB = '1') then
        op2 := A * 2;
else
        op2 := 0;
end;
if (LSB = '1') then
        op1 := A;
else
        op1 := 0;
end;
sum := op1 + op2;
```

The scheme is illustrated in Figure 5.3 and the summation (sum) is the result of the multiplication.

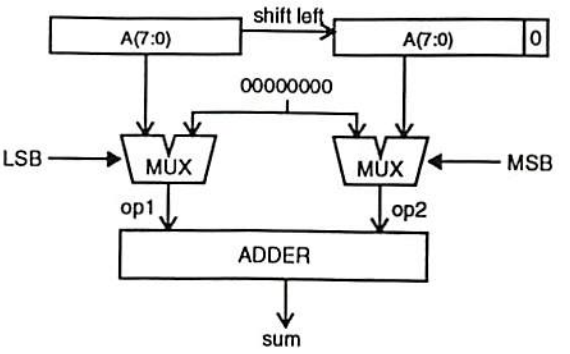


Figure 5.3: 2-bit multiplication using one adder.

5.4 Mapping of 1-D DCT Algorithm on the Video Coprocessor

There are two ways to partition the array for the multiplication and accumulation operations heavily used in the DCT operation. One is row-oriented and the other is column-oriented. Figure 5.4 shows the different configurations.

The main reason why the column-oriented configuration is preferred is that the data-path along the row is smaller and no shifting operation is necessary before accumulation. Take an 8-bit x 16-bit unsigned number multiplication using row-oriented configuration as an example. If regular structure is required, the data path along the column must be the maximum data bandwidth which is 8+16=24 bits. The bandwidth can be reduced for the left-side PE's but irregular structure will hamper the compact design of the PE array. In contrast, the data-path would be only 10+3=13 bits wide for the column oriented configuration where the 3-bit is contributed by the 8 additions.

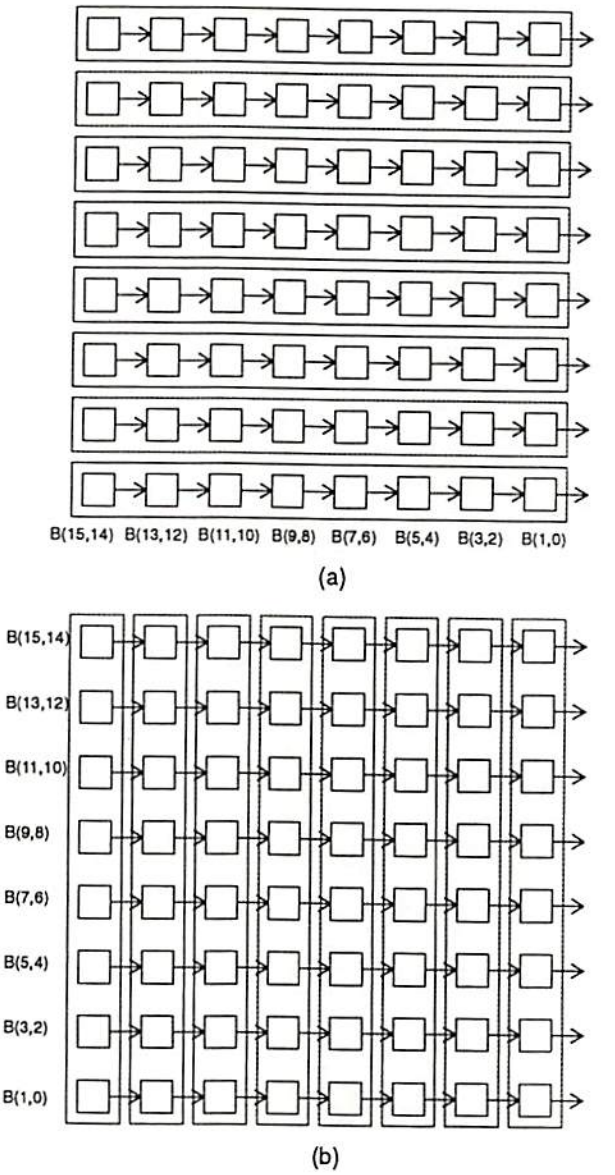


Figure 5.4: Two different configurations of the ADS-PE array: (a) Row-oriented configuration. (b) Column-oriented configuration.

If the 1-D DCT formula in matrix is expanded into individual equations, there will be eight of them. The first transformed coefficient can be achieved by the equation,

$$\begin{aligned} X(0) &= A \cdot x(0) + A \cdot x(1) + A \cdot x(2) + A \cdot x(3) + A \cdot x(4) \\ &+ A \cdot x(5) + A \cdot x(6) + A \cdot x(7). \end{aligned} \tag{5.6}$$

If x(0) .... x(7) are the pixel values of the first row in the image macro block, the result of the summation is X(0,0). If the second row in the image are fed next, the result of the summation will become X(1,0). If the pixel data keep feeding until the eighth row, the transformed coefficients of the first column come out in order. No additional transpose hardware is needed if the data are fed in this way.

In order to get the transformed second column, the DCT coefficients have to be updated. The new equation is,

$$\begin{aligned} X(1) &= D \cdot x(0) + E \cdot x(1) + F \cdot x(2) + G \cdot x(3) + (-G) \cdot x(4) \\ &+ (-F) \cdot x(5) + (-E) \cdot x(6) + (-D) \cdot x(7). \end{aligned} \tag{5.7}$$

If the image data are fed again from the first row until the eighth row, the transformed coefficients of the second column $X(0,1), X(1,1), ..., X(7,1)$ will come out.

By using the method described above to feed in the image pixels, the outputs of the transformed data have been transposed. Hereafter we will call the steps working on each of the rows as "pass1". From hardware implementation point of view, with the help of shift registers, output of each column can be gathered and sent out at a time when all components come out. Figure 5.7 shows the schemes for the serial-to-parallel shift register array used in the post processing unit. The summer is a 32-bit adder and the result will be rounded to 16-bit signed representation.

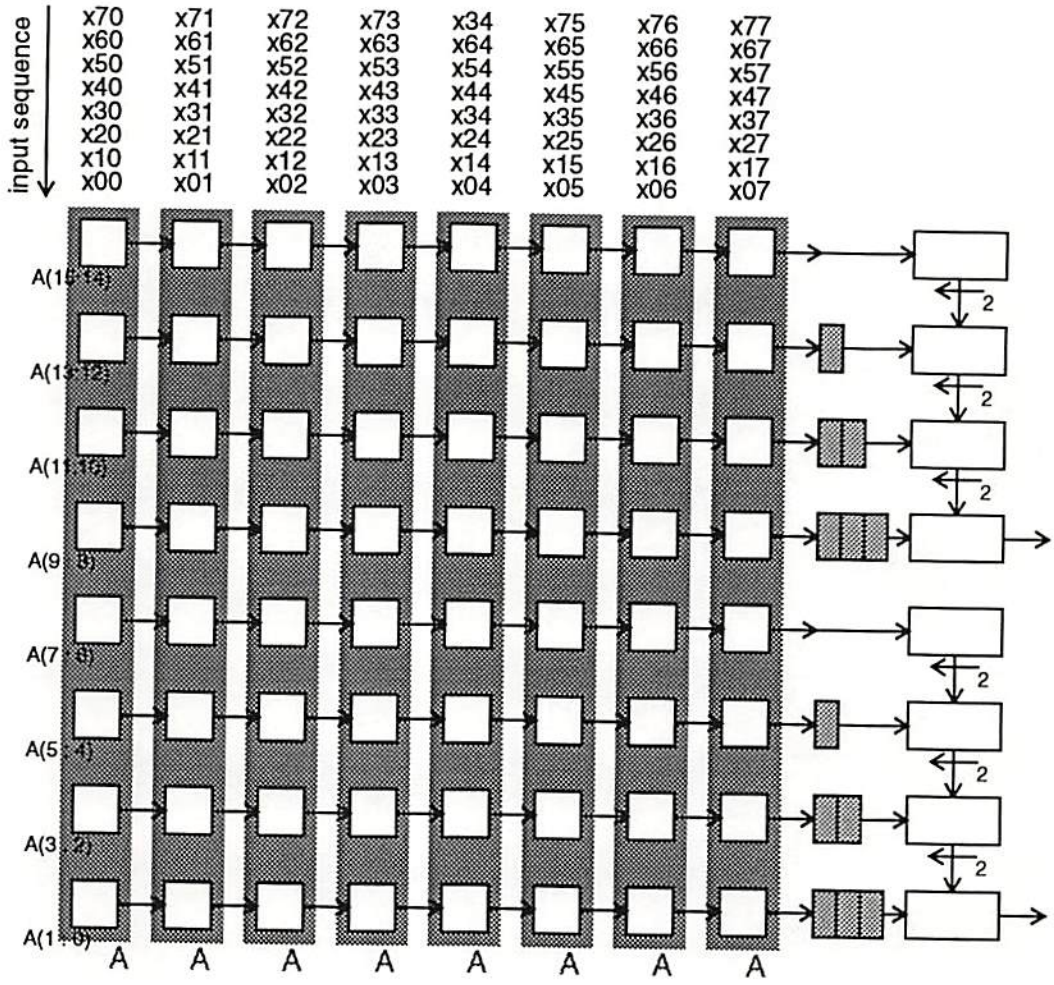


Figure 5.5: Configuration for the DCT Pass-1 while calculating the first column of the transformed data. The two outputs will be summed together (not shown).

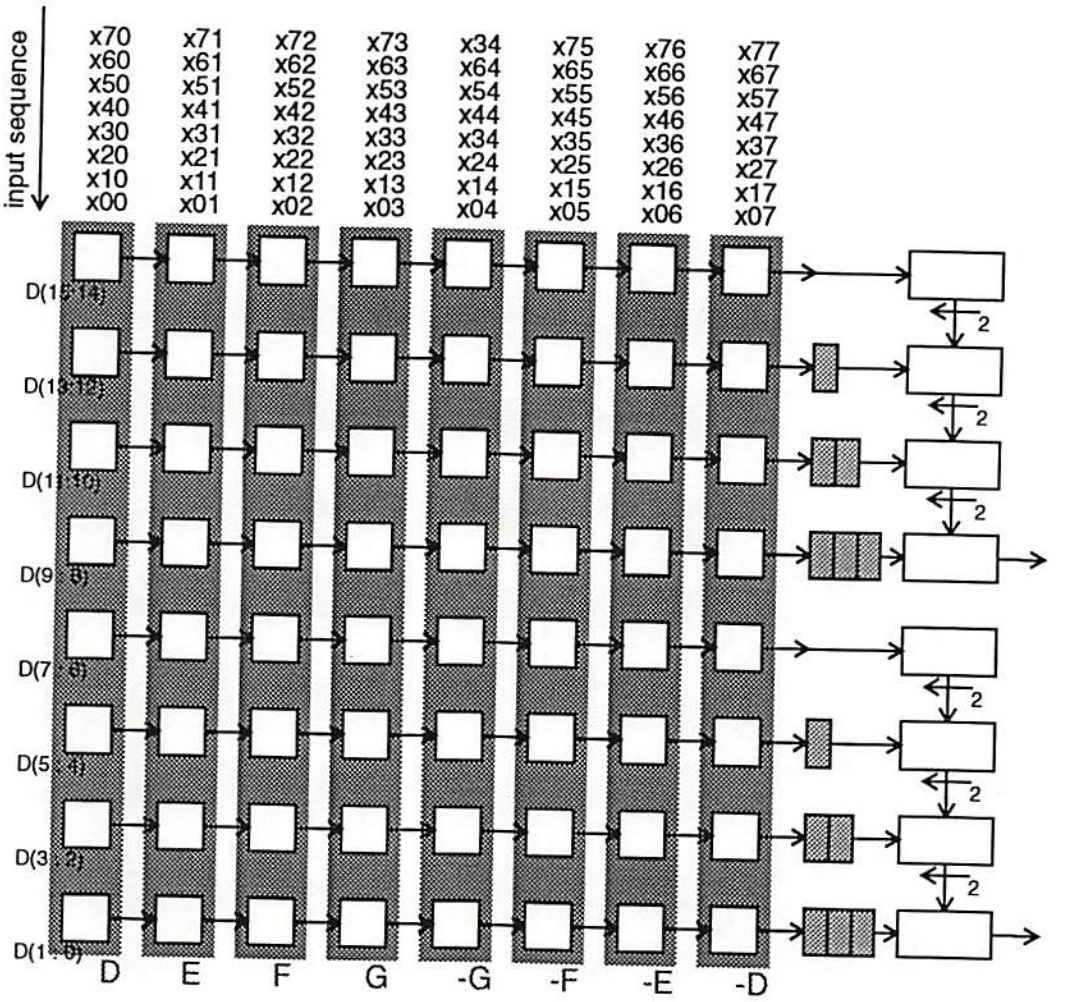


Figure 5.6: Configuration for the DCT Pass-1 while calculating the second column of the transformed data. The two outputs will be summed together (not shown).

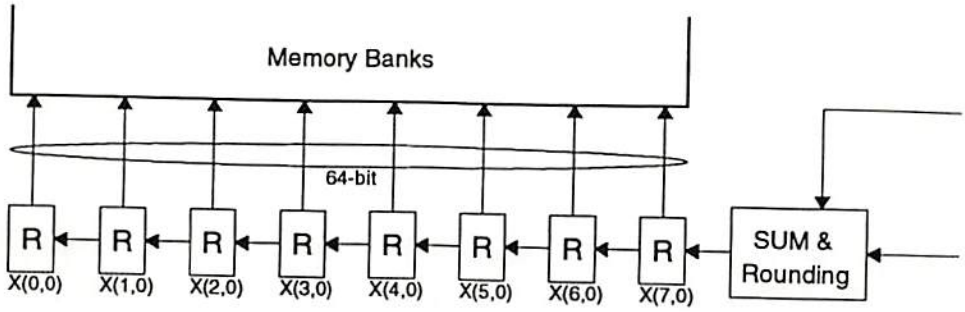


Figure 5.7: Schematic diagram of the serial-to-parallel shift register array, where the transformed outputs have been automatically transposed.

The operations on all columns (or "pass-2") are very similar. The transformed coefficients which are 16 bits wide from pass-1 are fed into the ADS-PE array with the same set of DCT coefficients. The rounding is done in the PPU and the final result has to be scaled back to 13-bit representation.

In the pass-2 where the length of the incoming data is more than 8 bits, only 8 bits (either the higher 8 bits or lower 8 bits) will be sent at a time due to the design of the data path. For signed number, only the first byte contains the sign information and the rest of bytes are unsigned numbers. Therefore, sign information has to be provided to help the PE's determine the data type when the multiplication operations are executing. A 4-bit circular-shift register can be used as shown in Figure 5.8. For different lengths and types of data, a preset value will be loaded into the shift register before the start of the operation. The sign signal is multiplexed with the CHANGE signal used for motion estimation operation to save the extra wiring. The propagation path is a little different with the one used in motion estimator. The most significant byte is loaded first, with the appropriate sign signal indicating whether it is a signed number or not. There is a special "sign function circuit" in the PE which can help to translate the input data into signed number format where all the addition operations hereafter are executed in signed-number representation.

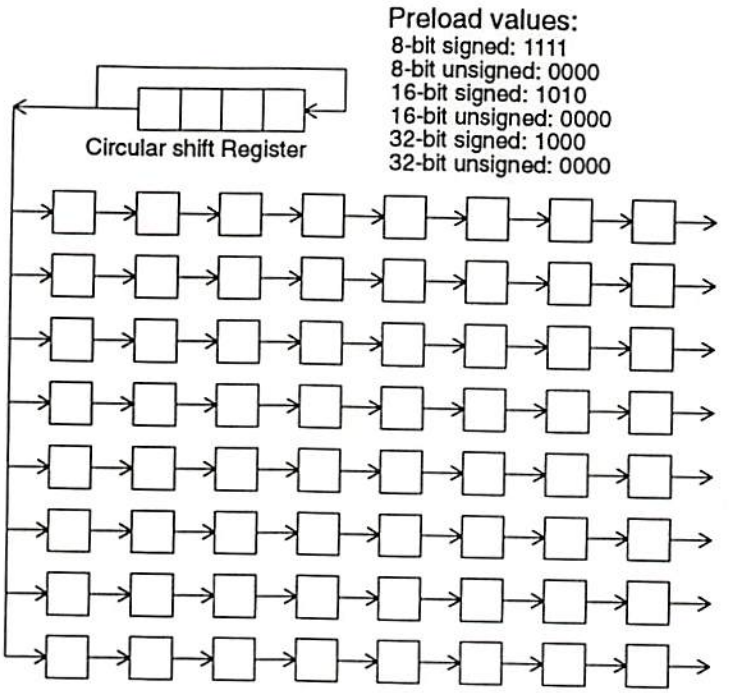


Figure 5.8: Signal flow of the sign information bit and table of preload values for different type of data.

The sign signal will propagate through the array just at the time when the data comes out of the delay elements. The circular operation of the shift register saves the effort to reload the preset value again and again.

5.5 Overview of System Architecture

The system architecture is shown in Figure 5.9, which consists of 5 major components:

ADS PE array The array contains 8x8=64 ADS PEs. Each PE maintains its simple structure which contains two adders (one 10-bit and the other one 13-bit), several registers and latches. Additional delay elements lie on top of the ADS-PE array to provide the necessary delay for the accumulation.

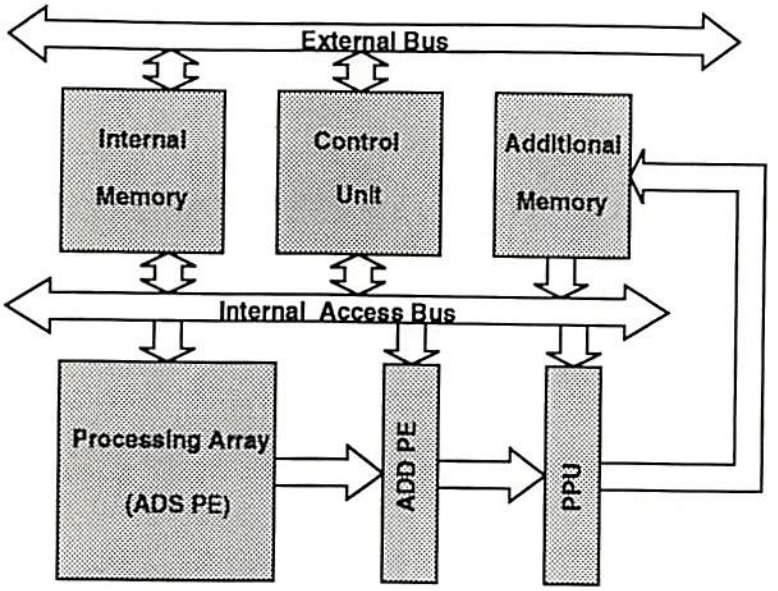


Figure 5.9: Block diagram of the video coprocessor system.

ADD PE array This array contains 8 ADD PEs. Each PE contains one 24-bit adder. Several delay elements are added for accumulation purpose.

Post Processing Unit (PPU) The PPU contains four **MIN/MAX** units for motion estimation operation and one 32-bit adder to accumulate the final result of DCT operation. After rounding, the data are sent to a serial-to-parallel shift register array. The data-path of each register is 16 bits. The external control signal can decide whether higher bytes or lower bytes are to be sent back to memory.

Video RAM (VRAM) Currently the size of the video RAM is 2K byte which is organized into 8 banks. Therefore the output data-path is 64-bit wide. The additional memory shown in Figure 5.9 is actually part of the internal video RAM and shares the same address space.

Control Unit Control Unit receives the commands from the host. It coordinates the operations among all the other components and schedules the execution sequence for different tasks. It is based on the Finite-State-Machine (FSM) concept.

The ADS-PE array maintains the original structure which is shown in Figure 5.10. In order to accommodate the new functions, slight modifications of the ADS-PE are necessary.

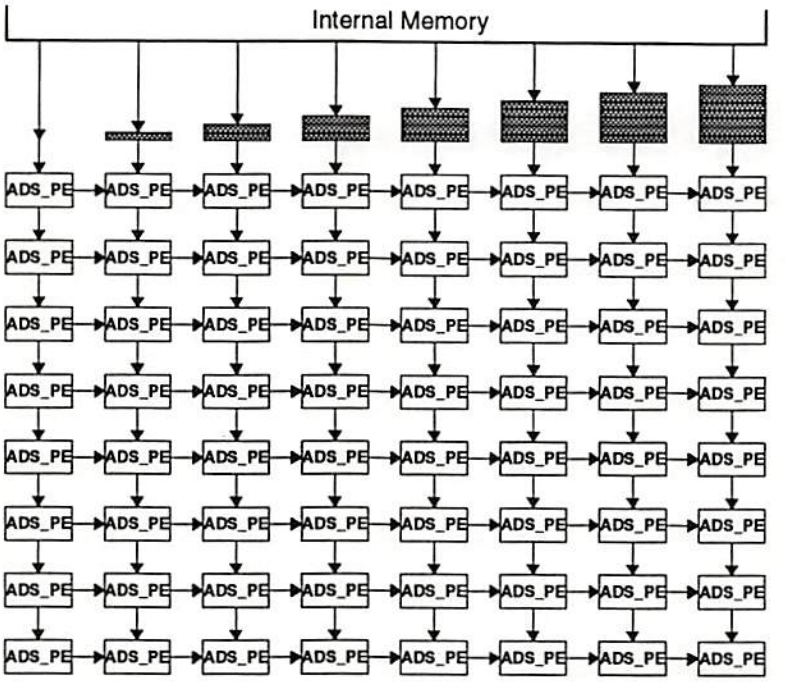


Figure 5.10: The systolic-array style connection of the ADS-PEs.

The input will be translated into signed-representation if it is an unsigned number. That can simplify further operations. The bandwidth of the first adder will be increased to 10 bits and one additional sign bit. The bandwidth of the second adder will be increased to 13 bits and additional one sign bit. The LOAD and CHANGE signal will be used for different purposes in different operations. That can save the

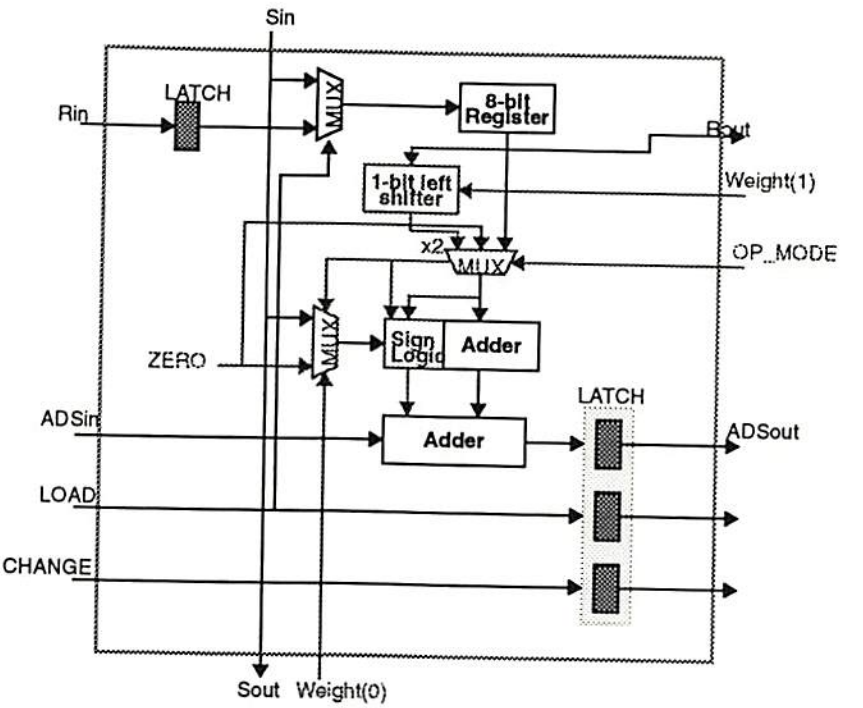


Figure 5.11: Detailed schematic diagram of the ADS-PE.

wire space for the layout routing. The detailed schematic diagram of the ADS-PE is shown in Figure 5.11.

The configuration of the ADD-PE vector and PPU is shown in Figure 5.12. The ADD-PEs are separated into two sets, the top four ADD-PEs and the bottom four ADD-PEs. This is because it can reduce the skew cycles before the data coming out from 8 to 4 cycles, and the number of delay elements from 28 to 12.

If it is operated for motion estimation, the accumulation still propagates through the whole 8 PEs. Since the range of the sum-of-difference is smaller, 24-bit adder will be enough. Besides, the smaller data-path width the adder uses, the faster the adder will be. In Figure 5.12 the dotted boxes indicate the new addition to the design. Whether the inputs of ADD-PE come from ADD-PEs directly or from the delay elements will be selected by MUX, which is not drawn here.

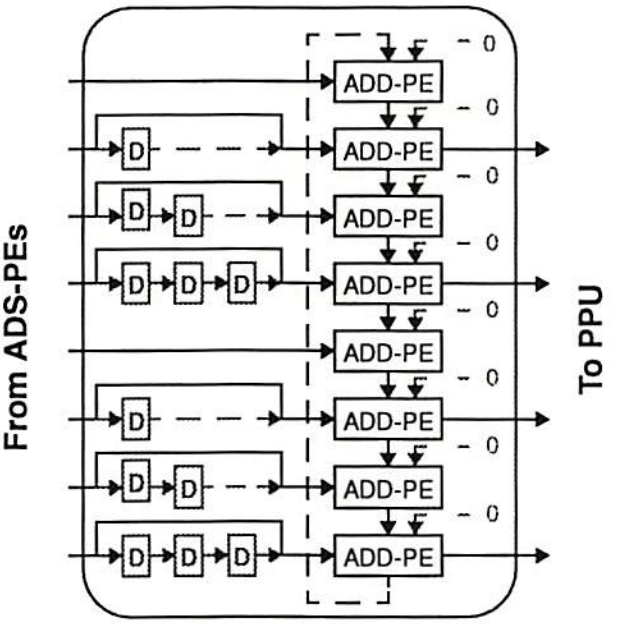


Figure 5.12: Schematic diagram of the ADD-PE.

Basically the ADS-PEs and ADD-PEs behave in the similar way no matter they are used in the motion estimation function or DCT function. But the operation of PPU is different for different functions. Figure 5.13 shows the block diagram of the PPU. When it is used in the motion estimation function, only the MIN/MAX blocks are activated to select the desired minimum results. When it is used for DCT operation, the 32-bit adder and the shift-to-parallel register arrays are in charge.

The operation of motion estimation is an asymmetric one. A lot of data comes in but only the minimum 3 sum-of-differences values are kept and sent back to the host. The I/O cycles are minimal. On the other hand, the DCT operations need a lot of I/O cycles to write the data back. The write-back can happen in two occurrences:

1. Use the interval between the generation of each transformed row, or

2. Use the interval between the generation of one entire block.

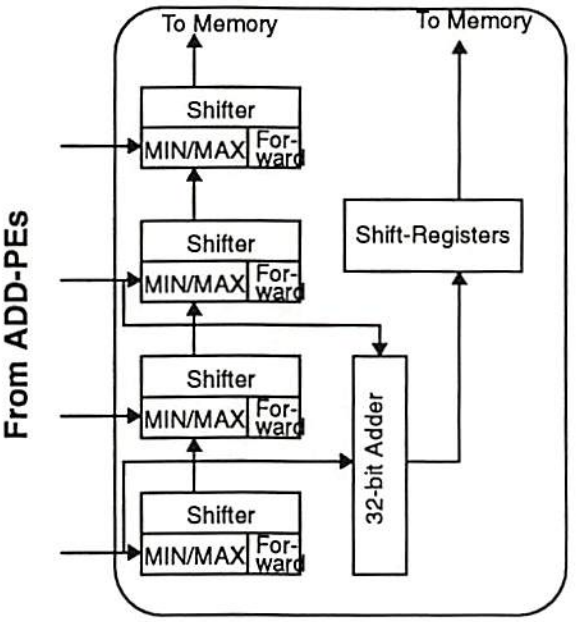


Figure 5.13: Schematic diagram of the PPU.

In the first method, no buffer is needed to temporarily store the transformed coefficients. However, extra cycles might be wasted to write the data back unless double-buffer memory is available. In the second method, buffers which can hold 64 transformed data (16 bits wide) are required.

The control unit (CU) is the coordinator of the whole coprocessor. It receives operation command from the host and extracts the necessary information. When a certain operation is initiated, CU presets the contents in the counters, start the operation micro-sequences and send control signals to the memory, ADS-PEs, ADD-PEs and PPU. When the operation is completed, it acknowledges the host about the completion and waits for further commands.

Depending on the trade-off among performance and programmability, the co-processor can be programmed in two different ways. The first one is for dedicated functions, such as motion estimation, 2-D DCT (both pass-1 and pass-2), or vector

quantization. Those are hardware-programmed routines in order to achieve maximum performance for the computation-intensive tasks. The user can only change the memory address where the input image pixels are stored. On the other hand, the filtering operation needs the software programmability to adapt to the functions the coprocessor is asked to do. The users can provide detailed information, such as filter length, input data type, and so on, via software routine and write all the settings to certain pre-defined registers.

The control unit is further divided into several macro blocks and the schematic diagram is shown in Figure 5.14. The op-code interpreter reads in the command, activates one of the core to start the execution. An internal control bus is used to share the control lines among those cores. In the final stage, the control signal generator (CSG) generates the necessary control signals to ADS-PEs, ADD-PEs, and PPU, while the address generator (AG) controls the read/write function of the internal video RAM by supplying appropriate address. At the same time, weight generator (WG) updates the weights if necessary.

5.6 Memory System

In order to retain high throughput of the PE array, it is desirable to spend most of the time to pass the data from memory to PE array. Besides, for the DCT operation, the data flow amount is asymmetric, i.e., the rate that data comes into the PE array is far more than the amount that comes out. Another set of register files is used to temporarily keep the data in the controller and the data is written back at certain intervals.

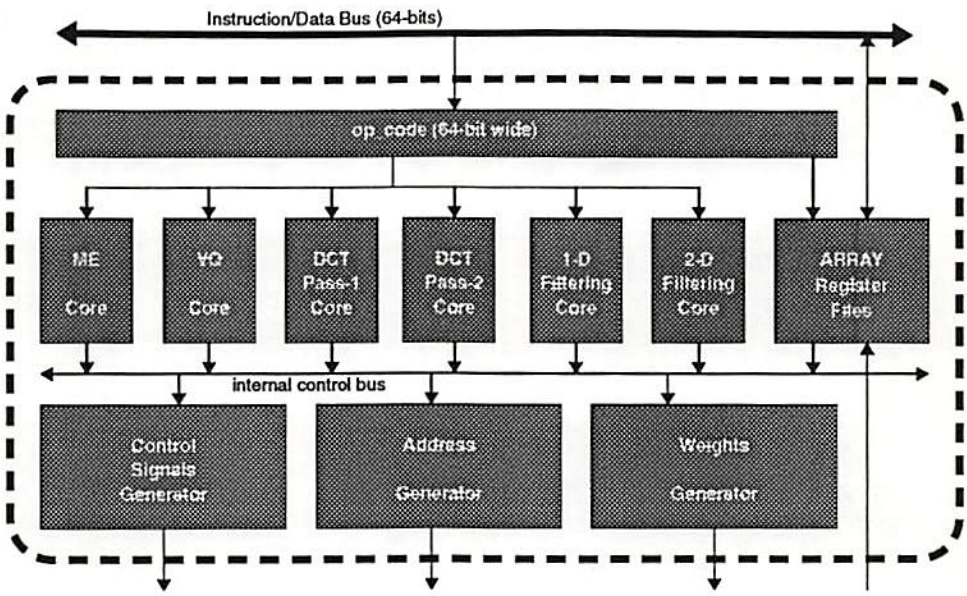


Figure 5.14: Diagram of the ARRAY control module.

The memory model used here is very simple. When EN = '1' and RW = '0', there will be 64-bit data out from the memory. If EN = '1' and RW = '1', 64-bit data on the bus will be written to the memory according to the address.

A non-linear address mapping can be used to minimize the space that the video RAM occupies. However, a new design is shown in Figure 5.15.

There are 4 row decoders and 4 column decoders which are designed to work independently. When it is operated for motion estimation operation, the whole memory space is used to hold the data sent into the ADS-PEs array. The first section of RAM can hold the macro block to be searched and the other 3 sections can hold the data of the search region. Figure. 5.16 shows the partition of the macroblock and how the data are stored.

At the instance while a portion of image to be compared crosses the video RAM section boundary, the row decoders have to make sure both sections are activated. It is up to the column decoder to decide which section provides the output data.

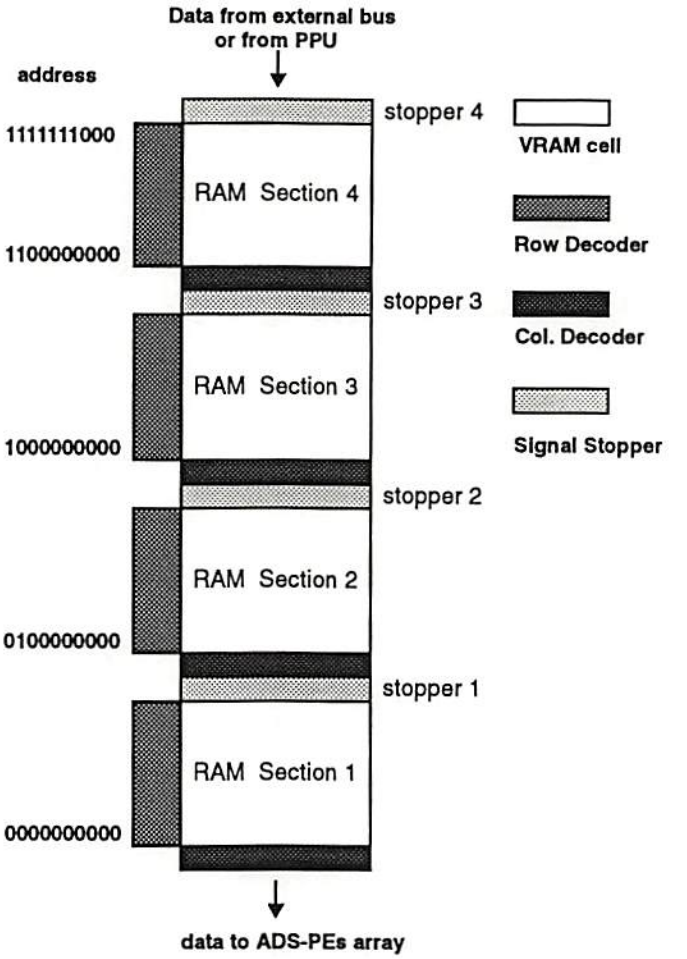


Figure 5.15: Block diagram of the draft new memory system design.

Thus the address mapping mechanism can be omitted. All the signal stoppers (1-4) will just behave like transmission lines.

When the memory is used for the DCT operation, the situation is more complicated. In the Pass-1 stage, the original data can be stored in video RAM section 1, and only the stopper 1 is activated to block the other 3 sections from the section 1. Therefore, the translated coefficients generated from PPU can be written back to section 2 without interference. In the pass-2 stage, Stopper 1 is deactivated but the Stopper 2 is activated. Therefore sections 3 and 4 are separated from sections 1

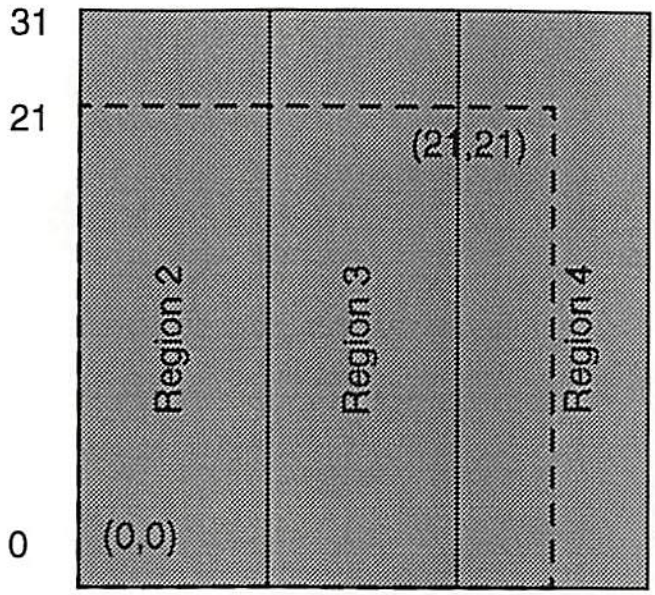


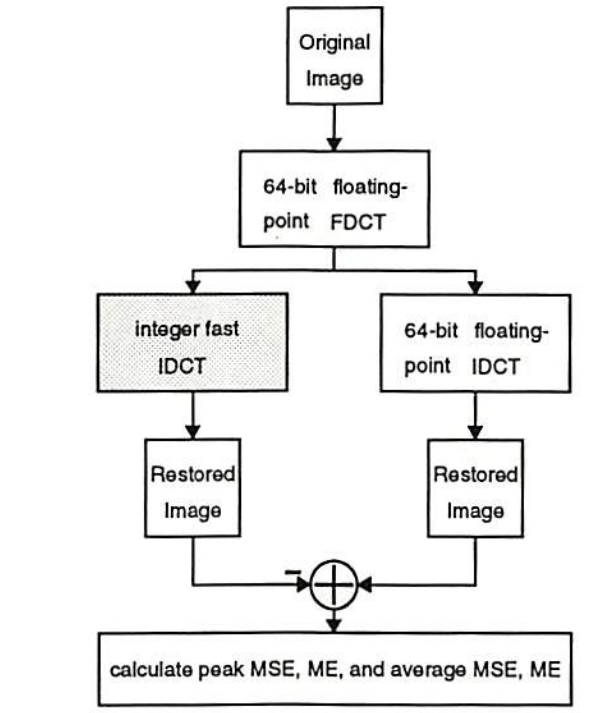
Figure 5.16: Partition of the search region which can fit to the memory design.

and 2. The array processor can thus read input data from section 2 and then write the translated data back to section 3. This could be the solution of memory system design when double-buffered memory is not available.

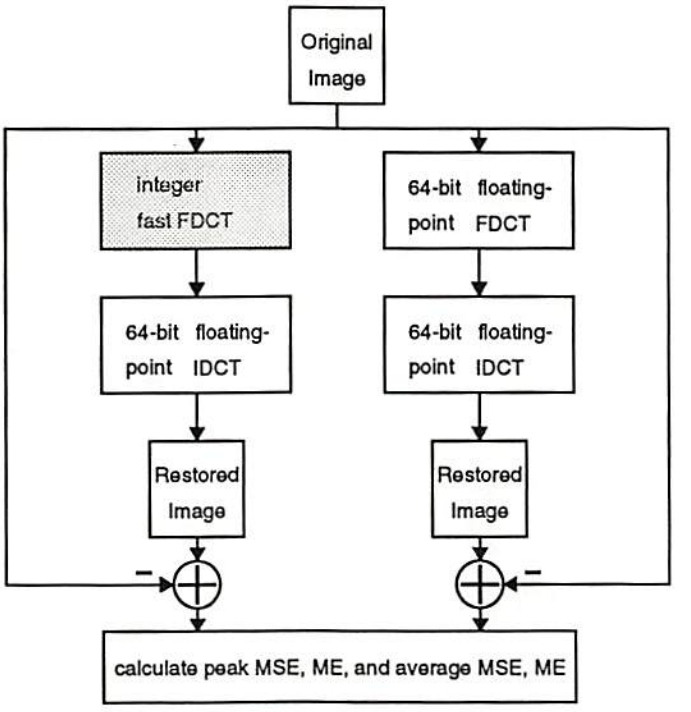
5.7 Accuracy Validation

Since integer operations are used instead of the more precise floating-point operations and a minimized amount of data coming into the PE arrays is more desirable, certain round-off (or truncation) operations are used. At the same time, errors could be accumulated during the DCT/IDCT operations and may cause the distortion and annoying block effects.

To ensure the accuracy of the design, certain tests have been recommended by IEEE [84]. The IDCT test taken is under the guidance of the Inverse Transform Accuracy Specification, where the block diagram of the IDCT test is illustrated in Figure 5.17 (a).



(a)



(b)

Figure 5.17: Schematic diagram for accuracy test. (a) Inverse 2-D IDCT test. (b) Forward 2-D DCT test.

When the two restored images are achieved, one by 64-bit floating-point IDCT and the other one by the integer method, a difference image is taken. Based on the difference image, the first step is to examine where there is any pixel difference (PD) larger than 1. If the answer is yes, the proposed method would fail. Then the peak mean error (PME), average mean error (AME), peak mean square error (PMSE), and average mean square error (AMSE) are calculated. The results must be within the IEEE recommended values. There are totally 10,000 blocks generated randomly and used for any one of the 6 test cases. It is clearly seen from table 5.1 that the integer IDCT algorithm mapping can successfully meet the IEEE requirement for the IDCT accuracy test.

Table 5.1: Accuracy test result for the IDCT operation validation (DCT coefficients are represented by 13-bit+1 sign bit number).

Test Items	PMSE	AMSE	PME	AME
L=256, H=255	0.0156	0.012384	0.0027	0.000075
L=5, H=5	0.0110	0.009152	0.0021	-0.000170
L=300, H=300	0.0170	0.013389	0.0031	0.000133
L=256, H=255 (RS)	0.0150	0.012230	-0.0028	-0.000095
L=5, H=5 (RS)	0.0102	0.008341	-0.0021	0.000119
L=300, H=300 (RS)	0.0171	0.013545	-0.0030	-0.00014
IEEE recommended	0.06	0.02	0.015	0.0015

Since only the inverse transform accuracy specification had been recommended, the following is the way the forward DCT algorithm is tested and Figure 5.17 (b) shows the schematic diagram of the DCT test. Basically it is very similar to the one used in the IDCT test. At first the 64-bit floating-point 2-D DCT and IDCT are applied and a new restored image is achieved. Then the integer fast 2-D DCT algorithm is applied on the original image and 64-bit floating-point 2-D IDCT is used for restoration. The two restored images are compared with the original image

where PME, AME, PMSE and AMSE can be calculated. Table 5.2 shows the result from program simulation. The test conditions are similar to the conditions used in the IDCT test and the results clearly indicate that the proposed method can achieve similar performance as the 64-bit floating-point DCT operations.

Table 5.2: Accuracy test result for the DCT operation validation (DCT coefficients are represented by 13-bit+1 sign bit number).

	Test Items	PMSE	AMSE	PME	AME
Proposed DCT + 64-bit IDCT	L=256, H=255	0.0891	0.083761	-0.0063	-0.000355
	L=5, H=5	0.0838	0.075911	0.0066	-0.0008
	L=300, H=300	0.0893	0.083781	-0.0055	-0.001084
	L=256, H=255 (RS)	0.0905	0.085414	0.0075	0.001561
	L=5, H=5 (RS)	0.0824	0.075605	0.0060	0.000433
	L=300, H=300 (RS)	0.0881	0.083495	-0.0064	-0.000414
64-bit DCT + 64-bit IDCT	L=256, H=255	0.0879	0.083345	-0.0062	0.000486
	L=5, H=5	0.0819	0.075553	-0.0064	0.000237
	L=300, H=300	0.0875	0.083645	0.0068	-0.000270
	L=256, H=255 (RS)	0.091	0.085302	0.0088	0.001830
	L=5, H=5 (RS)	0.0814	0.075647	-0.0066	-0.0002
	L=300, H=300 (RS)	0.0879	0.082909	-0.0060	0.000172

5.8 Performance Estimation

Assume that the video sequences are composed according to the MPEG-2 main profile, with resolution 720 × 480-pixel per frame. There are 30 frames per second and each frame contains 3 color components (R-G-B) which are 4:1:1 sub-sampled. The system clock will run at clock rate 100 MHz, i.e., 10 ns per cycle. Assume that the data access is always successfully with 100% hit rate and the on-chip video RAM is double-buffered which can support simultaneous read-write operation. That means the system will not halt while the data has to be written back to the video RAM. Table 5.3 shows the required cycles and the resource usage percentage.

Table 5.3: Performance Estimation of Proposed Video Coprocessor.

Operations	cycles per block	blocks per frame	cycles per frame	resource usage (%)
Motion Estimation	370	2025	749,250	22.5%
2-D DCT (Pass-1)	85	8100	688,500	20.7%
2-D DCT (Pass-2)	150	8100	1,215,000	36%

From the table shown above, the 3 operations will use about 80% of the computing resource of the coprocessor. Hence, the design can meet the performance requirement for real-time MPEG-2 main profile encoding operation.

Although no actual chip was fabricated, the table II in Chang, et al.'s paper [85] provided a good way to estimate the size and the transistor count of the VLSI design. The estimation is based on $0.8\mu m$ CMOS technology. Assume that a 16-bit adder needs 0.1052 mm^2 of area and uses about 700 transistors. There are totally 64 9-bit adders and 64 12-bit adders in the ADS-PE array, which account for about 58,000 transistors and 8.84 mm^2 of area. The 8-bit registers will use another 18,000 transistors and 1.74 mm^2 area. Thus the core processor will use roughly about 76,000 transistors and 10 mm^2 of silicon area. Compared with the results achieved by others [85, 76, 62], this approach not only saves the silicon area and transistor counts, but also further push the speed to a higher limit with the pipelined array structure. In fact the compact design provides a cost-effective solution for multimedia real-time applications.

5.9 Comparison with Others' Results

Table 5.4 gives a comparison of the proposed 2-D DCT/IDCT architecture with some existing designs in terms of several parameters. In summary, the proposed

approach has several advantages over others' results. Since it is based on a motion estimation search, it is more versatile with minimum additional hardware cost. The superpipelining execution of the adders in the processing elements can push the clock speed to a rate higher than 100 MHz by using a 0.5 μm CMOS fabrication technology. The speed can be further increased by using future more advanced semiconductor technologies. Massively parallel processing in the array which consists of totally 64 PEs can achieve the throughput rate of more than 10 giga-operations-per-second (GOPS). The high performance is critical for real-time multimedia MPEG-2/video applications. And the compact digital cell design proves to be a very cost-effective solution.

Table 5.4: Comparison with others' results.

	Array vs. Direct	Type of Multiplier	No. of Multiplier	No. of DCT Block	Trans-position	No. of Tran-sistors	Ref. No.
Duhamel-Guillemot			$N^2\log_2 N/2$		No		67
Cho-Lee	Array	Multiplier	$N^2\log_2 N/2$	2	No		66
Chiu-Liu			8N		No		68
Totzek-Manthiesen			2N		Yes		69
Ma			4N(N+1)		No		70
Chang-Wang	Array	Multiplier	N^2	2	No	340K	78
Wang-Chen	Array	Multiplier	$4N^2$	2	No		71
Yang-Bai	Array	Multiplier	N/2	1	No		72
Karathanasis	Array	ROM-based	0	2	No	102K	73
Madisetti-Willson	Array	Multiplier	4	1	Yes	67K	64
Jang-Kao	Direct	Multiplier	8	2	No		63
Wu-Chiou	Array	Multiplier	4	1	Yes	66K	62
Chan-Siu	Direct	ROM-based	12 ROM	2	Yes		61
Guo-Liu	Array	ROM-based	8	2	No		60
Uramoto-Inoue	Direct	ROM-based	8 ROM	2	No		76
Sun-Chen	Direct	ROM-based	32 ROM	2	Yes	73K	75
Proposed	Array	DA	0	1	No	70K	

Chapter 6

Conclusion

In this dissertation, a behavioral simulation methodology for the densely-connected analog array processor is presented. The proposed method is based on a Runge-Kutta 4-th order differential equation solver engine which can efficiently simulate the system dynamic behavior. System partition techniques can provide valuable information about the influence when multiple chips had to be used to construct a large system. Random noise added in the system would simulate the case when crosstalk noise generated during the fast data switching and test the robustness of the coefficient templates. The effect on the finite precision of analog system can also be analyzed with the simulator.

The paralleled computing architecture adapts a hybrid analog-digital scheme for the construction of a smart and powerful computing machine. A simulation environment which consists of several components is constructed. It includes the behavioral simulator, a graphical user interface, a compiler, and a template library manager. A parallel programming language is also defined to facilitate the use of the parallel computing architecture.

An innovative approach to fully utilize an array processor originally designed for motion estimation as a 2-D DCT/IDCT processor is also presented. Minimum hardware is added so the compact array structure can be maintained and maximum

flexibility is achieved. Pipelined execution can easily push the speed to be higher than 100 MHz in a chip fabricated by a $0.5\mu m$ CMOS technology while the number of latency cycles is still low. Multiple processing elements (8 by 8) can realize the throughput rate to be more than 10 giga-operations-per-second (GOPS) which is suitable for real-time applications. Because of its performance, flexibility and compact design, this array processor architecture could lead to a cost-effective solution for the multimedia video application.

Appendix A

Performance of Selected Microprocessors from the Industry

Year	Chip	Tran. #	Lmin (μm)	Power (W)	Speed (MHz)	MIPS	Size (cmxcm)
1986	NEC V60	375K	1.5		16	3.5	
1987	MC68030	300K			16		
1988	MC88100	1.2M	1		20		
1988	AMD29000		1.2		30		
1989	Intel 80486	1.2M	1		25	15	1.3*1.3
1989	Intel i860	1.2M	1		20	17	1.0*1.5
1990	MC68040	1.2M	0.8		25	20	164 mm^2
1990	NEC V80	980K	0.8		33	16.5	1.45*1.45
1990	Gmicro/300	900K	1	2	25	17	1.6*1.6
1991	Intel 486DX2	1.2M	0.8	7	66		81mm^2
1992	DEC Alpha 21064	1.68M	0.75	30	200		234 mm^2
1992	Super SPARC	3.1 M	0.6	14.2	60		256 mm^2
1992	Fujitsu	1.5M	0.5	5			1.575*1.6
1992	Hitachi	1.65M	0.6	9	66	132	1.09*1.6
1992	HP PA7100	0.85M	0.8	23	100		1.42*1.42
1992	PowerPC 601	2.8M	0.6	6.5	50		1.1*1.1
1993	Mistubishi	1.71M	0.8	6	40		1.63*1.27
1993	PowerPC 603	1.6M	0.5	3	80		85 mm^2
1993	Intel Pentium (P5)	3.1M	0.8	16	66		296mm^2
1993	Motorola 68060	2.4M	0.5	3.9	50		198mm^2
1993	DEC Alpha 21064A	2.8M	0.5	33	275		164 mm^2
1994	Intel Pentium (P54C)	3.1M	0.6	5	100		163mm^2
1994	TI TMS320C80	4M	0.5				

Year	Chip	Tran. #	Lmin (μm)	Power (W)	Speed (MHz)	MIPS	Size (cmxcm)
1994	PowerPC 604	3.6M	0.5	13	100		196 mm^2
1994	HP RISC 7200	1.2M	0.55	29	140		1.4*1.5
1994	MIPS R4200	1.4M	0.64	1.8	80		78 mm^2
1994	Silicon Graphics	2.6M	0.5	13	75	300	1.73*1.73
1994	DEC Alpha 21164	9.3M	0.5	50	300		161 mm^2
1995	Ultra Sparc	5.2M	0.5	30	167		315 mm^2
1995	NexGen	3.5M	0.5		94		1.41*1.41
1995	Intel Pentium (P54CS)	3.1M	0.35	10	120		163 mm^2
1995	Intel Pentium Pro (P6)	5.5M	0.35	14	133		306mm^2
1995	HP RISC 8000	3.9M	0.5		200		345 mm^2
1995	PowerPC 620	6.9M	0.5	30	100		311 mm^2
1995	MIPS R10000	5.9M	0.5	30	200		298 mm^2
1995	NEC	9.3M	0.5	50	300	1200	1.65*1.81

Appendix B

Network Simulation by Using HSPICE Circuit Simulator

HSPICE is a general-purpose circuit simulation program. Based on the equivalent circuits shown in Figure 2.5, the network can be described with circuit elements and simulated by using HSPICE. The signal is represented by voltage while the ideal voltage-controlled current source is used as the synapse weight multiplication scheme. The current can be summed together at a node where the resistor R and capacitor C are connected. The piecewise linear function can be simulated by using two diodes connected to a positive and negative voltage. Since there are voltage drops across the diodes, these two voltages have to be chosen carefully. In our experiment, setting them to ± 0.58 V makes the saturation voltage to ± 1 V. The subcircuit description of a connect-component-detector cell is shown here.

```
.subckt cell 1 6 3 11 12
* 1: Vx, 3: Vu, 6: f(Vy), 11, 12, : from neighbors
* feedforward from Vu
G0 0 1 3 0 0.0
* feedforward from neightboring inputs
G1 0 1 11 0 1
G2 0 1 12 0 -1
* feedback from Vy
G10 0 1 6 0 2
* Bias
I1 0 1 0.0
```

```
* R and C constant
R1 1 0 1
C1 1 0 1
* Vy and PWL circuit
E1 72 0 1 0 1
RR 72 2 10Meg
D1 5 2 dmodel
D2 2 4 dmodel
E2 6 0 2 0 1
* Decide the saturation voltage
Vp 4 0 0.58
Vn 5 0 -0.58
.ends
```

Assume that there is a 1×5 networks implementing the connected-component-detection function. The initial states are set as shown in Figure B.1(a). The rightmost pixel and the leftmost pixel are set to 1 (white) and the middle 3 pixels are set to -1 (black). As time evolves, the left white pixel gradually moves to the right as seen in Figure B.1 (b) and (c) and finally stops at the middle.

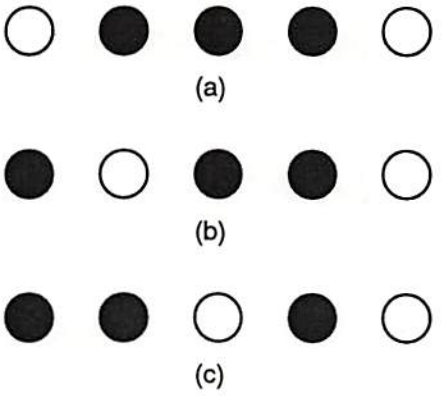


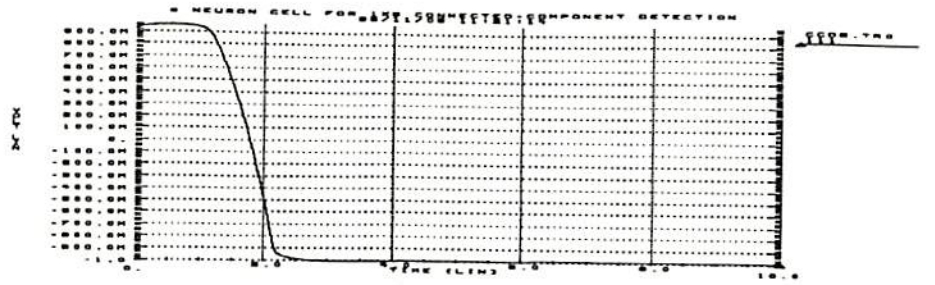
Figure B.1: The states of pixels during the connect-component detection operation. White pixels represent values of +1 and black pixels represent values of -1. (a) Initial state of the 1×5 network. (b) The movement of the white pixel from the left to the right. (c) The final state of the network.

The network can be described as the following:

```
Vp1 6 0 1
```

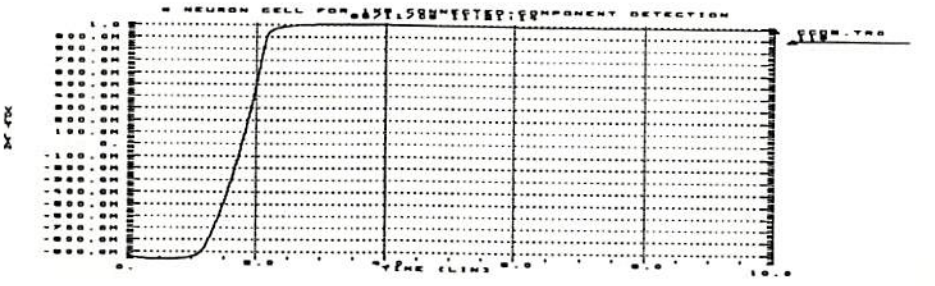
```
Vn1 7 0 -1
.IC V(211) 1
.IC V(212) -1
.IC V(213) -1
.IC V(214) -1
.IC V(215) 1
V11 11 0 1
V12 12 0 -1
V13 13 0 -1
V14 14 0 -1
V15 15 0 1

X11 211 111 11   7 112 cell
X12 212 112 12 111 113 cell
X13 213 113 13 112 114 cell
X14 214 114 14 113 115 cell
X15 215 115 15 114   7 cell
```

The dynamics of the simulation results are shown in Figure. B.2. At the top the behavior of the leftmost cell is plotted against the time. The peak value of 1 move to the right until it reaches the middle pixel. Since the values of the rightmost two cells do not change, they are plotted in the Figure B.2 (d).

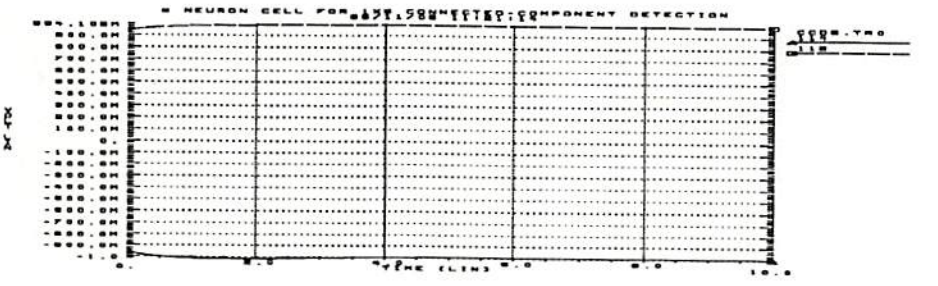


(a)

(b)



(c)



(d)

Figure B.2: The dynamic of a 1x5 compact network for connect-component detection operation by simulating using HSPICE. (a) The dynamic of the left most cell V_1. (b) The dynamic of the cell V_2. (c) The dynamic of the cell V_3. (d) The dynamics of the two right most cells V_4 and V_5.

Appendix C

Pattern Storage Behavior of Time-Delayed Discrete-Time Systems

In this appendix, a time-delayed discrete-time system which can associate certain input patterns with output patterns is described. Compared with the compact neural network, it is found that the cells are very similar when the synapse weights are replaced by the delayed ones.

C.1 Time-Delayed Discrete-Time Systems

Assume the time-delayed discrete-time system consists of N neurons and M external inputs. The schematic diagram of the neuron used is illustrated in Figure C.1 [86]. Each neuron can be described as

$$\begin{aligned} x_i[k+1] &= \sum_{j=1}^{M} v_{ij} \sum_{r=0}^{k} T^r u_j[k-r] - \sum_{j=1}^{N} w_{ij} \sum_{r=0}^{k} T^r y_j[k-r] - h_i, \\ y_i[k+1] &= f(x_i[k]), \end{aligned} \tag{C.1}$$

where $f(\cdot)$ is the hard-limiting function whose output is 0 when the input is smaller than 0, and 1 otherwise. The coefficients v_{ij} and w_{ij} represent the synapse weights from the external inputs u_j or the outputs y_j to the neuron i, respectively. Here,

h_i is the threshold value of neuron i and T is the decay parameter which is smaller than 1.

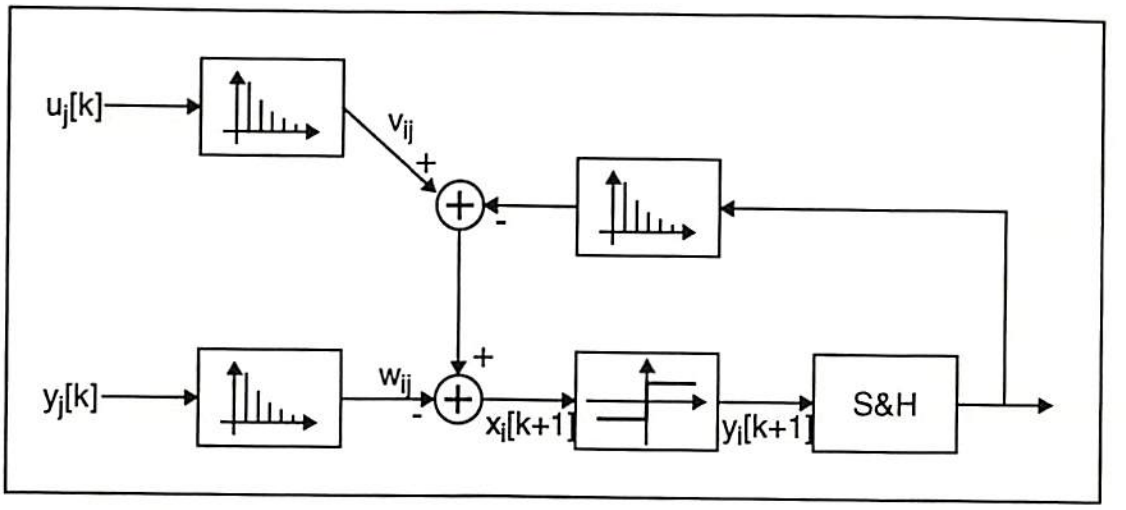


Figure C.1: Schematic diagram of the neuron [86].

Equation C.1 can be rewritten into the matrix form,

$$\begin{aligned} \mathbf{X[k+1]} &= \mathbf{V\ U[k]\ T[k] - W\ Y_x[k]\ T[k] - H}, \\ \mathbf{Y[k+1]} &= f(\mathbf{X[k]}), \end{aligned} \tag{C.2}$$

where

$$\begin{aligned} \mathbf{X[k+1]} &= [x_1[k+1]\ x_2[k+1]\ \cdots\ x_N[k+1]]^T \\ \mathbf{Y[k+1]} &= [y_1[k+1]\ y_2[k+1]\ \cdots\ y_N[k+1]]^T \\ \mathbf{T} &= \left[T^k\ T^{k-1}\ \cdots\ 1\right]^T \\ \mathbf{H} &= [h_1\ h_2\ \cdots\ h_N]^T \\ \mathbf{V} &= \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1M} \\ v_{21} & v_{22} & \cdots & v_{2M} \\ \vdots & & & \vdots \\ v_{N1} & v_{N2} & \cdots & v_{NM} \end{bmatrix} \end{aligned}$$

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & & & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NN} \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} u_1[0] & u_1[1] & \cdots & u_1[k] \\ u_2[0] & u_2[1] & \cdots & u_2[k] \\ \vdots & & & \vdots \\ u_N[0] & u_N[1] & \cdots & u_N[k] \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} y_1[0] & y_1[1] & \cdots & y_1[k] \\ y_2[0] & y_2[1] & \cdots & y_2[k] \\ \vdots & & & \vdots \\ y_N[0] & y_N[1] & \cdots & y_N[k] \end{bmatrix}$$

When the length of the memory element was shortened to 1 and $w_{ii} = 0$, the system will become a discrete-time Hopfield network. Furthermore, if the length is 1, the external inputs and incoming external outputs are within a limited range (neighborhood) and self-feedback is permitted, the system will behave as a discrete-time compact neural network.

C.2 Simulation Results

Figure C.2 shows the schematic diagram of the 2-neuron network that has been studied.

C.2.1 Simulation 1: No External Inputs

If no external inputs were applied, the final result will depend on the initial conditions. In the first case, the synapse weights are

$$w_{11} = -0.3,\ w_{12} = 0.6,\ w_{21} = 0,\ w_{22} = 0.3,\ \text{and}\ h_1 = h_2 = -0.3. \qquad \text{(C.3)}$$

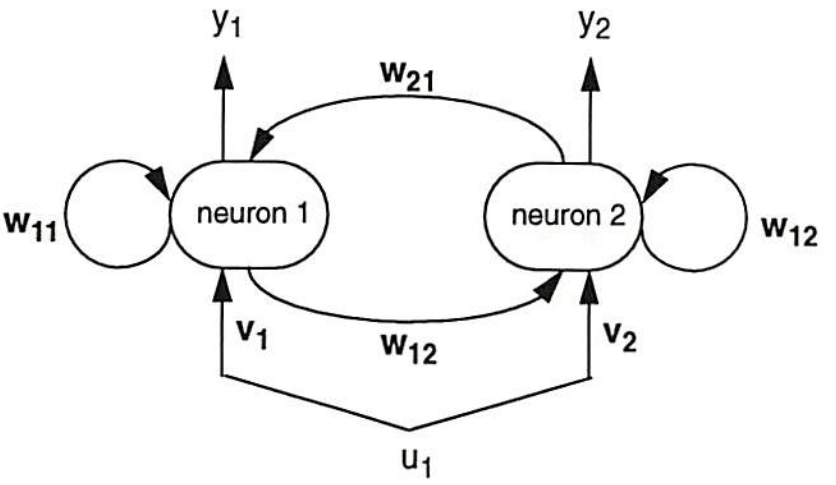


Figure C.2: Schematic diagram of the 2-neuron network.

Four different initial conditions, (1,1), (-1,1), (-1,-1) and (1,-1), which represent the points in four quadrants, were used. The final output is illustrated in Figure C.3. Two output patterns were found, which are

$$\mathbf{Y_a} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}, \text{ and } \mathbf{Y_b} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}. \tag{C.4}$$

The $i-th$ row represents the periodic discrete output sequence x_i. The period in this case is 2.

It is desirable to know what input pattern may lead to a certain output pattern. An experiment was performed by changing the initial points on each axis (x_1 and x_2) from -2 to 2, with 0.1 spacing. The result is shown in Figure C.4. If the initial points lie in the second quadrant, the output pattern is Y_b. Otherwise, the output pattern is Y_a.

The network will not be limited to remember just the above two patterns. By changing the synapse weight connection, more patterns can be memorized. In the second case, the synapse weights were changed to

$$w_{11} = -7.5,\ w_{12} = -9.0,\ w_{21} = 3,\ w_{22} = 4.5,\ \text{and}\ h_1 = 15.3, h_2 = -6.3. \tag{C.5}$$

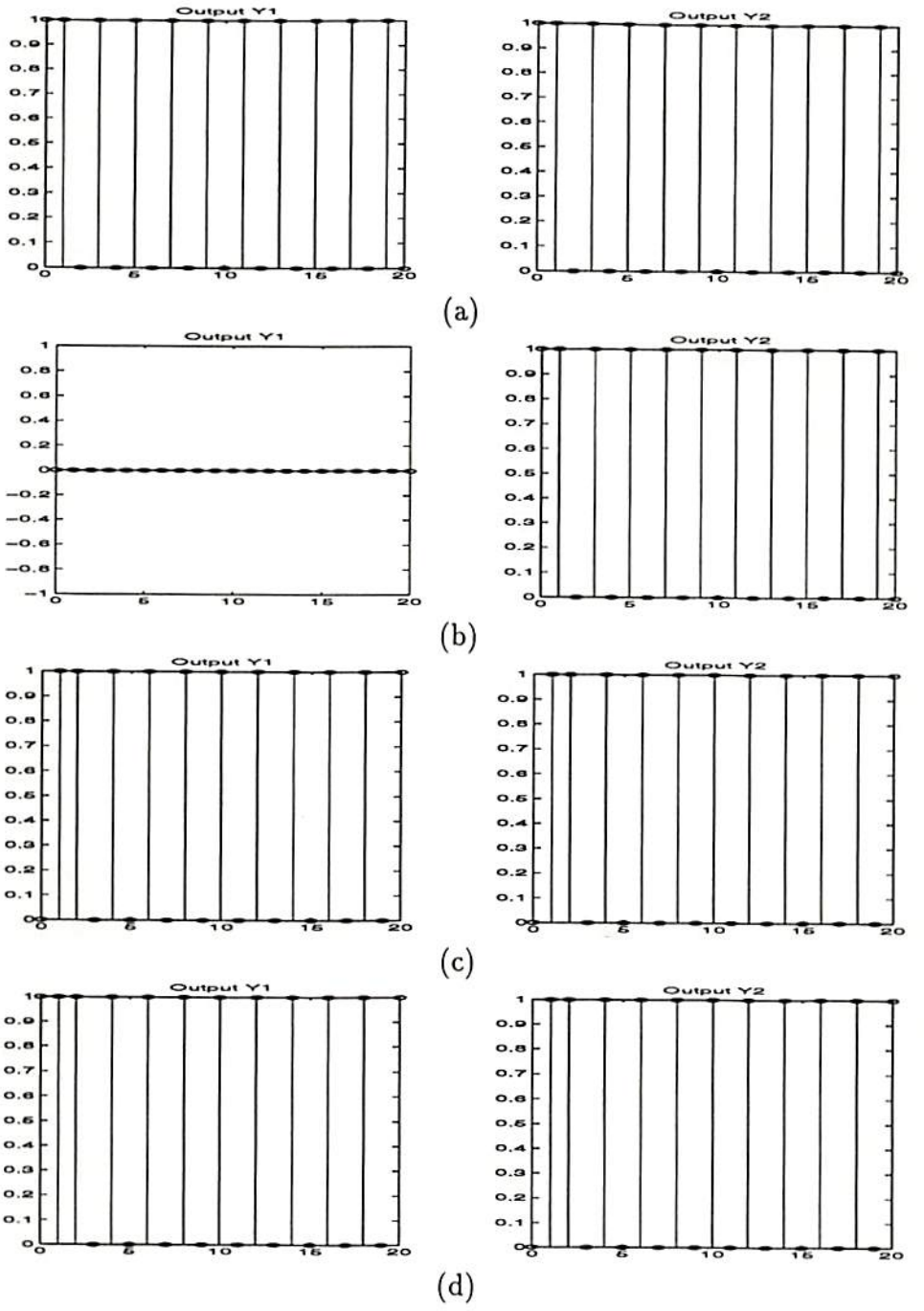


Figure C.3: Simulation results using different initial conditions with first set of synapse coefficients: (a) x[0]=(1,1). (b) x[0]=(-1,1). (c) x[0] = (-1,-1), (d) x[0] = (1,-1).

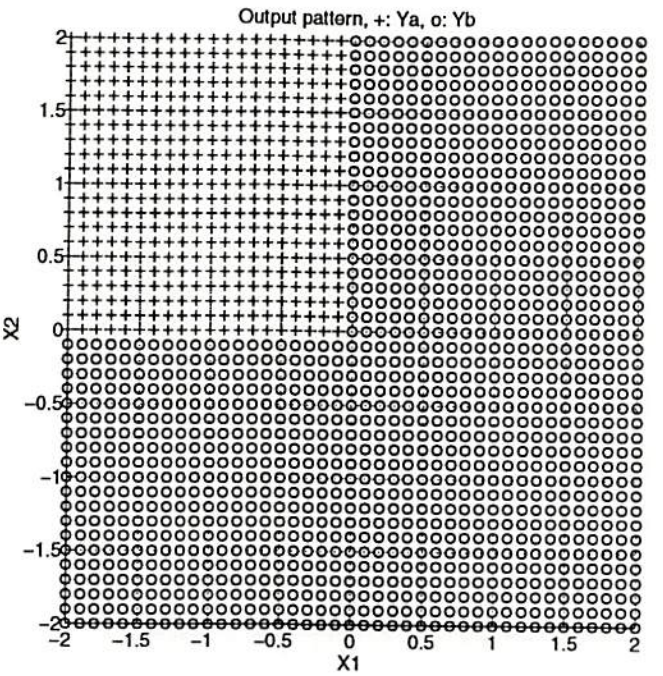


Figure C.4: The relationship of input states and output patterns.

The simulation result is shown in Figure C.5.

Three more patterns were found. They are

$$\mathbf{Y_d} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

$$\mathbf{Y_e} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ and}$$

$$\mathbf{Y_f} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix}. \tag{C.6}$$

Note that the periods of the output pattern in the second experiment can be either 4 or 5. That means a longer pattern is memorized.

The relationship of the initial states and the output patterns was plotted in Figure C.6. If the initial state points were all positive, the output pattern will be Y_d.

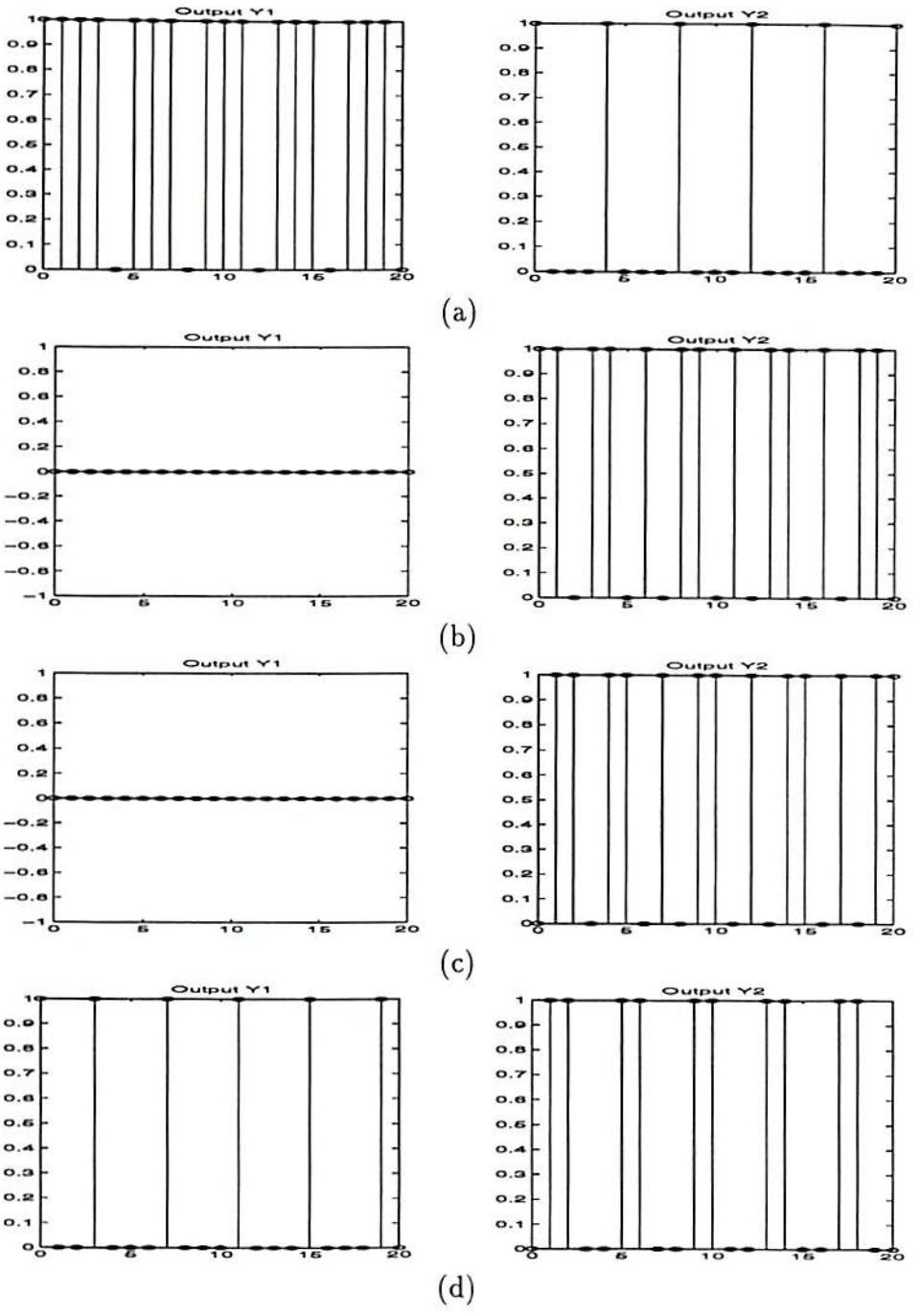


Figure C.5: Simulation results using different initial conditions with the second set of synapse coefficients: (a) x[0]=(1,1). (b) x[0]=(-1,1). (c) x[0] = (-1,-1), (d) x[0] = (1,-1).

Those initial states that lie in the fourth quadrant will lead to the output pattern Y_e. Otherwise the output patterns will converge to Y_f.

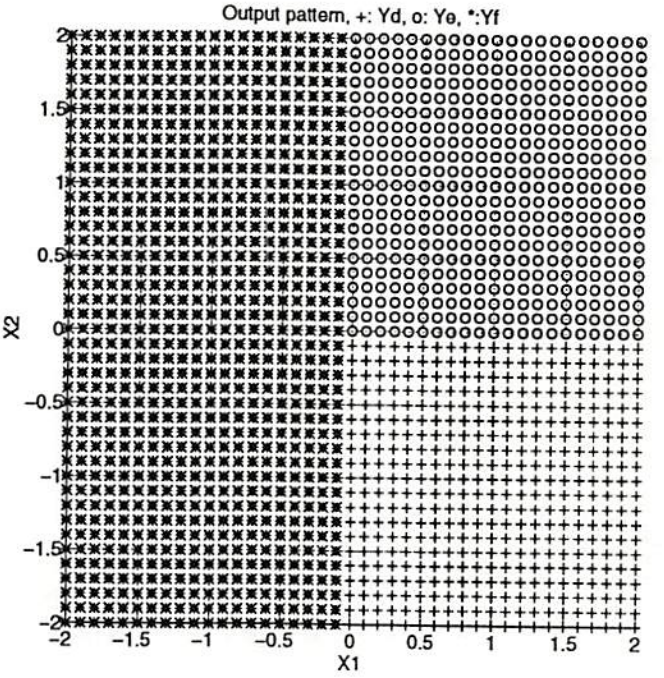


Figure C.6: The relationship of input states and output patterns.

C.2.2 Simulation 2: Network with External Inputs

The initial states will determine the eventual system output pattern when no external inputs were applied. However it is possible to change from a pattern to another pattern by the addition of external inputs. A network with parameters listed in C.3 was studied.

Assume the initial state is $(-1, 1)$ and the input is only applied to the 1-st neuron with strength 0.51, i.e., $\mathbf{W} = [0.51 \;\; 0]^T$. At the beginning no external input is added and the output will become pattern Y_a, as shown in Figure C.7. At time $t = 18$, there is an input impulse and the output sequence changes from pattern Y_a to Y_b.

At time $t = 29$, an input train with length 10 is applied and a new pattern Y_c is produced. The pattern Y_c can be described as

$$Y_c = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}. \tag{C.7}$$

When the input is returned to 0, the output pattern will remain the same. That means the pattern Y_c is memorized.

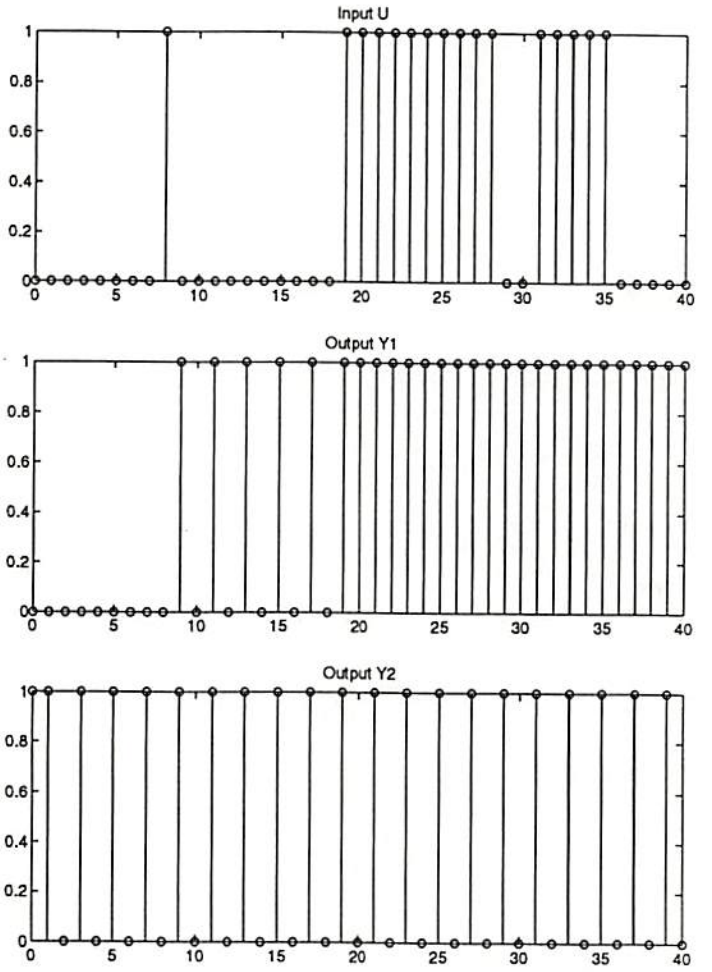


Figure C.7: The relationship of input states and output patterns while external inputs are imposed.

In the above experiment the output pattern changed from Y_a to Y_b, then from Y_b to Y_c. Another experiment is done without the impulse input at time 8. At this time

it is found that the output sequence will jump directly from pattern Y_a to pattern Y_c without passing through pattern Y_b. The simulation result is shown in Figure C.8.

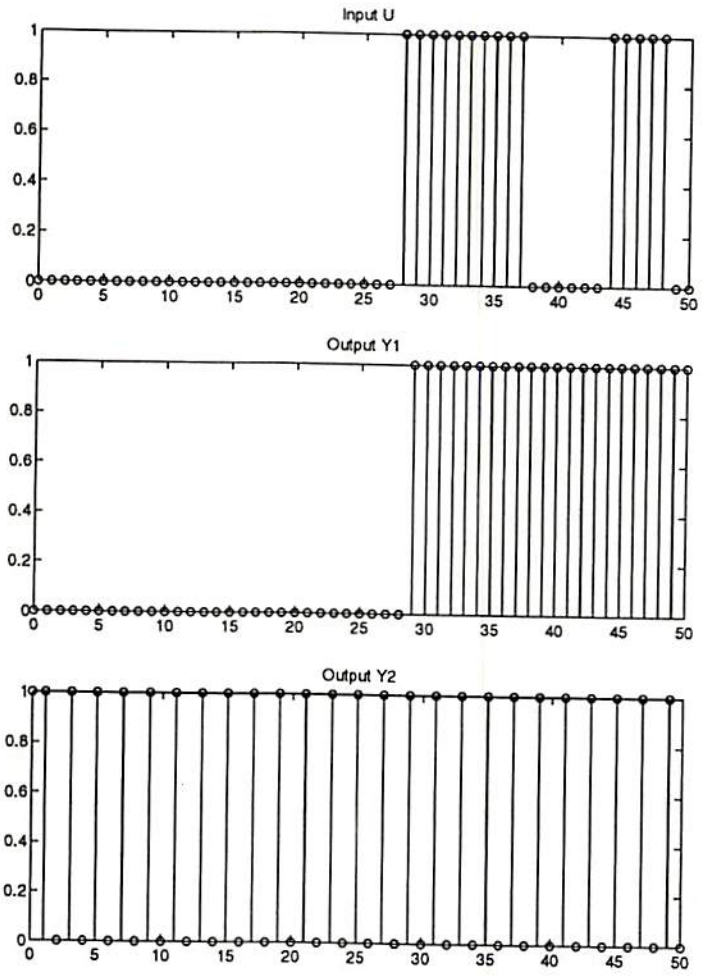


Figure C.8: The relationship of input states and output patterns while external inputs are imposed.

Appendix D

The Behavioral Simulator User's Guide

D.1 Execution of the Simulator

To invoke the program, execute the command *cnna* under the UNIX prompt as

```
prompt> cnna [command_file] <cr>
```

If no command file name is supplied, the program will look for the default command file *cnn.cmd*. The instructions used in command file will be explained in the next section. When the program is executed, it will read the commands from the command file, parse the sentence into tokens and extract the system configuration parameter values. For example, using the file hf5.cmd as the input command file, the screen will show as follows:

```
================================================
|       Compact  Neural Networks Simulator      |
|     Version 1.2.5, Last Updated Oct. 30 1995  |
|    Developed by Tony H. Wu, Dr. Bing J. Sheu   |
================================================

Initializing CNN ...
```

```
Reading from [edge32.cmd] and the parameters are ...
Using 4-th order Runge-Kutta Method.
Image size is 32 x 32.
Read input image from file [lenna.32]
Read initial states from file [lenna.32]
output file is [lenna32.out].
Time step = 0.050000 and Max Execution time = 50.000000
TempA size = 1.
 2.0000
TempB size = 3.
-0.2500 -0.2500 -0.2500
-0.2500  2.0000 -0.2500
-0.2500 -0.2500 -0.2500
Bias is -0.750000, Boundary value = -1.000000.
piecewise-linear function is used.
Tolerance is 0.010000. No annealing.
------------------------------------------------
Running CNN simulation ...
```

After execution, some statistics will appear on the screen. In the meantime output will be written to the file specified in the command file for further processing or viewing.

D.2 Template Library

A template library is provided and its name is called *template.lib*. A list of built-in templates and their names can be found in Chapter 4. Please put this file in the

same working directory so the *cnna* program can find it. The template file is also an ASCII file and will be scanned when a "USE" command is used. The syntax in the template file is

```
@DEF template_name
TSIZE x x
TEMPA ....
TEMPB ....
BIAS x
BOUNDARY x
@ENDEF
```

where TSIZE, TEMPA, TEMPB, BIAS, and BOUNDARY use the same syntax as usual. The user can add his/her own definition in the template library file. Currently the library only support the templates with constant template coefficients.

Users can also use the environment variables to specify a different path or a different template library file. The program *cnna* will use the environment variable **TEMPLIB_DIR** as the new path and the **TEMFILE** as the new template library filename. Therefore the library can be shared and this can avoid the duplication of the library files used by different users.

Data File Format

All the input, initial state, and output data are stored in the same format. Each cell will use one byte to store the data, which means the data range is 0 to 255. When the data were read in, they will be linearly scaled to values ranging from -1 to 1.

The data for the whole array are stored in column-first fashion. Assume a network contains cells $C(i, j)$ where $0 \leq i \leq n$, and $0 \leq j \leq m$, the data will be stored in the

order of $C(0,0)$, $C(0,1)$, $C(0,2)$, $\cdots$, $C(0,m)$, $C(1,0)$, $C(1,1)$, $\cdots$, $C(n,m)$. So the total image size will be just $n \times m$ bytes.

The ASCII text input and output are supported. The program will read the text input (or initial state) data in the order specified in the above. If the data cannot fit within one line, multiple line inputs can be accepted. But after each input line there has to be a \ character to let the parser know at least one more line needed to be read. There is no \ character at the last line of the data.

If the users want ASCII text output, the output will be sent to the screen instead of a file. '+1' will be represented by a '+' character and '-1' will be represented by '-'.

D.3 Syntax of Commands

The command file gives you the flexibility to change the system configurations without recompiling the whole program. The default command filename is *cnn.cmd*. Other command file will be used if its name is supplied. In the command file all instructions are given in plain ASCII text. Each line contains one instruction only and is composed of one command keyword (in capital letters) and several necessary or optional parameters. If the first character in the line is %, this line of text will be treated as comments and ignored during interpretation.

Syntax of each command is explained here according to the category it belongs to. The conventions used are

- each line started with a command keyword (in capital letters),
- the parameters enclosed by angle brackets are mandatory,
- the parameters enclosed by brackets are optional, and

- the parameters separated by / are choices to be made.

I/O Commands

`ISIZE <size_of_row> <size_of_column>`

Keyword ISIZE helps to specify the row size and the column size of the input V_u.

`INPUT ALL <value>`

`INPUT <filename>`

`INPUT ASCII pixel(0,0) pixel(0,1) ...`

Keyword INPUT defines the input data V_u, from either a data file or an uniformly presetting value. The input data file use 8-bit integers to store the values. Value 0 will be mapped to -1 and value 255 will be mapped to 1. Other values lie between will be linearly scaled.

ASCII text input is also supported. The pixel values will be read in the order explained in previous chapter. If the data cannot be fit in just one line, a \ character had to be appended at the end of the line except the last one.

`INITIAL ALL <value>`

`INITIAL <filename>`

`INITIAL ASCII pixel(0,0) pixel(0,1) ...`

Keyword INITIAL defines the initial state $V_x(0)$, from either a data file or a uniformly presetted value. The data file has the same properties as the input data file.

ASCII text input is supported as well. Please refer to the explanation of command 'INPUT'.

`OUTPUT <filename>`

`OUTPUT ASCII`

Keyword OUTPUT specifies the output filename. The output file stores the values of the output V_y. The values are mapped in the same way as what was done in the input file.

ASCII text output is also supported. The saturated output value '+1' will be represented by '+' character while value '-1' will be represented by '-' character. The output will be shown in the matrix form for easy viewing.

Simulation Commands

`METHOD RK4/ARK4`

Keyword METHOD specifies the integration method the solver engine will use. It can be either the 4-th order Runge-Kutta method (RK4) or the adaptive 4-th order Runge-Kutta method (ARK4).

`TRAN <step_size> <max_integration_period>`

Keyword TRAN defines the integration step size used and the maximum integration period (not the physical CPU time) the integrator will work. For the 4-th order Runge-Kutta method, the step size will be fixed and the simulator might stop if the output didn't change for a certain period. If the 4-th order adaptive Runge-Kutta method is used, the step size

defined here will be used as the initial step size. It will change its value according the characteristics of the system. However, the simulator will continue executing the program until the maximum integration period is reached. So be careful in choosing this parameter.

`ANNEAL N/Y`

Keyword ANNEAL specifies whether annealing processing is used. Users have to choose either N or Y.

`TOL <tolerance_value>`

Keyword TOL defines the tolerance value. If all neurons' changes are within this value for a certain number of time steps, the simulator stops.

`NOUTPUT PWL`

`NOUTPUT SIGMOID [lambda]`

Keyword NOUTPUT helps to define the output neuron type. It can be either piecewise-linear function (PWL) or sigmoid function (SIGMOID) which are described in (2.4). If λ value is not supplied, the default value is 2.

Template Commands

`TSIZE [size_of_TA] [size_of_TB]`

Keyword TSIZE specifies the size of the feedback matrix (T_A) and the control matrix (T_B), respectively. Square size of the template is assumed and the values can be either 0 , 1 , 3 or higher. Choose the possibly smallest one to speed up the simulation.

```
TEMPA coefficients...
TEMPA ALL <value>
```

Keyword TEMPA defines the entries in the feedback matrix (T_A). The terms are given in the order of T_A(1,1),T_A(1,2),.., T_A(1,N),T_A(2,1), All coefficients must lie in the same line and are separated by space. Each entry in the T_A matrix can also be set to the same value as specified.

```
TEMPB coefficients...
TEMPB ALL <value>
```

Keyword TEMPB defines the entries in the control matrix (T_B). The terms are given in the same order as T_A. Each entry in the T_B matrix can also be set to the same value as specified.

```
BIAS <bias_value>
```

Keyword BIAS defines the value of I_b.

```
BOUNDARY <boundary>
```

Keyword BOUNDARY specifies the boundary input value.

```
USE <template_name>
```

Instead of using the above 5 commands to specify the configuration of the compact neural network, the user can use the template that resides in the template library for short. The name had to exactly match and it is case-sensitive, which means that it is responsive to the difference between uppercase and lowercase letters.

Miscellaneous Commands

Commands described in this category in general is not necessary. They are mainly designed to simulate the network in some special conditions. For normal applications, commands can be omitted.

```
VXCUT <Vxcut_low> <Vxcut_high>
```

> Keyword VXCUT helps to simulate the case when the state variables will saturate when they approach V_{xcut_low} or V_{xcut_high}. That means the states values V_x will not exceed V_{xcut_high} or go below V_{xcut_low}.

Default Setting

In our implementation, R and C are normalized to 1 so all the time used hereafter are all specified in (normalized) time unit. The defaults for the inputs, initial states and boundary input values are all -1. The integration step size is 0.01 time unit and the integration period is 50 time units. The sizes of matrix T_A and T_B are 3, i.e. the neighborhood size is 1. No predefined template information is assumed and user has to specify the coefficients. The default bias is 0 and the tolerance value is 0.001. The default neuron output type is piecewise-linear function.

Sample Command File

The following is a complete command file without invoking the template library.

```
% comment: a sample example file
METHOD RK4
ISIZE 32 32
INPUT a.32
INITIAL a.32
OUTPUT a32.out
TRAN 0.05 100.0
```

```
TSIZE 3 0
TEMPA 0 0 0 1 2 -1 0 0 0
TEMPB 0
ANNEAL N
BIAS 0.0
TOL 0.001
BOUNDARY -1
```

If you want to use the template "CCD_V", a modified command file will look like:

```
% comment: a modified sample example file
METHOD RK4
ISIZE 20 20
INPUT ccd.20
INITIAL ccd.20
OUTPUT ccd.out
TRAN 0.05 30.0
ANNEAL N
TOL 0.0001
USE CCD_V
```

Appendix E

About the Author

E.1 Biography

Tony Hung-Yao Wu was born in Taiwan in 1967. He received the B.S. degree in electrical engineering from National Taiwan University, Taipei, in 1989. He received the M.S. and Ph.D. degrees both in electrical engineering from University of Southern California in 1992 and 1995, respectively. He will join Cirrus Logic, Fremont, CA, as a senior design engineer.

At USC, Mr. Wu was a teaching assistant for two graduate-level courses in multimedia technology and digital information highway. He worked as a graduate research assistant in the VLSI Signal Processing Laboratory where he also managed the computing facility and equipment. He had participated in many research topics including VLSI image processing and signal transmission, neural networks, and intelligent systems. In summer 1995, he worked in the AT&T Bell Labs. in Holmdel, NJ. He has been an active participant in IEEE activities. He served on the Technical Program Committee of the 1995 International Conference on Computer Design in the Architectures-and-Algorithm Track. He also served as an co-editor of the tutorial book for 1995 IEEE International Symposium on Circuits and Systems. He is a member of the IEEE.

E.2 Resume

EDUCATION

Jan. '93 - Dec. '95	Ph.D in Electrical Engineering, Univ. of Southern California, Los Angeles, CA (under the guidance of Prof. Bing J. Sheu)
Dec. 1992	M.S. in Electrical Engineering, Univ. of Southern California, Los Angeles, CA
Jun. 1989	B.S. in Electrical Engineering, National Taiwan University, Taipei, Taiwan.

EXPERIENCES

- Jun. 1995 - Aug. 1995
  Summer intern in AT&T Bell Labs. locating in Holmdel, New Jersey.
  Topics: Video DSP project
  - Design of new algorithm mapping based on the existing video co-processor which consists of 2-D systolic-array structure.
  - Work on both algorithm derivation and architecture mapping design.
  - System implementation and validation extensively using both high-level VHDL compiler/simulator and various physical layout/circuit simulation CAD tools.
  - The design is under internal evaluation of possible patent application.

- Jul. 1994 - present
  Graduate Research Assistant, VLSI Signal and Image Processing Lab, Dept. of Electrical Engineering, USC
  - Software-hardware codesign of low-power array processing systems.
  - Analysis and simulation of innovative technique to find optimal solutions in multiple-processor arrays.
  - Design of VLSI chip for parallel processing computing machine.
  - Design of software simulator for behavioral analysis and performance evaluation.
  - Management on computer and hardware facility.

- Jan. 1994-May 1994, Jan. 1995-May 1995
  Teaching Assistant, "Special Topics - Neural Network Processing and VLSI", Dept. of Electrical Engineering, USC
  - Guide weekly 1-hour discussion session.
  - Help to provide and prepare appropriate tools and supplementary materials for students.

- Sep. 1993-Dec. 1993, Sep. 1994-Dec. 1994
  Teaching Assistant, "Special Topics - VLSI Image Processing and Compression", Dept. of Electrical Engineering, USC
  - Teach VHDL language for high-level system simulation.
  - Teach MPEG-2 standards for 4 weeks while the instructor was absent.

- Lead students working on the final projects.
- Guide weekly 1-hour discussion session.
- Help to provide and prepare appropriate tools and supplementary materials for students.

- Jul. 1993-Jun. 1994,
  Graduate Research Assistant, VLSI Signal and Image Processing Lab, Dept. of Electrical Engineering, USC
  - Analysis and simulation of the hippocampal network.
  - Design of system-level architecture for hardware implementation.
  - Design of VLSI chips for emulation of brain memory function.

PROFESSIONAL SERVICES

- IEEE ICCD-95, Technical Program Committe on Architecture and Algorithm Track.
- Reviewer of IEEE Trans. on Circuits and Systems for Video Technology.
- Editor of IEEE ISCAS-95 Tutorial Book.

COURSE EMPHASIS

- Digital Image Processing
- Digital Signal Processing
- Advance Digital Signal Processing
- Special Topics: Low Power and High-Speed VLSI Design
- Medical Image Processing
- Communication System
- VLSI System Design
- Analog Integrated Circuits
- Computer Architecture

GPA = 3.90/4.0

COMPUTER SKILLS

- Computer Languages: C, Pascal, Fortran, 80x86 Assembly language, Perl, Unix Shell programming and DSP TI TMS320C3x/4x or Motorola DSP56000 assembly languages.
- Operating Systems: UNIX/ X-Window, VAX/VMS, MS-DOS, Macintosh OS and Linux.
- CAD Tools: VHDL, SPICE, MAGIC, SWITCAP2, IRSIM, Viewlogic, OrCAD, MATLAB, OASIS and various VLSI-CAD design tools.
- Familiar with MPEG-1/2.

HONOR/ACTIVITIES

- IEEE member: Circuit and System Society, Computer Society, and Signal Processing Society. 1992-present.
- Tau Beta Pi Honorary Member

PUBLICATIONS

• Books

- Chapter co-author of "Simulated Annealing, Boltzmann Machine and Hardware Annealing," in the book *The Industrial Electronic Handbook*, CRC Press, to be published in 1995.
- One of the editors of IEEE ISCAS-95 Tutorial Book *Microsystems Technology for Multimedia Applications: An Introduction*, IEEE Press, 1995.
- Assistant editor of *Neural Informational Processing and VLSI*, Kluwer Academic, 1995.

• Journal Papers:

- S.H. Bang, B.J. Sheu, **Tony H.-Y. Wu**, "Optimal Solutions for Cellular Neural Networks by Paralleled Hardware Annealing,", *IEEE Trans. on Neural Network* (accepted).
- **Tony H. Wu,** Bing J. Sheu, Eric, Y. Chou, "Behavioral Simulation of Densely-Connected Analog Cellular Array Processors for High-Performance Computing," *Analog Integrated Circuits and Signal Processing* (accepted).

• Conference Papers:

1. B.J. Sheu, C.-H. Chang, **Tony H.-Y. Wu**, "Neural information processing in analog current-mode VLSI," *Int. Symp. on Artificial Neural Networks*, pp. B-01-10, Hsinchu, Taiwan, R.O.C., Dec. 1993.
2. S.H. Bang, B.J. Sheu, **Tony H.-Y. Wu**, "Paralleled Hardware Annealing of Cellular Neural Networks for Optimal Solutions," *Proceeding of IEEE Int. Conference on Neural Networks*, pp. 2046-2051, 1994.
3. B. J. Sheu, R. C. Chang, **T. H. Wu**, "Applications, design, and test of mixed-signal ICs," Modern Engineering and Technology Seminar, Electronic Session, Taipei, Taiwan, Dec. 1994
4. B. J. Sheu, R. C. Chang, **T. H. Wu**, S. H. Bang, "VLSI-compatible cellular neural networks with optimal solution capability for optimization," IEEE Int. Symposium on Circuits and Systems, Seattle, WA, Apr. 1995.
5. Eric Y. Chou, Bing J. Sheu, **Tony H. Wu**, "VLSI Design of Densely-Connected Array Processors," IEEE International Conference on Computer Design, Austin, TX, 1995.

CHIPS DESIGNED

• Hippocampus Chip

- To mimic the function of the hippocampus in human brain for memory retrieval.
- Mixed analog-digital CMOS p-well 2-poly 2-μm technology (MOSIS) design.

• Cellular Neural Network Chip

- Prototype of a 5 by 5 cellular neural network with limited programmability.
- Mixed analog-digital CMOS p-well 2-poly 2-μm technology (MOSIS) design.

Reference List

[1] U. G. Committee, *Grand Challenges: High Performance Computing and Communications.* National Science Foundation, 1992.

[2] E. Holsinger, *How Multimedia Works.* Ziff-Davis Press, 1994.

[3] ISO/IEC JTC1/SC29/WG10. JPEG Committee Draft CD 10918, 1991.

[4] ISO/IEC JTC1/SC29/WG11. MPEG Committee Draft CD 11172, 1991.

[5] CCITT Recommendation H.261, 1990.

[6] S. Kung, *VLSI Array Processors.* Prentice-Hall, 1988.

[7] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* NY: McGraw-Hill, Inc, 1993.

[8] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective.* Addison-Wesley Publishing, 1993.

[9] H. Bakoglu, *Circuits, Interconnections, and packaging for VLSI.* Addison Wesley: Reading, MA, 1990.

[10] D. Hammerstrom, "A VLSI architecture for high-performance, low-cost, on-chip learning," in *Proc. IEEE/INNS Inter. Joint Conf. Nerual Networks*, vol. 2, pp. 537–543, 1990.

[11] D. H. D. Mueller, "A neural network systems component," in *Proc. IEEE Inter. Conf. Neural Networks*, vol. 3, (San Francisco), pp. 1258–1264, Mar. 1993.

[12] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, and B. Riley, "An 11-million transistor neural network execution engine," in *Tech. Digest IEEE Inter. Solid-State Circuits Conference*, (San Francisco, CA), pp. 180–181, Feb. 1991.

[13] D. Muelleer and D. Hammerstrom, "A neural network systems component," in *Proc. IEEE Inter. Conf. Neural Networks*, (San Francisco, CA), pp. 1258–1264, Mar. 1993.

[14] U. Ramacher, J. Beichter, and N. Brüls, "Architecture of a general-purpose neural signal processor," in *Proc. IEEE/INNS Inter. Joint Conf. Neural Networks*, vol. 1, (Seattle, WA), pp. 443–446, July 1991.

[15] S. Satyanarayana, Y. Tsividis, and H. Graf, "A reconfigurable VLSI neural network," *IEEE Jour. Solid-State Circuits*, vol. 27, pp. 67–81, Jan. 1992.

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

[17] P. W. Hollis and J. J. Paulos, "Artificial neural networks using MOS analog multipliers," *IEEE Jour. Solid-State Circuits*, vol. 25, pp. 849–855, Jun. 1990.

[18] B. W. Lee, H. Yang, and B. J. Sheu, "Analog floating-gate synapses for general-purpose VLSI neural computation," *IEEE Trans. Circuits and Systems*, vol. 38, pp. 654–658, Jun. 1991.

[19] A. Chiang and M. Chuang, "A CCD programmable image processor and its neural network applications," *IEEE Jour. Solid-State Circuits*, vol. 26, pp. 1894–1901, Dec. 1991.

[20] L. O. Chua and L. Yang, "Cellular neural networks: theory," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1257–1272, Oct. 1988.

[21] L. O. Chua and L. Yang, "Cellular neural networks: applications," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 1273–1290, Oct. 1988.

[22] L. O. Chua and T. Roska, "The CNN paradigm," *IEEE Trans. Circuits and Systems, Part I*, vol. 40, pp. 147–156, Mar. 1993.

[23] S. H. Bang, "Performance optimization in cellular neural networks and associated VLSI architectures," Tech. Rep. USC-SIPI Report No. 268, Signal and Image Processing Institute, University of Southern California, University Park/MC-2564, Los Angeles, CA 90089, 1994.

[24] T. Roska and L. Chua, "Cellular neural networks with non-linear and delay-type template elements and non-uniform grids," *Int Jour. Circuit Theory and Applications*, vol. 20, pp. 469–481, 1992.

[25] B. W. Lee, "VLSI design of adpative neural systems," Tech. Rep. USC-SIPI Report No. 152, Signal and Image Processing Institute, University of Southern California, University Park/MC-2564, Los Angeles, CA 90089, 1990.

[26] P. M. van Laarhoven and E. Aarts, *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, 1987.

[27] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, May 1983.

[28] B. W. Lee and B. Sheu, "General-purpose neural chips with electrically programmable synapses and gain-adjustable neurons," *IEEE Jour. Solid-State Circuits*, vol. 27, pp. 1299–1302, Sept. 1992.

[29] S. Bang, O.-C. Chen, J.-F. Chang, and B. Sheu, "Parallel hardware annealing in multi-level Hopfield neural networks for optimal solution," *IEEE Trans. Circuits and Systems, Part II*, vol. 41, Dec. 1994.

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

[31] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C.* Cambridge University Press, 1988.

[32] Visual Numerics, Inc., Houston, TX., *ISML Reference Manual, version 2.*

[33] J. Ortega and J. W.G. Poole, *An Introduction to Numerical Methods for Differential Equations.* Pitman Publishing Inc., 1981.

[34] T. Roska and J. Vandewalle, *Cellular Neural Network.* Wiley, 1993. In Chapeter: Cellular Neural Network Simulator User's Manual, ver. 3.6.

[35] J. P. de Gyvez, "XCNN: A software package for color image processing," in *Proceedings of the Third IEEE Inter. Workshop on Cellular Neural Networks and Their Applications*, pp. 219–234, Dec. 1994.

[36] R. Domínguez-Castro, S. Espejo, A. Rodríguez-Vázquez, I. García-Vargas, J. Ramos, and R. Carmona, "SIRENA: A simulation environment for CNNs," in *Proceedings of the Third IEEE Inter. Workshop on Cellular Neural Networks and Their Applications*, pp. 417–422, Dec. 1994.

[37] The simulator is written by J.A. Osuna, Additional Information can be found by ftp://ife.ethz.ch/pub/NeuroBasic.

[38] B. J. Sheu and J. Choi, *Neural Information Processing and VLSI.* Kluwer Academic Publishers: Boston, MA, Jan. 1995.

[39] S. Espejo, *VLSI Design and Modeling of CNNs.* Ph.D. Dissertation, University of Sevilla, Spain, Apr. 1994.

[40] "CNN analogic (dual) software library," Tech. Rep. Internal Report DNS-1-1993, Computer and Automation Institute, Hungarian Academy of Science, Jan. 1993.

[41] S. Bang, B. Sheu, and T. Wu, "Optimal solutions for cellular neural networks by paralleled hardware annealing," *IEEE Trans. on Neural Networks.* (Accepted).

[42] T. Matsumoto, L. Chua, and H. Suzuki, "CNN cloning template: connected component detector," *IEEE Trans. Circuits and Systems*, vol. 37, pp. 663–665, May 1990.

[43] T. Matsumoto, L. Chua, and R. Furukawa, "CNN cloning template: Hole filler," *IEEE Trans. Circuits and Systems*, May 1990.

[44] M. Ogorzalek, A. Dabrowski, and W. Dabrowski, "Hyperchaos, clustering and cooperative phenomena in CNN arrays composed of chaotic circuits," in *Proceedings of the Third IEEE International Workshop on Cellular Neural Networks and their Applications*, pp. 315–320, Dec. 1994.

[45] L. Chua and G.-N. Lin, "Canonical realization of Chua's circutis family," *IEEE Trans. Circuits and Systems*, vol. 37, no. 7, pp. 885–902, 1990.

[46] R. Saleh, *iSPLICES Version 3 User's Guide.* Dept. of Electrical and Computer Engineering, University of Illinoi, Urbana-Champaign.

[47] W. K. Pratt, *Digital Image Processing.* John Wiley & Sons, Inc.: New York, NY, 1991.

[48] D. I. Moldovan, *Parallel Processing: From Applications to Systems.* Morgan Kaufmann Publisher, 1993.

[49] T. Roska and L. Chua, "The CNN universal machine - an analogic array computer," *IEEE Trans. Circuits and Systems, Part II*, vol. 40, pp. 163–173, Mar. 1993.

[50] K. Crounse, T. Roska, and L. Chua, "Image half-toning with cellular neural networks," *IEEE Trans. Circuits and Systems, Part I*, vol. 40, pp. 267–283, Apr. 1993.

[51] L. Chua and B. Shi, "Multiple layer cellular neural networks: A tutorial," in *Algorithms and Parallel VLSI Architectures* (E. Deprettere and A. van der Veen, eds.), vol. A, pp. 137–168, New York: Elsevier, 1991.

[52] C.-C. Lee and J. P. de Gyvez, "Time-multiplexing CNN simulator," in *Proc. IEEE Int'l Symp. Circuits and Systems*, vol. 6, pp. 407–410, 1994.

[53] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools.* Addision-Wesley, 1985.

[54] B. W. Kernighan and D. M. Ritchie, *The* C *programming Language.* Prentice-Hall, 1988.

[55] W. Pennebaker and J. Mitchell, *JPEG: Still Image Data Compression Standard.* VonNostrand Reinhold, 1993.

[56] M. Sun and K.-M. Yang, "A flexible VLSI architecture for full-search block-matching motion-vector estimation," in *IEEE Int. Symp. on Circuits and Systems*, (Portland, OR), pp. 179–182, Oct. 1989.

[57] E. Chan and S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. on Consumer Electronics*, vol. 39, pp. 292–297, Aug. 1993.

[58] C. Wu and D. Yeh, "A VLSI motion estimator for video image compression," *IEEE Trans. on Consumer Electronics*, vol. 39, pp. 837–846, Nov. 1993.

[59] A. Costa, A. Degloria, P. Faraboschi, and F. Passaggio, "A VLSI architecture for hierarchical motion estimation," *IEEE Trans. on Consumer Electronics*, vol. 41, pp. 248–257, May 1995.

[60] J.-I. Guo, C.-M. Liu, and C.-W. Jen, "The efficient memory-based VLSI array designs for DFT and DCT," *IEEE Trans. Circuits and Systems, II*, vol. 39, pp. 723–733, Oct. 1992.

[61] Y.-H. Chan and W.-C. Siu, "On the realization of discrete cosine transform using the distributed arithmetic," *IEEE Trans. Circuits and Systems, I*, vol. 39, pp. 705–712, Sept. 1992.

[62] C.-M. Wu and A. Chiou, "A SIMD-systolic architecture and VLSI chip for the two-dimensional DCT and IDCT," *IEEE Trans. on Consumer Electronics*, vol. 39, pp. 859–869, Nov. 1993.

[63] Y.-F. Jang, J.-N. Kao, J.-S. Yang, , and P.-C. Huang, "A 0.8μm 100MHz 2-D DCT core processor," *IEEE Trans. on Consumer Electronics*, vol. 40, pp. 703–709, Aug. 1994.

[64] A. Madisetti and J. A.N. Willson, "A 100mhz 2-D 8x8 DCT/IDCT processor for HDTV applications," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, pp. 158–167, Apr. 1995.

[65] L.-W. Chang and M.-C. Wu, "A unified systolic array for discrete cosine and sine transforms," *IEEE Trans. on Signal Processing*, vol. 39, pp. 192–194, Jan. 1991.

[66] N. Cho and S. Lee, "DCT algorithms for VLSI parallel implementations," *IEEE Trans. on Acoustics, Speech and Signal Processing*, vol. 38, pp. 121–127, Jan. 1990.

[67] P. Duhamel and C. Guillemot, "Polynomial transform computation of the 2-D DCT," in *Proc. IEEE Int'l. Conf. Acoust., Speech, Signal Processing*, (Albuquerque, NM), pp. 1515–1518, Apr. 1990.

[68] C. Chiu and K. R. Liu, "Real-time parallel and fully pipelined two-dimensional DCT lattice structures with application to HDTV systems," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 2, pp. 25–37, Mar. 1992.

[69] U. Totzek and F. Matthiesen, "Two-dimensional discrete cosine transform with linear arrays," in *Proc. Int. Conf. Systolic Arrays* (J. McCanny, J. McWhirter, and J. E. Swartzlander, eds.), pp. 388–397, Hertfordshire: Prentice-Hall, 1989.

[70] W. Ma, "2-D DCT systolic array implementation," *Electron. Lett.*, pp. 201–202, Jan. 1991.

[71] C.-L. Wang and C.-Y. Chen, "High-throughput VLSI architectures for the 1-D and 2-D discrete cosine transforms," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, pp. 31–40, Feb. 1995.

[72] J.-F. Yang, B.-L. Bai, and S.-C. Hsia, "An efficient two-dimensional inverse discrete cosine transform algorithm for HDTV receivers," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, pp. 25–30, Feb. 1995.

[73] H. Karathanasis, "A low ROM distributed arithmetic implementation of the forward inverse DCT/IDCT using rotations," *IEEE Trans. on Consumer Electronics*, vol. 41, pp. 263–272, May 1995.

[74] LSI Logic Corp., Milpitas, CA, *CCITT Video Compression Databook*, Sep. 1991.

[75] P. Plantec, "MPEG-2: The new world standard for high-quality digital video," *Multimedia Today*, Apr. 1995.

[76] V. K. Madisetti, *VLSI Digital Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis.* IEEE Press, 1995.

[77] H.-D. Lin, "AVP-III Technical Report," tech. rep., AT&T Bell Labs., 1995.

[78] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, pp. 4–19, July 1989.

[79] S. Waser and M. Flynn, *Introduction to Arithmetic for Digital Systems Designers.* Holt, Rinehart and Winston Publishing, 1982.

[80] A. Jain, *Fundamentals of Digital Image Processing.* Prentice-Hall Publishing, 1989.

[81] W. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *IEEE Trans. Communication*, vol. 25, pp. 1004–1009, Sept. 1977.

[82] M. Sun, T. Chen, and A. Gottlieb, "VLSI implementation of a 16x16 discrete cosine transform," *IEEE Trans. Circuits and Systems*, vol. 36, pp. 610–617, Apr. 1989.

[83] S. Uramoto, Y. Inoue, A. Takabatake, J. Takeda, Y. Yamashita, H. Terane, , and M. Yoshimoto, "A 100MHz 2-D discrete cosine transform core processor," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 492–498, Apr. 1992.

[84] "IEEE standard specifications for implementation of 8x8 inverse discrete cosine transform," *IEEE Standard*, pp. 1180–1190, Mar. 1991.

[85] Y.-T. Chang and C.-L. Wang, "New systolic array implementation of the 2-D discrete cosine transform and its inverse," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 5, pp. 150–157, Apr. 1995.

[86] T. Kumagai, R. Hashimoto, and M. Wada, "Learning of limit cycles in discrete-time neural network," 1995. (Submitted).