VLSI STRUCTURES FOR DIGITAL COMMUNICATION RECEIVERS

by

Patrick Glenn Gulak

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
Electrical Engineering

Winnipeg, Manitoba

VLSI STRUCTURES FOR DIGITAL COMMUNICATION RECEIVERS

BY

PATRICK GLENN GULAK

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

A taxonomy of VLSI layouts are presented for the implementation of maximum likelihood sequence estimators realized by the Viterbi algorithm (VA), a dynamic programming solution to estimating a state sequence. These are classified in terms of increasing interprocessor wire area each of which are capable of increased data throughput. Cascade, Linearly and Orthogonally Connected Mesh, Shuffle-Exchange and Cube-Connected Cycles layouts can efficiently embed the VA in silicon. These structures are generalized to accommodate an arbitrary source alphabet size and algorithm memory length. The algorithm-structured layouts by implication are appropriate for convolutional decoding. The area * time and area * time^2 measures of complexity for the VA are presented and interpreted within the context of digital communications. One important result based on hardware considerations suggests that the algorithm memory length should be prime. Viterbi receivers for correlative encoded MSK, using first and second order encoding polynomials, are shown to reside in a generalized class of Cube-Connected Cycles layouts.

In addition, a Normalized Kolmogorov Metric Space is proposed which can be incorporated into the VLSI designs.

Though the simulation results are preliminary, this new metric space may find application in suboptimal soft decision decoding schemes.

ACKNOWLEDGEMENTS

The author wishes to thank Dr. E. Shwedyk for his patient supervision and encouragement throughout the extent of this thesis.

The author is also indebted to numerous colleagues, especially those associated with the Materials and Devices Research Laboratory (MDRL) at the University of Manitoba and the Computer Group (SF206) in the Electrical Engineering Department of the University of Toronto for various discussions and suggestions. Thanks to David Bowness for help with the planarity algorithm, and to Dr. J. Poltz for help in discovering more about full necklaces.

Financial assistance by the Natural Sciences and Engineering Research Council Postgraduate Scholarship Programme and the University of Manitoba Graduate Fellowship Programme is gratefully acknowledged.

CONTENTS

LIST OF FIGURES

SYMBOLS AND NOTATION

Symbol	Representation
$\underline{a}$	Vector of input symbols
a_k	Source symbol at time k
$\hat{a}_k$	Estimate of received source symbol at time k
A	Area of largest bounding rectangle
B	Carrier amplitude
d_k	encoded data bit at time k
D	Delay operator
Den	Denominator
f_c	carrier frequency
f_k	Equivalent discrete-time channel
f(D)	D polynomial of equivalent discrete-time channel
F	3dB channel filter bandwidth
h	modulation index
h_k	Discrete time channel impulse response
H_i	Hypothesis i
i,j,k,l	index variables
ln x	The natural logarithm of x
log x	The base two logarithm of x
$\log^y x$	$(\log x)^y$
m	Cardinality of the source alphabet set
mod x	modulo x
$M(X,\rho)$	Metric space on a set X with distance measure ρ

$n(t)$	Channel noise process
$\underline{n}$	Vector of discrete noise samples
n_k	White Gaussian noise sample value
n	Problem size index
$N_o/2$	Two-sided Spectral height of white Gaussian noise (joules)
Num	Numerator
$p(\bullet)$	Probability density function
P	A priori probability
P(e)	Probability of (symbol) error
$r(t)$	Received signal
R	Rate (1 / T)
R_i	Pulse autocorrelation coefficient
$s(t)$	Transmitted signal
t	time
T	Symbol Interval
u_k	Noiseless output of the whitened matched filter
w	Minimum bisection width
$\hat{x}(\alpha)$	Truncated survivor listing
y_k	Output sample of the whitened matched filter
α_k	State defined by $\{ \hat{a}_k, \hat{a}_{k-1}, \ldots, \hat{a}_{k-\nu+1} \}$
$\Gamma(\alpha)$	State metric
δ	Length of survivor listing
Δ	$h/(2T)$
$g(n)=\Theta(h(n))$	"g is theta of h," an exact bound within a constant factor. There exist positive constants c_1 and c_2 for which $c_1 h(n) \leq g(n) \leq c_2 h(n)$ for all sufficiently large n.

Θ	Phase offset
λ	Path metric in the Viterbi Algorithm or VLSI fabrication length unit (typ. $\lambda \sim 1\mu m$)
μ	Mean value (first moment)
ν	Viterbi Algorithm memory length
ξ	Random variable sample space
$g(n)=O(h(n))$	"g is big O of h," an upper bound within a constant factor. There exists a positive constant c for which $g(n) \leq ch(n)$ for all sufficiently large n.
ρ	Metric (Distance measure in a metric space)
σ^2	Variance (second central moment)
τ	Logic gate delay
ϕ	Information carrying phase
ψ	Element of the sample space ξ
$g(n)=\Omega(h(n))$	"g is omega of h," a lower bound within a constant factor. There exists a positive constant c for which $g(n) \geq ch(n)$ for all sufficiently large n.
$\lceil x \rceil$	"Ceiling of x": the least integer > x
$\lfloor x \rfloor$	"Floor of x": the greatest integer < x
$\overline{x}$	If x is m-ary, then $\overline{x}$ is any element from the set $\{0,1,\ldots,m-1\}$ such that $\overline{x} \neq x$. (Note: If x is binary (m=2), then $\overline{x}$ corresponds to the complement of x.)
$\lvert \bullet \rvert$	Absolute value
$\lVert \bullet \rVert$	Euclidean norm in L_2 (Hilbert Space)
<K>	Necklace of node K

TABLE OF ACRONYMS

Symbol	Representation
ACS	Add-Compare-Select
CC	Convolutional Code
CCC	Cube-Connected Cycles
CCD	Charge-Coupled Device
CLK	Clock
CMOS	Complementary Metal Oxide Semiconductor
CMOS/SOS	CMOS by Silicon on Sapphire
CPM	Continuous Phase Modulation
CORDIC	Coordinate Rotation Digital Computer
dB	Decibel
DCCC	Double Cube-Connected Cycles
DES	Data Encryption Standard
DFT	Discrete Fourier Transform
DLM	Distributed Logic Memory
FIR	Finite Impulse Response
FFT	Fast Fourier Transform
Hz	Hertz
IC	Integrated Circuit
IIR	Infinite Impulse Response
ISI	Intersymbol Interference
I/O	Input/Output
LIM	Logic in Memory
LEM	Logic Enhanced Memory

LPC	Linear Predictive Coding
MAP	Maximum a posteriori probability
Mbit/sec	Megabits per second
MHz	Mega-Hertz
ML	Maximum Likelihood
MLSE	Maximum Likelihood Sequence Estimation
MMSE	Minimum Mean Squared Error
MOS	Metal Oxide Semiconductor
MSE	Mean Squared Error
MSK	Minimum Shift Keying
PAM	Pulse Amplitude Modulation
pdf	Probability Density Function
PCCC	Pleated Cube-Connected Cycles
RL	Radial Line
ROM	Read Only Memory
RSA	Rivest-Shamir-Adelman
SE	Shuffle Exchange
SNR	Signal-to-Noise Ratio
TFM	Tamed Frequency Modulation
TP	Throughput
VA	Viterbi Algorithm
VHSIC	Very High Speed Integrated Circuits
VLSI	Very Large Scale Integration
WSI	Wafer Scale Integration
1-D	One Dimensional
2-D	Two Dimensional
3-D	Three Dimensional

Chapter I

INTRODUCTION

1.1 THE MOTIVATION

This thesis is motivated by the problem of digital communication over noisy time dispersive linear channels. Current research efforts in this area are due to the recent proliferation of distributed digital information sources/sinks (e.g.: telephony, satellites, computer networks) operating at very high data rates.

Increasing the data rate of a channel, reliably, involves a tradeoff between the total power and bandwidth allocated to the digital signal on one hand, and the complexity of the decision algorithm on the other hand. Until recently, the implementation of complex decision algorithms has been discouraged by the fact that sequential processing systems cannot compete in high data rate applications, while the design of a highly concurrent hardware implementation is disappointingly expensive and space consuming.

As an alternative, the technology of very large-scale integrated (VLSI) circuits opens unprecedented opportunities for realizing complex computational algorithms. However, the ability to integrate a tremendous amount of hardware on

a small silicon area is challenging many traditional digital logic design concepts. A distinct characteristic of VLSI circuits is that the on-chip data communication dominates the cost and performance of computing structures, whereas in traditional parallel processing it is assumed that the memories and the processors are the dominant factors (i.e., expensive interconnections vs. expensive devices).

1.2 OBJECTIVES

This thesis establishes how the potential of VLSI can be intelligently exploited in the realization of certain types of digital communication detectors. Specifically, this thesis will study well formed VLSI architectures that are capable of maximum likelihood sequence estimation (MLSE) as implemented by the Viterbi algorithm (VA). As an underlying theme it is maintained that architecture will not be separable from algorithms and often tradeoffs between time (baud interval) and implementation area are possible.

1.3 CONTRIBUTIONS OF THIS THESIS

Though several types of signal processing functions have been embedded in silicon, this thesis is the first attempt, to the author's knowledge, at investigating how certain types of decoders and detectors for digital communication can be realized as a VLSI circuit. The approach is novel in that we focus attention on embedding digital communication

algorithms in architectures that are developed using the VLSI grid model (which has made notable contributions to other problem domains in computer science since 1979). It is likely that this approach to the expression and evaluation of digital communication algorithms will redirect conventional thinking and become a standard technique in the future. Arising from this approach, the major contributions of this thesis are:

1. A new metric space, called a Normalized Kolmogorov Metric Space, is developed that has the property that distance in the signal space is bounded. This is a property of interest in a hardware realization that desires register and data paths of minimal width. In addition, the distance measure is parametric in the sense that it is a function of the probability density function of the noise source corrupting the signal. Though the results are preliminary, this metric space may find application in suboptimal soft decision decoding schemes.
2. A taxonomy of VLSI processor architectures are identified for implementing the VA concurrently. These are classified in terms of increasing interprocessor wire area. This complements the work of researchers studying other types of dynamic programming problems.
3. VLSI grid model layouts of the shuffle exchange graph are generalized, for the first time, to include m-shuffle exchange graphs.

4. All necklaces in a m-shuffle exchange graph are shown to be full when the algorithm memory length is prime. Cleaving the architecture into full necklaces is a reasonable point to start the integration of the design.
5. VLSI grid model layouts of the Cube-Connected Cycles (CCC) graph are generalized, for the first time, to accommodate m-ary alphabets while maintaining a vertex degree of four.
6. The CCC structure can be embedded in a tree of meshes graph because each is shown to have the same type of separator theorem.
7. Viterbi receivers for correlative encoded MSK are shown to fall within a generalized class of CCC structures.
8. It is demonstrated that for <u>any</u> VLSI implementation the area*time^2 product and the power dissipation of the VA is lower bounded by functions of the alphabet size and algorithm memory length alone.
9. Though specific reference is made throughout the thesis to the VA for MLSE of m-ary signals on inter-symbol interference (ISI) channels, some or all of the concurrent VLSI architectures described in the

following chapters directly apply to decoding (or detecting, as applicable):

- m-ary signals on ISI channels
- binary convolutional codes (CC)
- dual K codes (m-ary CC)
- punctured CC
- interleaved CC
- CC combined with MLSE on ISI channels
- coded modulation with multilevel/phase signals
- partial response signals
- correlative encoded MSK
- multi-h phase codes

Therefore, this research has identified a whole generation of VLSI microelectronic components for digital communication receivers.

1.4 THESIS ORGANIZATION

The subject material has been organized into five sections. Chapter 2 begins with a brief overview of digital communications in which important concepts are delineated. This leads to the development of a new metric space in which to operate a Viterbi receiver. Chapter 3 focusses concern on mapping algorithms into VLSI circuits. Initially, current VLSI implementations of digital communication algorithms are reviewed, followed by a more theoretically motivated introduction to computational models for VLSI. This forms the foundation for the development, in Chapter 4, of VLSI architectures for MLSE as realized by the Viterbi algorithm. The concurrent VLSI processor architectures are presented in order of increasing interprocessor wire area

each of which are capable of increased data throughput. It is important to note that these concepts can be used in the design of decoders for various types of transmission codes (e.g., convolutional codes, dc-free codes, run-length-limited codes, etc.) whose outputs can be modelled as outputs of a finite-state machine. As an illustration, Chapter 5 develops VLSI structures which implement Viterbi receivers for correlatively encoded MSK using first and second order encoding polynomials. Finally, Chapter 6 presents conclusions and describes some problems for further research.

Chapter II

DIGITAL COMMUNICATION RECEIVERS

2.1 INTRODUCTION

A digital communication system is one which communicates a discrete number (symbol) chosen from a finite set (alphabet). The process of "communication" involves transferring this data (symbol) from an information source through a medium to a destination or sink. This chapter presents a brief overview of important concepts in digital communications. Initially, Section 2.2 considers the PAM system model which establishes most of the notation used in later sections. Next, we consider the details of a specific digital communication receiver known as the Viterbi algorithm. Its attractive performance, in noise, motivates our quest to embed this algorithm in a VLSI format. One concept central to an implementation of the VA is that of path metric generation. This is discussed in more detail in Section 2.4. Finally, in Section 2.5 a new metric space is proposed in which path metric generation may be performed to particular advantage.

2.2 THE PAM SYSTEM MODEL

One of the simplest and most common digital modulation techniques [1] is pulse amplitude modulation (PAM). Increased data rates over a band-limited PAM system tend to induce interference among adjacent data pulses. This inter-symbol interference is usually the primary impediment to utilizing a reasonable fraction of the channel capacity when the signal to noise ratio is high.

A model of a basic PAM system is shown in Figure 2.1 . A sequence of data symbols, $\{ a_k \}$, each drawn independently from an alphabet of equally likely values $\{0,1,....,m-1\}$, are transmitted at a rate $R=1/T$ symbols per second. The linear channel with impulse response $h(t)$ is assumed to have a finite memory. The channel output signal is corrupted by additive Gaussian noise.

If a whitened matched filter is adopted as the receiving filter, it is more convenient to discuss the system in terms of the equivalent discrete time channel with impulse response $f(t)$, which is defined as the sampled output of the whitened matched filter due to a single impulse applied at the channel input. Thus, when the channel is characterized by $f(D) = f_0 + f_1 D + \ldots + f_\nu D^\nu$, where D is the delay operator and ν is the span of $f(t)$, the sampled output of the whitened matched filter at time k is

given by

$$y_k = \sum_{i=0}^{\nu} a_{k-i} f_i + n_k \qquad (2.1)$$

where n_k is an independent, additive Gaussian noise sample of variance σ^2. The receiver operates on the noisy distorted signal y_k to recover the transmitted data symbols a_k.

In order to combat the effects of intersymbol interference (ISI), a variety of linear and nonlinear receiver structures have been proposed [2]. Recently, a maximum likelihood sequence estimator (MLSE) implemented by the Viterbi algorithm (VA) has emerged as a seemingly ultimate solution to the problem of data transmission over time dispersive channels. Its performance can be shown to be as good as could be attained by any receiver structure. [3]

2.3 THE VITERBI ALGORITHM

The Viterbi algorithm was originally invented to detect convolutionally encoded data symbols [4]. Omura [5] showed the VA was equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph. About ten years ago, Forney [3] showed that the VA can be used to generate the maximum likelihood estimate of a transmitted sequence over a band-limited channel with intersymbol interference. The Viterbi processor requires that the ISI at the channel output be limited to a finite number of symbols and can be thought of as the dynamic programming

solution to the problem of estimating the state sequence of a finite-state Markov process observed in memoryless noise [6].

A survey of the Viterbi algorithm is given by Forney [7]. Its applicability to receivers for channels with intersymbol interference and correlative level coding (partial-response) coding was suggested by Kobayashi [8]. Application of the Viterbi algorithm to digital magnetic recording systems [9,10], adaptive delta modulation [11], speech and pattern recognition [12-15] have been discussed. Adaptive versions of Forney's receiver have been proposed [16] and its combination with decision feedback equalization has been suggested [17]. A thorough discussion of the use of the VA to combat ISI is given in Viterbi and Omura [18] and here we review some of the basic concepts for completeness.

The fundamentals of the Viterbi algorithm can be described by considering the discrete time model for the detector input, defined by

$$y_k = u_k + n_k \quad , \text{where} \quad u_k = \sum_{i=0}^{\nu} a_{k-i} f_i \qquad (2.2)$$

As derived from the Likelihood Ratio Test in white Gaussian noise take Γ_N , defined below for an N symbol sequence, as the objective function to be minimized

$$\Gamma_N = \sum_{k=1}^{N} (y_k - \hat{u}_k)^2 \qquad (2.3)$$

where $\hat{u}_k$ is derived from an estimate of the transmitted

sequence $\{\hat{a}_k\}$, given as

$$\hat{u}_k = \sum_{i=0}^{\nu} \hat{a}_{k-i} f_i \qquad (2.4)$$

for $N >> \nu$ ($N = 5\nu$ is satisfactory [18]).

If the noise n_k is white and Gaussian minimizing Γ_N corresponds to maximizing the log likelihood ratio for the detection of the data sequence, which minimizes the probability of sequence error. Equation 2.3 can be minimized using Bellman's dynamic programming technique [19].

Thus from (2.3), the recursive relation for time k is

$$\Gamma_k = \Gamma_{k-1} + (y_k - \hat{u}_k)^2 = \Gamma_{k-1} + \lambda_k \qquad (2.5)$$

Introducing the state definition

$$\alpha_{k-1} = \{\hat{a}_{k-1}, \hat{a}_{k-2}, \ldots, \hat{a}_{k-\nu}\} \qquad (2.6)$$

the Bellman equation of Dynamic programming yields

$$\Gamma_k^*(\alpha_k) = \min_{\alpha_{k-1} \to \alpha_k} [\Gamma_{k-1}^*(\alpha_{k-1}) + (y_k - \hat{u}_k)^2] \qquad (2.7)$$

where the minimum is taken over all possible predecessor states into each possible present state, and the asterisk denotes the optimum.

In order to implement the algorithm one must calculate and store $\Gamma^*(\alpha_k)$ for all α_k and, in addition, one must store the past sequence of state transitions (paths) into

each of the possible present states α_k. Figure 2.2 illustrates the Viterbi algorithm for a channel memory of two and a binary alphabet. The four possible states along with their corresponding state transitions are illustrated. This graphical representation is usually called a trellis.

As the procedure indicated in equation 2.7 is repeated for each state and for each time interval, a unique set of surviving paths result, one into each state. The dynamic programming equation is then only a method of finding the shortest route through a graph.

Eventually, at some point in time, all surviving paths merge backwards through the trellis. The merge time is a random variable and typically is assumed to occur within 5ν [18]. When a merge does occur, path detection can take place up to the merge point since further extensions will always originate from that state thereafter.

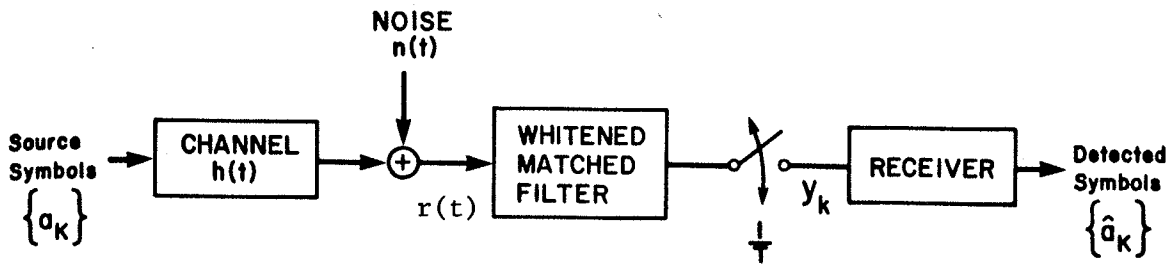


Figure 2.1: PAM COMMUNICATION SYSTEM

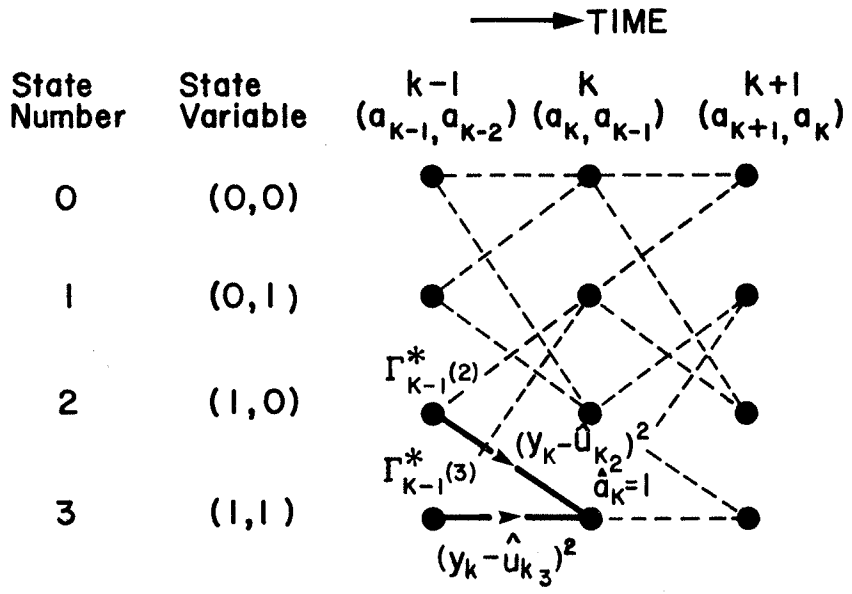


Figure 2.2: STATE AND PATH DIAGRAM FOR THE VITERBI ALGORITHM

2.4 PATH METRIC GENERATION

A typical hardware implementation of a Viterbi receiver for a binary alphabet and a channel memory of two is illustrated in Figure 2.3 . The structure consists of four identical processing cells each corresponding to one of the ISI states. The trellis is similar [7] to the computational flow diagram of the fast Fourier transform (FFT). Unlike the FFT, the basic operations within each cell consist of add, compare and select functions. In Figure 2.3, $\Gamma(\alpha)$ corresponds to the state metric or length, $\lambda(\alpha)$ corresponds to the path metric or length, $\hat{x}(\alpha)$ is the truncated survivor listing of δ symbols. The complexity of the algorithm is easily estimated. With an algorithm memory of ν and alphabet size of m, there are m^{ν} possible states α each associated with a particular ISI condition. The algorithm requires $2m^{\nu}$ storage locations, one for each of the state metrics $\Gamma(\alpha)$ and one each for the truncated survivor listing $\hat{x}(\alpha)$ of δ symbols (each symbol is one of m possible). Typically, $\delta = 5\nu$ is chosen with little degradation in performance. Computationally, in each unit of time the algorithm must make $m^{\nu+1}$ additions, one for each transition, and m^{ν} comparisons among the $m^{\nu+1}$ results. Thus the amount of storage is proportional to the number of states and the amount of computation to the number of transitions. It is important to note that the VA requires a fixed number of computations per symbol independent of the number of symbols received.

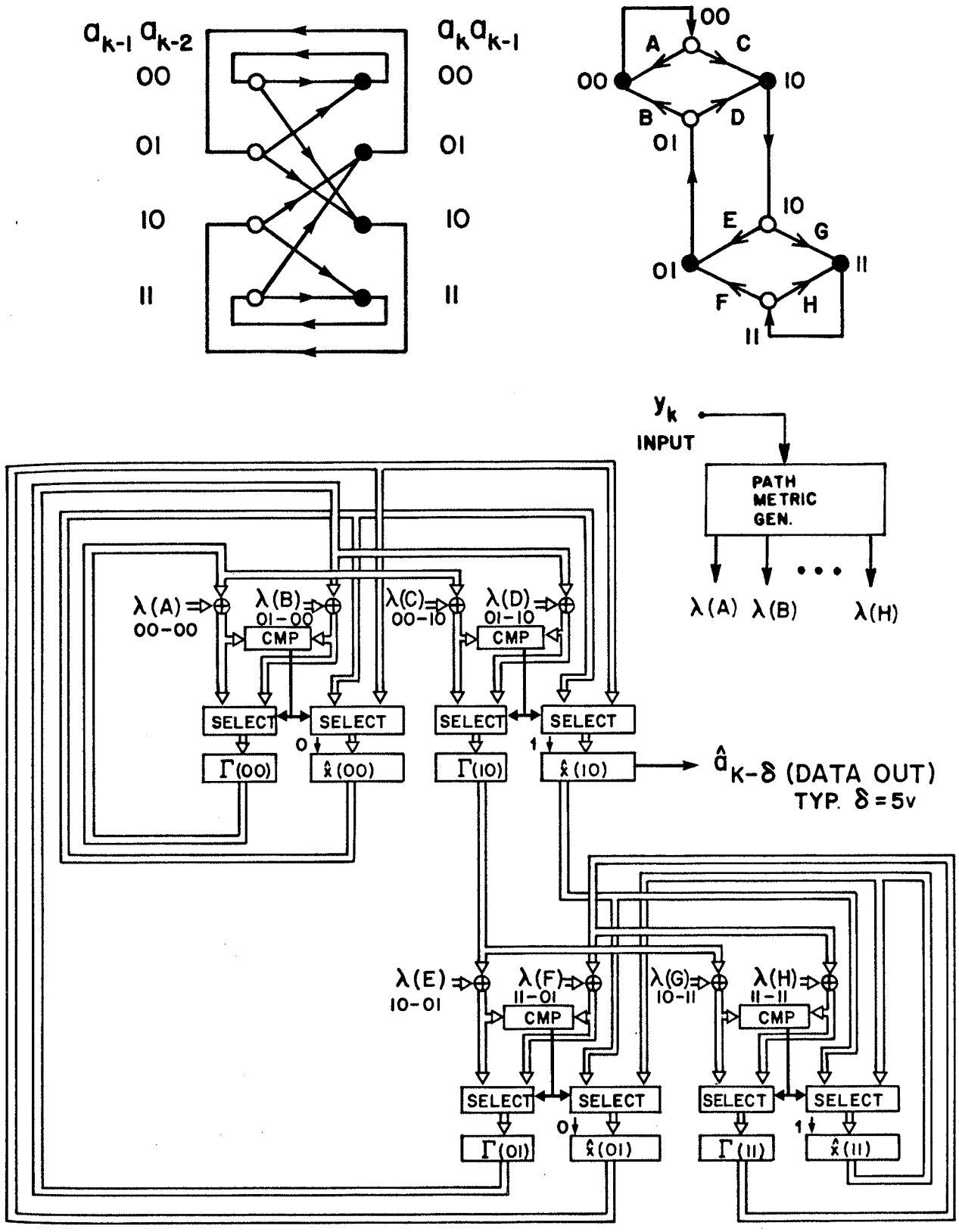


Figure 2.3: PARALLEL HARDWARE IMPLEMENTATION OF THE VA

In the previous paragraphs, the complexity involved in generating the incremental path lengths λ was ignored. For a discrete input sequence $\underline{a} = (a_1, a_2, a_3, \ldots, a_N)$ and if the receive filter is a whitened matched filter as discussed in Forney [3], then, $\Gamma_N(\underline{\hat{a}})$ becomes as in equation 2.3

$$\Gamma_N(\underline{\hat{a}}) = \| \underline{y} - \underline{\hat{u}} \|^2 = (\underline{y} - \underline{\hat{u}})^T (\underline{y} - \underline{\hat{u}}) = \sum_{k=1}^{N} (y_k - \hat{u}_k)^2. \qquad (2.8)$$

Hence, the path metrics λ_k are generated using

$$\lambda_k = \left[y_k - \sum_{i=0}^{\nu} \hat{a}_{k-i} f_i \right]^2 \qquad (2.9)$$

However, a whitened matched filter is not explicitly necessary, for as pointed out in Viterbi and Omura [18,p.274], if the receiver filter (with output y_k) is just a matched filter, an equivalent expression for path metric generation may be developed as shown below. When the channel is characterized by

$$r_k = \sum_{i=1}^{N} a_{k-i} h_i + n_k \qquad (2.10)$$

it follows that the maximum likelihood decision rule can be based on,

$$\sum_{k=1}^{N} (r_k - \sum_{i=1}^{N} \hat{a}_i h_{k-i})^2 \qquad (2.11)$$

which upon expansion yields,

$$\Gamma_N = \sum_{k=1}^{N} \left[r_k^2 - 2 \sum_{i=1}^{N} r_k \hat{a}_i h_{k-i} + \sum_{i=1}^{N} \sum_{j=1}^{N} \hat{a}_i \hat{a}_j h_{k-i} h_{k-j} \right] \qquad (2.12)$$

The first term $\sum_{k=1}^{N} r_k^2$ is an energy term independent of a_k and hence can be ignored in making comparisons. With the objective of simplifying the second term of equation 2.12 identify y_i as

$$y_i = \sum_{k=1}^{N} r_k \, h_{k-i} \tag{2.13}$$

This term can be identified as the sampled output of a matched filter for the channel. The third term can be simplified by observing that the pulse autocorrelation coefficients can be defined by

$$R_{i-j} = \sum_{k=1}^{N} h_{k-i} \, h_{k-j} \tag{2.14}$$

Now, the objective function to be minimized as derived from equation 2.12, can be rewritten as

$$\Gamma_N = -2 \sum_{i=1}^{N} \hat{a}_i \, y_i + \sum_{i=1}^{N} \sum_{j=1}^{N} \hat{a}_i \, \hat{a}_j \, R_{i-j} \tag{2.15}$$

Equation 2.15 can then be split into a form similar to that of equation 2.5 where

$$\Gamma_{k-1} = -2 \sum_{i=1}^{k-1} \hat{a}_i \, y_i + \sum_{i=1}^{k-1} \sum_{j=1}^{k-1} \hat{a}_i \, \hat{a}_j \, R_{i-j} \tag{2.16}$$

and

$$\lambda_k = -2\hat{a}_k \, y_k + 2\hat{a}_k \sum_{i=k-\nu}^{k-1} \hat{a}_i \, R_{k-i} + (\hat{a}_k)^2 R_0 \tag{2.17}$$

where ν is the channel memory length in units of T. Note that since the COMPARE operation is done between m transitions for the same state the constant $(\hat{a}_k)^2 R_o$ and the

constant factor 2 may be dropped from equation 2.17 producing:

$$\lambda_k = \hat{a}_k [\sum_{i=k-\nu}^{k-1} (\hat{a}_i R_{k-i}) - y_k] \qquad (2.18)$$

If the source symbols a_k are binary all the multiplications in 2.18 become additions. For time varying channels estimated R_i's would be supplied to the path metric calculation hardware each symbol interval. Symmetries in R_i allow 2.18 to be simplified further [18, p.275].

2.5 THE NORMALIZED KOLMOGOROV METRIC SPACE

The receiver defined by the Viterbi algorithm uses a metric λ for the branches of the trellis diagram. This metric may correspond to many possible forms such as those defined by ML, MAP, or MSE criterion. In this section, the performance of the VA in a metric space, which allows us to cater for noise sources with various types of probability density functions (pdf), is studied.

In Gaussian noise, the appropriate ML metric is the Euclidean squared distance. Other distance measures are possible with a subsequent degradation in probability of error performance. These may be justified by other considerations such as hardware complexity and processor throughput. The performance degradation may even be desireable in pseudo-error rate monitoring [20].

One such example would be the use of an absolute value distance measure $|y_i - \hat{u}_i|$. This distance measure is optimal, in a maximum likelihood sense, for Laplacian noise. However, the simplicity of hardware implementation may tempt one to consider its use in a Gaussian noise environment.

In fact, the distance between the received signal and any of the ISI states can be measured by any convenient metric. A large number of metrics have been suggested in the pattern recognition literature [21-24], each having particular advantages and drawbacks. Some of the more important

metrics used as interclass distance measures [21] are: Minkowski, City Block, Euclidean, Chebychev, Quadratic, Nonlinear.

Establishing the likelihood ratio always provides the appropriate "distance measure" required. However, for non-Gaussian noise sources the analysis may not be mathematically tractable. Despite this fact, the main criticism to applying one of the above mentioned distance measures indiscriminately is that they are not closely related to the error probability.

A more appropriate metric which reflects the local probability structure of data can be obtained by using more sophisticated probabilistic distance measures. For the two class separability problem, the following measures have been suggested:

Chernoff
Bhattacharyya
Matusita
The Divergence
Patrick-Fisher
Lissack-Fu
Kolmogorov Variational Distance

All these measures provide an upper (lower) bound to the error probability. Of particular interest is the Kolmogorov variational distance:

$$\int |p(\xi|H_1)P_1 - p(\xi|H_2)P_2| d\xi \qquad (2.19)$$

which is closely related to the Bhattacharyya coefficient

[25]. Note that in (2.19) the integral is simply the area indicated in Figure 2.4 .

At this point, we would like to introduce a new metric space derived from the Kolmogorov Variational Distance.

Consider the metric space $M(X,\rho)$ where:

the set X = {stationary random variables ε $\rho(\xi|H_1)$, $\rho(\xi|H_2)$}

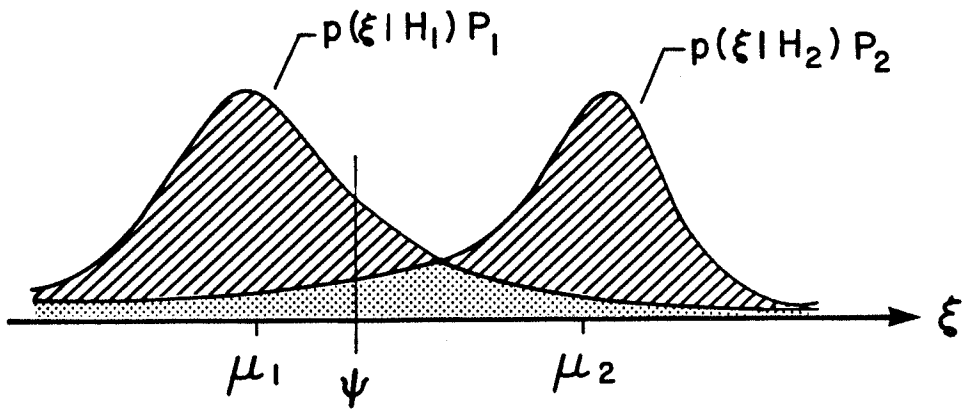
and the distance measure ρ is defined as

$$\rho_{H_1}(\psi) = \frac{\int_{-\infty}^{\psi} \max[p(\xi|H_1)P_1, p(\xi|H_2)P_2]d\xi - \int_{-\infty}^{\psi} |p(\xi|H_1)P_1 - p(\xi|H_2)P_2|d\xi}{\frac{1}{2} \cdot \left[1 - \int_{-\infty}^{+\infty} |p(\xi|H_1)P_1 - p(\xi|H_2)P_2|d\xi \right]} \qquad (2.20)$$

$$\rho_{H_2}(\psi) = 1 - \rho_{H_1}(\psi) \qquad (2.21)$$

where ψ is an element of the sample space ξ and the subscript on ρ indicates "with respect to". This definition relates the distance of a sample from a specified hypothesis to its contribution to the probability of error. Note that the distance measure is the distribution function of the error probability for the maximum likelihood (ML) receiver.

The proof that $\rho_{H_i}(\psi)$ satisfies the three metric space axioms [26] is given in Appendix A. For example, equiprobable hypotheses H_1, H_2 with Gaussian conditional densities



$$\blacksquare = \int_{-\infty}^{+\infty} | p(\xi|H_1)P_1 - p(\xi|H_2)P_2 | \, d\xi$$

$$\blacksquare = 1 - 2 \cdot \square$$

Figure 2.4: THE KOLMOGOROV VARIATIONAL DISTANCE

of mean μ_1, μ_2 and standard deviation σ_1, σ_2, respectively, would have the following distance definition ρ, when $\mu_2 > \mu_1$:

$$\rho_{H_1}(\psi) = \frac{Q\left(\frac{\mu_2 - \psi}{\sigma}\right)}{2Q\left(\frac{|\mu_1 - \mu_2|}{2\sigma}\right)} \quad , \forall \ \psi \leq \frac{\mu_1 + \mu_2}{2}$$

$$= 1 - \frac{Q\left(\frac{\psi - \mu_1}{\sigma}\right)}{2Q\left(\frac{|\mu_1 - \mu_2|}{2\sigma}\right)} \quad , \forall \ \psi > \frac{\mu_1 + \mu_2}{2} \tag{2.22}$$

$$\rho_{H_2}(\psi) = 1 - \rho_{H_1}(\psi) \tag{2.23}$$

where: $$Q(\alpha) = \frac{1}{\sqrt{2\pi}} \int_{\alpha}^{\infty} e^{-\beta^2/2} \, d\beta \tag{2.24}$$

This particular metric exhibits the property that samples separated by a Euclidean distance of more than 3σ are essentially the same distance (within 1.0%) using the new metric space. A graphical representation of such an effect is presented in Figure 2.5 . Note that the distance measure always has a finite range between zero and one. This is a property of interest in a hardware realization that desires register and data paths of minimal width.

The case of data dependent noise, where the noise variance is dependent on the hypothesis, is a representative model for certain channel nonlinearities. For hypothesis H_1, H_2 with corresponding μ_1 , μ_2 and σ_1 , σ_2 and a priori

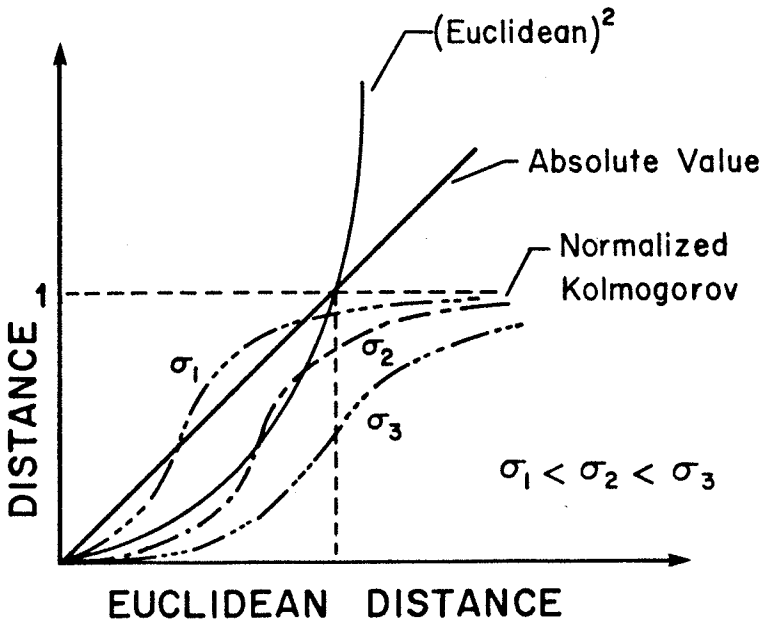


Figure 2.5: THE NORMALIZED KOLMOGOROV METRIC
Note that for the one-dimensional case, the Euclidean distance equals the absolute value distance measure.

probabilities P_1, P_2 the following distance definition would apply:

$$\rho_{H_1}(\psi) = \frac{P_2 Q(\frac{\mu_2 - \psi}{\sigma_2})}{P_1 Q(\frac{|\mu_1 - \mu_2|}{2\sigma_1}) + P_2 Q(\frac{|\mu_1 - \mu_2|}{2\sigma_2})} \quad , \forall \psi \leq \xi_0$$

(2.25)

$$= 1 - \frac{P_1 Q(\frac{\psi - \mu_1}{\sigma_1})}{P_1 Q(\frac{|\mu_1 - \mu_2|}{2\sigma_1}) + P_2 Q(\frac{|\mu_1 - \mu_2|}{2\sigma_2})} \quad , \forall \psi > \xi_0$$

where:

(2.26)

$$\xi_0 = \frac{\mu_1^2\sigma_2^2 - \mu_2^2\sigma_1^2}{\sigma_1^2 - \sigma_2^2 + 2\mu_1\sigma_2^2 - 2\mu_2\sigma_1^2} + \frac{2\sigma_1^2\sigma_2^2}{\sigma_1^2 - \sigma_2^2 + 2\mu_1\sigma_2^2 - 2\mu_2\sigma_1^2} \ln[\frac{P_1}{P_2}]$$

Development of the Normalized Kolmogorov Metric Space is based on a priori knowledge of the form of the conditional density function. By the principle of maximum entropy, it is reasonable to choose a normal density when the mean and variance are the only known parameters. (Several methods are available for estimating these parameters.) When parametric estimation is not sufficient it becomes necessary to estimate the density function by direct functional approximation. An iterative technique employed with an appropriate training sequence can estimate the true pdf to an arbitrary degree of closeness [27].

In order to evaluate the utility of the proposed metric a computer program (Appendix D) was written to simulate the operation of the architecture illustrated in Figure 2.3 .

Register Transfer Level computer simulations of this receiver, evaluating the probability of error as a function of SNR [28], were carried out for binary data on a single pole channel of memory two, algorithm memory of two and an 8 bit path memory. The data rate, R =1/T, was fixed at four times the 3dB channel filter bandwidth F (i.e., F /R = 1/4). The motivation was to study the effect of implementation losses on performance [29,30] hence the simulation modelled each binary function within the processor. Control of the bus width (in bits), input signal quantization, path metric computations and other parameters critical to the operation of the processor allowed for an analysis of each of these factors on performance. The results of the simulation are presented in Figure 2.6.

The relative performance (symbol P(e) .vs. SNR) of the new normalized Kolmogorov metric space, proposed in this chapter, the least squares and the absolute value metrics is shown in Figure 2.6. The Euclidean squared type metric has the best performance followed by the Normalized Kolmogorov metric and lastly the Euclidean metric. At a P(e) of 10^{-3} all metrics are within 0.5 dB of one another, when at least 25 error events per SNR value were observed. A limited number of simulation results in both Gaussian and Laplacian noise suggest a similar ordering holds for P(e) versus agc, carrier phase and register width. If the metric space calculations within the Viterbi receiver are table driven,

the new proposed metric poses no additional computational burden on the Viterbi processor. Though these results are preliminary in nature they do appear to justify further research aimed at establishing performance bounds in different noise environments, channel characteristics and implementation errors.

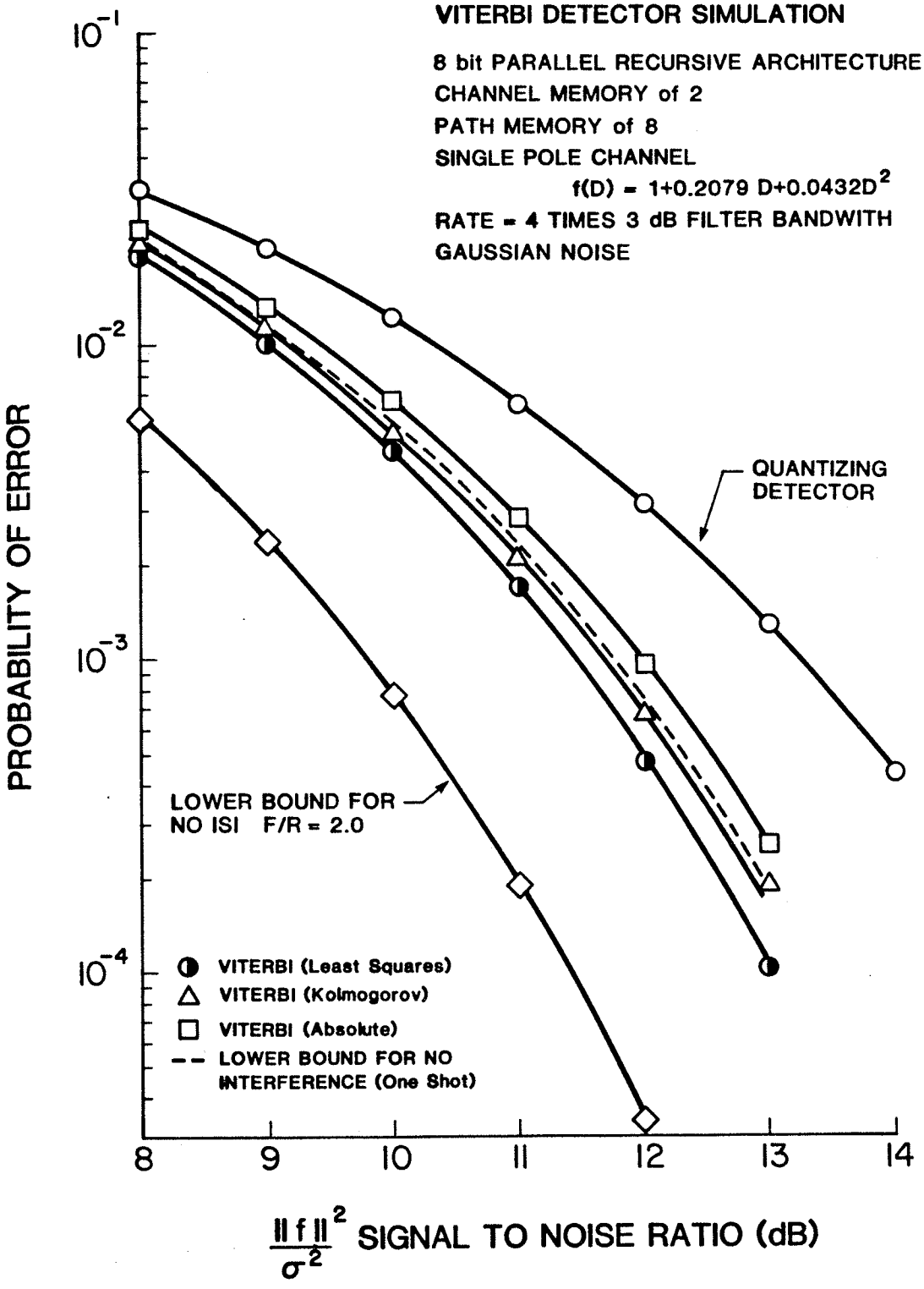


Figure 2.6: P(e) versus SNR IN VARIOUS METRIC SPACES

Chapter III

MAPPING ALGORITHMS INTO VLSI CIRCUITS

3.1 INTRODUCTION

Very Large Scale Integration (VLSI) technology offers the potential of implementing complex algorithms directly in hardware [31,32] with ten million transistor chips being promised shortly [33]. This resource encourages one to cast an algorithm as a structure of concurrent processes promising as a reward tangible increases in performance over traditional von-Neumann architectures. However, the design of significant computing structures using VLSI is notoriously expensive [34]. The task of VLSI design involves bridling the complexity of algorithm realization, using a surface of thousands of simultaneously active computing elements. Complexity control is achieved by defining a topologically regular processor and wire layout which allows a system of concurrent processes to efficiently move information from where it was produced to where it is needed in the next time instant.

This chapter presents the background material necessary in the development of well structured, highly concurrent VLSI layout strategies for realizing Maximum Likelihood

Sequence Estimators via the Viterbi algorithm (VA). Section 3.2 briefly surveys the marriage of VLSI and digital communications to date. Absent from the literature is an appreciation of how the Viterbi algorithm can be efficiently implemented in the VLSI domain.

In Section 3.3, a computational model for VLSI circuits is introduced which provides a high level of abstraction at which to start the design of VLSI circuits, thus allowing one to think in algorithmic rather than electrical terms. Efficient, practical circuits can be designed using this approach. By taking a realistic account of the placement of processing elements and their interconnection, the VLSI model of computation allows us to identify which compositions will result in: modularity and ease of layout, local communication paths, regular control and timing structures, extensibility and minimum die area (yield is an inverse exponential function of die area).

Finally, in Section 3.4, concepts are presented that will let us talk about the complexity of problems in a formal way. In later chapters, this will allow us to lower bound commodities such as area and time to within a constant factor, that the VA requires if it is to be implemented on an integrated circuit.

3.2 VLSI AND DIGITAL COMMUNICATIONS - A LITERATURE REVIEW

The progress in VLSI technology, to date, has yielded economical hardware realizations of highly sophisticated signal processing algorithms.

Commercial products currently range from special purpose devices such as CCD Sampled Analog Delay Lines [35], Sampled Analog Correlators/Convolvers [36] and Sampled Digital Correlators/Convolvers [37] useful in the realization of matched filters, programmable transversal filters and adaptive equalizers, to general purpose programmable digital signal processors [38-41] for the realization of digital filters, codecs, LPC synthesizers, etc. Beyond basic devices and general purpose processors numerous publications have focused on novel architectures and techniques for many important signal processing algorithms.

Ahmed et al. [42,43] suggests that algorithms appearing to possess additions and multiplications as fundamental operations are actually more naturally formulated in terms of elementary plane and hyperbolic rotations (CORDIC). Applications to speech synthesis and least mean square adaptive equalization are outlined.

Alternatively, bit serial techniques are proposed as a simple and efficient way to implement digital filters and other signal processing systems in silicon [44]. Bit-serial architectural techniques do not overconstrain the system

level architecture and do not necessarily limit the applications to low-bandwidth problem domains.

Inner product step computer realizations have been discussed extensively (see [45,46] for example). Pragmatic approaches to digital signal processing circuits for VLSI are outlined in Bloch et al. [47].

Logic-In-Memory (LIM) [48,49], Distributed Logic Memories (DLM) [50] and more recently Logic-Enhanced-Memory (LEM) [51,52] systems have been proposed to relieve the computational burden of matched filtering, pattern recognition and other signal processing tasks. In a rather different context, error detection and correction for memory systems can also be accomplished using VLSI devices [53].

Systolic Architectures [54,55] have been developed for FIR filtering, IIR filtering, convolution [56-58] and median filtering [59]. Several high speed charged coupled signal processing architectures [60-62] have been proposed for matched filtering, DFT's and image transforms, some using multivalued logic.

There are currently available high speed (up to 16 Mbits/sec) VLSI implementations of symmetric data encryption algorithms such as the Data Encryption Standard or DES [63] type and asymmetric or public key data encryption algorithms such as the Rivest-Shamir-Adelman or RSA [64] scheme and knapsack type [65].

Symbol slice or layout slice techniques [66] have been proposed as a design methodology for the realization of Reed Solomon Encoders [67].

Efficient VLSI structures for solving dynamic programming problems [68] have been the subject of several recent publications [69-71]. This work is particularly interesting as the Viterbi algorithm can be thought of as a dynamic programming solution to estimating a state sequence. This naturally leads us to consider how the Viterbi algorithm could be efficiently implemented in a VLSI format.

Considerations for the hardware implementation of the Viterbi algorithm [9,72-74] have generally dealt with discrete IC's where the major concern was the hardware (cell) cost and not the communication (wire area) costs associated with a highly concurrent VLSI computing environment. When the hardware costs are minimized (often implying global communication paths in a von Neumann architecture) without regard to exploiting the inherent parallelism in the algorithm, as in recent microprocessor realizations [75-77], throughput is drastically reduced. A recent VLSI implementation of the Viterbi algorithm for convolutional decoding [78] exploited the inherent parallelism of the algorithm to achieve up to a 46 Mbits/sec throughput rate. However, it was found that 38% of the space on the IC was occupied by the wires that carry control and data to the functional blocks. Analog Viterbi decoding structures [79], which may

provide throughput rates of up to 200 Mbits/sec suffer from the same expensive interconnection problems that digital implementations suffer from.

The remaining chapters of this thesis, attempt to explore different layout strategies for the Viterbi algorithm and its derivatives that acknowledges the design effort required in laying out communication paths (wiring) that are a good fit to VLSI. The first step towards this goal is the development of an abstraction of integrated circuit layouts, known as the grid model, which is presented in the next section.

3.3 A COMPUTATIONAL MODEL FOR VLSI

A VLSI chip can be viewed as a computation graph whose vertices are called nodes and whose edges are called wires. Nodes or processing elements are a collection of transistors and are responsible for information processing (computation of Boolean functions). Wires are just electrical connections, and are responsible for both the transfer of information between nodes and the distribution of power supply and timing waveforms to nodes. This correspondence seems straight forward enough but practical considerations force us to restrict the discussion to classes of graphs with vertex degrees that are bounded by a constant, and by further assuming that vertices require only a constant area of silicon. This is not to trivialize the design effort in

building a functional node but rather this assumption allows one to simplify the structure so as to develop insights into the wiring required to realize a particular algorithm.

Following the Thompson model for VLSI [80], often referred to as the grid model, we assume that there is only one layer of interconnect material, and that wires are laid out on a rectangular grid. Thus, wires may meet only at right angles. Wires may also cross over each other at right angles if one of them makes a run in a heavily doped "channel" in the silicon substrate. The assumption of one layer of interconnect material (planar embedding) is not overly restrictive in the sense that the availability of q layers of interconnect material can only reduce the area requirements of a layout by at most a constant factor of q^2.

There is a natural unit of area for VLSI. Manufacturing and physical limitations give rise to a minimal spacing between centers of parallel wires (i.e., pitch). In the terminology of Mead and Conway [31], this minimal spacing is called λ (not to be confused with path metrics defined earlier - usage should be clear from context). Thus a convenient area unit is the square of this length, λ^2 . The area of a chip can be expressed in terms of unit squares with one unit square being just large enough to contain one small transistor or one wire cross-over. A 64K RAM has an area of about $10^5 \lambda^2$, and chips of $10^8 \lambda^2$ may be possible [80].

The salient features of the Thompson model may be summarized as follows:

1. One bit of storage or logic at a node requires O(1), that is, Order-At-Most one or constant area.

2. Wires are O(1) units wide and are laid out on a rectangular grid. Thus, wires can cross only at right angles.

3. The unit of time is defined by specifying that a unit-width wire has at most unit bandwidth. A wire of length K has a driver of area K, which consists of log K stages of amplification. Therefore, the delay of the wire and driver is together O(log K). However, the amplifier stages are individually clocked and pipelining can be used to transmit one bit every O(1) units of time through the wire (propagation time independent of wire length).

Other VLSI models have been proposed [81], after [80], which differ chiefly in the time required for a bit of information to propagate across a wire.[1] Attention has focussed on this aspect of the model because the time to communicate between processors [82] generally dominates over processing time, and thus sets a lower limit to the achievable performance. One of the fundamental realizations that VLSI has brought into computing is that the expense in area or time of transmitting a result from where it is produced to where it is needed can often equal or exceed the cost of producing the result in the first place. As a result, algorithms that exhibit local communication are most desir-

---

[1] One other point of contention within the various VLSI grid models, not considered in this thesis, is how to adequately model chip I/O. We implicitly use a grid model which assumes that the system is: synchronous, semellective, word-local, when-determinate and where-determinate.

able. Ultimately, the global time parameter of interest is the output period, T_p, of the circuit defined as the maximum time between two successive data passages at the output port when the circuit is used in a pipelined fashion at the highest data rate. This time, T_p, will limit the maximum symbol interval, T, of the input data stream.[2]

The total area, A, of a VLSI chip may be bounded in two respects. Lower bounds on area, to within a constant factor, may be derived by measuring only the area actually occupied by wires (i.e., the product of the number of vertical tracks and the number of horizontal tracks containing a processor or wire of the network). The area of the smallest bounding rectangle is used to describe the upper bounds. Although the assumption that processors can be represented by points is clearly false in practice, good Thompson model layouts can still be used to develop good practical layouts.

A unit of energy is defined by the product of the units of area and time (AT). When a signal is sent from one MOS transistor (charge control device) to another, the driver must charge (or discharge) the capacitance presented by both the wire and receiver. Thus, one unit of energy is consumed by one unit of chip area every time it is involved in the transmission of a signal [80, 83].

[2] No distinction is made between T_p and T throughout this thesis, thus providing an upper bound to the data rate that can be supported by a given architecture.

A related topic, the problem of embedding a graph in a two-dimensional grid with minimum area, minimum number of edge crossings, minimum maximum edge length or fixed aspect ratio has gained a great deal of attention in the recent literature [84-99]. As a result, much has been learned about the design and analysis of efficient chip layouts for VLSI systems under certain constraints. In addition, von Neumann's early work on cellular automata [100,101] elucidates some aspects of the area-time tradeoff.

3.4 THE ROLE OF MODELS IN ALGORITHM DESIGN

The task of the digital communication engineer is twofold: (i) to find good (min P(e), MMSE, etc.) algorithms for decoding, detection or estimation and (ii) to implement these algorithms in real terms (optimally with respect to some cost function such as area or power).[3] The expression of these algorithms, in the context of VLSI, can be achieved by resorting to the VLSI grid model.

The VLSI grid model is justified on the basis that its area and time charges are sufficiently realistic to represent real computation and that it allows one to model all instances of a problem. In order to clarify this concept, let us formalize the notion of a problem.

---

[3] This twofold task is often referred to as competence (what the algorithm does) versus performance (how the algorithm is to be performed).

A problem instance consists of a list of Boolean input variables $(x_1, x_2, \ldots, x_r)$, a list of Boolean output variables $(y_1, y_2, \ldots, y_s)$ and a Boolean relation $y_i = f_i(x_1, x_2, \ldots, x_r)$, $1 \leq i \leq s$. The problem size parameter is a natural number that allows for a convenient and consistent physically related description of each problem instance. For example, the size parameter may be the number of input variables r, the dimension of an input matrix or the amount of memory contained in the relation f , etc. A problem is an infinite sequence of problem instances, one for each of an infinity of values of n, the problem size parameter. Solutions to problems are sequences of circuits, within the VLSI grid model, one for each instance of the problem. By relating the variables of the problem instance to the inputs and outputs of the circuit, by a schedule, we are able to say that an integrated circuit has an algorithm embedded in it (solves a certain problem instance).

The role of VLSI models in algorithm design is that:

1. It allows us to develop and discuss topological properties of a solution to a problem instance in isolation,

2. It allows us to discuss an infinite family of circuits, one for each n, by referring to the growth rate of some cost function (often involving the commodities of area and/or time) as n gets large,

3. It allows us to prove theorems of the form, "Every VLSI circuit that solves problem P requires area $\Omega(n^3)$", for example. Thus we can develop lower bounds to within a constant factor that allows us to judge the asymptotic quality of a solution to a problem (for a fixed λ).

Traditional complexity theory, in dealing with time and space, parallels these concepts in associating with a problem an integer, called the size of the problem, which is a measure of the quantity of input data. The time needed by an algorithm, expressed as a function of the size of a problem, is called the time complexity of the algorithm. Analogous definitions can be made for space complexity where space is the amount of memory used to store intermediate results. However, this is where traditional complexity theory diverges from the VLSI grid model in that the notion of space gives no consideration to the VLSI area required to transmit (route) these results to processing elements. Associated with this is an energy cost of communicating information throughout the system not considered in the classical case. This is the inherent strength of computational models for VLSI.

Having developed the necessary background we are now in a position to formally state the problem domain that will guide the development of the solutions presented in the next two chapters. Our concern will focus principally on the following problem:

Let P(m) be a problem of maximum likelihood sequence estimation implemented by the Viterbi algorithm with alphabet size m and let ν (the algorithm memory) be a value of the problem size parameter for which the input and output sets of vari-

ables of P(m) are the path metrics[4] and truncated survivor sequences, respectively.

Given P(m), the next chapter is devoted to generating families of area efficient VLSI layouts that are solutions to P(m). We will find that solutions to P(m) that contain more wire, and hence occupy more area, generally can support higher baud rates.

---

[4] The input set of variables can be essentially reduced to that of the sampled whitened matched filter output, by accepting a constant area or time penalty at each node. This approach only weakly influences the results developed in the later chapters.

Chapter IV

COMPLEXITY ANALYSIS: CASCADE, MESH, SE, CCC AND TREE ARCHITECTURES

4.1 INTRODUCTION

The problem, P(m), is not demanding because its algorithm is complex in a conceptual sense. Rather, the essence of the algorithm is a relatively simple procedure that must be applied to a huge number of basic "units" or "nodes". Unfortunately, the number of nodes grows exponentially with the problem size parameter ν, the algorithm memory length. This fact may be tolerable, as the technology of VLSI is capable of realizing chips with the hundreds of thousands of transistors (neglecting wiring) required to realize the VA for channel memory lengths of commercial interest. What is not especially clear or obvious at this point is the type of topologies that are appropriate, and how "expensive" is the wiring, for embedding or arranging these transistors? In other words, what solutions to P(m), realize this important digital communication algorithm? Furthermore, of all possible solutions to P(m), what solutions are optimal with respect to cost functions of area and execution time? In addition, how close do known designs approach those best possible (even though these best realizations may still be

unknown)? The answers to some of these questions are the subject of this chapter.

This chapter is organized into four major sections. The first two sections present several recirculation type communication graphs capable of implementing highly concurrent forms of the Viterbi algorithm. Initially, we consider layouts with small wire area. These designs are compact requiring little silicon area, however, they are relatively slow, thus restricting the baud rate of the input data stream. The solutions to P(m) generated in this section fall into the class of linearly and orthogonally mesh connected processor arrays.

In the second section, section 4.3, we consider layouts dominated by large interprocessor wire area. Communication graphs based on Shuffle-Exchange, Cube-Connected Cycles, and Tree-of-Meshes topologies are considered. These designs, though requiring relatively large areas of silicon, are capable of supporting maximum likelihood sequence estimation of high speed data streams.

All of the structures presented can be generalized to accommodate arbitrary source alphabet sizes and channel memory lengths. Special features, intrinsic to particular layout strategies, of interest to the digital communication engineer, are highlighted.

In section 4.4, we demonstrate that certain functions of the problem size parameter are sufficient to lower bound particular cost functions, such as area*time (AT), that any VLSI implementation (within the grid model) of problem P(m) must satisfy. In particular, the AT and AT^2 measures of complexity are presented and interpreted within the context of digital communications. The Mesh, Shuffle-Exchange and Cube-Connected Cycles solutions for P(m) presented in this chapter are good solutions in that they asymptotically approach the AT^2 lower bound to within a constant factor.

Finally, in the last section, we discuss issues related to the realization of practical devices, the neglected constant factors of the layouts described in this chapter.

4.2 LAYOUTS WITH SMALL WIRE AREA

4.2.1 Uniprocessor Layouts

The VA can be implemented on a single Add-Compare-Select (ACS) processing element driven by a programmed control unit (e.g. microprocessor) using a direct sequential algorithm. This is the degenerate case of a concurrent realization of this algorithm and we include it here, in this chapter, only for completeness. The implementation suffers from being processor poor and from being I/O bound. The uniprocessor must coordinate $O(m^{\nu})$ random accesses to the processor's I/O memory and perform $O(m^{\nu+1})$ arithmetic operations each symbol interval T. Consequently, the hardware logic speed must be

$\Omega(m^{\nu+1}/T)$. Since the processor/memory ratio is so low, (the design is almost all memory), the throughput of such a system is disappointing even though this is the smallest area solution to P(m) imaginable. Adding more processing elements has the potential of increasing throughput, as demonstrated in the following section.

4.2.2 Cascade Layouts

The VA can be parallelized on a linear array of ν processors each with geometrically varying memory sizes, in a manner similar to that used for sorting numbers [102]. In the case of binary alphabets, each processor contains two sets of ACS circuits arranged in a butterfly configuration. Associated with each processor P_j $(1 \leq j \leq \nu)$ are two auxiliary $(2^{\nu-j})$-word FIFO queues, as illustrated in Figure 4.1. The total memory of the system is proportional to the sum of this geometric series. In our example, $2(2^3 - 1) + 1$ words are required. This is a factor of two larger than necessary as the function of each FIFO pair can be shared with one FIFO of the same length. However, associated with this memory reduction is control hardware of increased complexity to coordinate memory interleaving. Each word in the FIFO is responsible for storing a state metric and an associated survivor sequence.

State metrics and survivor sequences migrate or recirculate, a limited distance, around the ring of processors,

each processing cycle. A processing cycle is defined to be the time required for a processor to take its inputs and perform an ACS operation. Unidirectional information transfers between processors are coordinated by programmed switches. The control algorithms for each switch are a function of the current discrete time index k, as defined in Figure 4.1.

The feature to note is that the topology is small, regular and compact with essentially little interprocessor wire area. The regular structure provides for extensibility such that the architecture can accommodate arbitrary problem size instances, in a controlled way. This is a prerequisite for developing an appropriate silicon compiler for automated design. The structure also allows one to partition the circuit into processor/FIFO pairs for incorporation onto separate chips (or printed circuit boards) as available technology dictates.

Each symbol interval, the circuit accepts a digitized whitened matched filter output into a dual ported memory or FIFO queue of ν words. Switch S5 routes operands from this queue to the appropriate path metric generator, as determined by Procedure S5, in Figure 4.1. Each path metric generator serves a unique processing element and is responsible for providing a set of four path metrics each valid processing cycle, the clock cycle during which the ACS processor is active, of which there are $m^{\nu-1}$ in ν symbol

```
Procedure S1
 if k mod4 = k mod8
   then: A -> B
   else: A -> C
```

```
Procedure S2
 if k mod4 = k mod8
   then:
     if k mod2 = k mod4
       then: D -> F
       else: D -> G
   else:
     if k mod2 = k mod4
       then: E -> F
       else: E -> G
```

```
Procedure S3
 if k mod2 = k mod4
   then:
     if k mod2 = 0
       then: H -> J
       else: H -> K
   else:
     if k mod2 = 0
       then: I -> J
       else: I -> K
```

```
Procedure S4
  if k mod2 = 0
    then: L -> N
    else: M -> N
```


Figure 4.1: CASCADE LAYOUT OF THE VA Architecture for binary alphabet and memory three. P_1, P_2, and P_3 are ACS butterfly processors. Also illustrated are the control algorithms for switches S1-S5. (See Appendix F for more detail)

```
Procedure S5
 if (k mod8 = 4) then: P -> U
 if (k mod8 = 6) then: P -> V
 if (k mod8 = 7) then: P -> W
     else:  No Connection  (Path Metric Generators hold previous values)
```

intervals. If the first state metric generated in a group of 2^{ν} processing cycles is subtracted from all others, state metrics can be conveniently normalized to control register overflow. Fixed delay maximum-likelihood estimates of the transmitted data sequence are available from the truncated survivor sequence of any state at each stage of the trellis. A convenient method to tap into these survivors is to extract the last ν items in the survivor list once every m^{ν} processing cycles which are available from processing element P_{ν}. The implicit assumption, of course, is that the oldest items in the 5ν-element survivor list have merged indicating that all states agree on a common ancestry. An output queue of length ν allows the output to present one output estimate each symbol interval T.

Each processor/FIFO set is responsible for processing states associated with one stage of the trellis. Pipelining is possible in this recirculation network because newly generated state information produced by processor P_j can be passed to the immediate neighbor $P_{j(\mathrm{mod}\ \nu)+1}$ so that states associated with the next stage of the trellis can be processed even before all states associated with P_j have been evaluated. Pipelining allows ν symbols to be processed each m^{ν} processing cycles. Consequently, hardware logic speed must be $\Omega(m^{\nu}/\nu T)$, a respectable improvement over the one processor implementation. Even with complete pipelining each processor P_j is idle for one-half of the processing cycles.

Although the layout has been directed toward realizations for binary alphabets, versions of this processor for arbitrary alphabet sizes should be apparent [103].

The cascade (like the uniprocessor) solution to P(m) is mainly dominated by storage. The next section presents a layout strategy that contains as many ACS processors as storage elements, to within a constant factor.

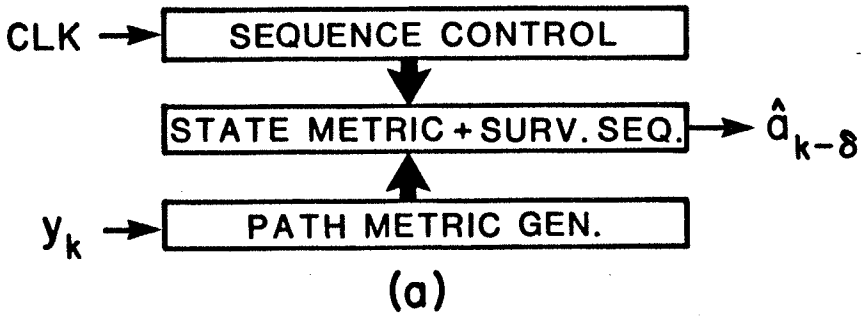
4.2.3 Linearly Connected Layouts

A linearly connected network of processors is illustrated in Figure 4.2. It consists of three rows of processing cells where each element of a row is fitted with word-parallel interconnections to its near neighbor. Each row of this architecture is homogeneous in function. This allows us to define a column of three processing cells as a standard building block. Linearly connecting $m^{\nu+1}$ of these building blocks together comprises a a solution to P(m) with problem size parameter ν . Since this architecture can be viewed as a systolic array, the results of reference [104] are particularly relevant to a wafer-scale implementation. Though this architecture has an unpleasant aspect ratio for channel memory lengths of interest, say $\nu > 4$, a theorem due to Leiserson [105, p.94] can be used to establish the comforting fact that a topologically equivalent layout can be enclosed in a square whose area is at most three times the area of the original rectangular layout.

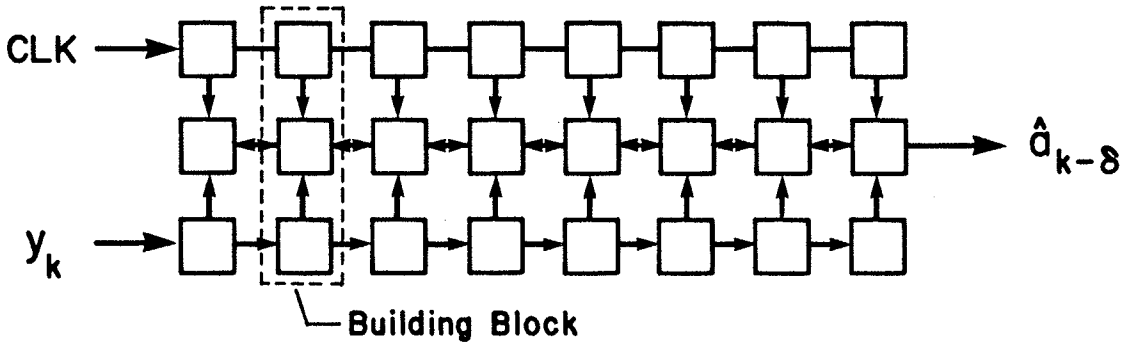
The backbone of this architecture is formed by the middle row of processing cells, while the top and bottom rows can be viewed as support hardware. The top row takes a clock signal as input and produces as output a set of sequence control signals, one for each processor in the middle row. The bottom row accepts the input signal y_k and generates a unique path metric, one for each processing element in the middle row. In some implementations this row may be comprised exclusively of ROM hardware. Each processing cell in the middle row contains a state metric register, survivor sequence register, ACS circuit and control circuitry. The control circuitry exchanges state metrics and survivor sequences with a near neighbor as dictated by the sequence control signals generated by the top row of processors.

Path metrics supplied by the bottom row are used to update the state metric registers each symbol interval. As output, the middle row provides fixed delay estimates of the transmitted data sequence. These can be extracted from the truncated survivor sequence resident in the rightmost processing element.

The sequence of events during one symbol interval in the center row of processing cells is illustrated in Figure 4.3. The initial configuration consists of m^{ν} sets of m identical state metrics (and survivor sequences). In a series of near neighbor transpositions, state metrics are moved (in only $m^{\nu}-1$ time steps) to appropriate positions in the one-



(a)



(b)

Figure 4.2: LINEARLY CONNECTED PROCESSOR LAYOUT OF THE VA
(a) Functional Block Diagram
(b) Architecture for P(2) with $\nu = 2$
(binary alphabet and memory two)

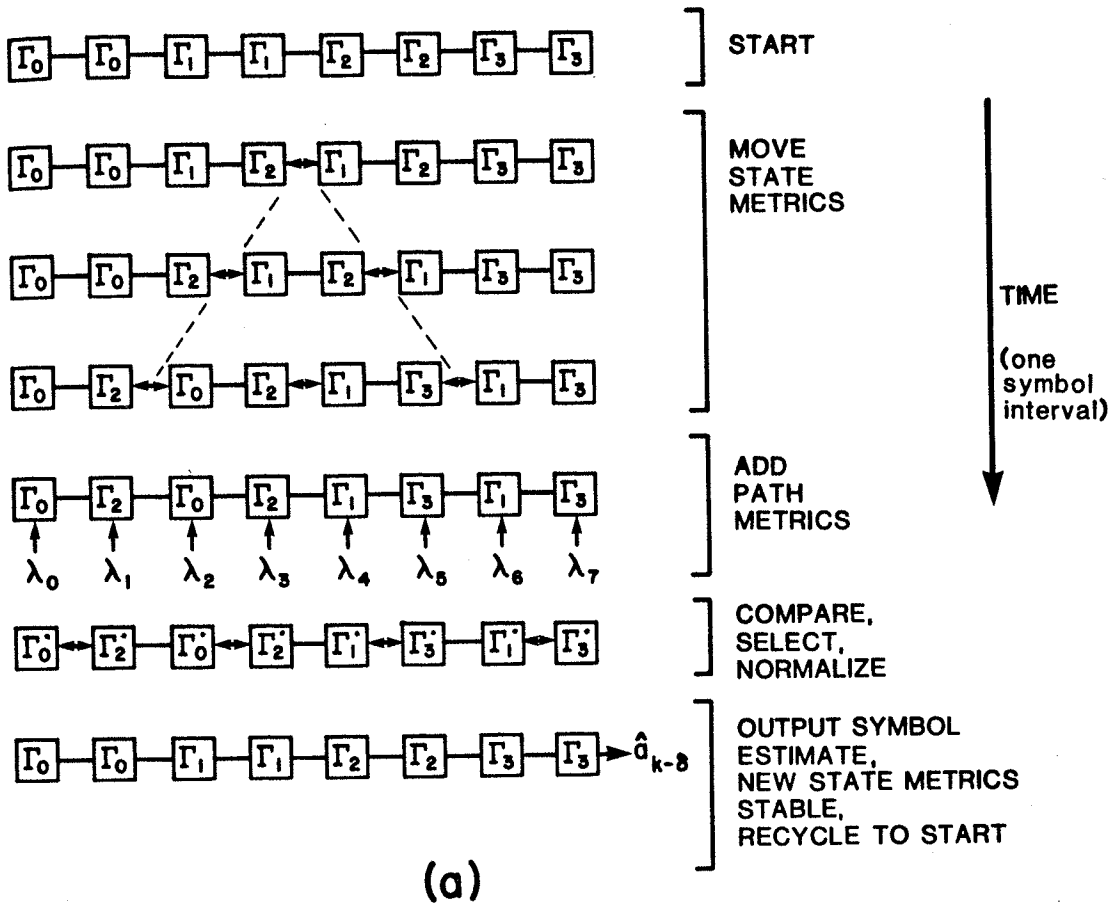
dimensional array in anticipation of the path metrics generated by y_k. This sequence of steps can be viewed as unpacking in a stable manner (i.e. without altering their original order) the items initially in the left half into the even positions in the array, and those in the right half into the odd positions of the array. The triangular structure of the series of transpositions is characteristic of the control algorithm required for any size problem instance of P(2).

After the path metrics have been added to the state metrics, the bottom of the "triangle" consists of having each even numbered processor compare its state metric with that of its odd numbered adjacent neighbor. The smallest state metric of this pair is chosen, normalized to prevent overflow and then duplicated in its odd-even processor pair. At the same time the survivor sequence registers are updated and an estimate of a transmitted symbol in the past history is ejected from the last processing cell in the array. The events described in the last few paragraphs are then repeated during the next symbol interval.

Overflow control is achieved by selecting one state metric and subtracting it from all others. For an efficient implementation one of the processors in the center of the array should distribute its state metric into a special register during the transposition operations. At the bottom of the "triangle" each processor will have a copy of this

one state metric to be subtracted from all the newly generated state metrics.

Hardware logic speed must be $\Omega(m^{\nu}/T)$ in this algorithm structured VLSI network. The simpler control algorithm this structure enjoys is paid for by the increased processor area and reduced throughput relative to the cascade design. Problem P(m) can be embedded not only in a one-dimensional array of processors but also in two-dimensional arrays as demonstrated in the next section.



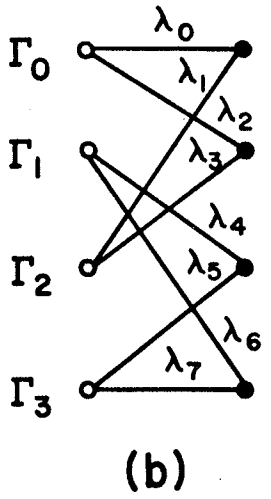


Figure 4.3: LINEARLY CONNECTED EVENT SEQUENCE
(a) A schematic presentation of the sequence of events in a linearly connected processor layout for P(2), ν=2; $\longleftrightarrow$ denotes transposition.
(b) The corresponding trellis diagram.

4.2.4 Mesh Layouts

In this section, we demonstrate that solutions to P(m) can take the form of a compact mesh-type recirculation network. The two-dimensional mesh layout has a higher throughput than the linearly connected layouts studied in the previous section because operands do not always have to migrate as far through the network of processors. However, the connectivity is inferior and hence time charges are greater than the high wire area layouts to be studied in the next section in that operands in a mesh must circulate beyond immediate cell neighbors before they reach the correct cell.

The rectangular mesh interconnection pattern consists of $N=m^{\nu}$ identical processors (one for each state), arranged in a two dimensional array of size $m^{\lfloor \frac{\nu}{2} \rfloor}$ x $m^{\lceil \frac{\nu}{2} \rceil}$. Each processor is connected to adjacent neighbors, as shown in Figure 4.4(a). Processors at the perimeter have two or three rather than four neighbors; there are no end-around connections. The feature to note is that this structure is a small and compact design that requires essentially little interprocessor wire area.

Each cell is a message driven processing element with the ability to generate, forward and receive messages. Each cell contains an add-compare-select circuit, a path metric generator (or table lookup), transceiver and multiplexers

and a microcoded control processor. Conceivably, Figure 4.4(a) could implement a Viterbi receiver with a memory of four (2^4=16 states). Two bidirectional communication paths would be provided to each neighbor for path survivors and for state metrics. The way in which the processors are indexed determines the routing algorithm used to move data between processors. The objective is to use index schemes which minimize the time spent in routing. A row-major index scheme, as illustrated in Figure 4.4(a), yields a simple routing algorithm for each cell. The routing algorithm is ultimately determined by the trellis diagram which has been rearranged in Figure 4.4(b) to exploit a divide and conquer solution paradigm [106].

The routing steps and the corresponding migration of state metrics is illustrated in Figure 4.4(c-f) for a 16-state binary VA receiver implemented on a 4 x 4 square mesh. One stage of an m^{ν} state trellis (the trellis diagram for one symbol interval) is implemented by a collection of routing steps. The sequence of routing steps, defined by the discrete time index k, repeats every ν symbol intervals since the state metrics are back in their original starting location. Communication of information from one cell to its nearest neighbor is called a unit distance route [107] and takes time t_R. Note that some routing steps require multiple unit distance routes, to move information from where it was produced to where it is needed next. This is

accomplished by having cells swap information with near neighbors. In addition to routing time, processing cells have an associated processing delay, t_{ACS}. The throughput of the VA, executed on a square mesh is limited by the worst case number of unit distance routes required in a symbol interval and the delay of the ACS processing cells. It can take as much as $O(m^{\lceil \nu/2 \rceil})$ time to rearrange the data among the processors in preparation for the next ACS step. Fortunately, only a few of the ACS+route operations take this amount of time. The average time for a ACS+route for 2^{ν} states is only $O(2^{\lceil \nu/2 \rceil}/\nu)$. Consequently, the maximum symbol interval (T) this structure could support is lower bounded by:

$$T = \Omega\left(t_{ACS} + \frac{2^{\lceil \nu/2 \rceil}}{\nu} t_R\right) \qquad (4.1)$$

Thus, the required logic speed (operations/second) of the structure must be $\Omega(2^{\lceil \nu/2 \rceil}/(\nu T))$.

Three aspects of the mesh implementation are important.

1. First, in order to spread the routing step time penalty equally among each symbol interval a FIFO queue of depth ν should be used on the detector input data stream. The output of the FIFO queue is globally broadcast to all processing cells. This is the only global wiring required in the design (besides power and timing signals).

2. Second, overflow control of the state metric registers is not straightforward, if we are determined to use only the local near neighbor communication paths. One possible strategy involves normalizing state metrics once every ν processing steps (i.e., once every ν steps into the trellis). This provides for the luxury of selecting one state metric, say state zero, and separating this value from the main computational data path. During the first $\lceil \nu/2 \rceil$ processing cycles, this value can be broadcast to all processing cells, in the same row, using only nearest neighbor broadcasts (on a dedicated connection). At this point in time each column has at least one processing cell with this value stored in a register. During the next $\lfloor \nu/2 \rfloor$ processing cycles, this value can be broadcast to all processing cells in each column. Now, all state metrics can be normalized using this value. This normalization routine is then repeated in the next ν processing cycles. The penalty paid in this type of scheme is that state metric registers would have to be several bits wider than if state metrics were normalized each processing cycle.
3. Third, the detector output is available at each symbol interval from the truncated survivor sequence of any processing cell.

Figures 4.5 and 4.6 illustrate layouts where the number of states are not a perfect square and where the alphabet size is not necessarily binary.

The number of states to be processed could be larger than N, the number of processors. An efficient means of handling this situation is very similar to that devised for odd-even merge sort described in reference [108, p.155]. Naturally, there is a corresponding performance degradation associated with achieving varying degrees of parallelism.

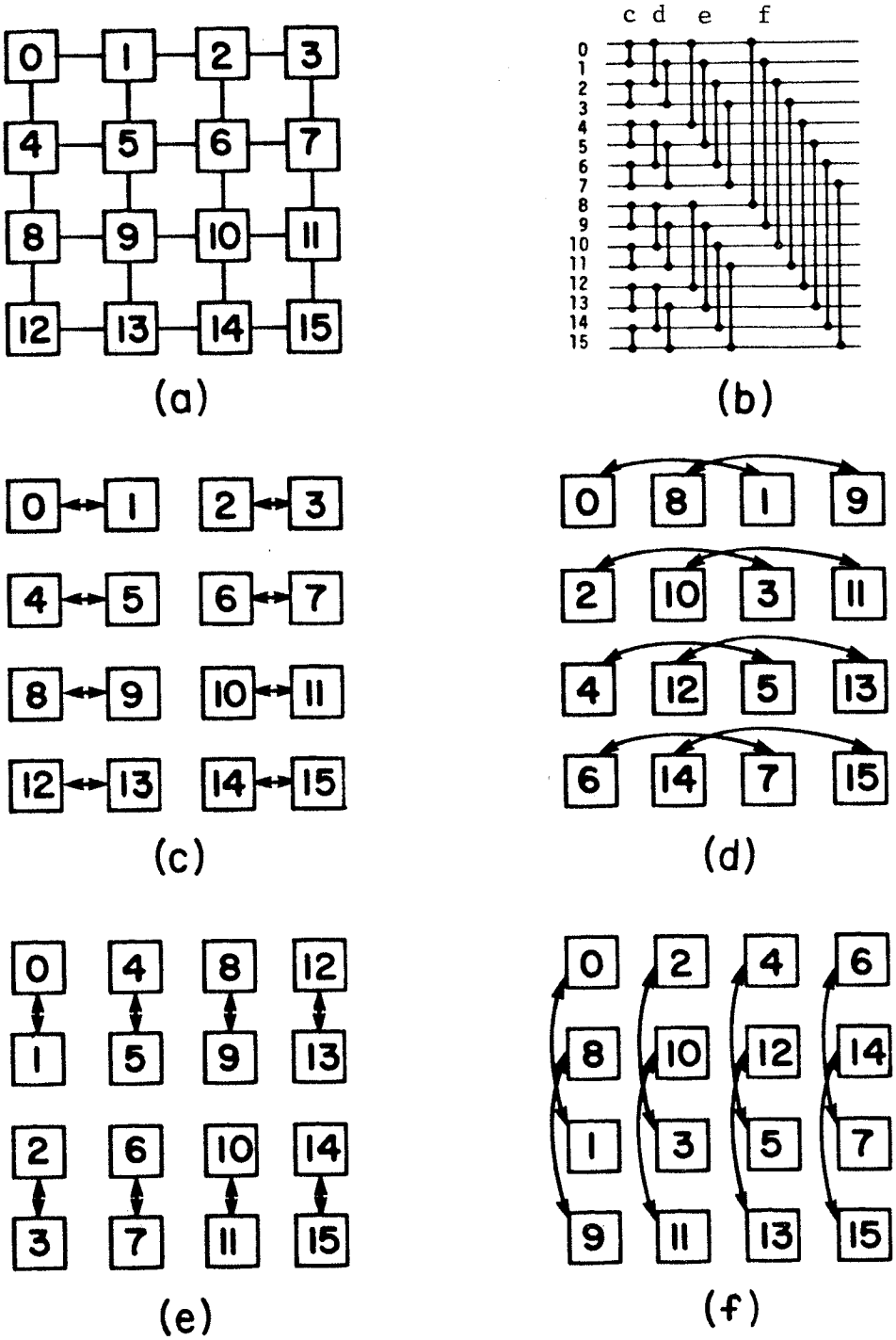


Figure 4.4: THE VA IN A 4X4 SQUARE MESH
(a) The 4x4 Mesh of Processors labelled in row major order
(b) The rearranged trellis diagram presented in a notation similar to that in Knuth [109, p.222].
(c)-(f)
Routing steps and Migration of State Metrics of a 16-state binary VA receiver implemented on a 4x4 Square Mesh.
The numerals correspond to state metric labels.
The execution sequence is: cdefcdefcd....

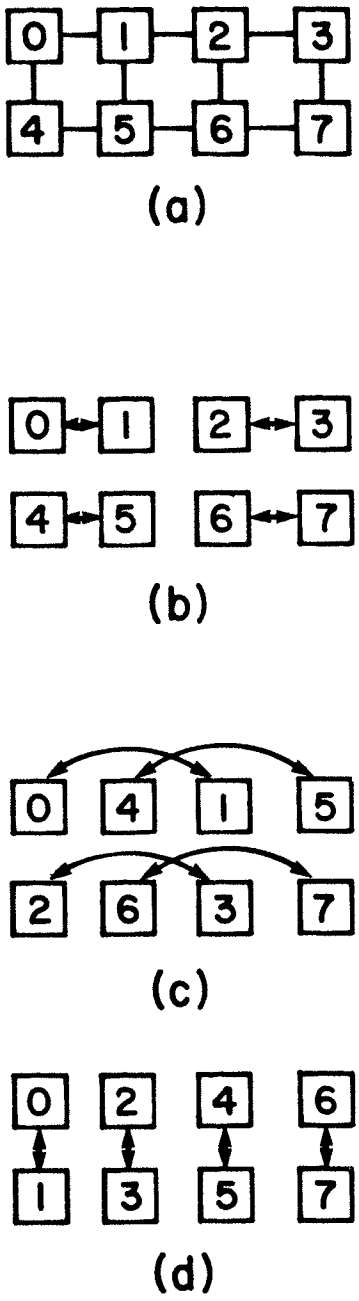


Figure 4.5: THE VA IN A 2X4 RECTANGULAR MESH
Routing steps and Migration of State Metrics of a 8-state binary VA receiver implemented on a 2x4 Rectangular Mesh. The numerals correspond to state metric labels. The execution sequence is: bcdbcdbc....

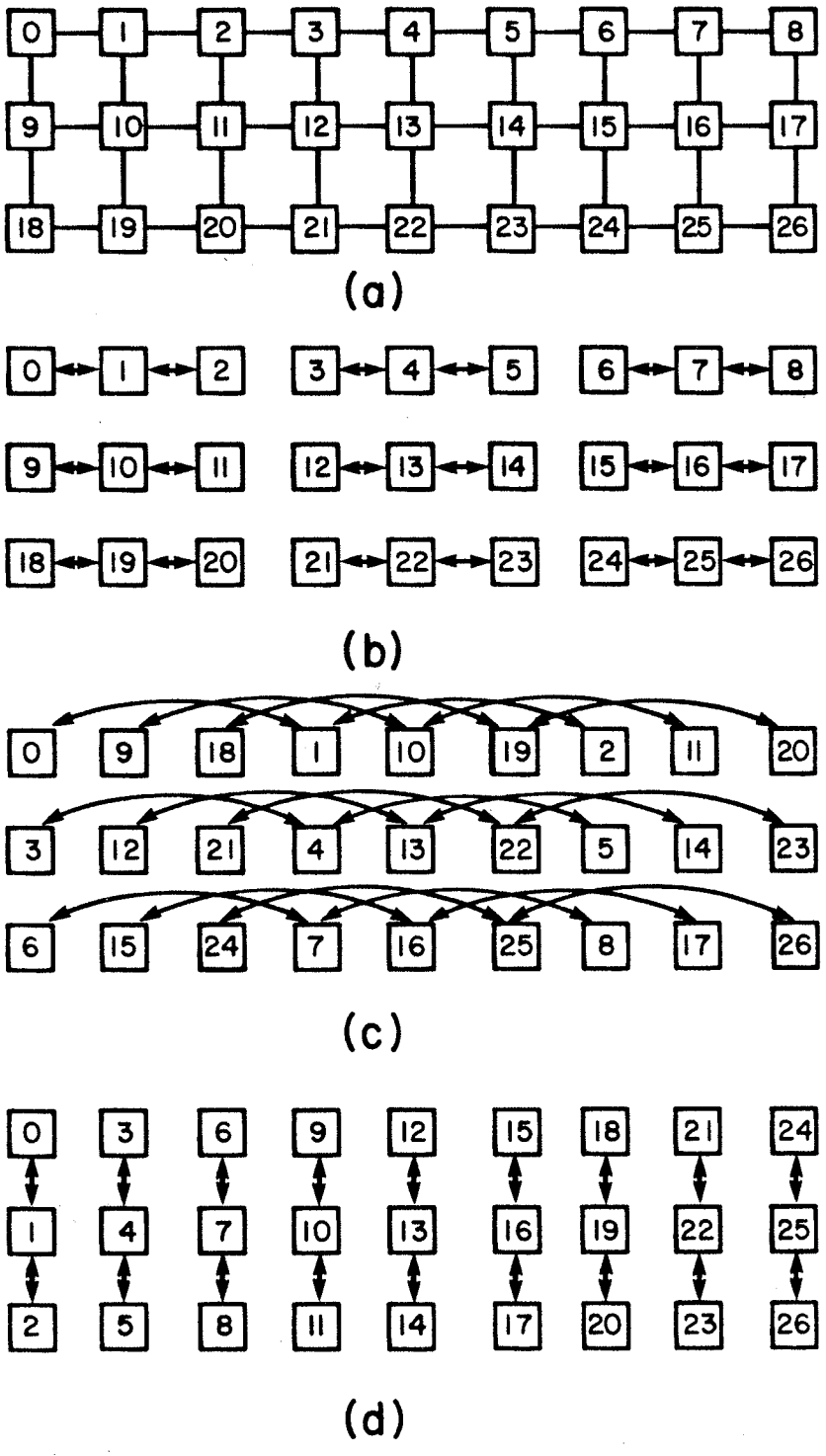


Figure 4.6: THE VA IN A 3X9 RECTANGULAR MESH
Routing steps and Migration of State Metrics of a 27-state ternary VA receiver implemented on a 3x9 Rectangular Mesh. The numerals correspond to state metric labels. The execution sequence is: bcdbcdbc....

4.3 LAYOUTS WITH LARGE WIRE AREA

4.3.1 Shuffle-Exchange Layouts

In this section we propose a shuffle-type recirculation network to handle the data flow of the Viterbi algorithm rather than decomposition into one step subtrellis components (i.e., butterfly) proposed by other authors [7,72,76]. However, the shuffle-exchange (SE) network as defined in the published literature can only be used to implement the VA for binary alphabets (denoted here as 2-SE graphs). Well known VLSI layouts for 2-SE graphs are extended and generalized in this section such that these new layouts realize the VA for m-ary alphabets. The shuffle exchange pattern is an ideal recirculation network in that one pass through the network is sufficient to move data to appropriate nodes at the next time instant. Solutions to P(m) based on the SE topology are faster than those based on the mesh since the performance of the mesh is limited by the routing time. However, to counterbalance their improved time performance SE circuits are much larger than mesh based ones due to the increased area required for interprocessor wiring. Nevertheless, by exploiting the parallelism of the algorithm in shuffle-type layout slices, arbitrarily large channel memory lengths (extensibility) can be accommodated while maintaining regular communication and control paths.

Shuffle exchange graphs were [110] originally proposed in 1971 as an interconnection methodology for parallel computa-

tion. It has been shown that algorithms to compute the fast Fourier transform (FFT), matrix multiplication and sorting among others [111-118] can be efficiently implemented using this scheme.

With the advent of VLSI, the question of how to best layout the shuffle exchange graph on a grid using as little area as possible is of practical as well as theoretical importance. Thompson [80] in his Ph.D. dissertation was the first to address this issue and showed that any layout of the n-node ($n=2^{\nu}$) shuffle-exchange graph requires $\Omega(n^2/\log^2 n)$ area. Several researchers in the following year attempted to find layouts which achieved Thompson's lower bound [96,97]. The area layout question was finally settled by Kleitman et. al. [91]. The $O(n^2/\log^2 n)$-area layout for the shuffle exchange graph in [91] is asymptotically optimal, however, it is not optimal for small values of n (i.e., $n<10$). Leighton and Miller [119] describe techniques for finding good layouts for small shuffle exchange graphs. Their technique is generalized in this paper to allow an area efficient embedding in silicon of the Viterbi algorithm with arbitrary alphabet size and memory length.

The 2-SE graph consists of $n=2^{\nu}$ nodes and $3n/2$ edges. Each node is associated with a unique ν-bit binary string $s_{\nu-1}, s_{\nu-2}, \ldots, s_0$. Two nodes K and K' are linked via a shuffle edge if K' is a left or right cyclic shift of K (i.e., if $K = s_{\nu-1}, \ldots, s_0$ then $K' = s_{\nu-2}, \ldots, s_0, s_{\nu-1}$

or $K' = s_0, s_{\nu-1}, \ldots, s_1$). Two nodes K and K' are linked via an exchange edge if K and K' differ only in the first bit. (i.e., if $K = s_{\nu-1}, \ldots, s_1, s_0$ then $K' = s_{\nu-1}, \ldots, s_1, \bar{s}_0$). For example, we have drawn the 8-node shuffle exchange graph on the right side of Figure 4.7(a) with the shuffle edges drawn as dashed lines and the exchange edges drawn as solid lines.

The above definition for the shuffle-exchange graph as used by Thompson and others [80] will be denoted by G_t. The exchange edge can be defined in another way. Two nodes K and K' are linked via an exchange edge if $K = s_{\nu-1}, \ldots, s_1, s_0$ and $K' = s_{\nu-2}, \ldots, s_1, s_0, \bar{s}_{\nu-1}$. This graph, denoted by G_s, corresponds to the Viterbi algorithm trellis structure for binary alphabets and is similar to Stone's perfect shuffle depiction. The two graphs G_t and G_s are illustrated in Figure 4.7 for $\nu = 3$. Appendix B shows that there is a sequence of elementary contractions that will map G_t into G_s. Consequently, G_t can be embedded with algorithms that reside in G_s. By dealing with G_t exclusively it is not necessary to distinguish between these two structures.

The collection of all cyclic shifts of a node K is called a necklace and is denoted by <K>. For example, the necklace generated by 0011 is <0011> = { 0011, 0110, 1100, 1001 }. Each necklace corresponds to a cycle in the shuffle-exchange graph and shuffle edges always link nodes which are in the

same necklace. If the necklace contains precisely ν nodes, then it is said to be full. Otherwise, a necklace contains less than ν nodes and is said to be degenerate. For example, <0011> is full while <0000> is degenerate. The partition of the shuffle edges into necklaces is a key part of the layout technique used to embed the Viterbi algorithm into silicon.

In the layouts presented in Figures 4.8-4.13 each necklace appears as a dashed rectangle consisting of arbitrarily long segments of two vertical tracks and unit length segments of two horizontal tracks and each exchange edge appears as a solid horizontal line segment. Figures 4.8-4.11 are sample solutions to P(2) while Figures 4.12 and 4.13 are sample solutions to P(3). The nodes or ACS processors in the VLSI grid model layouts are numbered such that each corresponds to the state metrics they evaluate when numbered in a "natural" ordering. It is not known how best to order the necklaces in general to minimize the maximum overlap of the horizontal exchange edges. One simple heuristic proposed for the binary case [119] is to order the necklaces from left to right so that the minimum value of the nodes in each necklace form an increasing sequence.

As can be observed in Figure 4.11, the structure of the layout facilitates efficient chip manufacture and data management, with only seven ACS processors per layout slice each connected by nonoverlapping horizontal wiring.

Several other points are worthy of note. The adjacency matrix of the SE graph has a regular structure. This allowed an investigation as to the planarity of the graph generated by considering processing nodes to be one-step trellis components (i.e. butterfly). The Hopcroft and Tarjan planarity algorithm [120,121] established that for $4<\nu<10$ the digraphs are nonplanar (Appendix E). This complements the work of Thompson [80] who showed that the shuffle exchange graph cannot be embedded in silicon using area linearly proportional to the number of nodes. (This has recently been established [122] to be a necessary but not sufficient condition for nonplanarity.) This test was motivated by the existance of a general purpose divide and conquer layout algorithm that produces low area layouts for a wide variety of families of graphs including planar graphs of degree four or less [98].

To generalize the 2-SE structure presented earlier to an m-shuffle exchange graph, which corresponds to solutions for P(m), we proceed as follows. Associate each state (node) with a unique ν-digit m-ary string. Shuffle connections are described by cyclic shifts as before. Exchange connections are defined by nodes with the same first $\nu-1$ digits and the last digit being any valid number within the number radix m (i.e., 0,1,2,...,m-1). The resulting graph has a maximum vertex degree of four. The collection of all cyclic shifts of a node K is called a necklace, as before, and is denoted

by <K>. For example, the necklace generated by 012 is <012> = {012, 120, 201}. Each necklace corresponds to a cycle in the m-shuffle-exchange graph and shuffle edges always link nodes which are in the same necklace. If the necklace contains precisely ν nodes, then it is said to be full. Otherwise, a necklace contains less than ν nodes and is said to be degenerate. If a necklace contains only one node it is said to be a self-loop. As an example, for ν=3, <012> is full while all degenerate necklaces, such as <222>, are self-loops. It is possible to have a degenerate necklace that is not a self-loop such as <012012>, for the case ν=6. The partition of the shuffle edges into necklaces is the layout technique used to embed the VA into silicon.

As in the binary case, it is not known how best to order the necklaces in general to minimize the maximum overlap of the horizontal exchange edges. However, the same simple heuristic developed for the binary case [119] appears to be equally useful in the m-ary case where the necklaces are ordered from left to right so that the minimum value of the nodes in each necklace form an increasing sequence.

In Figure 4.12, a shuffle-exchange layout for the Viterbi algorithm with ternary alphabet and memory two, is shown. Detailed data flow within the the shuffle exchange graph for this design is presented in Figure 4.13(a). Note that each collection or group of exchange edges is local to three nodes. As usual, each node is an ACS circuit. The layout

can be extended to arbitrary memory size, as shown in Figure 4.13(b).

In general, for an m-ary alphabet and memory ν there are:

(i) m^{ν} nodes.

(ii) m self-loops.

(iii) m nodes in each "exchange" edge.
In this context "exchange" edge refers to the collection of edges defined by the exchange operation with at least one common node.

(iv) $m^{\nu-1}$ "exchange" edges, m of which contain a node that carries a self-loop.

(v) $2m^{\nu} - m^{\nu-1}$ edges. This is established as follows: $m^{\nu}-m$ nodes are contained in the necklaces, which because of their cyclic nature also contain $m^{\nu}-m$ edges. The $m^{\nu-1}$ "exchange" edges each contain m-1 edges, as illustrated in Figure 4.13(b). Finally, there are the m self loops that are not usually illustrated, for a total of $2m^{\nu}-m^{\nu-1}$ edges. As a check, consider when $m^{\nu}=2^{\nu}=n$. Then we have $2m^{\nu}-m^{\nu-1}=2n-n/2=3n/2$, as was known previously.

(vi) ν nodes (ACS units) in each full necklace.

(vii) No more than $\lfloor (m^{\nu} - m)/\nu \rfloor$ full necklaces.

Property (vii), above, leads to the following theorem.

DEFINITION: In the domain of natural numbers, let S be a set whose elements are either primes or pseudo primes to the base m.

THEOREM: In an m-shuffle exchange graph, there can be no more than $(m^{\nu}-m)/\nu$ full necklaces, which occurs iff ν is an element of S.

PROOF: An m-shuffle exchange graph with m^{ν} nodes has m self-loops. Consequently, only $m^{\nu} - m$ nodes are contained in the necklaces. If all necklaces are full then ν must be a factor of $m^{\nu} - m$ since a full necklace, by definition, has ν nodes. By Fermat's Theorem [26], ν is a factor of $m^{\nu} - m$ when ν is prime.

All the other numbers that are not prime that satisfy the equality $m^{\nu} - m = 0 \pmod{\nu}$ are, by definition, pseudo primes to the base m. For example, 15 is a pseudo prime to the base 4, because $4^{15} - 4 = 0 \pmod{15}$ and 15 is composite (5 * 3). Since S is the union of the two subsets defined by the primes and pseudo primes to the base m, a sufficient condition for all necklaces to be full is that ν be an element of S.

Necessity is proved from the definition: if ν is not prime then it must be composite, and a composite number that satisfies $m^{\nu} - m = 0 \pmod{\nu}$ must be a pseudo prime to the base m. ■

As an aside, it is interesting to note that the set of Carmichael numbers [123] are, by definition, pseudo primes to every base m. In addition, the set of prime numbers are prime irregardless of the base m. Therefore, if ν is either a prime or a Carmichael number then ν is a factor of $m^{\nu} - m$ independent of the value of m.

Having established an upper bound to the number of full necklaces we can expect in a m-SE graph, we will next show the conditions under which all necklaces are full.

THEOREM: In a m-shuffle exchange graph all necklaces, other than the m self-loops, are full independent of the value of m iff ν is a prime number.

PROOF:

Each node can be identified by a ν-digit string or label $x_{\nu-1}, x_{\nu-2}, \ldots, x_j, \ldots, x_1, x_0$ where each x_j is drawn from the set $\{0,1,2,\ldots,m-1\}$.

The theorem amounts to proving that for any given node, other than that involved in a self-loop, no cyclic shifts less than ν will regenerate the node label iff ν is prime.

Assume there exists a cyclic shift p for which the string is regenerated. In this case the following sets of equations must be satisfied.

$$\begin{aligned} x_{\nu-1} &= x_{\nu-p-1} \pmod{\nu} \\ x_{\nu-2} &= x_{\nu-p-2} \pmod{\nu} \\ &\vdots \\ x_1 &= x_{\nu-p+1} \pmod{\nu} \\ x_0 &= x_{\nu-p} \pmod{\nu} \end{aligned} \qquad (4.2)$$

Let us reorganize the system of equations such that after choosing any one of the equations in 4.2 the others are selected by subscript and arranged in order as follows:

$$\begin{aligned} x_{\ell_1} &= x_{r_1} \\ x_{\ell_2=r_1} &= x_{r_2} \\ &\vdots \\ x_{\ell_k=r_{k-1}} &= x_{r_k=\ell_1} \end{aligned} \tag{4.3}$$

Note that there are k equations generated by such a procedure, where $0<k<\nu$. To prove that there is always a sequence of substitution steps that terminates in the last equation selected with $r_k=\ell_1$ let us do this by contradiction. Assume there exists another $r_k=r_j\neq\ell_1$ which is the subscript on the right side of the last equation. This however, cannot be the last r_k as r_j must be on the left since each of the indices is represented once and only once on both the left and right sides of the equality in 4.2 and has not been eliminated previously by substitution.

At this point we have established that the $r_k=\ell_1$ and that $0<k<\nu$. Now we will try to evaluate k.

Using the first equation selected, the subscript on the right is related to that on the left, for a shift of p, by the following relation:

$$\ell_1 - p \pmod{\nu} = r_1 \tag{4.4}$$

while that of the second equation selected is related by:

$$\ell_2 - p \pmod{\nu} = r_2 \tag{4.5}$$

or equivalently from 4.3:

$$r_1 - p \pmod{\nu} = r_2 \tag{4.6}$$

which upon substitution of 4.4 into 4.6 yields:

$$\ell_1 - 2p \pmod{\nu} = r_2 \tag{4.7}$$

In general, for the k^{th} substitution,

$$\ell_1 - kp \pmod{\nu} = r_k \tag{4.8}$$

However this is precisely the step where $r_k = \ell_1$ (the last step). In other words, the following equation holds:

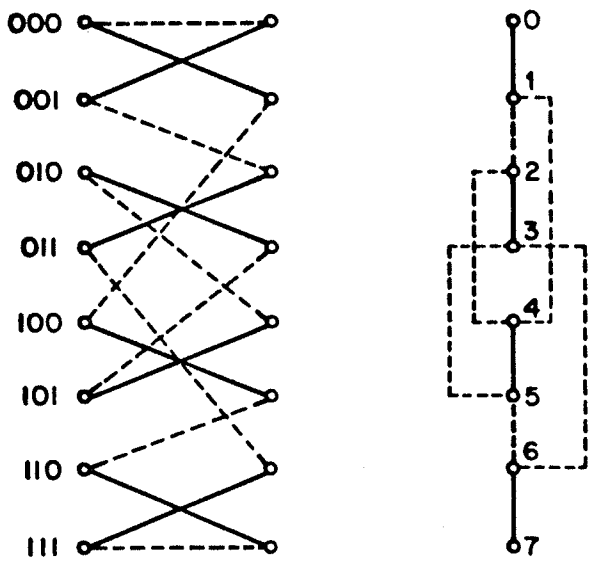
$$\begin{aligned} &\ell_1 - kp \pmod{\nu} = \ell_1 \\ &\Leftrightarrow kp \pmod{\nu} = 0 \text{ , where } 0 < p < \nu \end{aligned} \tag{4.9}$$

The only way that $kp \pmod{\nu} = 0$, for ν prime, is either when $k \pmod{\nu}=0$ or when $p \pmod{\nu}=0$. Since $p<\nu$ we know that ν is not a factor of p and hence $p \pmod{\nu}=0$ can be discarded. However, there is a solution to $k \pmod{\nu}=0$ which is $k=\nu$. But this means that all the x_j's are equal to x_{ℓ_1} which implies that the only degenerate necklaces for ν prime are the self-loops. Therefore, ν prime is a sufficient condition for <u>all</u> necklaces, other than the self-loops, to be full.

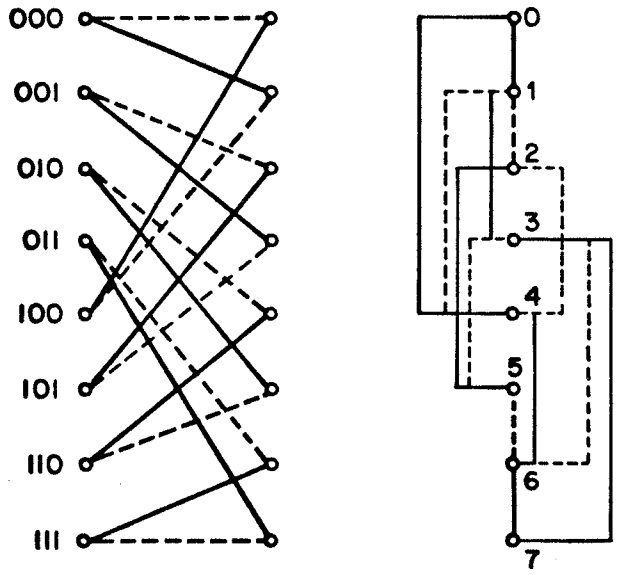
Necessity is proved by the following argument. If ν is not prime it must be composite. Suppose, in this case, ν has a factor q, where $1<q<\nu$. Then take the node label or string $x_{\nu-1}, x_{\nu-2}, \ldots, x_0$ and segment it into contiguous portions of q digits. Choose values for each digit in a substring of q digits from the set {0,1,2,...,m-1} such that all the digits are not equal in value. (This is always possible since q>1 and m>1.) Place identical copies of this substring in each portion of the segmented node label. After q shifts the node label is the same. Since q is less than ν, the necklace that contains this node is degenerate. Hence, if ν is <u>not</u> prime there is at least one necklace, other than the self-loops, that is <u>not</u> full. ■

This theorem is significant in that for a particular implementation all the layout slices are identical for prime channel memory lengths. Fortunately, near term technological interest will probably be concerned with $1<\nu<20$, precisely where eight prime numbers reside.

A detailed block diagram of the shuffle layout slice is shown in Figure 4.14, for P(2) and $\nu = 2$. The quantized input data signal y_k is fed to the branch metric lookup table (ROM) or combinational circuitry. The corresponding λ's are delivered to the appropriate adders. Two ACS units are detailed in this layout slice along with registers for the path survivors $\hat{x}(\alpha)$.



(a)



(b)

Figure 4.7: A COMPARISON OF THE BIPARTITE GRAPHS G_t AND G_s
(a) The Shuffle-Exchange graph G_t used by Thompson [80]
(b) Viterbi trellis structure (G_s) for binary alphabets similar to the Perfect Shuffle structure by Stone [110]
The bipartite graphs on the left side are folded about a vertical axis to generate the graphs on the right side.
---- shuffle operation; —— exchange operation.

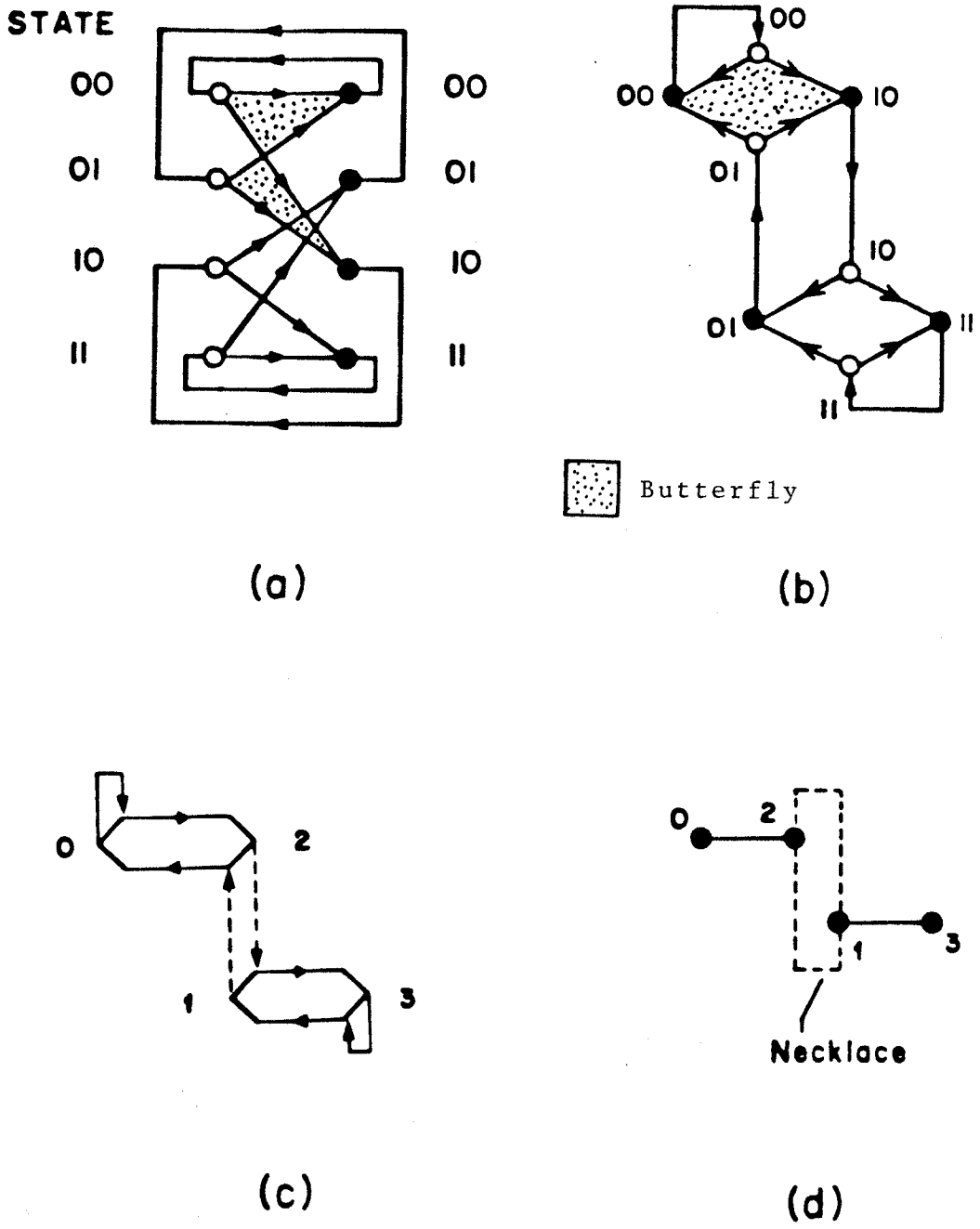


Figure 4.8: VA GRAPHS FOR BINARY ALPHABET AND MEMORY 2
(a) Single stage trellis
(b) The trellis diagram in (a) unfolded
(c) The ACS units in (b) separated
(d) Equivalent grid model layout for the 4 node 2-SE graph
---- corresponds to the shuffle operation
—— corresponds to the exchange operation

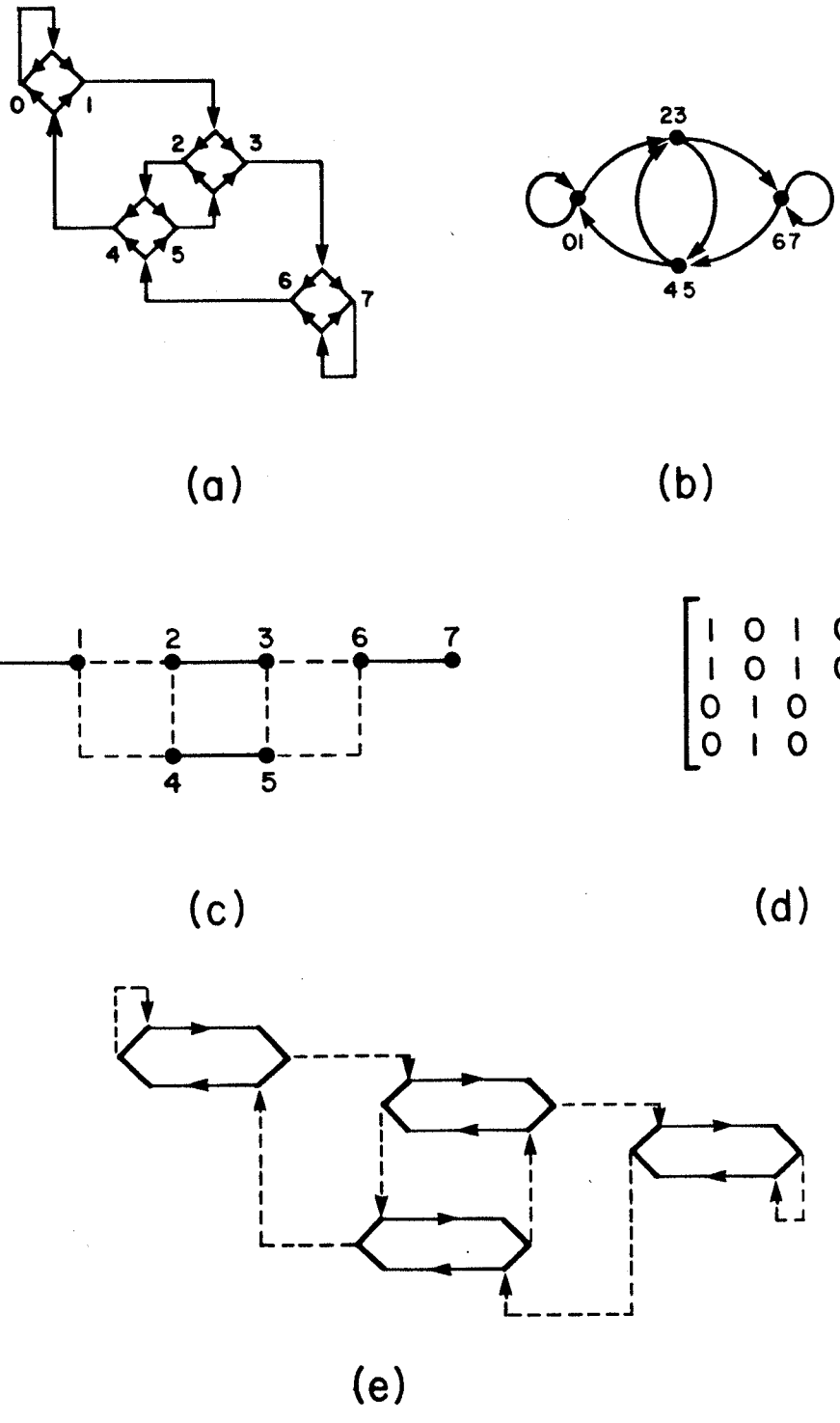
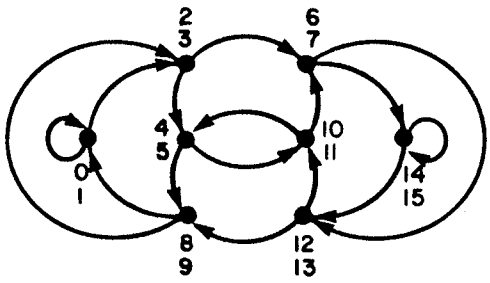


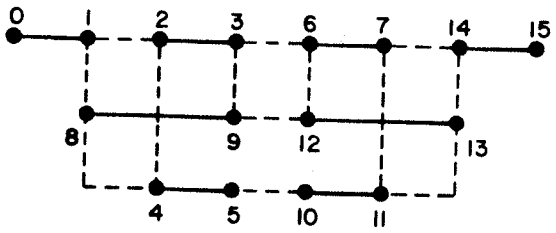
Figure 4.9: VA GRAPHS FOR BINARY ALPHABET AND MEMORY 3
(a) Butterfly network (Signal Flow Graph)
(b) Digraph
(c) Grid model (optimal area) layout for 8 node 2-SE graph
(d) Adjacency Matrix for (b)
(e) Butterfly network on the 2-SE graph



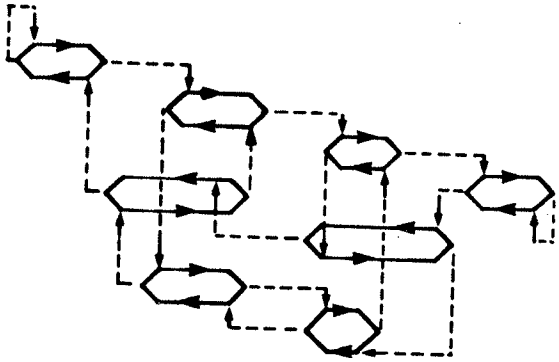
(a)

$$\begin{bmatrix} 1&0&0&0&1&0&0&0 \\ 1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&1 \\ 0&0&0&1&0&0&0&1 \end{bmatrix}$$

(b)



(c)



(d)

Figure 4.10: VA GRAPHS FOR BINARY ALPHABET AND MEMORY 4
(a) Digraph
(b) Adjacency Matrix for (a)
(c) Grid model (optimal area) layout for 16 node 2-SE graph
(d) Butterfly network on the 2-SE graph

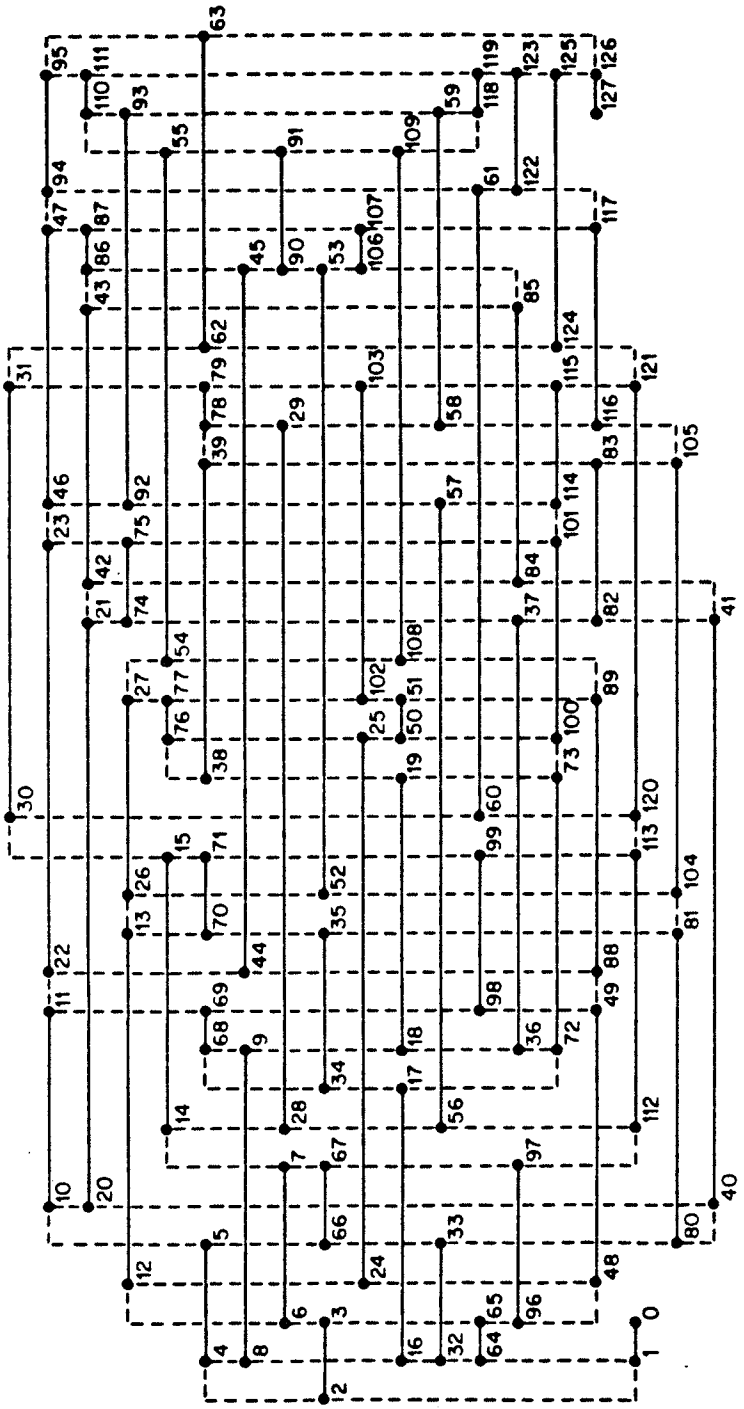


Figure 4.11: GRID MODEL 2-SE LAYOUT FOR P(2) AND $\nu = 7$ USING THE LAYOUT TECHNIQUE IN [119].

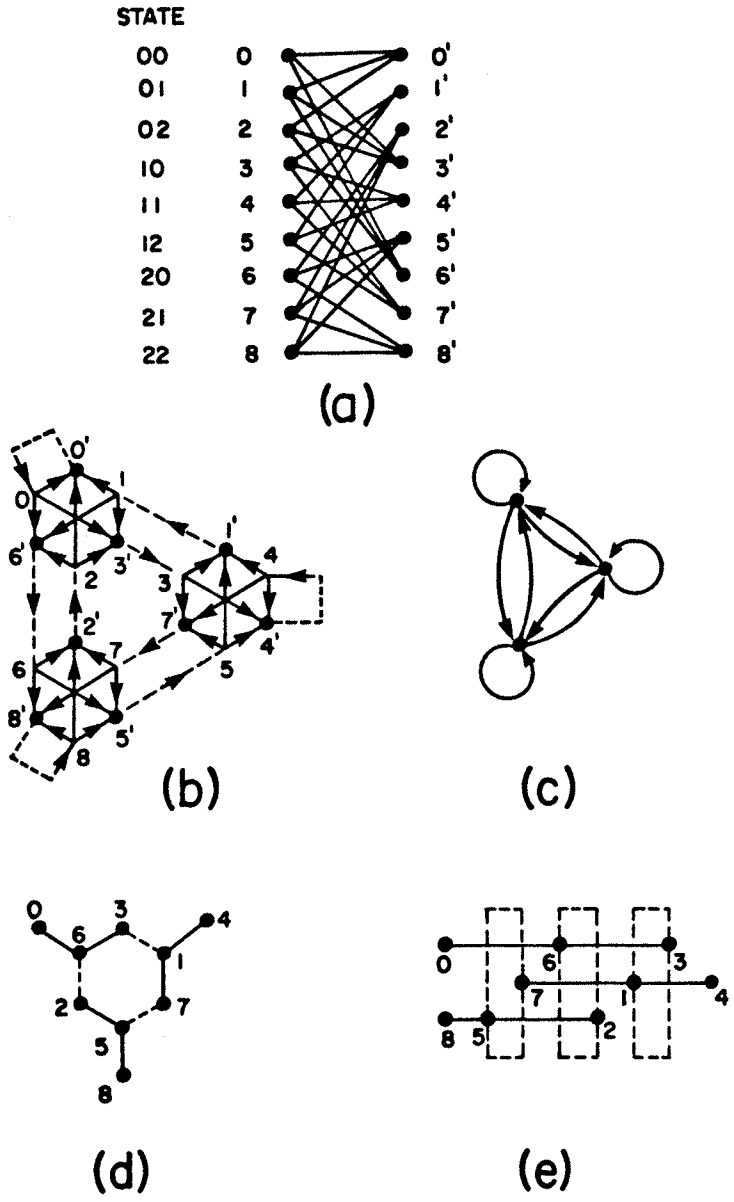
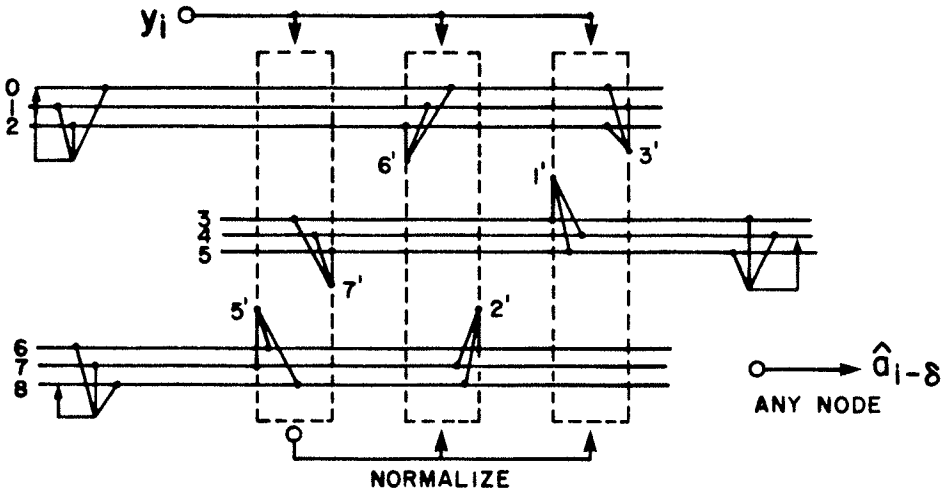
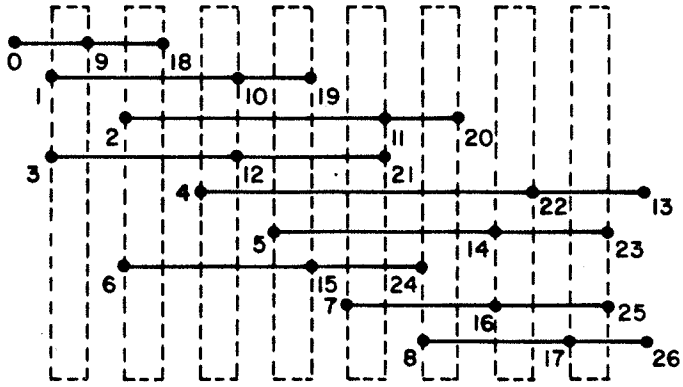


Figure 4.12: VA GRAPHS FOR TERNARY ALPHABET AND MEMORY 2
(a) Single Stage Trellis
(b) Butterfly network (Signal Flow Graph)
(c) Digraph
(d) Hex layout for the 9-node 3-Shuffle-Exchange Graph
(e) A 7x3 layout for the 9-node 3-SE Graph



(a)



(b)

Figure 4.13: FURTHER EXAMPLES FOR TERNARY ALPHABETS
(a) Detailed signal flow of Figure 4.12(e)
(b) A 9x16 layout for the 27-node 3-S-E graph

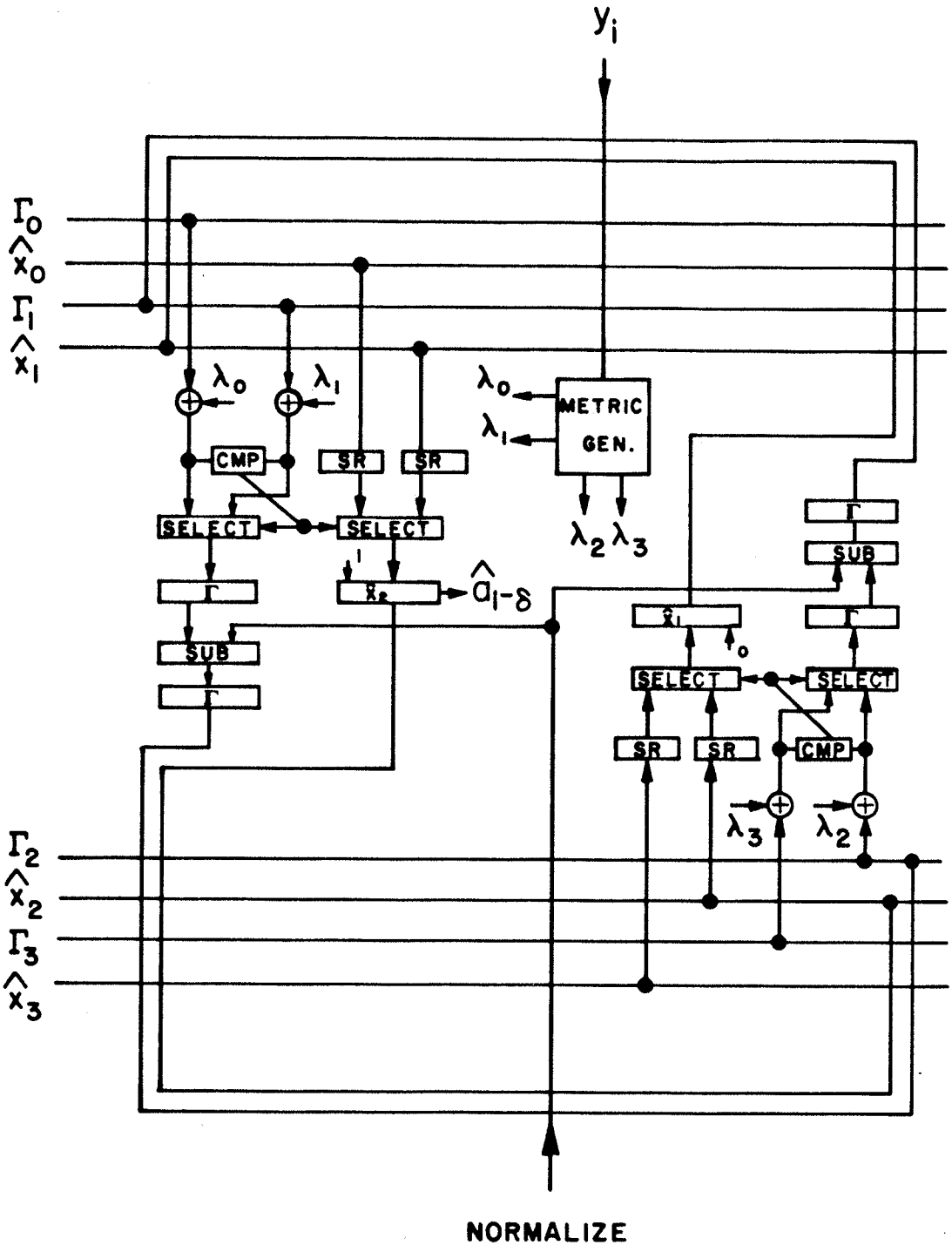


Figure 4.14: DATA PATHS WITHIN THE VA 2-SE LAYOUT SLICE

Γ	State Metric	CMP:	Comparator
λ	Path Metric	SR:	Shift Register
$\hat{x}$	Survivor Sequence	SELECT:	2-TO-1 Multiplexer
$\hat{a}$	Output Data Seq.	SUB:	Subtract Unit

(See Chapter II for a description of the nomenclature)

4.3.2 Cube - Connected Cycles Layouts

A feasible substitute for the shuffle-exchange network is a relatively new interconnection scheme known as the Cube-Connected Cycles. The CCC has been shown to be capable of efficiently solving a large class of problems that include the fast Fourier transform, multiplication, polynomial product, sorting, permutations and derived algorithms [84]. This section demonstrates that the CCC structure can be used to solve dynamic programming problems of the type suitable for implementing the Viterbi algorithm. The topology of this network can be derived from a boolean hypercube of 2^k vertices by replacing each vertex with a cycle of k vertices, for a total of $k2^k$ vertices. It has a compact and regular VLSI type structure. In addition, the CCC is of particular interest because its vertex degree is independent of the problem size parameter, unlike the hypercube. Unfortunately, the CCC structure as it stands is capable only of implementing the VA for binary alphabets. In this section, well known VLSI layouts for the CCC are extended and generalized such that these new structures can realize the VA for m-ary alphabets. These new computation graphs have vertex degree less than five.

As opposed to the SE, which implements a recursive version of the trellis diagram for one time step, the CCC directly implements several stages of the trellis diagram, as illustrated in Figure 4.15 for two time steps. The

correspondence between the trellis and the CCC layout can be visualized by restructuring the trellis diagram as in Figure 4.15(a).

The CCC concept can be generalized for m-ary alphabets in the same manner as the SE was. Since each node must have a unique label, associate each node with a unique $\nu+1$ digit string where all digits are m-ary except the first digit which is ν-ary (i.e., nodes are labelled from 0,0,0,...,0 to $\nu-1$,m-1,m-1,...,m-1). Cycle connections, illustrated by the vertical wires, are defined by nodes with the same last ν digits, the first digit being any number from 0 to $\nu-1$. (Two nodes K and K' are linked via a cycle connection for K = $s_\nu, s_{\nu-1}, \ldots, s_0$ and K' = $\bar{s}_\nu, s_{\nu-1}, \ldots, s_0$ where $\bar{s}_\nu$ is any digit from 0, ..., $\nu-1$ other than s_ν.) Cube connections, illustrated by the horizontal wires, are defined by nodes which have the s_ν^{th} digit to the right of $s_{\nu-1}$ being any number from 0 to m-1. (Two nodes K and K' are linked via a cube connection for K = $s_\nu, s_{\nu-1}, \ldots, s_j, \ldots, s_0$ and K' = $s_\nu, s_{\nu-1}, \ldots, \bar{s}_j, \ldots, s_0$ for $j = \nu - 1 - s_\nu$.)

For an m-ary alphabet and memory ν, the properties of the generalized CCC are the following:

(i) νm^ν nodes.

(ii) m^ν cycles.

(iii) ν nodes in each cycle.

(iv) $\nu m^{\nu-1}$ cube connections.

(v) m nodes in each cube connection.

(vi) $2\nu m^\nu - \nu m^{\nu-1}$ edges. This is established as

follows: ν nodes are contained in each of the m^{ν} cycles, which because of their cyclic nature also contain νm^{ν} edges collectively. Then there are $\nu m^{\nu-1}$ cube connections each with m-1 edges. In total, $2\nu m^{\nu} - \nu m^{\nu-1}$ edges.

Area efficient VLSI layouts for the CCC are generated when cycles are ordered from left to right so that the minimum value of the nodes in each cycle form an increasing sequence. This concept is illustrated for binary alphabets in Figure 4.16(b) and for ternary alphabets in Figure 4.17(c).

The SE processor has a slight area advantage over the CCC processor, because of its simpler control algorithm. However, the CCC is a somewhat more regular interconnection pattern, so that it may be easier to wire up in practice.

The regular interconnection pattern can be exploited by defining a CCC building block, useful in the construction of VA systems with large channel memory lengths, as illustrated in Figure 4.18 . These building blocks would be manufactured as separate chips and arranged on a printed circuit board or the dies would be integrated onto a silicon wafer which contains a regular interconnection pattern. This interconnection pattern would be personalized by the placement of appropriate solder dots, as illustrated in Figure 4.18(c) and Figure 4.19, after selecting the functional

processors from those available on the wafer in each radial line of processors.

Only processors in one stage of the trellis are active at each time step. This implies that each cycle layout requires that the λ's generated by a quantized received data signal be fed to successive nodes each time step. This feature is advantageous in serially transmitting the survivor sequences, which normally require more bits than the state metrics, over long interprocessor interconnections. In addition, this feature may allow the structure to be multiplexed for the detection of data sequences from several data sources. At most, ν independent data streams can be decoded in a CCC structure with νm^{ν} nodes, resulting in full hardware utilization. Note however that the appropriate path metrics would have to migrate from node to node around the cycle connection with the state metrics.

The CCC structure may also be used to decode one data stream ν times as fast as the speed of a single ACS processing node and associated routing step. This feature can be exploited in digital communication over burst noise channels.

One technique for achieving reliable transmission on a burst noise channel is the use of time diversity or interleaving [124]. The approach requires no knowledge of channel memory other than its approximate length, and is

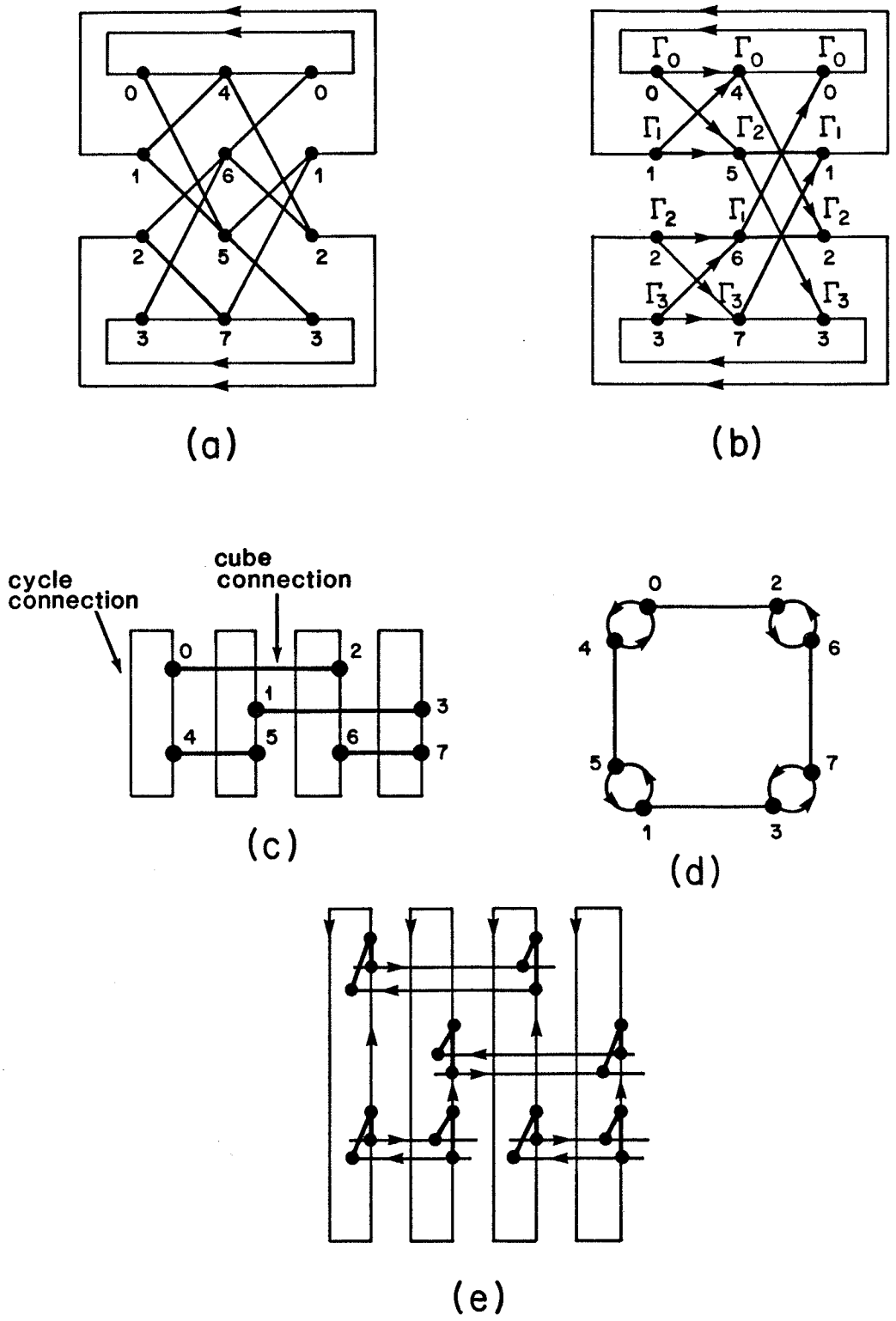
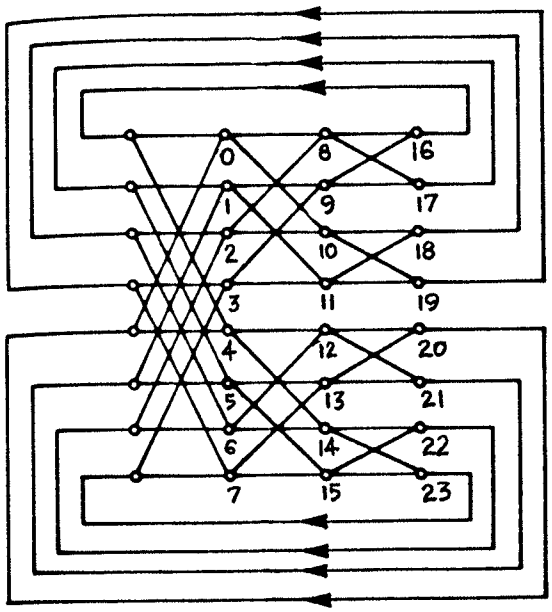
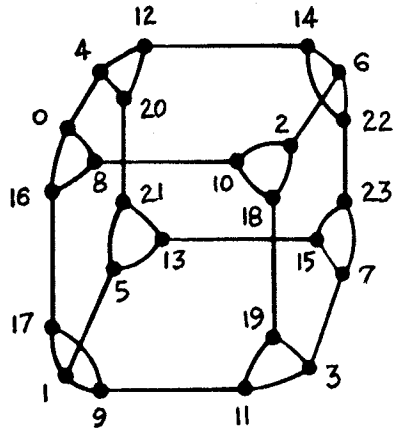


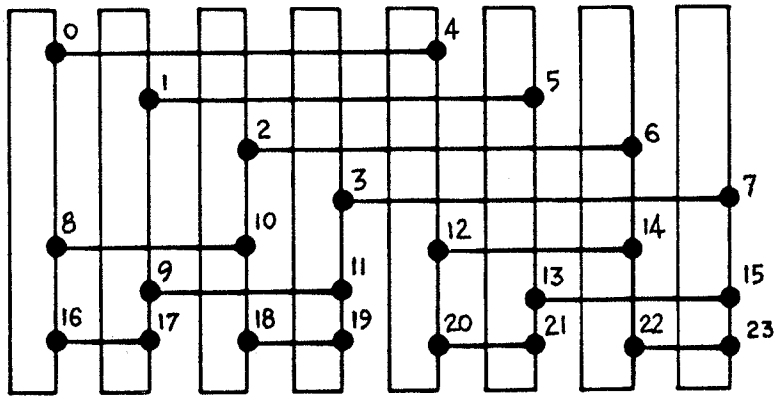
Figure 4.15: CCC GRAPHS FOR BINARY ALPHABET AND MEMORY 2
(a) Conventional Trellis Diagram Structure
(b) Restructured Double Time Step Trellis Diagram
(c) Grid model (optimal area) layout for 8 node CCC graph
(d) Grid model in (c) rearranged to illustrate cycles on a 2-D hypercube
(e) Detailed Grid Layout With Bus Structure



(a)



(c)



(b)

Figure 4.16: CCC GRAPHS FOR BINARY ALPHABET AND MEMORY 3
(a) Restructured Trellis Diagram
(b) Grid model (optimal area) layout for 24 node CCC graph
(c) The CCC illustrated on a cube

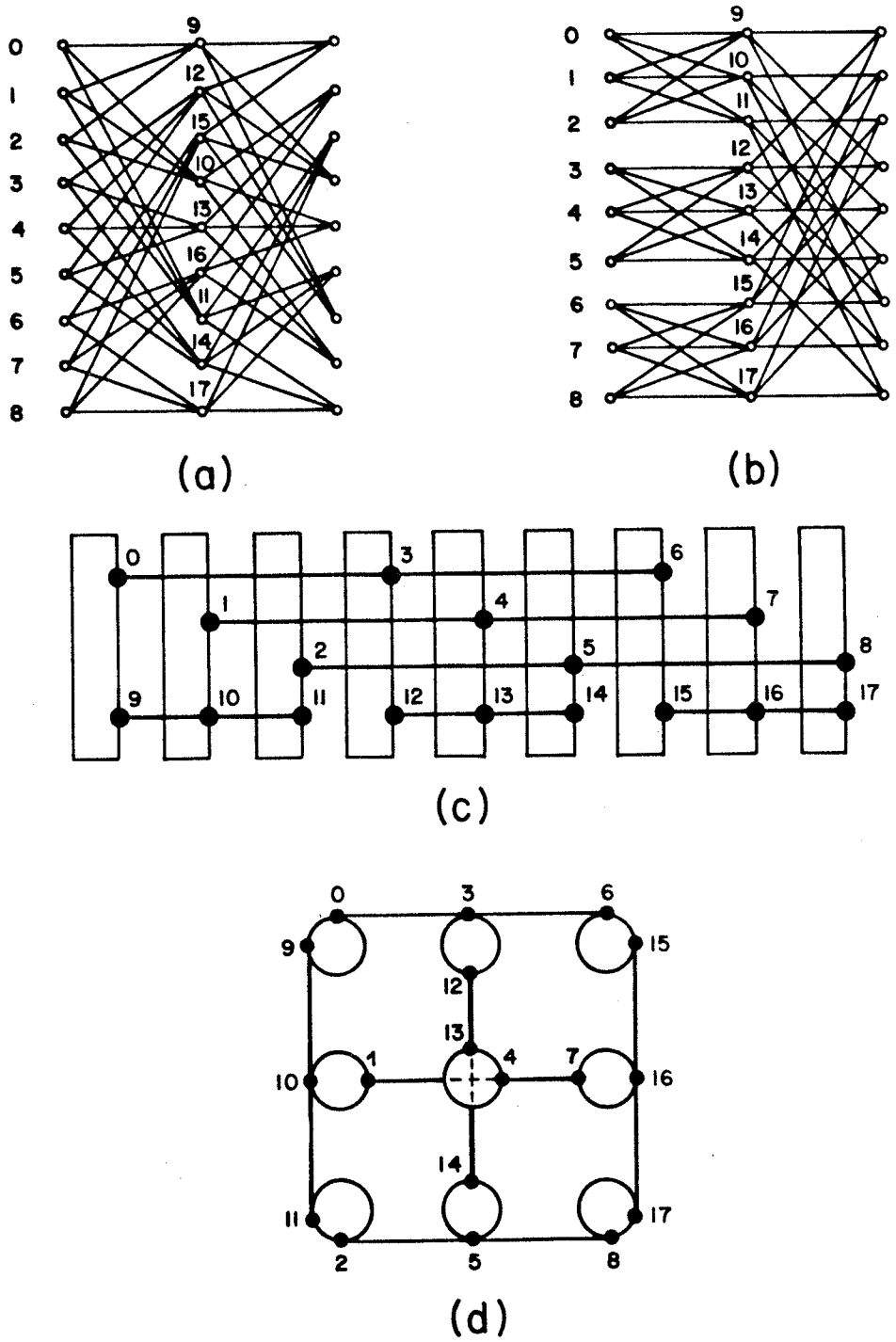


Figure 4.17: CCC GRAPHS FOR TERNARY ALPHABET AND MEMORY 2
(a) Conventional Trellis Diagram
(b) Restructured Ternary Trellis Diagram
(c) Grid model (optimal area) layout for 18 node CCC graph
(d) Grid model in (c) rearranged to illustrate cycles on a set of 2-D hypercubes

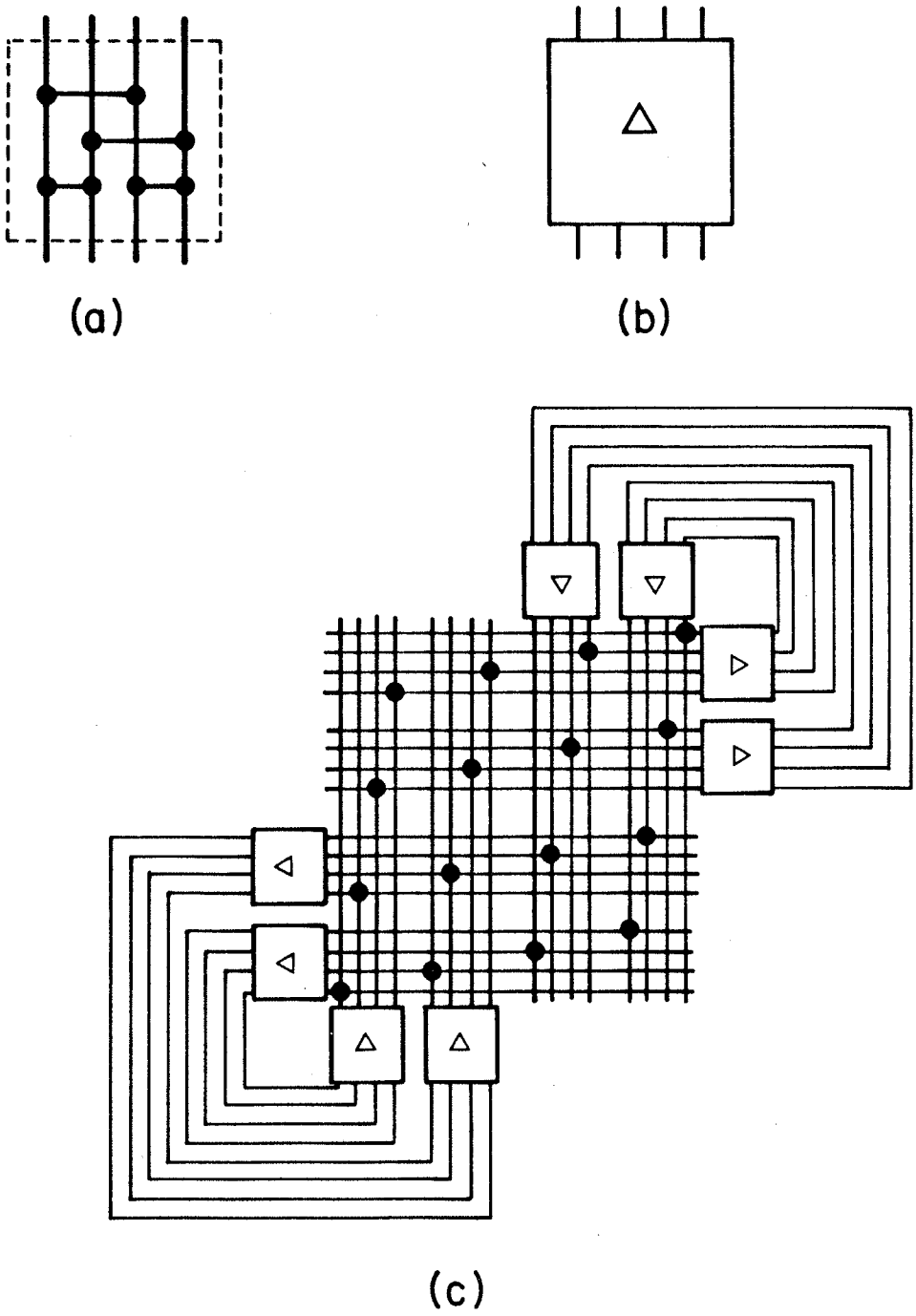


Figure 4.18: A CCC BUILDING BLOCK FOR BINARY ALPHABETS
(a) Detail of building block primitive (refer to Fig. 4.15)
Note that each heavy dot is an ACS circuit
(b) Schematic representation of the building block
(c) Layout for CCC network for binary alphabet and memory 4
Note orientation of 8 building blocks (chips) and
placement of solder dots in central programming plane

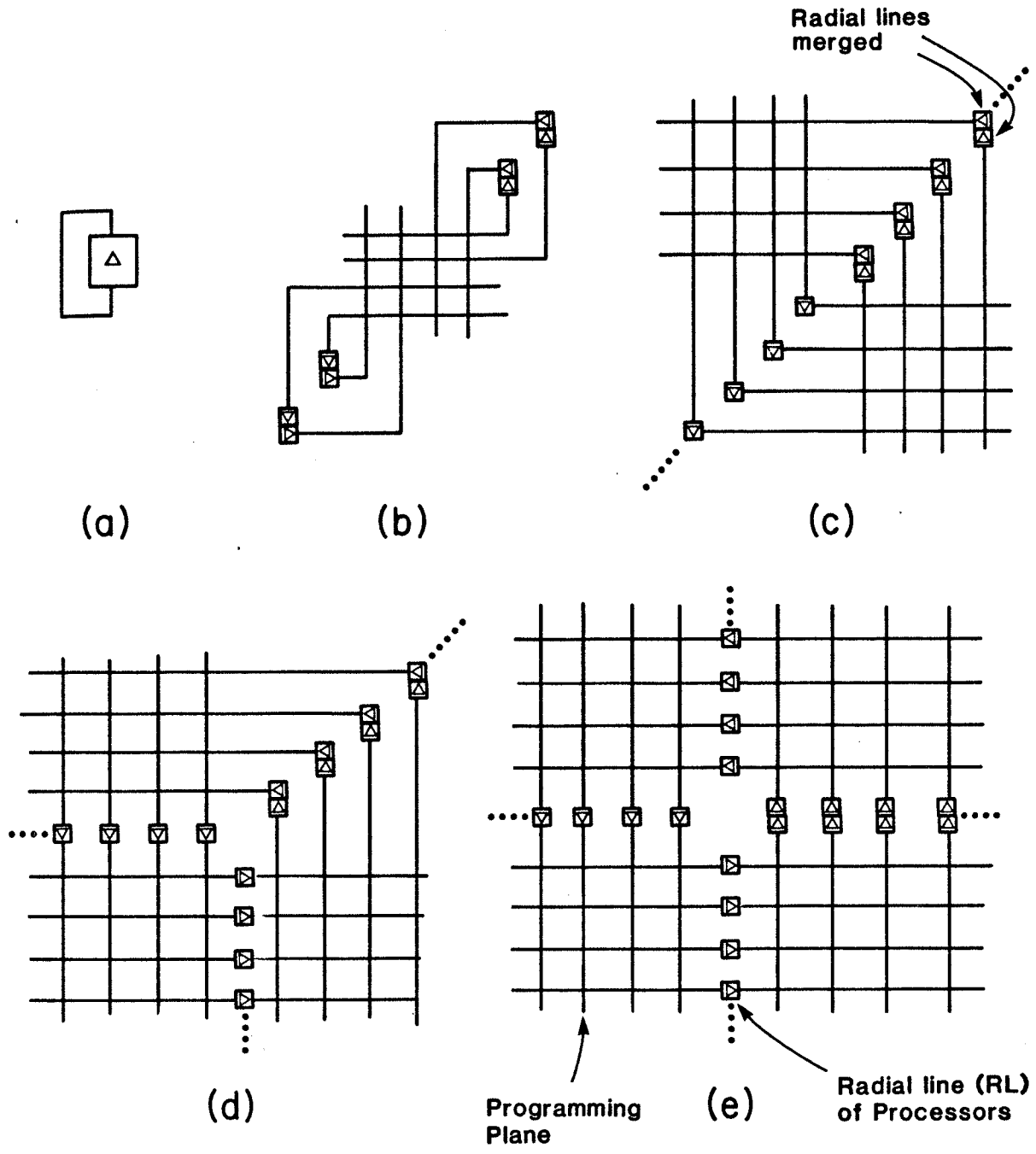


Figure 4.19: CCC CHIPS IMPLEMENT THE VA (BINARY ALPHABET)

(a)	Memory 2:	1 chip /RL:	1 RL :	TOTAL	1 chip
(b)	Memory 4:	4 chips/RL:	2 RLs:	TOTAL	8 chips
(c)	Memory 6:	16 chips/RL:	3 RLs:	TOTAL	48 chips
(d)	Memory 8:	64 chips/RL:	4 RLs:	TOTAL	256 chips
(e)	Memory 10:	256 chips/RL:	5 RLs:	TOTAL	1280 chips

Note: Solder dots absent and relative sizes not illustrated

consequently very robust to changes in memory statistics. From a conceptual standpoint, the incoming stream of binary data is separated into a fixed number, say ν, of data streams. Each of the ν data streams is then separately encoded and the encoded sequences are interleaved together for transmission through the channel. The constant ν is known as the interleaving degree. At the channel output, the receiver demultiplexes (unscrambles) the received data streams into ν streams, each stream is separately decoded, and the decoded data is finally commuted together again.

The idea behind this technique is that successive symbols within any code word will be separated on the channel by ν symbol time units. Thus in the case of practical channels, where memory decreases with time separation, the channel noise affecting successive letters in a code word will be essentially independent for sufficiently large ν. Consequently, any of the coding techniques for memoryless channels can be used on a burst noise channel in conjunction with interleaving.

If the coding technique is a convolutional code of constraint length K then the appropriate VLSI realization of the convolutional decoder based on the Viterbi Algorithm is a demultiplexer and a binary CCC structure with $\nu 2^{\nu}$ nodes, when the interleaving degree is a multiple of the convolutional code input memory. The hardware is fully utilized each symbol interval. The channel data rate is limited to ν

times the speed of a single ACS processing node and routing step.

The structure has the following performance characteristics. An error burst of length ν on the channel will look like single errors to each of the separate decoders in the structure. Hence, if each decoder is capable of correcting b errors in a constraint length, then, with interleaving, b or fewer error bursts of length ν or less relative to a guard space of length at most $\nu(K-b) = \nu((\nu+1)n-b)$ will be corrected, for a n-output convolutional encoder [124].

4.3.3 Tree of Meshes Layouts

Rather than observing that the trellis is in fact a folded tree structure, this section considers a divide and conquer layout strategy for the CCC solution to P(m) that capitalizes on the planar embedding property of binary trees. Our motivation is provided by the fact that certain classes of graphs can be partitioned recursively into pieces with few connecting edges. This section presents recursive geometries (floor plans) for realizing the VA in silicon, in the form of H-tree and Y-tree of meshes. The H-tree of meshes concept as proposed for other application areas is known while the Y-tree of meshes construct is presented here for the first time. These layouts have attractive synchronous clock distribution [125] characteristics (vis-a-vis self timed systems) and may be technologically important in

the automated design and functional partitioning of VLSI chips [126,127].

The binary tree of meshes [122] is formed by replacing each node of a complete binary tree with a mesh and each edge by several edges which link the meshes together. More precisely, the root of the binary tree is replaced by an n x n square mesh (where n is a power of 2), its sons are replaced by n/2 x n meshes, their sons are replaced by n/2 x n/2 meshes, and so on until the leaves are replaced by 1 x 1 meshes. In all cases connections are made between fathers and sons so as to preserve the column and row order of the nodes and to insure that the resulting graph is planar. By modifying the familiar H-tree layout for binary trees the N-node tree of meshes can be embedded without edge crossings in a square region, as illustrated in Figure 4.20. The resulting graph, referred to as the n x n H-tree of meshes has $N = 2n^2 \log n + n^2$ nodes. The N-node tree of meshes has an $O(N^{\frac{1}{2}}/\log^{\frac{1}{2}} N)$-separator [122].

Leighton in his doctoral dissertation [122] showed how to embed any N-node planar graph in an O(N log N)-node tree of meshes. This result has been generalized to arbitrary graphs. By modifying the standard H-tree layout, an N-node tree of meshes can be embedded without edge crossings in area O(N log N).

Previously, a VLSI layout for the Cube-Connected Cycles network on $N = k2^k$ vertices was presented [84]. The CCC graph has an $O(N/\log N)$-separator theorem since removing all edges in one dimension of the original hypercube bisects the graph, removal of those in another bisects the halves, and so forth in all k dimensions (a recursive generalization of bisector). The CCC and H-tree of meshes have identical separators as shown in equations 4.10 and 4.11 below.

$$
\begin{aligned}
O[(\frac{N}{\log N})^{\frac{1}{2}}] &= O\{[\frac{n^2\log n + n^2}{\log(n^2\log n + n^2)}]^{\frac{1}{2}}\} \\
&= O\{[\frac{2^{2k}\log 2^k + 2^{2k}}{\log(2^{2k}\log 2^k + 2^{2k})}]^{\frac{1}{2}}\} \text{ when } n = 2^k \\
&= O[(\frac{k2^{2k}}{\log k2^{2k}})^{\frac{1}{2}}] \qquad (4.10) \\
&= O[(\frac{k2^{2k}}{k})^{\frac{1}{2}}] \\
&= O(2^k)
\end{aligned}
$$

$$
\begin{aligned}
O(\frac{N}{\log N}) &= O(\frac{k2^k}{k}) \qquad (4.11) \\
&= O(2^k)
\end{aligned}
$$

Consequently, the CCC can be embedded within the H-tree of meshes topology. Figure 4.21(b) illustrates an example of the CCC layout of Figure 4.16(b) embedded in the H-tree of meshes T_4.

The CCC layouts for ternary alphabets, presented earlier, lead to the development of a new construct called the Y-tree of meshes, the counterpart of the H-tree of meshes. The development of the Y-tree of meshes is guided by the requirement it have a separator theorem identical to that of the ternary CCC layouts. However, the concept of a separator theorem must be generalized to accomplish this task. In the case of ternary alphabets we are interested in the cut set which trisects the computation graph.

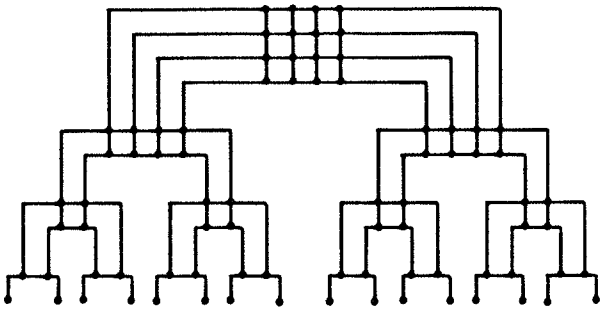
Definition: Let S be a class of graphs closed under the subgraph relation, that is, if G is an element of S, and G' is a subgraph of G, then G' is also an element of S. The class S is said to have an f(n)-triseparator, or is f(n)-triseparable, if the following condition is true:

> There exists a constant c such that if G is an n-vertex graph in S, then by removing at most $c \cdot f(n)$ edges, G can be partitioned into three disjoint subgraphs G_1, G_2 and G_3 each having at least n/4 vertices.

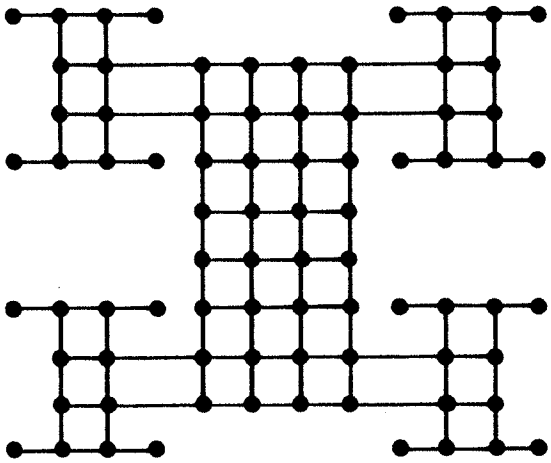
A family of graphs, S, has a strong f(n)-triseparator if the conditions for an f(n)-triseparator hold, and in addition G_1, G_2, and G_3 have at most (n+1)/3 nodes each.

The N-node CCC layout for ternary alphabets has a strong O(N/log N) triseparator identical to the ternary tree of meshes defined below.

The ternary tree of meshes is formed by replacing each node of a complete binary tree with a mesh and each edge by several edges which link the meshes together as before. More precisely the root of the binary tree is replaced by an n x n x n hexagonal mesh (where n is a power of 3), their sons are replaced by n/3 x n/3 x n/3 meshes, and so on until the leaves are replaced by 1 x 1 x 1 meshes. Like the H-tree layout of the binary tree, the Y-tree layout is a recursive embedding of the complete ternary tree in a hexagonal mesh. By modifying the Y-tree layout of Figure 4.22(a) slightly the Y-tree of meshes is constructed as illustrated in Figure 4.22(c). The Y-tree layout has a maximum edge length less than that of CCC layouts embedded in a rectangular grid. This is a desireable feature in that the Y-tree layout attempts to minimize propagation delay of the interprocessor connections. In a stylized representation, larger channel memory lengths are easily depicted, as in Figure 4.23. This visual notation highlights the recursive partitioning into pieces with few connecting edges that this class of graphs enjoy.

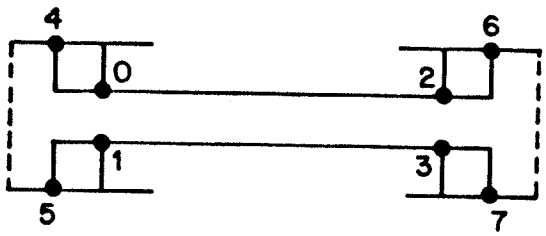


(a)

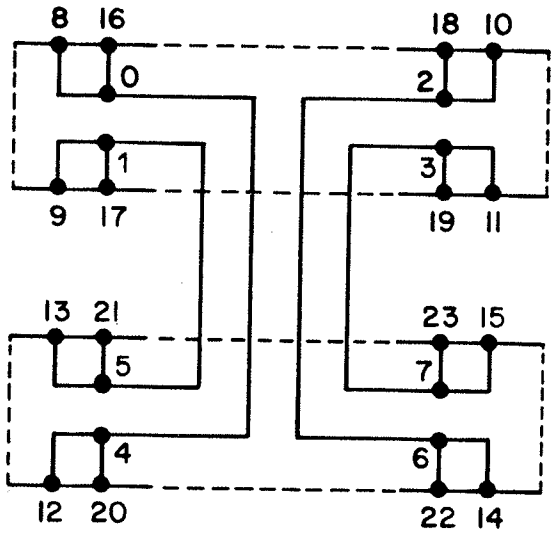


(b)

Figure 4.20: TREE STRUCTURES
(a) The 4x4 tree of meshes
(b) H-tree layout of the 4x4 tree of meshes



(a)



(b)

Figure 4.21: CCC EMBEDDED IN A TREE OF MESHES
(a) CCC of Fig. 4.15 embedded in the H-tree of meshes T_4
(b) CCC of Fig. 4.16 embedded in the H-tree of meshes T_4

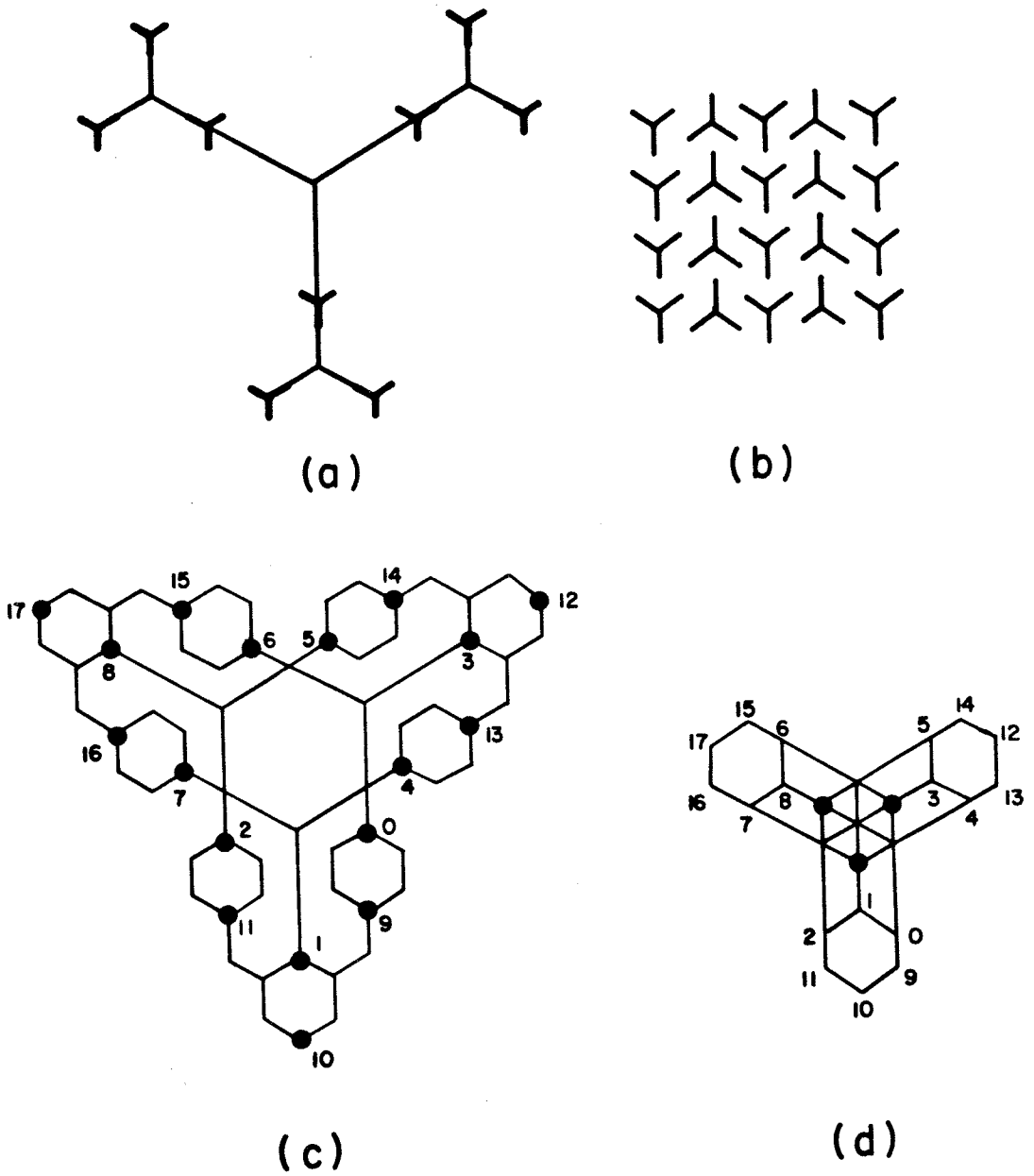


Figure 4.22: THE Y-TREE LAYOUT
(a) A Y-tree layout of a complete ternary tree of height 3
(b) A forest of Y-trees packed in a bounding rectangle
(c) Y-tree layout of Fig. 4.17 (P(3); ν = 2)
(d) Stylized Y-tree of meshes notation for (c)
Note placement of solder dots in central programming plane

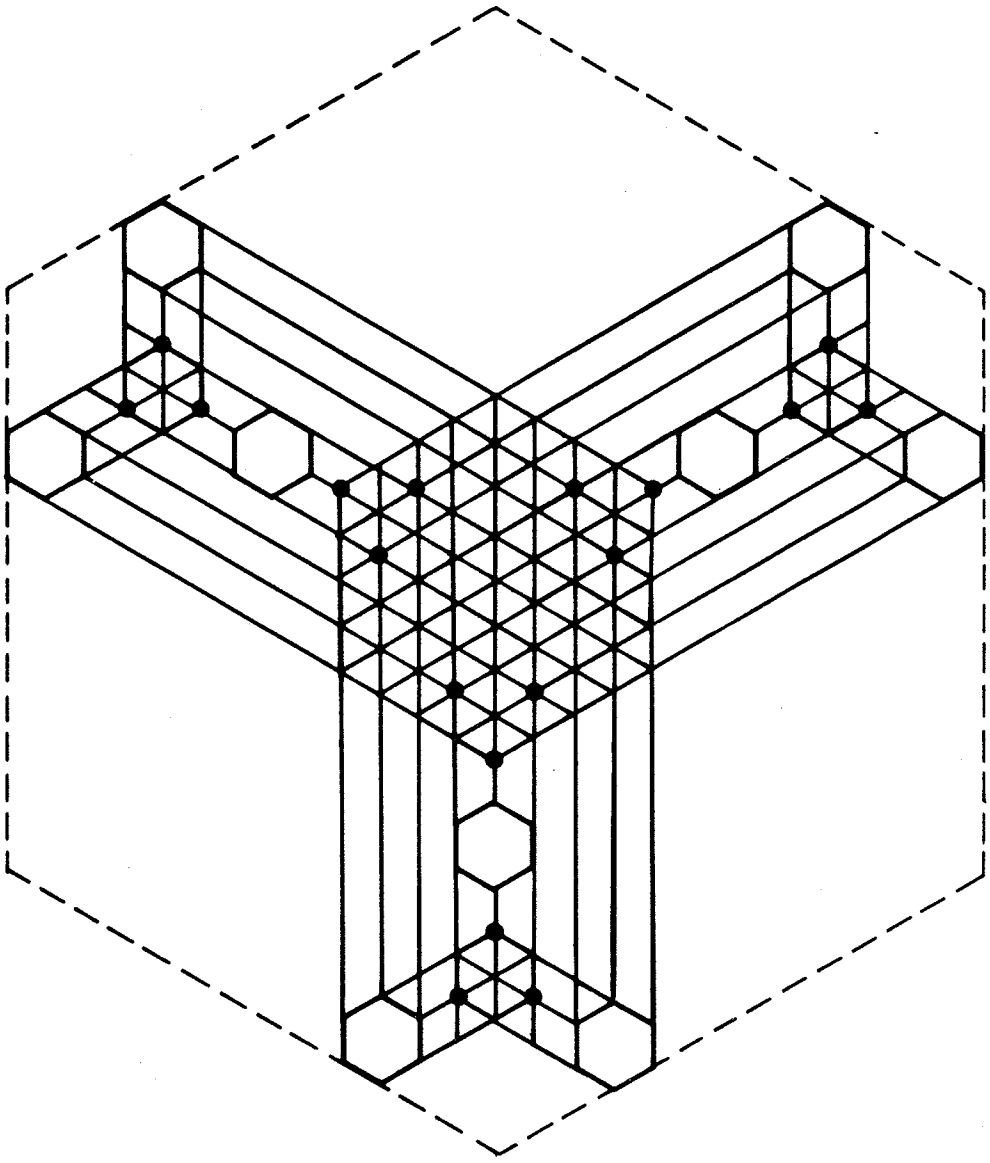


Figure 4.23: THE Y-TREE OF MESHES LAYOUT
CCC for ternary alphabet and memory 3 in a Y-tree of meshes T_9
Small hex regions are a collection of ACS processors.
Note placement of solder dots in programming plane.

4.4 AREA-TIME COMPLEXITY MEASURES

Using the grid model described in the last chapter, we demonstrate, in this section, that any VLSI implementation of problem P(m) must obey certain functions of the problem size parameter that lower bounds particular types of cost functions such as area*time. This is done by a technique originally developed by Thompson, using the VLSI grid model which proves that a minimum amount of information must be shipped from one part of the circuit to another to solve a particular problem instance.

As demonstrated above, the speed of a VLSI design may be limited either by the time taken by arithmetic operations or by the time taken to get intermediate results to the proper place. It is the latter that generally places a stricter limit on large VLSI designs. For algorithms that must pass data from one side of the communication graph to the other, the bottleneck in data flow is ultimately quantified and bounded by the minimum bisection width of the graph.

The minimum bisection width of a graph is the smallest number of edges whose removal disconnects one half of the vertices from the other. The set of removed edges is called the cut set of the bisection. For example, the minimum bisection width of Figure 4.4(a) is four. In general, the minimum bisection width of a square mesh of N nodes is $N^{1/2}$.

Thompson found that any VLSI design for an N-point DFT, with a communication graph of minimum bisection width w, is lower bounded in area and time by $w^2/4$ and $N/(2w)$ respectively. Thus he proved the following theorem [128]: If a VLSI design with area A computes an N-point DFT in time T, then $AT^2 > N^2/16$. Since the trellis structure of the m-ary VA is isomorphic to the signal flow graph of a radix-m DFT, one would expect similar results to apply to the Viterbi algorithm. In order to prove this conjecture we need the following lemma.

LEMMA: Given the graph G with m^{ν} nodes and $m^{\nu+1}$ edges, where each node is given a m-ary label $x_{\nu-1},\ldots,x_1,x_0$ and each node can source m edges (and sink m edges) according to the following rule:

$$x_{\nu-1},\ldots,x_1,x_0 \longrightarrow \begin{bmatrix} x_{\nu-2},\ldots,x_0,\overline{x_{\nu-1}} \\ x_{\nu-2},\ldots,x_0,x_{\nu-1} \end{bmatrix}$$

then all partitions that separate the nodes into two disjoint subsets, a "left side" that contains $\lfloor m/2 \rfloor m^{\nu-1}$ nodes and a "right side" that contains $\lceil m/2 \rceil m^{\nu-1}$ nodes have the property that $\Omega(m^{\nu})$ edges cross the partition separating the left and right side.

DEFINITION: Let j be an element from a subset of $\lfloor m/2 \rfloor$ integers drawn from the set {0,1, ..., m-1}. In addition, let j' be an element of the remaining $\lceil m/2 \rceil$ integers in the set {0,1, ..., m-1}.

PROOF:

Step 1: THE NUMBER OF EDGES FROM THE LEFT TO THE RIGHT SIDE

1. The $\lfloor m/2 \rfloor m^{\nu-1}$ nodes on the left are characterized by $x_{\nu-1}=j$, for example.
2. The nodes on the right hand side are characterized by $x_{\nu-1}=j'$ for a total of $\lceil m/2 \rceil m^{\nu-1}$ nodes.
3. For the nodes on the left side, $\lfloor m/2 \rfloor$ of these nodes have $x_{\nu-2}=j$ while $\lceil m/2 \rceil$ of these nodes have $x_{\nu-2}=j'$.
4. Only those nodes on the left with $x_{\nu-2}=j$ have edges which remain exclusively on the left side, while those with $x_{\nu-2}=j'$ have edges which go exclusively to the right side of the chip.
5. On the left, $\lfloor m/2 \rfloor * \lceil m/2 \rceil * m^{\nu-2}$ nodes go to the right side of the chip.
6. Each node has an outdegree of m. However, each edge is delivering the same state metric to the same side of the chip so only one wire per node is required to cross the partition.
7. The total number of edges from the left side to the right side is: $1 * \lfloor m/2 \rfloor * \lceil m/2 \rceil * m^{\nu-2} = \Omega(m^{\nu})$.

Step 2: THE NUMBER OF EDGES FROM THE RIGHT TO THE LEFT SIDE

8. Of the $\lceil m/2 \rceil * m^{\nu-1}$ nodes on the right side, $\lfloor m/2 \rfloor$ of these nodes have $x_{\nu-2}=j$.
9. Each of these nodes source m edges from the right side to the left side for a total of $1 * \lceil m/2 \rceil * \lfloor m/2 \rfloor * m^{\nu-2}$ edges.

Therefore, the total number of edges that cut the partition equals $\Omega(m^{\nu}) + \Omega(m^{\nu}) = \Omega(m^{\nu})$. ■

It can be shown that even the divide and conquer approach, which uses more area, has the same bisection width. Using the above lemma, we are now in a position to prove the following theorem.

THEOREM: If a VLSI design with area A executes the Viterbi algorithm with alphabet m and memory ν in symbol interval T, then $AT^2 > \Omega(m^{2\nu})$.

PROOF: The proof proceeds as an extension of [80].

1. A communication graph of minimum bisection width w has an area greater than $w^2/4$.

2. Associate a graph, G, of minimum bisection width w, with each VLSI design of the VA. At least m^{ν}/w time is required to compute the VA with alphabet m and memory ν on a VLSI design that corresponds to G. This is a consequence of the following property of the VA. With alphabet m and memory ν the algorithm has m^{ν} states. The algorithm must pass m^{ν} words along valid state transitions amongst m^{ν} states in each symbol interval T. If G is partitioned into two subgraphs each containing $\lfloor m/2 \rfloor * m^{\nu-1}$ and $\lceil m/2 \rceil * m^{\nu-1}$ nodes, then by the above lemma for some symbol interval the minimax number of operand transfers between subgraph states in the trellis is $\Omega(m^{\nu})$. Since $\Omega(m^{\nu})$ unique signal flow edges as defined by the trellis cut the bisector, it takes $\Omega(m^{\nu}/w)$ time to pass $\Omega(m^{\nu})$ operands over w wires.

The arguments presented in 1. and 2. can be immediately combined to give the theorem: If a VLSI design with area A executes the Viterbi algorithm with alphabet m and memory ν in symbol interval T then $AT^2 > \Omega(m^{2\nu})$. ■

At least three designs approach this lower bound for p(2), the VA for binary alphabets, those with either a perfect shuffle, CCC or a mesh-type interconnection pattern.

Furthermore, as shown below, the energy consumption during each symbol interval (power) defined by the AT measure of complexity is given by $AT = \Omega(m^{3\nu/2})$. The lower bound is nearly tight for P(m) in a technology such as CMOS which has very low static power dissipation (vis-a-vis dynamic power consumption). This measure can also be interpreted as the reciprocal of throughput per unit area.[4] A completely pipelined circuit optimal with respect to this criterion can be claimed to make best use of this area.

COROLLARY: Any grid model layout of area A that takes minimax time T to solve P(m) with algorithm memory ν, is lower bounded by the relation $AT = \Omega(m^{3\nu/2})$.

Proof: The area of any VLSI design for P(m) with algorithm memory ν must be $\Omega(m^{\nu})$ since each of the m^{ν} ACS processing nodes occupies at least unit area. In addition, as stated previously, a communication graph of minimum bisection width w is lower bounded in area and time by $w^2/4$ and $\Omega(m^{\nu}/w)$, respectively. Consequently, these statements can be

[4] The United States Military development program for Very High Speed Integrated Circuits (VHSIC) uses a processing throughput per unit area (TP) figure of merit defined by:
TP = (gate density)*(clock rate) = gate-Hz/mm^2.
Contemporary microprocessors achieve a TP of $\simeq 10^{10}$ gate-Hz/mm^2.

combined to produce:

$$AT^x = \Omega((m^\nu + \frac{w^2}{4}) * \left[\frac{m^\nu}{w}\right]^x) \tag{4.12}$$

$$= \Omega(m^{\nu(1+x)} w^{-x} + m^{\nu x} w^{2-x}) \tag{4.13}$$

AT^x can be minimized with respect to w since the first term decreases with increasing w while the second term increases with increasing w, for $0<x<2$. Take the derivative of equation 4.13 and equate to zero,

$$\frac{\partial}{\partial w} (m^{\nu(1+x)} w^{-x} + m^{\nu x} w^{2-x}) = 0, \tag{4.14}$$

$$-xm^{\nu(1+x)} w^{-x-1} + (2-x) m^{\nu x} w^{1-x} = 0, \tag{4.15}$$

$$\frac{w^{1-x}}{w^{-x-1}} = \frac{xm^{\nu(1+x)}}{(2-x)m^{\nu x}} \tag{4.16}$$

$$w^2 = \frac{x}{2-x} \cdot m^\nu \tag{4.17}$$

Therefore,

$$w = \Theta(m^{\nu/2}) \tag{4.18}$$

optimizes AT^x for $0<x<2$. This result can be combined with equation 4.13 to produce,

$$AT^x = \Omega\ (m^{\nu(2+x)/2}),\ 0 < x < 2 \tag{4.19}$$

For the case, x=1:

$$AT = \Omega\ (m^{3\nu/2}). \qquad (4.20)$$

proving the Corollary. ■

4.5 THE CONSTANT FACTORS IN VLSI IMPLEMENTATIONS

This chapter has demonstrated the existence of an area-time tradeoff for VLSI implementations of the Viterbi algorithm, at least for asymptotically large problem instances. Table 4.1 presents the asymptotic ordering of architectures, discussed in this chapter, in terms of increasing wire area which corresponds closely with increasing throughput. This ordering may be destroyed in implementations of small problem instances, since the processor area and control circuitry - the neglected constant factors - may dominate the wire area in these situations, forcing certain types of architectures to lose their asymptotic size advantage. (Hence, those with simple wiring schemes may be more costly areawise to implement than those with complex wiring patterns.) The tradeoffs amongst various architectures may not be very tractable below some critical problem size parameter ν_0 (i.e., below some critical channel memory length ν_0). Several questions arise from such considerations. First, can we establish an estimate of ν_0? Second, is ν_0 of technological interest? If so, can problem instances of $\nu > \nu_0$ be fabricated (for a given baud rate) using present technology?

TABLE 4.1

PERFORMANCE SUMMARY

LAYOUT TYPE	LOGIC SPEED FOR A SYMBOL INVERVAL OF T	WIRE AREA	NUMBER OF PROCESSORS	REMARKS
UNIPROCESSOR	$\Omega(\frac{m^{\nu+1}}{T})$	$O(1)$	1	• processor/memory ratio too low • compute bound
CASCADE	$\Omega(\frac{m^{\nu}}{\nu T})$	$O(\nu)$	ν	• complex path metric controller • pipelined • input queue required
1-D ARRAY	$\Omega(\frac{m^{\nu}}{T})$	$O(m^{\nu+1})$	$m^{\nu+1}$	• simplified processors • simple state and path metric controller
2-D ARRAY	$\Omega(\frac{m^{\lceil \frac{\nu}{2} \rceil}}{\nu T})$	$O(m^{\nu})$	m^{ν}	• complex path metric controller • input queue required
SE	$\Omega(\frac{1}{T})$	$O(m^{\nu}(m^{\nu}-1))$	m^{ν}	• choose ν to be prime
CCC	$\Omega(\frac{1}{T})$	$O(\nu m^{\nu}(\nu m^{\nu}-1))$	νm^{ν}	• can decode ν data streams with same logic speed
H-tree;Y-tree	$\Omega(\frac{1}{T})$	?	νm^{ν}	• specialized CCC layout

In order to answer these queries, consider the basic arithmetic circuitry required for binary alphabets. The ACS circuitry can be built in about $10^5\lambda^2$, if the word length is eight bits. This allows room for two adders, a comparator, two multiplexers, a subtractor for normalization, a few registers to hold operands and survivors, and several dozen gates for control logic and buffering. Accordingly, up to 64 ACS cells (ν=6) could fit on a present day chip that has $10^7\lambda^2$ units of area. By partitioning the design onto several integrated circuits (perhaps using the necklace approach), which conservatively could expand the design by a factor of 16, problem size instances of ν=10 could be realized.

By the year 1990, it is expected that approximately 1024 ACS cells could be formed on a 6 cm diameter silicon wafer. Such a circuit would be capable of handling binary alphabets with an algorithm memory of 10. Partitioning the design onto several wafers would allow problem size instances of ν =14 to be implemented.

The interconnections between the cells have yet to be considered. In the mesh type layouts, assume that the $10^5\lambda^2$ cell is 320λ on a side. As a consequence, a 30 to 50 wire bus is easily accommodated contributing negligible area to the layout. For ν=6, this would allow all survivor and state metrics to be transferred simultaneously. The layout area of an N-element mesh based design is O(N), where $N=m^{\nu}$.

On a shuffle-exchange type design, N ACS cells require $O(N^2)$ area of which $O(N^2-N)$ area is wire. The area devoted to wiring could be reduced somewhat by resorting to pipelining and bit serial transmission of state metrics and survivors, yielding perhaps a 4 to 16 factor in area savings. In addition, the routing control logic is much simpler for the shuffle-exchange layout than for the mesh. The mesh cell, in fact, may be as much as a factor of ν more expensive in area, due to this control circuitry. These considerations imply that for ν_0 in the range of 3 to 7, both the mesh and shuffle-exchange layout areas are equal. Below this range of ν_0 the size advantage of the mesh-based design may not be apparent. This concept is illustrated in Figure 4.24.

Now, with regards to the processing speed of each of the ACS cells. The adder used in the ACS unit can take the form of a carry-save adder in which ripple carries are used between stages. The carry circuits can be designed such that there is only one logic delay per carry. The total carry propagation delay for an 8-bit adder is then only 8τ. Approximately, 8 ns for current CMOS/SOS technology with one nS gate delays. Look-ahead-carry techniques offer speed advantages, but not without a corresponding hardware penalty.

The comparator is implemented by subtracting the two state metric sums from one another. The actual difference

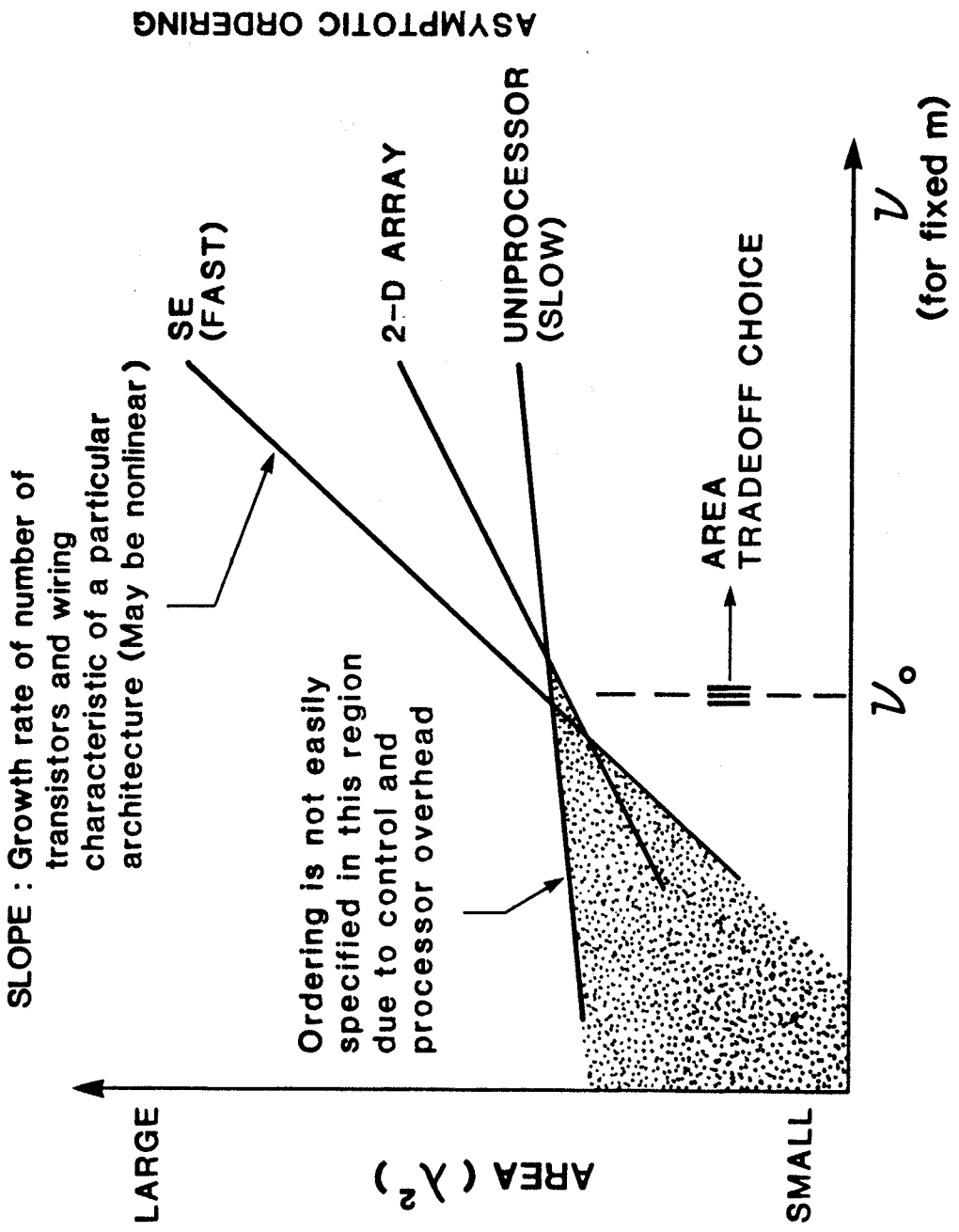


Figure 4.24: THE AREA TRADEOFF
The Constant Factors in the VLSI Implementations makes an an Area Tradeoff Available only after some Critical Problem Size Parameter ν_0. (For clarity only three layout types are presented in the asymptotic ordering. Table 4.1 presents a complete listing of classified architectures.)

is not of interest, only the carry out of the most significant bit of the subtractor. Hence, only the carry portion of the subtractor need be implemented. The total add compare time is only one logic gate delay more than the add time alone.

Two gate delays are required in the multiplexer circuit. Eight logic delays for normalization of the state metrics. And finally, eight gate delays are required in reading out of and into the ACS output registers. In total approximately 27τ logic gate delays are required between clocked sections.

As mentioned earlier, the shuffle-exchange design would be expected to be faster than the mesh design above some threshold problem size ν_0. In either case, each ACS-normalize step takes about 27 ns. The maximum routing time during one symbol interval on a mesh-based design for $\nu = 6$, is $\sqrt{64}/2 = 4$ unit distance routing steps. Allowing two or three clock pulses per unit route for synchronization and buffering, routing takes 20 ns if a 5 ns clock is used. Consequently, in such an implementation a minimum symbol interval of 50 ns could be supported, for a 20 MHz throughput rate.

Routing on the shuffle-exchange design is somewhat faster with only one routing step each symbol interval. The total ACS-normalize route cycle is therefore about 33 ns for a

throughput of 30 MHz. The speed advantage of the shuffle-exchange layout is even more apparent with larger problem instances as the mesh design requires ever greater unit distance routing steps.

In conclusion, problem instances of ν_0=6 appear to be just large enough for the asymptotic time and area analysis. Above this threshold of ν_0 the results of Table 4.1 would appear to apply. In addition, it appears that current fabrication technology is mature enough to implement problem instances in the range of ν=6 to ν=10, operating at 10 MHz to 50 MHz.

With regard to very large problem instances, Seitz [82] states that in a fully mature MOS technology signal speed on wires can be expected to be 1cm/3ns, or 18ns across a 6 cm wafer. Hence, wafer scale integration [104,129] of the architectures discussed could conceivably operate in the 1 MHz to 10 MHz range.

A currently feasible, three dimensional microelectronic packaging scheme for the VA can be constructed using a stack of silicon wafers, as illustrated in Figure 4.25. This type of layout strategy is appropriate to problem instances of all sizes. Signals are passed vertically through the stack along wire-like data lines composed of feedthroughs (through the wafers), and microbridge interconnects (between wafers).

The microbridge interconnection concept presents very low parasitic impedances for the feedthrough thus permitting the use of very small, low power devices to drive the data and control lines. The overhead in area associated with a microbridge is approximately the same as that of an ordinary bonding pad [130].

The architecture of this scheme is configured such that data flows in a parallel fashion out of the elements of one wafer into all elements of another adjacent wafer. Three elemental wafer types are sufficient to implement the VA.

The first wafer accepts a digitized input signal y_k and uses this to address several small lookup tables (<256 bytes/table) in which appropriate path metrics have been stored. The topology of this wafer is regular and compact, being based on the extremely mature ROM type technology. Alternatively, this wafer may contain appropriate digital correlators and/or arithmetic logic to generate the required path metrics, perhaps like that described by Frenette and Peppard [131].

The second wafer, accepts path metrics generated by the first wafer and calculates appropriate state metrics at each baud interval. The digital hardware residing on this wafer consists of adders, comparators and multiplexers

$\lceil \text{Log } m \rceil$ wires from the comparator output of each processing element are fed to the underlying third wafer to

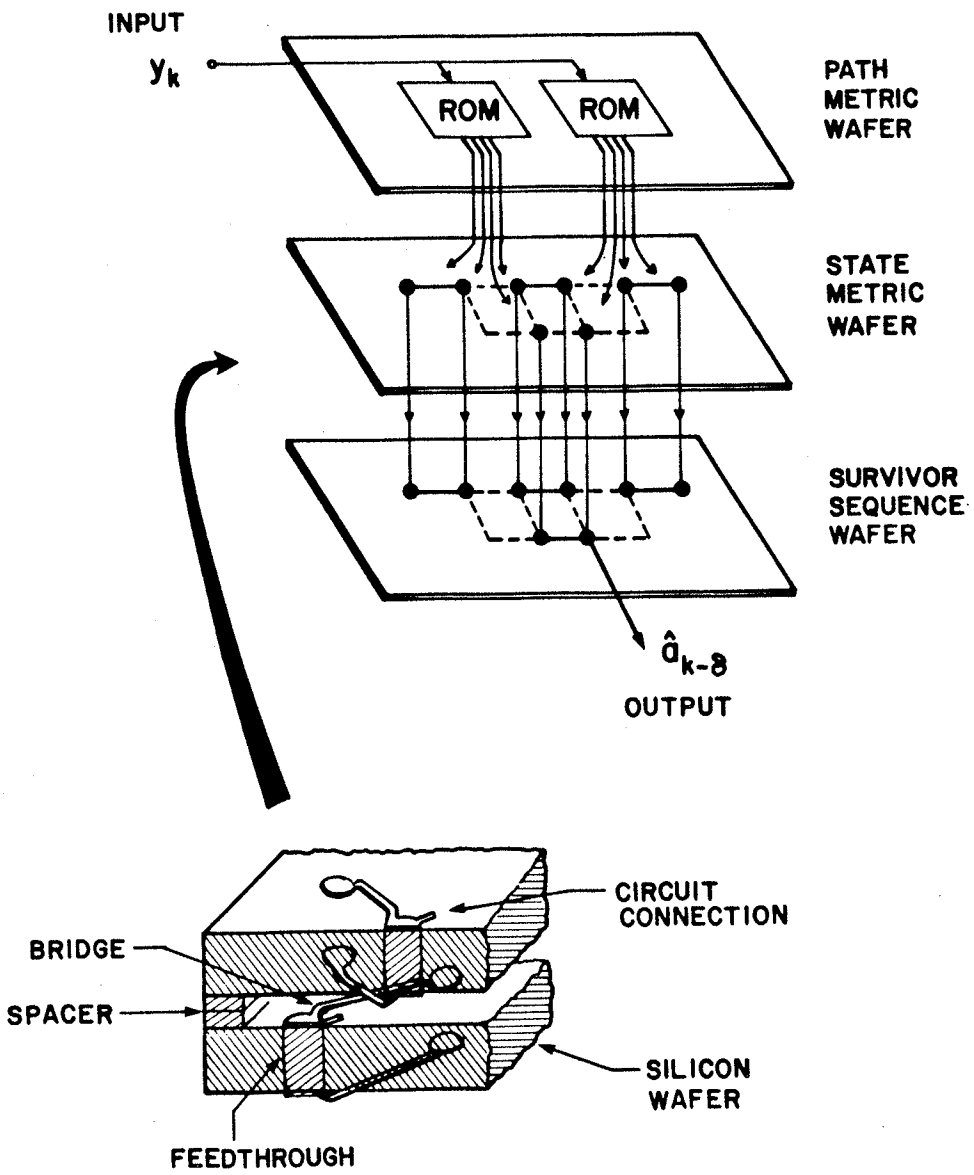


Figure 4.25: 3-D MICROELECTRONIC PACKAGING SCHEME FOR THE VA

Microbridge interconnections are used between wafers as developed by Hughes Research Laboratories [130]. Typical dimensions: 7000 μm x 7000 μm x 2000 μm for the package of three wafers.

indicate the minimum state metric that was selected. This third wafer is then responsible for maintaining updated truncated survivor sequence listings for each state. The digital hardware residing on this wafer consists only of multiplexers and 20 to 60 bits of memory per node (in a typical case). Fixed time lag estimates of the transmitted data (i.e. the output of the VA) can be obtained from the truncated survivor sequence of any state. This wafer could be replaced with a binary tree of comparators to produce a minimum path detector if state metrics were transported from the second wafer.

One important feature to note is that inputs from the outside world interact with only the first wafer; outputs from the device are extracted only from the bottom layer. Of course, clock signals and power would have to be fed to all layers. The topology or arrangement of the m^{ν} processing elements on each of the second and third wafers could be any of those developed in Sections 4.2 or 4.3. As an alternative, on the third wafer, the survivor sequence registers may be consolidated as illustrated in Appendix C.

A significant characteristic of this 3-D embedding of the VA in silicon is its potential for extremely low-cost fabrication. The assembly of a 3-D computer consists of simply stacking wafers on top of each other, where microbridge interconnections between circuits are made simultaneously.

Circuit testing is greatly enhanced with this packaging concept because of the cellular and functional partitioning of the circuit elements. The various wafers, each homogenous in function can be tested independently. The problems of very large state space searches encountered in the testing of unpartitioned VLSI circuitry are thereby eliminated.

Additional economies could be anticipated [132], since avoiding obstacles in a two dimensional environment can require circuitous routing of wires. One would expect average wire length to be shorter, with a subsequent savings in active surface area and power dissipation.

It also appears that optical interconnections for VLSI systems will offer attractive design features over traditional wiring methodologies. Of particular interest, to the high speed implementation (several hundred MHz) of the architectures described in this thesis, is the optical distribution of clock signals and the optical perfect shuffle network described in Goodman et. al. [133].

Chapter V

VLSI STRUCTURES FOR CORRELATIVE ENCODED MSK RECEIVERS

5.1 INTRODUCTION

In addition to decoding convolutional codes, and the demodulation of of intersymbol interference and partial response PAM signals the Viterbi algorithm is applicable to maximum likelihood demodulation of bandwidth efficient continuous phase modulations (CPM).

Within the class of CPM signalling schemes, this chapter presents a VLSI design methodology for synthesizing highly concurrent computing structures which directly implements the Viterbi receiver for Correlative Encoded MSK signals. When the source symbols are correlatively encoded using a first order polynomial, the appropriate Viterbi receiver takes the form of a Cube-Connected Cycles (CCC) Structure, studied in Chapter 4. Second order encoding polynomials give rise to a new type of area efficient VLSI structure which is a generalization of the CCC structure. The results are important from the perspective that simple, practical VLSI layouts are generated, by a structured design methodology, which commercial silicon foundries can fabricate.

Our interest is focussed on the MSK type of CPM scheme because: (i) this technique gives rise to signals with excellent bandwidth efficiency, (ii) the receivers are not very complex as they either require four or eight states to be processed and stored during each symbol interval. The Viterbi receiver specified in this paper can be commercially realized today with dies containing less than 32,000 transistors (excluding synchronization hardware and correlators for path metric generation) with throughputs on the order of 10^7 bits per second.

The presentation, which follows that of McLane [134] closely, is organized into three sections. The first section introduces details of correlative MSK modulation which are relevant to the design. In the second section, we establish that the Viterbi receiver for MSK modulation, using first and second order encoding polynomials, falls within a generalized class of Cube-Connected Cycles processing structures. VLSI grid model layouts are presented for these constructs. The final section summarizes our findings and presents extensions to multi-h phase codes and phase estimation.

5.2 CORRELATIVE ENCODED MSK MODULATION

The mathematical representation of a CPM signal is:

$$x(t) = \mathbf{B} \cos[2\pi f_c t + \phi(t) + \theta] \qquad (5.1)$$

where B is the carrier amplitude, f_c is the carrier frequency, θ is the phase offset, and $\phi(t)$ is the information carrying phase. We assume perfect carrier phase coherence and hence take $\theta = 0$ without loss of generality.

The information carrying phase $\phi(t)$, with modulation index h=0.5, can be written in the following form:

$$\phi(t) = \phi((k-1)T) + \frac{\pi}{2} \cdot d_k \left[\frac{t-(k-1)T}{T}\right], \quad (k-1)T \leq t \leq kT \qquad (5.2)$$

where k is an integer, T is the bit period and d_k the correlative encoded data bit (implicitly rectangularly shaped) for the k^{th} bit interval.

In the simple case of no encoding $d_k = a_k$, where a_k is a source symbol drawn from the finite alphabet [-1,+1]. This is the minimum shift keying (MSK) modulation format which is just continuous phase digital FM with modulation index one-half.

For duobinary MSK, the encoding polynomial is (1+D)/2, and thus $d_k = (a_k + a_{k-1})/2$. The incentive to correlate the data symbols prior to modulation is that duobinary MSK has less phase variation than MSK and consequently has better bandwidth efficiency.

The Viterbi receiver, in this case, is specified by use of the modulation state diagram in Figure 5.1(a). The system states can be divided into two classes, one class (Type A) occupied at odd bit times and the other class (Type B) occupied at even bit times. The two classes are shown in the recursive trellis diagram of Figure 5.1(b). Type A transitions terminate at states 2, 3, 6, 7 while Type B transitions terminate at states 1, 4, 5, 8 in Figure 5.1(b).

For tamed frequency modulation [135], the memory in the modulation is increased by one over that for duobinary MSK, providing additional bandwidth efficiency over duobinary MSK. The encoding polynomial in this case is $(1+D)^2/4$, hence $d_k=(a_k+2a_{k-1}+a_{k-2})/4$. Thus, an eight state Viterbi processor can be derived for the MSK modulation with correlative encoding using the TFM encoding polynomial. The modulation state diagram of Figure 5.2(a) specifies the appropriate Viterbi receiver. Type A transitions terminating in states 2, 4, 6, 8, 10, 12, 14, 16 are occupied at odd bit times. Type B transitions terminating in states 1, 3, 5, 7, 9, 11, 13, 15 are occupied at even bit times. The two classes are shown in the trellis diagram of Figure 5.2(b).

5.3 VLSI REALIZATIONS

In this section we demonstrate that correlative encoded MSK type trellis structures can be implemented in a fully parallel manner on the Cube-Connected Cycles processor interconnection scheme.

For duobinary MSK, the recursive two step trellis diagram of Figure 5.1(b) can be equivalently implemented by the CCC structure of Figure 5.1(c). Cycle connections can be identified as the four vertical loops. Note that the data flow is unidirectional and counterclockwise in each of the loops. Cube connections, illustrated by the horizontal wires, handle bidirectional data. Each node contains an add-compare-select logic circuit for generating state metrics and a survivor sequence register; no more than 2,000 transistors per node are required to implement the required boolean operations. In addition, it is important to realize that the path metric for each state transition is obtained by the correlation function between the received waveform and the expected signal waveform. In duobinary MSK three pairs of correlators as illustrated in Figure 5.3(a) are required for this task. In Figure 5.3(b) the complete duobinary MSK Viterbi receiver floorplan is illustrated. Note that nodes in each cycle "slice" require three unique correlator outputs. Cycle slices can be grouped into two pairs such that three correlators are local to a pair. This is the reasoning behind the rearrangement of cycle slices presented in Figure 5.3(b).

Fixed time lag estimates of the data can be obtained alternately from the truncated survivor sequence of any Type A and Type B processing node. Overflow control of the finite length state metric registers can be achieved by choosing one state metric and subtracting it from all other state metrics of the same type at the appropriate alternating bit period. The carrier and timing signals needed in the receiver structure are obtainable from the technique given by deBuda [136].

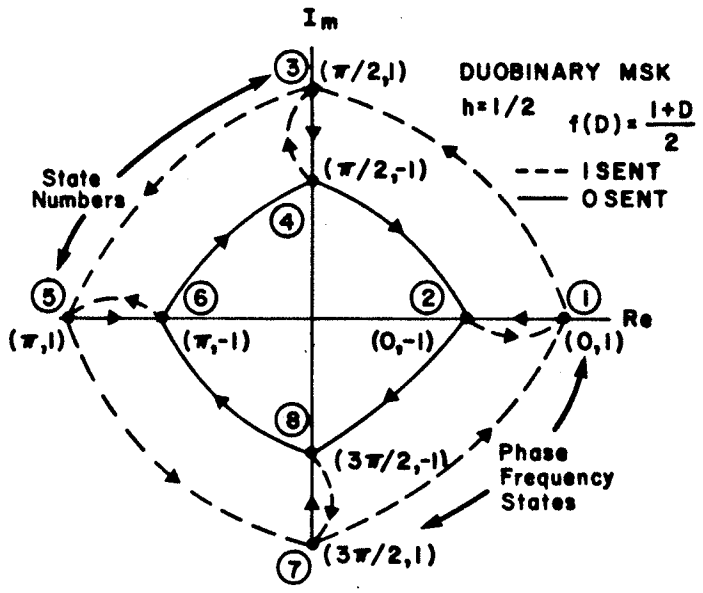
MSK modulation with correlative encoding using the TFM polynomial has a trellis structure which forces us to generalize the CCC structure into a new type of area efficient VLSI structure which we refer to as the "double CCC", or DCCC, shown in Figure 5.4(a). This name is derived from the fact that an implementation of the trellis of Figure 5.2(b) requires double the number of cube connections of a standard CCC, as illustrated in Figure 5.4(b). Note that even though there are four cycles in the embedding of Figure 5.4(a) the direction of the data flow in each of the cycle loops is not the same. One other important difference over duobinary MSK is that five pairs of correlators are required to generate the required path metrics [134].

5.4 DISCUSSION

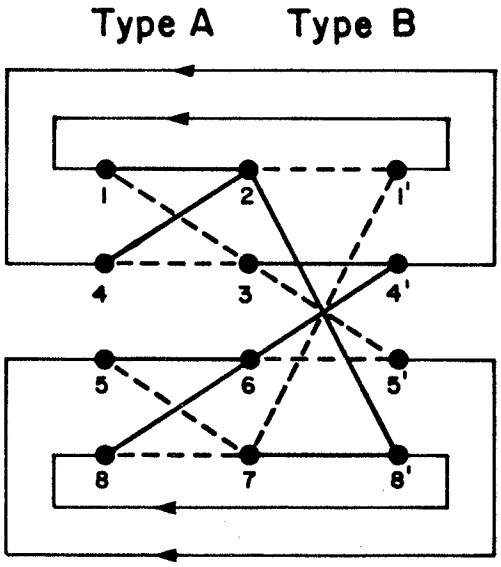
The Viterbi algorithm technique as applied to correlative encoded MSK is just a special case of a dynamic programming solution to modulo-2π phase sequence estimation. The same techniques that were presented in this paper can be extended to develop special types of digital VLSI phase lock loop equivalents [137].

In addition, the VLSI realization of complex trellis structures, generated by multi-h phase codes can be realized by an analogous approach. Figure 5.5 shows the trellis structure and VLSI grid model implementation of a Viterbi receiver for {2/4, 1/4} constraint length 2 phase code [138].

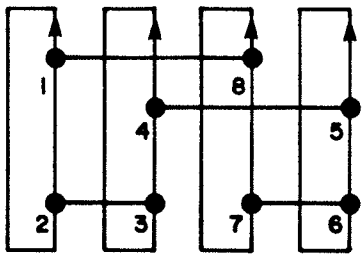
In conclusion, well structured VLSI layout strategies have been identified for realizing Correlative Encoded MSK type Viterbi receivers. When the source symbols are correlatively encoded using a first order polynomial, the appropriate Viterbi receiver takes the form of a Cube-Connected Cycles Structure. Second order encoding polynomials give rise to a new type of area efficient VLSI structure called a DCCC, which is a generalization of the CCC structure.



(a)



(b)



(c)

Figure 5.1: DUOBINARY MSK
(a) Phasor State Diagram (from reference [134])
(b) Recirculating Trellis
(c) Equivalent CCC structure

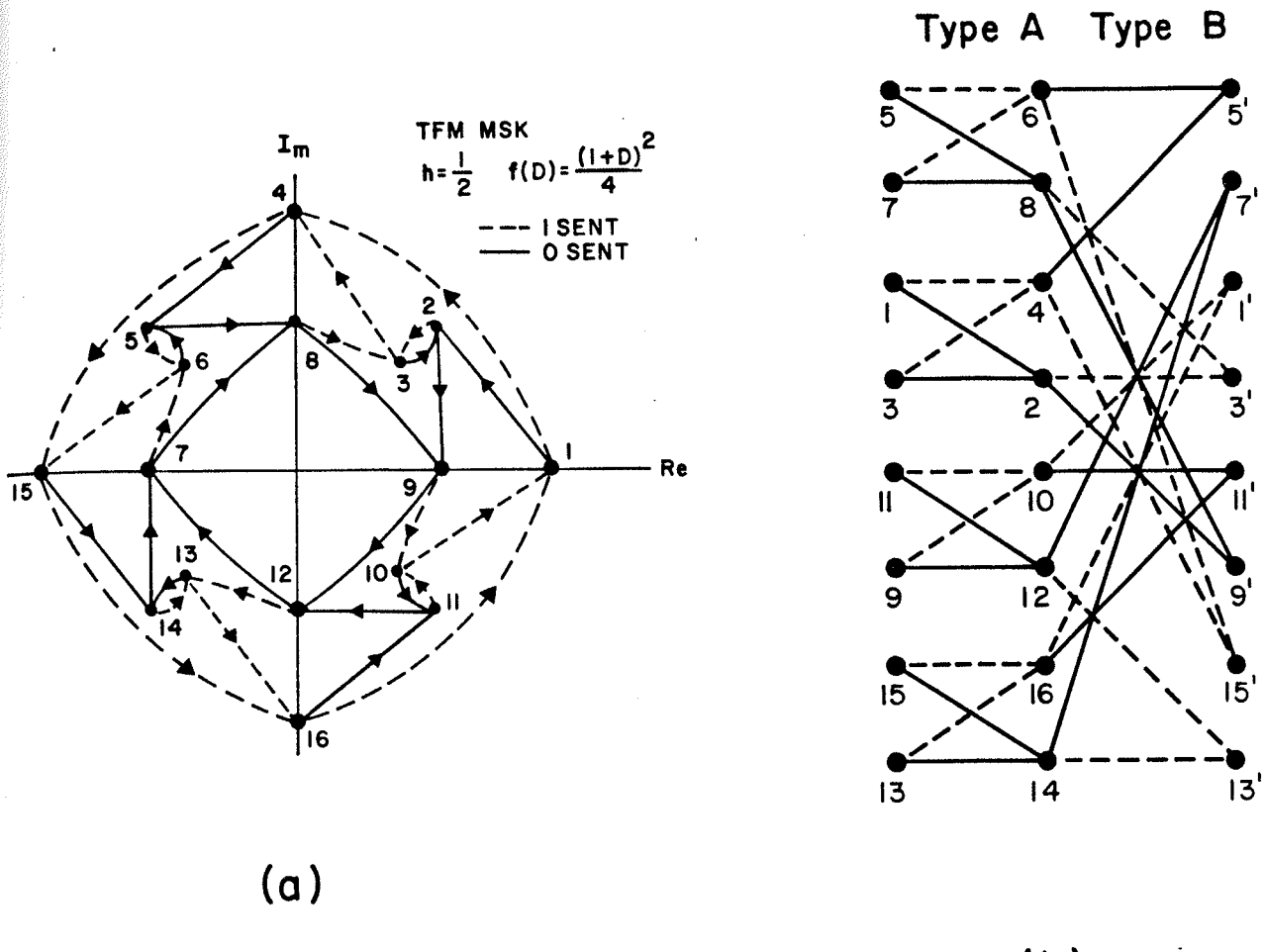
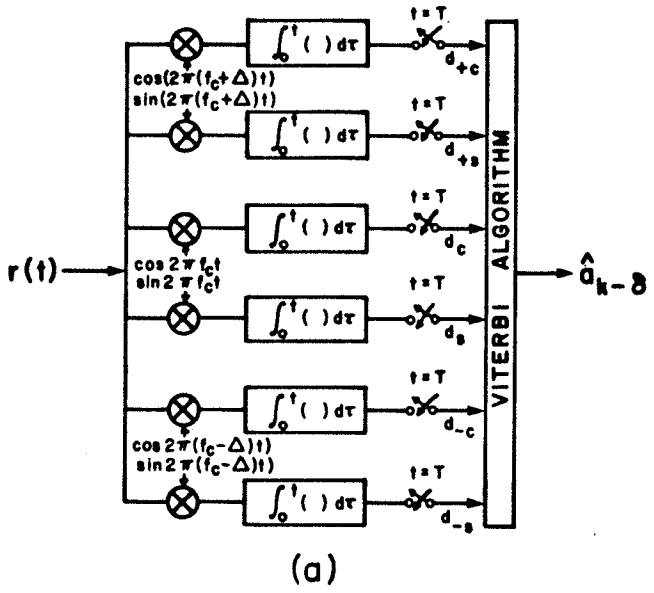


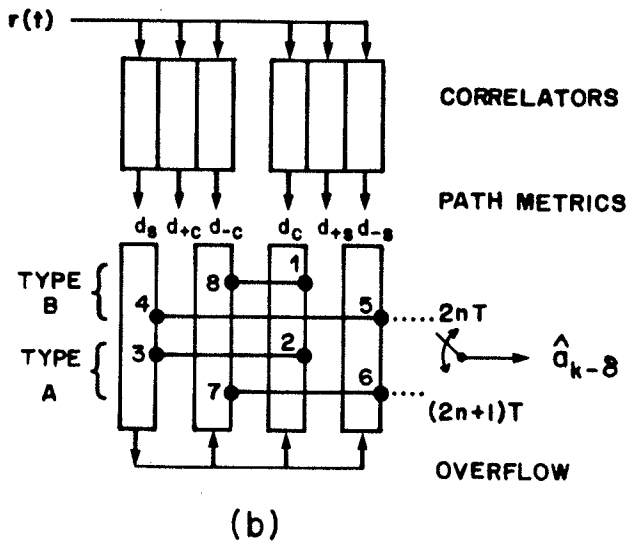
Figure 5.2: TFM MSK

(a) Phasor State Diagram
(Initial phase state = 0 ;
Initial encoder state all 0's or all 1's.)

(b) Recirculating Trellis

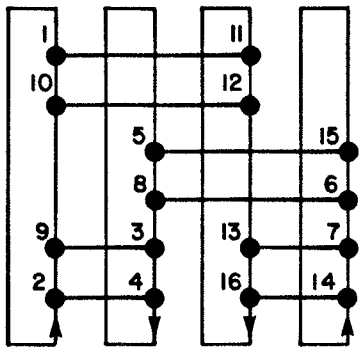


(a)

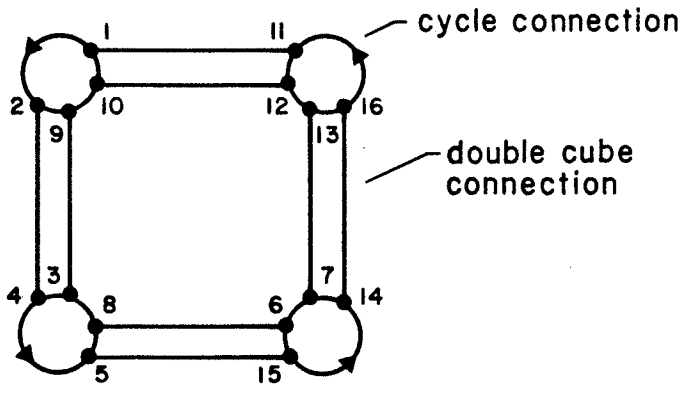


(b)

Figure 5.3: DUOBINARY MSK VITERBI RECEIVER
(a) Generation of Path Metrics ($\Delta = 0.25/T$) (from reference [134])
(b) Floorplan of the Duobinary MSK Receiver (n is integer)

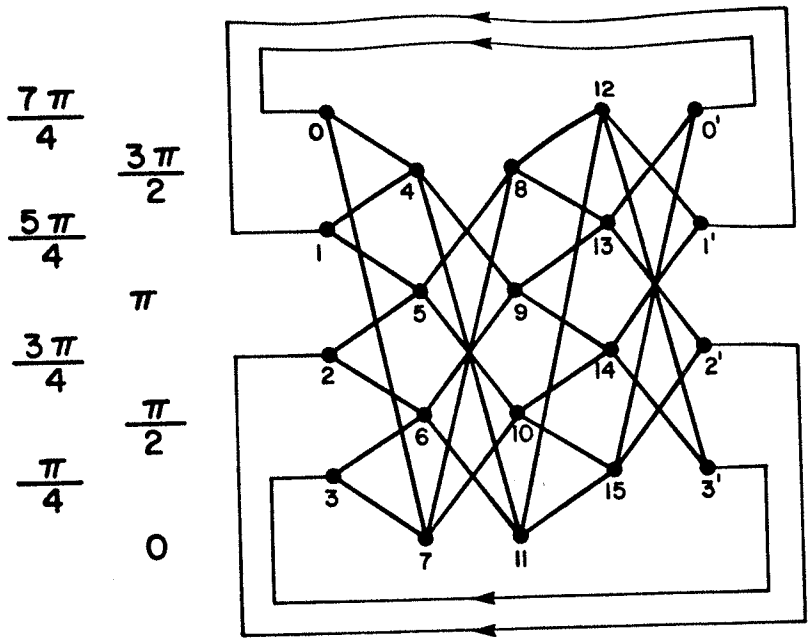


(a)

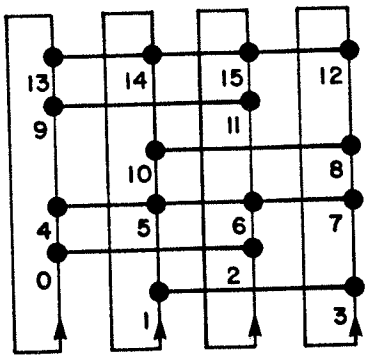


(b)

Figure 5.4: TFM MSK VITERBI RECEIVER
(a) VLSI Grid Model Layout of the DCCC (from Fig. 5.2(b))
(b) Illustration of the Double Cube Connection



(a)



(b)

Figure 5.5: MULTI-H PHASE CODES
(a) Trellis structure of {2/4, 1/4} constraint length 2 phase code
(b) VLSI Grid Model Layout

Chapter VI

CONCLUSIONS AND SUGGESTIONS FOR FURTHER STUDY

6.1 SUMMARY AND CONCLUSIONS

The proliferation of digital information sources and sinks has brought with it a greater need to convey this commodity accurately, rapidly and inexpensively under the constraints of finite bandwidth and finite power. In response to this need, this thesis has investigated an approach to building, in a VLSI format, digital communication receivers based on the Viterbi algorithm. Recently, Ford Aerospace Corporation and Rockwell International have jointly developed a convolutional decoding VLSI circuit, based on the VA, containing 16 ACS processors on a 0.50 cm by 0.73 cm die, using 2 μm CMOS/SOS technology. Though this feat was a "brute force" existence proof that VLSI technology is mature enough to implement small problem instances of the VA on silicon, no general design methodology, until this time, was available or identified to guide future developments.

In addition, there have been persistent efforts in the literature directed at investigating dynamic programming structures in VLSI, for various applications. Since the VA

is a dynamic programming solution to estimating a state sequence this prompts the query, "Is there a relationship between the VA and concurrent computer architectures that are a good fit to VLSI?". The preceding chapters have successfully resolved this question by establishing the nexus between concepts in theoretical computer science and digital communications. This highlights the importance of synergistically tracking the research developments in both disciplines. The major results established in this thesis are summarized below.

6.2 SUMMARY OF MAJOR CONTRIBUTIONS

In the preceding chapters several new concepts have been developed.

1. The concept of a Normalized Kolmogorov Metric Space has been introduced for implementation within the Viterbi algorithm. It has the property that distance in the signal space is bounded. This is a property of interest in a hardware realization that desires register and data paths of minimal width. In addition, the distance measure is parametric in the sense that it is a function of the probability density function of the noise source corrupting the signal. Though the results are preliminary, this metric space may find application in suboptimal soft decision decoding schemes for Gaussian and non-Gaussian noise sources.

2. A taxonomy of VLSI processor architectures are identified for implementing the VA concurrently. These are classified in terms of increasing interprocessor wire area. In order of increasing throughput and wire area: Cascade, Linear and Orthogonally Connected Mesh, Shuffle-Exchange and Cube-Connected Cycles layouts can efficiently embed the VA in silicon. These structures are easily generalized to accommodate arbitrary source alphabet sizes and channel memory lengths. In all cases, good practical layouts are generated by a structured design methodology, which commercial silicon foundries can fabricate.
3. VLSI grid model layouts of the shuffle exchange graph are generalized, for the first time, to include m-shuffle exchange graphs. The structures, in all cases, facilitate efficient chip manufacture and data management.
4. All necklaces in a m-shuffle exchange graph are shown to be full when the algorithm memory length is prime. Cleaving the architecture into full necklaces is a reasonable strategy to use for the integration of the design. This implies that channel memory lengths should always be equalized so that they are a prime number of symbol intervals.
5. VLSI grid model layouts of the Cube-Connected Cycles graph are generalized, for the first time, to accommodate m-ary alphabets while maintaining a vertex

degree of four. The regularity of the layout allowed us to define a standard building block which could realize arbitrarily large memory lengths. It has the potential to decode multiplexed data streams or interleaved convolutional codes for burst noise channels.

6. The CCC structure can be embedded in an H-tree of meshes graph because each is shown to have the same type of separator theorem. A new recursive construct called a Y-tree of meshes was introduced. This structure can efficiently embed the CCC realization of the VA for ternary alphabets.
7. It is demonstrated that the area*time^2 product and the power dissipation of the VA is lower bounded by functions of the alphabet size and algorithm memory length alone. In particular, it was shown, using the VLSI grid model, that if a VLSI design with area A executes the Viterbi algorithm with alphabet m and memory ν in symbol interval T, then $AT^2 > c \cdot m^{2\nu}$, where c is a constant dependent on the technology. Furthermore, any VLSI design of the VA has a power consumption (reciprocal of throughput per unit area) that is lower bounded by $AT = \Omega(m^{3\nu/2})$.
8. Viterbi receivers for correlative encoded MSK are shown to fall within a generalized class of CCC structures.

In summary, the global unifying concept appears to be that many scientific problems can be classified into grid-point problem instances represented as a lattice in space and time. (Wires provide interconnection in space; memories provide interconnection in time.) This lattice structure often provides for a natural concurrent decomposition in showing how to orchestrate a single computation so that it can be distributed across an ensemble of processors.

6.3 SUGGESTIONS FOR FUTURE RESEARCH

Now that several feasible strategies have been unveiled for implementing MLSE in VLSI, there is a basis upon which to ask many questions and propose variations to the architectures. Some of these are now put forth as areas for further investigation.

1. A study should be carried out to establish the criterion that is being optimized when selecting the shortest path length through the trellis as measured in a Normalized Kolmogorov Metric Space. Are there certain types of non-Gaussian noise processes where this criterion is advantageous? Do quantized distance measures that saturate, either through design or through implementation, have to be optimized?
2. The following evolution towards the commercial exploitation of the concepts presented in this thesis

is envisaged. Initially, a linearly connected processor slice, a CCC building block of Figure 4.18 and a 2-SE necklace (remember, make ν prime) should be realized as a minimum path receiver, using hard decision decoding. The lack of both complex path metric generation and survivor sequences in such a structure will eliminate some design issues and highlight throughput bottlenecks. The ability of these layout slices to handle soft decision decoding and survivor sequences should then be integrated into the design, realizing the VA in its complete form. If convolutional decoders are of interest, puncturing [139] will provide hardware simplification. Wafer scale integration technology should then be used to realize "complete" systems on silicon. The development of an appropriate silicon assembler (or silicon compiler) would help expedite this implementation strategy.

3. Instead of handling and storing the entire survivor sequence list at each processing node for each symbol interval, interpreting the survivor sequence list as a sequence of pointers, as suggested in [72], may remove the necessity to transport the entire survivor sequence field to successive nodes. Perhaps a suboptimal scheme can be contrived that contains only one survivor sequence list for each necklace in a SE layout or for each cycle in a CCC layout. This would

reduce the memory devoted to storing survivor sequences by a factor of ν. (Approximately an order of magnitude reduction in memory, not to mention the associated reduction in wire area, power and/or transport time, in most cases of interest.) The performance of such a system would definitely be better than a minimum path detector which, by definition, contains no survivor sequences at all. Computer simulations for channels of interest would establish the magnitude of degradation, however analytic bounds on the performance of such a system may be derivable.

4. It is well known that at least $\Omega(2^{2\nu}/\nu^2)$ units of area are required to embed a 2-SE graph in the plane under the VLSI grid model assumptions. Since the nodes themselves only take up $O(2^{\nu})$ area, the average edge length in a planar embedding of the 2-SE graph is $\Omega(2^{\nu}/\nu^2)$. Though this thesis has developed good compact layouts for small SE and CCC graphs, those precisely of interest in practical applications, it would be insightful to prove that asymptotically a m-SE graph (and the corresponding CCC graph for that matter) requires at least $\Omega(m^{2\nu}/\nu^2)$ units of area.
5. The power of recursive structures is their concise expression. Are there other recursive structures, conceived in the shadow of the tree of meshes, that are appropriate for m-ary versions of the VA? One

construct worth exploring immediately is that of the Pleated Cube-Connected Cycles (PCCC), proposed in [140].

6. The work by Moldovan [141] should be investigated to determine if the mathematical techniques introduced could be successfully applied to embedding the Viterbi algorithm in other types of structures.
7. Decoders for other classes of codes defined by their finite state machine representation (which implies specific trellis structures) should be analysed for their AT^x performance. Novel computing structures for their realization should be pursued.
8. The $(volume)^x*(time)^y$ lower bound should be established for 3-dimensional embeddings of the VA.
9. The AT^2 lower bound was established for the VA, does this apply to any algorithm for MLSE, in general?

6.4 CONCLUDING REMARKS

This thesis has shown that there is promise in building VLSI devices that can implement the Viterbi algorithm for important digital communication tasks. These types of systems will provide great economies and performance. The success of such systems will depend on whether appropriate development costs are expended.

The challenge remains, as always, to find VLSI implementable algorithms for other important digital communication tasks.

Appendix A

PROOF THAT THE NORMALIZED KOLMOGOROV DISTANCE IS A METRIC

PRELIMINARY: A metric space M is a set X in which we can talk sensibly about the distance ρ between any two elements in X. For M to qualify as a metric space, the distance function $\rho(x,y)$ must satisfy the three metric axioms:

(i) $\rho(x,y) \geq 0$ AND $\rho(x,y) = 0$ IFF $x = y$

(ii) $\rho(x,y) = \rho(y,x)$

(iii) $\rho(x,z) \leq \rho(x,y) + \rho(y,z) \quad \forall\, x,y,z \in X$

NOTATION: In Chapter 2 a stylized notation was utilized to represent the two elements between which the distance is to be measured. Note, that since $\mu_1 < \mu_2$, in all instances:

$$\rho_{H_1}(\psi) = \rho(-\infty, \psi) = \rho(x,y)$$

$$\rho_{H_2}(\psi) = \rho(\psi, +\infty) = \rho(x,y)$$

In general, $\rho_{H_1}(\psi)$ indicates that the distance ρ is to be measured between the received signal ψ and the appropriate corresponding supremum or infimum of Hypothesis H_i.

THEOREM: The space $M(X,\rho)$ defined by:

the set X = {stationary random variables $\in \rho(\xi|H_1), \rho(\xi|H_2)$}

and the distance measure,

$$\rho(x,y) = \frac{\left|\int_x^y \max[p(\xi|H_1)P_1, p(\xi|H_2)P_2]d\xi - \int_x^y |p(\xi|H_1)P_1 - p(\xi|H_2)P_2|d\xi\right|}{\frac{1}{2}\cdot\left[1 - \int_{-\infty}^{+\infty} |p(\xi|H_1)P_1 - p(\xi|H_2)P_2|d\xi\right]}$$

constitutes a valid metric space, given that x, y $\in$ X

Note: The elements x and y only determine limits of integration.

PROOF: (i) The denominator of $\rho(x,y)$ is an intersection of two conditional probability densities $\rho(\xi|H_1)$ and $\rho(\xi|H_2)$ of ξ under the condition of H_1 and H_2. Consequently,

$$0 \leq \text{Num}(x,y) \leq \text{Den}(-\infty, +\infty) \leq 1 \quad \text{AS} \quad [x,y] \subseteq [-\infty, +\infty]$$

$$\Rightarrow \quad 0 \leq \frac{\text{Num}(x,y)}{\text{Den}(-\infty, +\infty)} \leq 1$$

$$\Rightarrow \quad 0 \leq \rho(x,y) \leq 1$$

If x=y then Num(x,y)=0. Therefore, the $\rho(x,y)=0$ condition is satisfied.

If the noise is hardlimited such that $\sup(x)=a_2$ and $\inf(x)=a_1$, then in order to enforce the $\rho(x,y)=0$ iff x=y condition we must enforce the following constraint:

$$[x,y] \subset [a_1, a_2]$$

OR $$[x,y] \subseteq (a_1, a_2)$$

■

(ii) Consider the numerator of $\rho(x,y)$.

$$\text{Num}(x,y) = \left| \int_x^y f(\xi)d\xi - \int_x^y g(\xi)d\xi \right|$$

$$\text{Num}(y,x) = \left| \int_y^x f(\xi)d\xi - \int_y^x g(\xi)d\xi \right|$$

$$= \left| -1 \cdot \left(\int_x^y f(\xi)d\xi - \int_x^y g(\xi)d\xi \right) \right|$$

$$= \left| \int_x^y f(\xi)d\xi - \int_x^y g(\xi)d\xi \right|$$

$$= \text{Num}(x,y)$$

The denominator of ρ is equivalent in each case.

Consequently,

$$\rho(x,y) = \rho(y,x)$$

■

(iii) Let x,y and z be three sample values from the sample space of the conditional densities $\rho(\xi|H_1), \rho(\xi|H_2)$.

$$\rho(x,y) + \rho(y,z) > \rho(x,z) \quad \text{where } \rho(x,y) = \frac{\text{Num}(x,y)}{\text{Den}(-\infty, +\infty)}$$

$$\Longleftrightarrow \text{Num}(x,y) + \text{Num}(y,z) > \text{Num}(x,z) \qquad (A.1)$$

Since Num(x,y) is the overlapped part of $\rho(\xi|H_1)$ and $\rho(\xi|H_2)$ between x and y.

Let f(x) be the functional representation of the overlapped part of the two probability densities. Clearly, $f(x) \geq 0$, $\forall$ x.

Now (A.1) may be equivalently expressed as:

$$\int_x^y f(\xi)d\xi + \int_y^z f(\xi)d\xi \geq \int_x^z f(\xi)d\xi$$

Since this is clearly a property of definite integrals the triangle inequality holds. ■

Q.E.D.

Appendix B

PROOF THAT G_t IS CONTRACTIBLE TO G_s

THEOREM: There is a sequence of elementary contractions that will map the recursive, bipartite (shuffle-exchange) graph G_t of cardinality $N=2^n$, $G_t(N)$, into the recursive, bipartite (shuffle) graph G_s of cardinality $N/2=2^{n-1}$, $G_s(N/2)$, where $G_t(N)$ and $G_s(N/2)$ is defined by the following relation of incidence that associates with each edge a pair of vertices X and X' :

$$G_t(N): \quad X_t \rightarrow X'_t \quad \forall \quad X_t, \qquad x_0, x_1, \ldots, x_{n-1} \quad \varepsilon\{0, 1\}$$

$$X_t = x_{n-1}, x_{n-2}, \ldots, x_1, x_0$$

$$X'_t = x_{n-2}, \ldots, x_0, x_{n-1} \cup x_{n-1}, \ldots, \bar{x}_0$$

$$G_s(N/2): \quad X_s \rightarrow X'_s \quad \forall \quad X_s, \qquad x_0, x_1, \ldots, x_{n-2} \quad \varepsilon\{0, 1\}$$

$$X_s = x_{n-2}, x_{n-3}, \ldots, x_1, x_0$$

$$X'_s = x_{n-3}, \ldots, x_0, x_{n-2} \cup x_{n-3}, \ldots, x_0, \overline{x_{n-2}}$$

DEFINITION:

1. The IMAGE of the node $x_{n-1}, \ldots, x_0$ in X is the corresponding node $x_{n-1}, \ldots, x_0$ in X' .
2. The CARDINALITY of graph G is defined to be the number of elements in its vertex subset X (or X').

OBSERVATION: The edge defined by f: $x_{n-1}, \ldots, x_0 \rightarrow x_{n-1}, \ldots, x_0$ is implicit. Each node in X is connected to its image X' since G_t and G_s are recursive. (i.e., A node can always talk to its image at the next clock tick.) In fact, G can be drawn such that X is coincident with X' .

PROOF:

Step 1: Reduce the cardinality (merge nodes) through an elementary contraction of G_t by applying the following rule:

$$g: \begin{bmatrix} x_{n-1}, \cdots, x_0 \\ x_{n-1}, \cdots, \bar{x}_0 \end{bmatrix} \rightarrow x_{n-1}, \cdots, x_1 \quad \forall X_t$$

This implies that exchange edges occur between $x_{n-1}, x_{n-2}, \cdots, x_1$ and its image.

Note: Henceforth, a node in G_t will be referred to by the label $x_{n-1}, \cdots, x_1$ or by the pair of labels $\begin{bmatrix} x_{n-1}, \cdots, x_0 \\ x_{n-1}, \cdots, \bar{x}_0 \end{bmatrix}$ of which it was previously comprised.

Step 2: As a consequence of STEP 1 the rule:

$$h: x_{n-1}, x_{n-2}, \cdots, x_0 \rightarrow x_{n-2}, \cdots, x_0, x_{n-1}$$

bifurcates. We now have two rules (two shuffle edges) for each node. In order to establish these rules, consider the following node in X_t:

$$\begin{bmatrix} x_{n-1}, \cdots, x_0 \\ x_{n-1}, \cdots, \bar{x}_0 \end{bmatrix}$$

Since each $x_0, \cdots, x_{n-1} \in \{0, 1\}$ one element of the above pair will have either $x_{n-1} = x_0$ or $x_{n-1} = \overline{x_0}$.

Apply $h: x_{n-1}, x_{n-2}, \cdots, x_0 \rightarrow x_{n-2}, \cdots, x_0, x_{n-1}$ to each node to get:

$$x_{n-2}, \cdots, \bar{x}_0, x_{n-1}$$

$$x_{n-2}, \cdots, x_0, x_{n-1}$$

Assume that $x_0 = x_{n-1}$

(the same derivation applies if we assume $x_0 = \overline{x_{n-1}}$)

These nodes can be rewritten as:

$$x_{n-2}, \cdots, x_1, \overline{x_{n-1}}, x_{n-1}$$

$$x_{n-2}, \cdots, x_1, x_{n-1}, x_{n-1}$$

By the transformation g defined in STEP 1 we know these portions of the corresponding node pair belong to the following mapping:

$$\begin{bmatrix} x_{n-2}, \cdots, x_1, \overline{x_{n-1}}, \overline{x_{n-1}} \\ x_{n-2}, \cdots, x_1, \overline{x_{n-1}}, x_{n-1} \end{bmatrix} \rightarrow x_{n-2}, \cdots, x_1, \overline{x_{n-1}}$$

$$\begin{bmatrix} x_{n-2}, \cdots, x_1, x_{n-1}, \overline{x_{n-1}} \\ x_{n-2}, \cdots, x_1, x_{n-1}, x_{n-1} \end{bmatrix} \rightarrow x_{n-2}, \cdots, x_1, x_{n-1}$$

which was mapped from the vertex $x_{n-1}, \cdots, x_1$ in X.

i.e.:

$$x_{n-1}, \cdots, x_1 \rightarrow x_{n-2}, \cdots, x_1, \overline{x_{n-1}} \cup x_{n-2}, \cdots, x_1, x_{n-1}$$

STEP 3: Subtract one (1) from each indices

$$x_{n-2}, \cdots, x_0 \rightarrow x_{n-3}, \cdots, x_0, x_{n-2} \cup x_{n-3}, \cdots, x_0, \overline{x_{n-2}}$$

These are precisely the rules which define G_s.

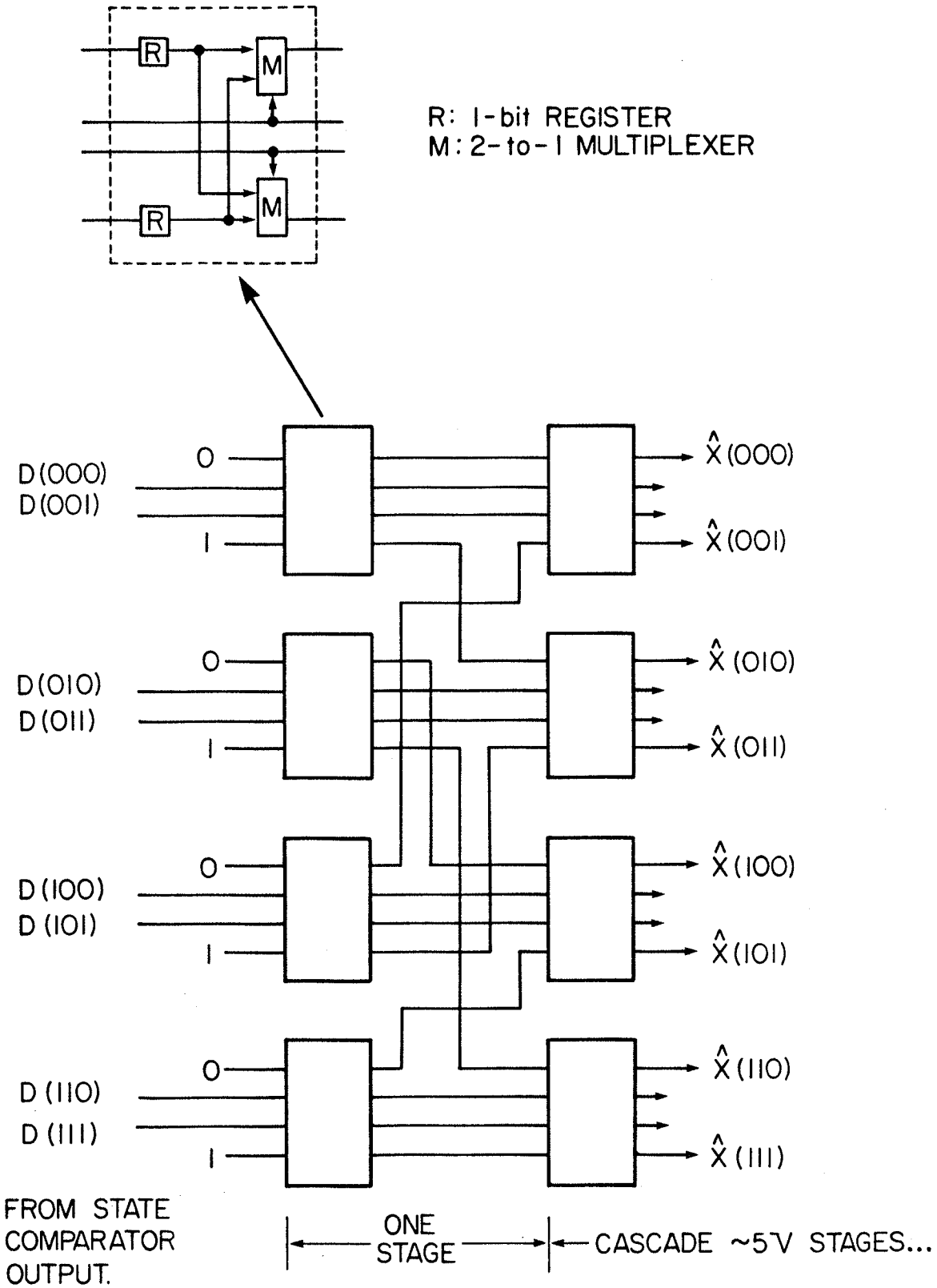
Hence, $G_t(N) = G_s(N/2)$

Q.E.D.

Appendix C

CONSOLIDATED SURVIVOR SEQUENCE MEMORY LAYOUT

FOR P(2) AND $\nu = 3$


R
M
R
M
R: 1-bit REGISTER
M: 2-to-1 MULTIPLEXER
D(000)
D(001)
D(010)
D(011)
D(100)
D(101)
D(110)
D(111)
0
1
$\hat{X}$(000)
$\hat{X}$(001)
$\hat{X}$(010)
$\hat{X}$(011)
$\hat{X}$(100)
$\hat{X}$(101)
$\hat{X}$(110)
$\hat{X}$(111)
FROM STATE COMPARATOR OUTPUT.
ONE STAGE
CASCADE ~5V STAGES...

Appendix D

VITERBI SIMULATION SOFTWARE

```
10.  //GULAK JOB ',,,T=30,C=0','GLEN'
20.  // EXEC WATFIV,SIZE=256K
30.  //GO.SYSIN DD *
40.  $JOB  WATFIV   GLENN,NOEXT,NOWARN
50.        IMPLICIT INTEGER(A-Y)
60.        DOUBLE PRECISION ZSEEDI, ZSEEDO, ZSEED, ZSEEDL
70.        REAL GGUBFS, Q, GAIN, DEG, RAD
80.        INTEGER APM(256),BPM(256),CPM(256),DPM(256),
90.       &        EPM(256),FPM(256),GPM(256),HPM(256)
100.       INTEGER ABPNEW(8),CDPNEW(8),EFPNEW(8),GHPNEW(8),
110.      &        ABPOLD(8),CDPOLD(8),EFPOLD(8),GHPOLD(8)
120.       INTEGER  REFDO(8), REFDN(8), DATA(3), MINPO(8), MINPN(8)
130.       INTEGER QUANN(8), QUANO(8)
140.       INTEGER IT,IOP(3),IER,I3(1),I4,I5
150.       REAL TBL(27,5),P1(1),P2(1),R(1),RNOISE
160.       EXTERNAL SUBRF
170.       DIMENSION  ZCHAN(3)
180.  C
190.       DATA MINPO/8*0/, MINPN/8*0/, QUANN/8*0/, QUANO/8*0/
200.       DATA REFDO/8*0/, REFDN/8*0/, DATA/3*0/
210.       DATA ABPOLD/8*0/,CDPOLD/8*0/,EFPOLD/8*0/,GHPOLD/8*0/
220.  C
230.  C-------------------------------------------------------
240.  C
250.  C       SIMULATION OF AN LOG2(BUSW) -BIT WIDE VITERBI RECEIVER
260.  C       FOR A CHANNEL MEMORY OF 2  AND PATH MEMORY OF 8
270.  C
280.  C       ELECTRICAL ENGINEERING DEPARTMENT
290.  C       UNIVERSITY OF MANITOBA
300.  C       WINNIPEG, MANITOBA  R3T 2N2
310.  C
320.  C
330.  C              VERSION 2.2
340.  C
350.  C            (C)  P.G. GULAK
360.  C
370.  C                MARCH 1983
380.  C
390.  C       NOTE: IMSL SUBROUTINES REQUIRED: GGVCR, GGUBFS
400.  C
410.  C-------------------------------------------------------
420.  C
430.  C
440.       IOP(1) = 23
450.       IOP(2) = 1
460.       IOP(3) = -1
470.  C
480.       I4=IOP(1)+1
490.       I5=27
500.       IT=27
510.       ZSEEDL=12457.0D0
520.  C
530.  C    SET PARAMETERS
540.  C
550.       ZSEEDI = 1999.0D0
560.       ZSEED = 123457.0D0
570.       TOUT = 10054
580.       TSTOP = 10064
590.       BUSW = 256
600.       SUPPRESS = 0
610.       ZSNR = 10.00
620.       ZSTDEV = 1.25/(10 ** (ZSNR/20.0))
630.       ZB = SQRT (3.00) * ZSTDEV
640.  C
```

```
      P1(1) = ZSTDEV
C
      GAIN = 1.000
      DEG = 0.00
      RAD = (3.14159265/180.0) * DEG
C
C
C     INITIALIZE
C
      BITS = 0
      ERRCNT = 0
      MINPER = 0
      QUANER = 0
      ABSMO=0
      CDSMO=0
      EFSMO=0
      GHSMO=0
C
C     SET CHANNEL RESPONSE
C
      ZCHAN(1) = 1.00000
      ZCHAN(2) = 0.20788
      ZCHAN(3) = 0.04321
C
C
C     SET EXPECTED REFERENCE VALUES FOR  F0/R0  VALUE
C
      AREF=(BUSW/2)*(1.0 + (ZCHAN(1)+ZCHAN(2)+ZCHAN(3))/5.0)
      BREF=(BUSW/2)*(1.0 + (ZCHAN(1)+ZCHAN(2)-ZCHAN(3))/5.0)
      CREF=(BUSW/2)*(1.0 + (-ZCHAN(1)+ZCHAN(2)+ZCHAN(3))/5.0)
      DREF=(BUSW/2)*(1.0 + (-ZCHAN(1)+ZCHAN(2)-ZCHAN(3))/5.0)
      EREF=(BUSW/2)*(1.0 + (ZCHAN(1)-ZCHAN(2)+ZCHAN(3))/5.0)
      FREF=(BUSW/2)*(1.0 + (ZCHAN(1)-ZCHAN(2)-ZCHAN(3))/5.0)
      GREF=(BUSW/2)*(1.0 + (-ZCHAN(1)-ZCHAN(2)+ZCHAN(3))/5.0)
      HREF=(BUSW/2)*(1.0 + (-ZCHAN(1)-ZCHAN(2)-ZCHAN(3))/5.0)
      PRINT 59, AREF,BREF,CREF,DREF,EREF,FREF,GREF,HREF
   59 FORMAT ( '1', 10X, 'EXPECTED ISI REF VALUES ', /,
     &          8(5X,I5), /// )
C
C     GENERATE METRIC SPACE
C
      THRES = BUSW / 7
      SEP = AREF - HREF
      ZSEP = FLOAT ( SEP )
      HALF = SEP / 2
      ZHALF = FLOAT ( HALF )
      ZSIGAD = ( ZSTDEV * ( BUSW / 10.0 ))
      ZSCALE = 0.5 / Q( ZHALF / ZSIGAD )
C
      DO 88 I=1,BUSW
      DIF = IABS( I - AREF )
      ZDIF = FLOAT ( DIF )
      ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
      ZXX = ZDIF/(ZSIGAD)
      IF ( DIF .LE. HALF )
     &     THEN
               APM(I) = Q(ZX) * THRES * ZSCALE
           ELSE
               APM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
        ENDIF
      DIF = IABS( I - BREF )
      ZDIF = FLOAT ( DIF )
      ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
      ZXX = ZDIF/(ZSIGAD)
```

```
1290.        IF ( DIF .LE. HALF )
1300.       &     THEN
1310.                 BPM(I) = Q(ZX) * THRES * ZSCALE
1320.             ELSE
1330.                 BPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1340.         ENDIF
1350.        DIF = IABS( I - CREF )
1360.        ZDIF = FLOAT ( DIF )
1370.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1380.        ZXX = ZDIF/(ZSIGAD)
1390.        IF ( DIF .LE. HALF )
1400.       &     THEN
1410.                 CPM(I) = Q(ZX) * THRES * ZSCALE
1420.             ELSE
1430.                 CPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1440.         ENDIF
1450.        DIF = IABS( I - DREF )
1460.        ZDIF = FLOAT ( DIF )
1470.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1480.        ZXX = ZDIF/(ZSIGAD)
1490.        IF ( DIF .LE. HALF )
1500.       &     THEN
1510.                 DPM(I) = Q(ZX) * THRES * ZSCALE
1520.             ELSE
1530.                 DPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1540.         ENDIF
1550.        DIF = IABS( I - EREF )
1560.        ZDIF = FLOAT ( DIF )
1570.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1580.        ZXX = ZDIF/(ZSIGAD)
1590.        IF ( DIF .LE. HALF )
1600.       &     THEN
1610.                 EPM(I) = Q(ZX) * THRES * ZSCALE
1620.             ELSE
1630.                 EPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1640.         ENDIF
1650.        DIF = IABS( I - FREF )
1660.        ZDIF = FLOAT ( DIF )
1670.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1680.        ZXX = ZDIF/(ZSIGAD)
1690.        IF ( DIF .LE. HALF )
1700.       &     THEN
1710.                 FPM(I) = Q(ZX) * THRES * ZSCALE
1720.             ELSE
1730.                 FPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1740.         ENDIF
1750.        DIF = IABS( I - GREF )
1760.        ZDIF = FLOAT ( DIF )
1770.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1780.        ZXX = ZDIF/(ZSIGAD)
1790.        IF ( DIF .LE. HALF )
1800.       &     THEN
1810.                 GPM(I) = Q(ZX) * THRES * ZSCALE
1820.             ELSE
1830.                 GPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1840.         ENDIF
1850.        DIF = IABS( I - HREF )
1860.        ZDIF = FLOAT ( DIF )
1870.        ZX = ( ZSEP - ZDIF ) / (   ZSIGAD )
1880.        ZXX = ZDIF/(ZSIGAD)
1890.        IF ( DIF .LE. HALF )
1900.       &     THEN
1910.                 HPM(I) = Q(ZX) * THRES * ZSCALE
1920.             ELSE
```

```
1930.                   HPM(I) = (1.0/ZSCALE - Q(ZXX)) * THRES * ZSCALE
1940.            ENDIF
1950.  C        IF ( APM(I) .GT. THRES ) APM(I)=THRES
1960.  C        IF ( BPM(I) .GT. THRES ) BPM(I)=THRES
1970.  C        IF ( CPM(I) .GT. THRES ) CPM(I)=THRES
1980.  C        IF ( DPM(I) .GT. THRES ) DPM(I)=THRES
1990.  C        IF ( EPM(I) .GT. THRES ) EPM(I)=THRES
2000.  C        IF ( FPM(I) .GT. THRES ) FPM(I)=THRES
2010.  C        IF ( GPM(I) .GT. THRES ) GPM(I)=THRES
2020.  C        IF ( HPM(I) .GT. THRES ) HPM(I)=THRES
2030.     88    CONTINUE
2040.           PRINT , CPM
2050.  C
2060.  C-------------------------------------------------------
2070.  C
2080.  C        A    00-00 STATE TRANSITION
2090.  C        B    00-01 STATE TRANSITION
2100.  C        C    10-00 STATE TRANSITION
2110.  C        D    10-01 STATE TRANSITION
2120.  C        E    01-10 STATE TRANSITION
2130.  C        F    01-11 STATE TRANSITION
2140.  C        G    11-10 STATE TRANSITION
2150.  C        H    11-11 STATE TRANSITION
2160.  C
2170.  C        AB   00 STATE
2180.  C        CD   10 STATE
2190.  C        EF   01 STATE
2200.  C        GH   11 STATE
2210.  C
2220.  C        SM   STATE METRIC
2230.  C        PM   PATH METRIC
2240.  C
2250.  C        P    PATH ( STATE TRAJECTORY )
2260.  C
2270.  C        N    NEW
2280.  C        O    OLD
2290.  C
2300.  C
2310.  C-------------------------------------------------------------
2320.  C
2330.  C
2340.           DO 99 ITIME = 1,TSTOP
2350.  C
2360.  C              GENERATE NOISE SAMPLE
2370.  C ----- LAPLACIAN
2380.           CALL GGVCR(SUBRF,TBL,P1,P2,I3,I4,IT,IT,IOP,ZSEEDL,R,IER)
2390.           ZNOISE = R(1)
2400.  C
2410.  C ----- GAUSSIAN
2420.  C        CALL GAUSS( ZSTDEV, 0.0, ZNOISE, ZSEEDI, ZSEEDO )
2430.  C        ZSEEDI = ZSEEDO
2440.  C
2450.  C ----- UNIFORM
2460.  C        ZR = GGUBFS ( ZSEEDI )
2470.  C        ZNOISE = ( 2.0 * ZR - 1.0 ) * ZB
2480.  C
2490.  C              GENERATE PSEUDO RANDOM DATA
2500.  C
2510.           ZDATA = GGUBFS ( ZSEED )
2520.           IF ( ZDATA .LE. 0.5 )
2530.          &      THEN DO
2540.                       DATA(1) = -1
2550.                  ELSE DO
2560.                       DATA(1) = 1
```

```
2570.         ENDIF
2580.   C
2590.         RBIN = DATA(1)
2600.         IF ( RBIN .EQ. -1 ) RBIN = 0
2610.         CALL SHIFT ( REFDO, REFDN, RBIN, RBOUT )
2620.   C
2630.   C           GENERATE CHANNEL RESPONSE FOR PSEUDO-RANDOM DATA
2640.   C
2650.         ZVOLTS = DATA(1)*ZCHAN(1)+DATA(2)*ZCHAN(2)+DATA(3)*ZCHAN(3)
2660.         ZVOLTS = GAIN * ((ZVOLTS * COS(RAD)) + ZNOISE)
2670.         IF ( ZVOLTS .GT. 5.00) ZVOLTS = 5.00
2680.         IF ( ZVOLTS .LT. -5.00) ZVOLTS = -5.00
2690.   C
2700.         DATA(3) = DATA(2)
2710.         DATA(2) = DATA(1)
2720.         DO 9 I=1,8
2730.      9  REFDN(I) = REFDO(I)
2740.   C
2750.   C        DELAY QUANTIZER DECISION FOR LATER COMPARISON
2760.   C
2770.         QUANT = 1
2780.         IF ( ZVOLTS .LE. 0.0 ) QUANT = 0
2790.         CALL SHIFT ( QUANO, QUANN, QUANT, QBOUT )
2800.         DO 12 I=1,8
2810.     12  QUANN(I) = QUANO(I)
2820.   C
2830.   C    A/D CONVERT AND USE BINARY WORD AS AN INDEX TO METRIC SPACE
2840.   C
2850.         IDATA =  ( BUSW/2 ) - IFIX ( ZVOLTS * BUSW/10.0 )
2860.         IF (IDATA .EQ. 0) IDATA = 1
2870.   C
2880.   C
2890.   C
2900.   C-----------------------------------------------------------------
2910.   C        START FIRST BUTTERFLY
2920.   C
2930.   C
2940.         LTEMP = ABSMO + APM(IDATA)
2950.         LTEMP = MOD(LTEMP,BUSW)
2960.         RTEMP = EFSMO + BPM(IDATA)
2970.         RTEMP = MOD(RTEMP,BUSW)
2980.   C
2990.         IF (LTEMP.LT.RTEMP)
3000.        &    THEN DO
3010.                   ABSMN = LTEMP
3020.                   DO 1 I=1,8
3030.      1            ABPNEW(I)=ABPOLD(I)
3040.              ELSE DO
3050.                   ABSMN = RTEMP
3060.                   DO 2 I=1,8
3070.      2            ABPNEW(I)=EFPOLD(I)
3080.           ENDIF
3090.   C
3100.   C-----------------------------------------------------------
3110.   C
3120.   C
3130.         LTEMP = ABSMO + CPM(IDATA)
3140.         LTEMP = MOD(LTEMP,BUSW)
3150.         RTEMP = EFSMO + DPM(IDATA)
3160.         RTEMP = MOD(RTEMP,BUSW)
3170.   C
3180.         IF (LTEMP.LT.RTEMP)
3190.        &    THEN DO
3200.                   CDSMN = LTEMP
```

```
3210.                 DO 3 I=1,8
3220.    3            CDPNEW(I)=ABPOLD(I)
3230.              ELSE DO
3240.                 CDSMN = RTEMP
3250.                 DO 4 I=1,8
3260.    4            CDPNEW(I)=EFPOLD(I)
3270.           ENDIF
3280.  C
3290.  C-------------------------------------------------------------
3300.  C       END FIRST BUTTERFLY
3310.  C
3320.  C-------------------------------------------------------------
3330.  C
3340.  C       START SECOND BUTTERFLY
3350.  C-------------------------------------------------------------
3360.  C
3370.  C
3380.          LTEMP = CDSMO + EPM(IDATA)
3390.          LTEMP = MOD(LTEMP,BUSW)
3400.          RTEMP = GHSMO + FPM(IDATA)
3410.          RTEMP = MOD(RTEMP,BUSW)
3420.  C
3430.          IF (LTEMP.LT.RTEMP)
3440.         &    THEN DO
3450.                 EFSMN = LTEMP
3460.                 DO 5 I=1,8
3470.    5            EFPNEW(I)=CDPOLD(I)
3480.              ELSE DO
3490.                 EFSMN = RTEMP
3500.                 DO 6 I=1,8
3510.    6            EFPNEW(I)=GHPOLD(I)
3520.           ENDIF
3530.  C
3540.  C-------------------------------------------------------------
3550.  C
3560.  C
3570.          LTEMP = CDSMO + GPM(IDATA)
3580.          LTEMP = MOD(LTEMP,BUSW)
3590.          RTEMP = GHSMO + HPM(IDATA)
3600.          RTEMP = MOD(RTEMP,BUSW)
3610.  C
3620.          IF (LTEMP.LT.RTEMP)
3630.         &    THEN DO
3640.                 GHSMN = LTEMP
3650.                 DO 7 I=1,8
3660.    7            GHPNEW(I)=CDPOLD(I)
3670.              ELSE DO
3680.                 GHSMN = RTEMP
3690.                 DO 8 I=1,8
3700.    8            GHPNEW(I)=GHPOLD(I)
3710.           ENDIF
3720.  C
3730.  C-------------------------------------------------------------
3740.  C       END SECOND BUTTERFLY
3750.  C
3760.  C-------------------------------------------------------------
3770.  C
3780.  C       OVERFLOW PROTECT
3790.  C                 FIND MINIMUM STATE METRIC
3800.  C
3810.          IF ( ABSMN .LT. CDSMN )
3820.         &     THEN DO
3830.                 MIN1=ABSMN
3840.                 MINP1 = 0
```

```
3850.                ELSE DO
3860.                     MIN1=CDSMN
3870.                     MINP1 = 1
3880.           ENDIF
3890.           IF ( EFSMN .LT. GHSMN )
3900.          &     THEN DO
3910.                     MIN2=EFSMN
3920.                     MINP2 = 0
3930.                ELSE DO
3940.                     MIN2=GHSMN
3950.                     MINP2 = 1
3960.           ENDIF
3970.          IF ( MIN1 .LT. MIN2 )
3980.          &     THEN DO
3990.                     MIN = MIN1
4000.                     MINP = MINP1
4010.                ELSE DO
4020.                     MIN = MIN2
4030.                     MINP = MINP2
4040.           ENDIF
4050.  C
4060.  C      GENERATE TWO'S COMPLEMENT & ADD TO EACH STATE
4070.  C
4080.          MIN = BUSW - MIN
4090.          IF ( MIN .EQ. BUSW ) MIN = 0
4100.  C
4110.          ABSMN = ABSMN + MIN
4120.          ABSMN = MOD ( ABSMN, BUSW )
4130.  C
4140.          CDSMN = CDSMN + MIN
4150.          CDSMN = MOD ( CDSMN, BUSW )
4160.  C
4170.          EFSMN = EFSMN + MIN
4180.          EFSMN = MOD ( EFSMN, BUSW )
4190.  C
4200.          GHSMN = GHSMN + MIN
4210.          GHSMN = MOD ( GHSMN, BUSW )
4220.  C
4230.  C
4240.  C      DELAY MINIMUM PATH DECISION FOR LATER COMPARISON
4250.  C
4260.          CALL SHIFT( MINPO, MINPN, MINP, MPOUT )
4270.          DO 11 I=1, 8
4280.    11    MINPN(I) = MINPO(I)
4290.  C
4300.  C
4310.  C-------------------------------------------------------------------
4320.  C
4330.  C      OUTPUT REGISTER STATUS FOR MACHINE
4340.  C
4350.          IF ( SUPPRES .EQ. 1 ) GO TO 77
4360.          IF ( ITIME .LT. TOUT ) GO TO 77
4370.  C
4380.          PRINT 21, ITIME,ABSMO,CDSMO,EFSMO,GHSMO
4390.    21    FORMAT ( '  TIME: ', I6, '     REGISTERS: ',4(I5,5X))
4400.          PRINT 20, ABPOLD,CDPOLD,EFPOLD,GHPOLD
4410.    20    FORMAT ( 4(/,15X,8I2), /  )
4420.  C
4430.    77    CONTINUE
4440.  C
4450.  C-------------------------------------------------------------
4460.  C
4470.  C      END OF BIT INTERVAL AND PROCESSING
4480.  C                        MAKE FINAL REGISTER TRANSFERS
```

```
4490.  C
4500.        ABSMO = ABSMN
4510.        CALL SHIFT ( ABPOLD,ABPNEW,0,BOUT1)
4520.  C
4530.        CDSMO = CDSMN
4540.        CALL SHIFT ( CDPOLD,CDPNEW,1,BOUT2)
4550.  C
4560.        EFSMO = EFSMN
4570.        CALL SHIFT ( EFPOLD,EFPNEW,0,BOUT3)
4580.  C
4590.        GHSMO = GHSMN
4600.        CALL SHIFT ( GHPOLD,GHPNEW,1,BOUT4)
4610.  C
4620.  C     MAJORITY VOTE FOR FINAL DECISION  -- MBOUT
4630.  C
4640.        MBOUT = 0
4650.        BSUM = 0
4660.        BSUM = BOUT1 + BOUT2 + BOUT3 + BOUT4
4670.        IF ( BSUM .GE. 2 ) MBOUT = 1
4680.        IF ( SUPPRES .EQ. 1 ) GO TO 55
4690.        IF ( ITIME .LT. TOUT ) GO TO 55
4700.        PRINT 27, RBOUT,MBOUT,MPOUT,QBOUT
4710.    27  FORMAT ( 15X, 'TRANSMIT ', I3, '  ESTIMATE ', I3,
4720.       &        ' MIN. PATH ', I3, ' QUANTIZER ', I3, /// )
4730.    55  CONTINUE
4740.  C
4750.  C-------------------------------------------------------------
4760.  C
4770.  C    ACCUMULATE ERROR STATISTICS
4780.  C
4790.        IF ( ITIME .LE. 64 ) GO TO 99
4800.        BITS = BITS + 1
4810.        IF ( RBOUT .EQ. QBOUT ) GO TO 97
4820.        QUANER = QUANER + 1
4830.    97  IF ( RBOUT .EQ. MPOUT ) GO TO 98
4840.        MINPER = MINPER + 1
4850.    98  IF ( RBOUT .EQ. MBOUT ) GO TO 99
4860.        ERRCNT = ERRCNT + 1
4870.  C
4880.    99  CONTINUE
4890.  C
4900.  C     OUTPUT STATISTICS FOR SIMULATION
4910.  C
4920.        ZBUSW = FLOAT (BUSW)
4930.        WIDTH = ALOG10 ( ZBUSW ) / ALOG10 ( 2.0 )    + 0.2
4940.        PRINT 46, WIDTH
4950.    46  FORMAT ( '1', 10X, I3, ' BIT PARALLEL RECURSIVE ARCHITECTURE'
4960.       &       , ///// )
4970.  C
4980.        ZBER = FLOAT(ERRCNT) / FLOAT(BITS)
4990.        PRINT 47
5000.    47  FORMAT(      10X, ' VITERBI RECEIVER ', // )
5010.        PRINT 28, ERRCNT,BITS,ZBER,ZSNR
5020.    28  FORMAT ( 10X, I7, ' ERRORS IN ', I8,
5030.       &    ' BITS READ: BER = ',E12.5,' FOR SNR = ',F4.1,' DB',//)
5040.  C
5050.        ZBER = FLOAT(MINPER) / FLOAT(BITS)
5060.        PRINT 48
5070.    48  FORMAT ( /////, 10X, ' MINIMUM PATH DETECTOR ', // )
5080.        PRINT 28, MINPER,BITS,ZBER,ZSNR
5090.  C
5100.        ZBER = FLOAT(QUANER) / FLOAT(BITS)
5110.        PRINT 49
5120.    49  FORMAT ( /////, 10X, ' THRESHOLD DETECTOR ', // )
```

```
5130.          PRINT 28, QUANER,BITS,ZBER,ZSNR
5140.    C
5150.    C
5160.    C
5170.          STOP
5180.          END
5190.          SUBROUTINE SHIFT(OLD,NEW,BITIN,BITOUT)
5200.          INTEGER OLD(8),NEW(8),BITIN,BITOUT
5210.    C
5220.    C ----- SHIFT REGISTER PROCEDURE
5230.    C
5240.          N=8
5250.          N1 = N - 1
5260.          OLD(1)=BITIN
5270.          DO 99 I=1,N1
5280.          OLD(I+1) = NEW(I)
5290.       99 CONTINUE
5300.          BITOUT = NEW(N)
5310.          RETURN
5320.          END
5330.          SUBROUTINE GAUSS(ZS,ZM,ZNOISE,DSEEDI,DSEEDO)
5340.          DOUBLE PRECISION DSEEDI,DSEEDO
5350.    C
5360.    C ----- PROCEDURE TO GENERATE A GAUSSIAN R.V.
5370.    C
5380.          A = 0.0
5390.          DO 1 I=1,48
5400.       1  A = A + GGUBFS(DSEEDI)
5410.          ZNOISE = (( A - 24.0 ) / 2.0 ) * ZS + ZM
5420.          DSEEDO = DSEEDI
5430.          RETURN
5440.          END
5450.          FUNCTION Q(X)
5460.          REAL Q,X
5470.    C
5480.    C ----- Q FUNCTION FOR GAUSSIAN NOISE
5490.    C
5500.          X1=X
5510.          IF(X.LT.0.0) X1 = -X
5520.          IF(X1.GE.13.0) GO TO 12
5530.          XS = X*X
5540.          Z = 0.39894228*EXP(-0.5*XS)
5550.          IF(X1.LE.6) GO TO 19
5560.          Q = Z*(1.0 - 1.0/XS)/X1
5570.          IF(X.LT.0.0) Q = 1.0 - Q
5580.          RETURN
5590.       19 T = 1.0/(1.0+0.2316419*X1)
5600.          B1 =  0.319381530
5610.          B2 = -0.356563782
5620.          B3 =  1.781477937
5630.          B4 = -1.821255978
5640.          B5 =  1.330274429
5650.          F = T*(B1+T*(B2+T*(B3+T*(B4+T*B5))))
5660.          Q = F*Z
5670.          IF(X.LT.0.0) Q = 1.0 - Q
5680.          RETURN
5690.       12 Q = 0.0
5700.          IF(X.LT.0.0) Q = 1.0
5710.          RETURN
5720.          END
5730.    C     FUNCTION Q(X)
5740.    C     REAL Q,X,RT2
5750.    C
5760.    C ----- Q FUNCTION FOR LAPLACIAN NOISE
```

```
C
C     RT2 = 1.414213562
C     IF ( X .LT. 0 ) GO TO 11
C     Q = 0.5 * EXP ( -1.0 * RT2 * X )
C     GO TO 22
C  11 Q = 1.0 - 0.5 * EXP ( RT2 * X )
C  22 CONTINUE
C     RETURN
C     END
      SUBROUTINE SUBRF(TBL,P1,P2,I3,I4,I5)
      INTEGER I3(1),I4,I5
      REAL TBL(I5,1),P1(1),P2(1)
C
C ----- LAPLACIAN TABLE GENERATION FOR GGVCR (IMSL)
C
      A = SQRT(2.0) / P1(1)
      TBL(3,1) = -12.0 * P1(1)
      TBL(3,2) = 0.0
      DO 1 I=4,I4
      TBL(I,1) = (-11.0 + (I-3)) * P1(1)
      IF (TBL(I,1) .LT. 0.0) GO TO 99
            TBL(I,2) = 1.0 - ( 0.5 * EXP( - A * TBL(I,1) ))
            GO TO 1
   99 TBL(I,2) = 0.50 * EXP( A * TBL(I,1))
    1 CONTINUE
      TBL(I4+1,1) = +12.0 * P1(1)
      TBL(I4+1,2) = +1.0
      RETURN
      END
$ENTRY
```

```
10.  //GULAK JOB ',,,T=1M,C=0','GLEN'
20.  // EXEC WATFIV,SIZE=256K
30.  //GO.SYSIN DD *
40.  $JOB  WATFIV   GLEN,NOEXT,NOWARN
50.        COMMON H(21),R(21),KS(101),DT(30),LG(30),IS(30,100)
60.        DATA LG/30*1/, IS/3000 * 0/
70.  C_____________________________________________________________________
80.  C
90.  C       PROGRAM TO EVALUATE  DMIN  BY EXHAUSTIVE SEARCH OF
100. C       BINARY ERROR SEQUENCES.
101. C            SPAN OF CHANNEL IMPULSE RESPONSE: 20 MAX
110. C            FIRST 30 SMALLEST ERROR SEQUENCES
120. C            ERROR SEQUENCE LENGTH: 100 MAX
130. C
140. C_____________________________________________________________________
150. C
160. C ----- READ AND ECHO CHANNEL IMPULSE RESPONSE
170.       PRINT 10
180.    10 FORMAT (///,10X,' L,',12X,'(H(K),K=0,L-1)' )
190.       READ , LL,(H(I),I=1,LL)
200.       PRINT , LL, (H(I),I=1,LL)
210.       MAX=4*LL
220. C
230. C ----- CALCULATE PULSE AUTOCORRELATION COEFFICIENTS (ONE-SIDE)
240.       DO 21 J=1,LL
250.       SUM = 0.0
260.       LAST = LL-J+1
270.       DO 20 K=1,LAST
280.       SUM = SUM + H(K) * H(K+J-1)
290.    20 CONTINUE
300.       R(J) = SUM
310.    21 CONTINUE
320. C
330.       PRINT , (R(I),I=1,LL)
340. C
350. C ----- INITIALIZE DISTANCE SQUARED VECTOR (DT) TO LENGTH 100
360.       DO 22 M=1,30
370.       DT(M) = 100.0
380.    22 CONTINUE
390. C
400. C -----  INITIALIZE TRIAL ERROR SEQUENCE VECTOR (KS) WITH INIT ERROR
410.       KS(1) = 1
420.       DO 23 L=2, 101
430.       KS(L) = 0
440.    23 CONTINUE
450. C
460. C-------------------------------------------------------------------
470. C
480. C ----- SET TRIAL ERROR SEQUENCE LENGTH (LS) POINTER
490.       KS(2) = 0
500.       LS = 2
510. C
520. C ----- SCAN ERROR SEQUENCES UP TO LENGTH 3 * CHANNEL MEMORY
530.       WHILE (LS .LE. MAX .AND. LS .LE. 100 ) DO
540. C
550.            DO 41 J=2, LS
560.            IF(KS(J) .EQ. 2 ) KS(J) = -1
570.    41      CONTINUE
580. C
590. C ----- CALCULATE DISTANCE OF TRIAL ERROR SEQUENCE
600.            CALL DSTNC(DQ,LS,LL)
610. C
620. C ----- REORDER ACCUMULATED LISTS IF NECESSARY ELSE DISCARD TRIAL
630.            CALL REORDER(DQ,LS)
```

```
C
C ----- GENERATE NEW TRIAL ERROR SEQUENCE
            DO 42 I=2, LS
            IF(KS(I).EQ. -1) KS(I) = 2
   42       CONTINUE
C
            KS(2) = KS(2) + 1
            DO 43 M=2, 100
            IF (KS(M) .EQ. 3 )
     &         THEN
                  KS(M) = 0
                  KS(M+1) = KS(M+1) + 1
               ENDIF
   43       CONTINUE
C
C ----- SET ERROR SEQUENCE LENGTH (LS) POINTER
            DO 44 N=2, 101
            IF(KS(N) .NE. 0 ) LS=N
   44       CONTINUE
C
       END WHILE
C
C ---------------------------------------------------------------------
C
C ----- TABULATE RESULTS
       PRINT 50
   50 FORMAT ( ///,' DISTANCE SQUARED    ERROR SEQUENCE',/)
       DO 52 K=1,30
       IF(DT(K).EQ. 100.0 ) GO TO 99
       LN = LG(K)
       PRINT 51, DT(K),(IS(K,L),L=1,LN)
   51 FORMAT(1X,F12.6,6X,25I2,3(/,19X,25I2))
   52 CONTINUE
   99 STOP
       END
       SUBROUTINE DSTNC(DQ,LS,LL)
       COMMON H(21),R(21),KS(101)
C
C---------------------------------------------------------------
C
C        TAKES THE TRIAL ERROR SEQUENCE OF LENGTH (LS) AND
C        DETERMINES THE DISTANCE (DQ) GIVEN THE CORRELATION
C        COEFFICIENTS OF THE CHANNEL
C
C---------------------------------------------------------------
C
C ----- CALCULATE DISTANCE (DQ) CONTRIBUTED BY R(1)
      SUM = 0.0
      DO 10 L=1,LS
      KSL = KS(L)
      IF(KSL .EQ. 0 ) GO TO 10
      SUM = SUM + KSL * KSL
   10 CONTINUE
      DQ = SUM * R(1)
C
C ----- CALCULATE DISTANCE (DQ) CONTRIBUTED BY REST OF R VECTOR
      SUM = 0.0
      DO 21 M=2, LS
      LAST = M-1
      KSM = KS(M)
      IF( KSM.EQ.0) GO TO 21
C
      DO 20 N=1,LAST
      MN = M - N +1
```

```
1280.          KSN = KS(N)
1290.          IF ( MN .GT. LL .OR. KSN .EQ. 0 ) GO TO 20
1300.          SUM = SUM + KSM*KSN*R(MN)
1310.       20 CONTINUE
1320.       21 CONTINUE
1330.  C
1340.  C ----- FINAL DISTANCE FOR THIS TRIAL ERROR SEQUENCE
1350.          DQ = DQ + 2.0 * SUM
1360.          RETURN
1370.          END
1380.          SUBROUTINE REORDER(DQ,LS)
1390.          COMMON H(21),R(21),KS(101),DT(30),LG(30),IS(30,100)
1400.  C
1410.  C-----------------------------------------------------------------------
1420.  C
1430.  C         TAKES THE TRIAL ERROR SEQUENCE AND DETERMINES WHETHER ITS
1440.  C         'DISTANCE' (DQ) IS LESS THAN ANY DISCOVERED SO FAR.
1450.  C         IF SO, THEN IT IS INSERTED IN THE APPROPRIATE ASCENDING
1460.  C         ORDER OF THE LIST.
1470.  C
1480.  C-----------------------------------------------------------------------
1490.  C
1500.  C ----- DETERMINE WHETHER DQ IS LESS THAN ANY DT IN THE LIST
1510.          DO 10 I=1,30
1520.          IDX=I
1530.          IF (DQ.GE.DT(IDX)) GO TO 10
1540.          GO TO 11
1550.       10 CONTINUE
1560.  C ----- DQ IS GREATER THAN THOSE IN THE LIST ---> EXIT.
1570.          GO TO 99
1580.  C ----- LAST ENTRY IN THE LIST - NOTHING TO PUSH DOWN - INSERT DIRECT
1590.       11 IF ( IDX.EQ.30 ) GO TO 30
1600.          LEFT = 30 - IDX
1610.  C
1620.  C ----- PUSH DOWN OTHER ENTRIES IN THE LIST FOR NEW MEMBER
1630.          DO 20 M=1, LEFT
1640.          K = 30 - M
1650.          DT(K+1) = DT(K)
1660.          LG(K+1) = LG(K)
1670.          LN = LG(K+1)
1680.  C
1690.          DO 20 N=1,LN
1700.          IS(K+1,N) = IS(K,N)
1710.       20 CONTINUE
1720.  C
1730.  C ----- PUSH NEW DISTANCE AND ERROR SEQUENCE LENGTH INTO LIST
1740.       30 DT(IDX) = DQ
1750.          LG(IDX) = LS
1760.  C
1770.  C ----- PUSH NEW ERROR SEQUENCE INTO THE LIST
1780.          DO 31 L=1,LS
1790.          IS(IDX,L) = KS(L)
1800.       31 CONTINUE
1810.  C
1820.       99 RETURN
1830.          END
1840.  $ENTRY
1850.  7  1.000 0.250 0.000 0.500 0.000 0.000 0.500
```

Appendix E

PLANARITY TESTING SOFTWARE

```
10.  //GLENN JOB ',,,T=8,C=0,L=4,I=8','EMBED IN PLANE',MSGLEVEL=(1,1)
20.  //     EXEC PL1OCG,X=NX,AG=NAG,A=NA,CSIZE=512K LSIZE=512K
30.  //PL1.SYSIN DD *
40.   /*              PLANAR EMBEDDING                                    */
50.  0/*-------------------------------------------------------------------*/
60.  0/* AN IMPLEMENTATION OF THE ALGORITHM FROM                          */
70.   /* "EFFICIENT PLANARITY TESTING" IN THE JACM VOL.12 (1974),         */
80.   /* BY J. HOPCROFT AND R. TARJAN.                                    */
90.  0/*-------------------------------------------------------------------*/
100. 0PLANE:   PROC   OPTIONS( MAIN )   REORDER;
110. 0  DECLARE
120.     ( NU,                          /* NUMBER OF VTXS                */
130.       EPS,                         /* NUMBER OF EDGES               */
140.       MAXDEG,                      /* MAX DEGREE IN GRAPH           */
150.       PATHLEN,
160.       BADPATH,
170.       ROOT                         INIT( 1 ),
180.       ADJ( *,* )                   /* ADJACENCY MATRIX              */
190.                           CTL )    FIXED,
200.       1 VX( * )           CTL,
210.         2 NBR             FIXED,   /* SET BY "DFS" PROC             */
220.         2 DEG             FIXED,   /* # OF ADJ VTXS                 */
230. 0     1 NUM( * )          CTL,     /* VTX LIST BY ORDER OF DEPTH    */
240.         2 ID              FIXED,
250.         2 DEG             FIXED,
260. 0                         /* ADJACENCY STRUCTURE:                  */
270.       1 ADJ_LIST( *,* )   CTL,     /* A LIST OF ADJ VTXS, FOR EACH VTX */
280.         2 JOIN            BIT(1),  /* SELECT DIRECTED EDGES         */
290.         2 AV              FIXED,   /* IDENTIFY ADJ VTXS             */
300. 0     ( AL( *,* ),                 /* ADJ LISTS IN ORDER OF VTX DEPTH */
310.         DEP( *,* ),                /* DEPENDENCY GRAPH              */
320.         LOW0(*), LOW1(*) )         /* LOWEST VTXS REACHED BY FROND  */
330.                                    /* FROM DESCENDANTS OF VTX       */
340.                           CTL FIXED,
350. 0     ( SEGTOP, SEGEND )  PTR,     /* LIST OF SEGMENTS IN GRAPH     */
360.       1 SEG               BASED,
370.         2 NXT             PTR,     /* PT TO NEXT SEGMENT            */
380.         2 ( PTOP, PEND )  PTR,     /* PT TO LIST OF PATHS IN SEG    */
390. 0     1 PATH              BASED,   /* A PATH IN A SEG               */
400.         2 NXT             PTR,     /* NEXT PATH IN SEG              */
410.         2 LEN             FIXED,   /* LENGTH OF CURR PATH           */
420.         2 NODE( PATHLEN REFER( LEN ))/* VTXS IN PATH                */
430.                           FIXED,
440.     ( MORE,                        /* FLAG END OF SYSIN FILE        */
450.       PLANAR    )                 /* T. IF GRAPH IS STILL EMBEDDABLE */
460.                           BIT(1) INIT('1'B),
470.       NULL                BUILTIN,
480.     ( SYSIN, SYSPRINT )   FILE;
490. 1   ON ENDFILE( SYSIN ) MORE = '0'B;
500.     OPEN FILE( SYSPRINT ) PRINT STREAM OUTPUT,
510.          FILE( SYSIN )    STREAM INPUT;
520. 0/*-------------------------------------------------------------------*/
530. 0/*              *   M A I N L I N E   *                              */
540.  /*                                                                   */
550.  /* ESSENTIALLY THIS ALGORITHM IS TWO DEPTH-FIRST SEARCHES:           */
560.  /*                                                                   */
570.  /* 1. THE FIRST FINDS A SPANNING PALM TREE, P(G).                    */
580.  /*    USING SOME RESULTS FROM THIS SEARCH, THE VTXS ARE              */
590.  /*    ORDERED AS REQUIRED BY THE EMBEDDING PROCEDURE.                */
600.  /*                                                                   */
610.  /* 2. APPLY ANOTHER SEARCH TO P(G), TRAVERSING THE VTXS IN A         */
620.  /*    PARTICULAR ORDER.  THIS SEARCH FINDS PATHS IN G WHICH          */
630.  /*    ARE THEN EMBEDDED WITH PREVIOUSLY FOUND PATHS.                 */
640.  /*    FINDING THE PATHS IN THE CORRECT ORDER IS NECESSARY FOR        */
```

```
 650.   /*    THE EMBEDDING PROCEDURE TO WORK PROPERLY AND EFFICIENTLY.      */
 660.   /*                                                                   */
 670.   /* THIS PROGRAM EXECUTES IN A TIME THAT IS LINEARLY DEPENDENT        */
 680.   /* ON THE NUMBER OF VERTICES IN G.  G SHOULD BE BICONNECTED.         */
 690.  0/*-------------------------------------------------------------------*/
 700.  0  GET FILE( SYSIN ) LIST( NU );
 710.     DO WHILE( MORE );
 720.         CALL INITIALIZE;
 730.         CALL DFS( (ROOT), 0, 0 );
 740.         CALL ORDER_ADJ_LISTS;
 750.         CALL HALFWAY;
 760.         CALL EMBED( AL, NUM.DEG, EPS, NU );
 770.         CALL WRITE_RESULT;
 780.         GET FILE( SYSIN ) LIST( NU );
 790.     END;
 800.  1  HALFWAY:   PROC;
 810.  0/*-------------------------------------------------------------------*/
 820.  0/* - DISPLAY ADJACENCY MATRIX.                                       */
 830.   /* - FREE STRUCTURES NO LONGER NEEDED.                               */
 840.  0/*-------------------------------------------------------------------*/
 850.  0    PUT FILE( SYSPRINT ) EDIT
 860.         ( (110) '=', ' ' )( SKIP(3),A, SKIP(2),A );
 870.       CALL WRITE_GRAPH( ADJ, VX(*).NBR, NU,
 880.         'GIVEN GRAPH:' );
 890.       FREE VX, ADJ, ADJ_LIST;
 900.     END HALFWAY;
 910.  1  WRITE_RESULT:   PROC;
 920.  0    DECLARE
 930.       ( PATHCNT, SEGCNT )    FIXED,
 940.       ( P, S, T1, T2 )       PTR;
 950.  0/*                    DISPLAY LIST OF PATHS                          */
 960.  0    PATHCNT, SEGCNT = 0;
 970.       PUT FILE( SYSPRINT ) EDIT
 980.         ('PATH','NUMBER')( SKIP(2),COL(2),A,SKIP,A );
 990.       DO S = SEGTOP   REPEAT T1   WHILE( S ¬= NULL );
1000.           SEGCNT = SEGCNT + 1;
1010.           PUT FILE( SYSPRINT ) EDIT
1020.             ('(SEGMENT',SEGCNT,')',' ')
1030.             ( SKIP(2),COL(6),A,F(3),A,SKIP,A );
1040.  0        DO P = S-> PTOP   REPEAT T2   WHILE( P ¬= NULL );
1050.               PATHCNT = PATHCNT + 1;
1060.               PUT FILE( SYSPRINT ) EDIT
1070.                 ( PATHCNT,'.....',P->NODE )
1080.                 ( SKIP,F(4), A,(P->PATH.LEN)F(3) );
1090.               T2 = P-> PATH.NXT;
1100.               FREE P-> PATH;
1110.           END;
1120.  0        T1 = S-> SEG.NXT;
1130.           FREE S-> SEG;
1140.       END;
1150.  0/*                    IF PLANAR THEN DISPLAY DEPENDENCY GRAPH        */
1160.  0    IF PLANAR THEN
1170.         DO;
1180.           PUT FILE( SYSPRINT ) EDIT
1190.             ('<< *** PLANAR GRAPH *** >>')( SKIP(2),COL(5),A );
1200.           ALLOCATE   VX( PATHCNT );
1210.           CALL WRITE_GRAPH( DEP, VX(*).NBR, -PATHCNT,
1220.             'DEPENDENCY GRAPH:' );
1230.         END;
1240.       ELSE
1250.  0      IF BADPATH < 0 THEN
1260.           PUT FILE( SYSPRINT ) EDIT
1270.             ('<< ??? NOT PLANAR:  TOO MANY EDGES ??? >>')
1280.             ( SKIP(2),COL(5),A );
```

```
1290.          ELSE
1300.              PUT FILE( SYSPRINT ) EDIT
1310.                ('<< ??? NOT PLANAR:  EMBEDDING STOPPED AT PATH',
1320.                  BADPATH,' ??? >>' )
1330.                ( SKIP(2),A,F(3),A );
1340.        FREE VX, NUM, AL;
1350.      END WRITE_RESULT;
1360.   1INITIALIZE:   PROC;
1370.   0/*-------------------------------------------------------------------*/
1380.   0/* - ALLOCATE STRUCTURES BASED ON "NU"                               */
1390.    /* - SET ADJACENCY LISTS                                            */
1400.   0/*-------------------------------------------------------------------*/
1410.   0  DECLARE
1420.      ( R, C,                          /* INDICES FOR "ADJ"             */
1430.        IX )                           /* INDEX FOR "AV"                */
1440.                              FIXED,
1450.        P                     POINTER,
1460.      ( SUM, REPEAT )         BUILTIN;
1470.   0  MAXDEG, EPS = 0;
1480.      ALLOCATE ADJ( NU, NU ), VX( NU ), LOW0( NU ), LOW1( NU );
1490.      GET FILE( SYSIN ) LIST( ADJ );
1500.   0  DO R = 1 TO NU;
1510.           VX( R ).NBR = 0;
1520.           VX( R ).DEG = SUM( ADJ(R,*) );
1530.           IF VX(R).DEG > MAXDEG THEN MAXDEG = VX(R).DEG;
1540.           EPS         = EPS + VX(R).DEG;
1550.      END;
1560.   0  EPS = EPS / 2;
1570.      IF EPS > 3*( NU - 2 )
1580.        THEN CALL NONPLANAR( -1 );
1590.        ELSE PLANAR = '1'B;
1600.      ALLOCATE ADJ_LIST( NU, MAXDEG );
1610.      DO R = 1 TO NU;
1620.           IX = 1;
1630.           ADJ_LIST( R,* ).JOIN = '0'B;
1640.           DO C = 1 TO NU;
1650.             IF ADJ( R,C ) > 0 THEN
1660.               DO;
1670.                 AV( R, IX )= C;
1680.                 IX         = IX + 1;
1690.               END;
1700.           END;
1710.      END;
1720.    END INITIALIZE;
1730.   1DFS:   PROC( V, P, DEPTH )   RECURSIVE;
1740.   0/*-------------------------------------------------------------------*/
1750.   0/* "DEPTH-FIRST SEARCH"                                             */
1760.    /*                                                                  */
1770.    /* TRAVERSE GRAPH WHILE STACKING VTXS AS THEY ARE CROSSED.          */
1780.    /* EACH NEW VTX INDICATES A TREE ARC; AN OLD VTX (WHICH             */
1790.    /* IS NOT THE CURR VTX'S PARENT) INDICATES A FROND.  UPON           */
1800.    /* REACHING THE END, BACK DOWN THE STACK TO THE PREV VTX            */
1810.    /* THEN CONTINUE ALONG ITS NEXT EDGE.                               */
1820.    /*                                                                  */
1830.    /* "LOW " VALUES, WHICH ARE REQUIRED TO ORDER THE VTXS              */
1840.    /* OF THE SPANNING PALM TREE FOR THE "EMBED" PROC, ARE              */
1850.    /* COLLECTED DURING "DFS".                                          */
1860.   0/*-------------------------------------------------------------------*/
1870.   0  DECLARE
1880.      ( V,                        /* CURR VTX IN SPANNING TREE BEING BUILT */
1890.        P,                        /* PARENT OF "V"                     */
1900.        DEPTH,                    /* LAST DEPTH # ASSIGNED TO A VTX    */
1910.        IX,                       /* TO INDEX THRU ADJ LIST            */
1920.        W )                       /* VTX ADJ TO "V"                    */
```

```
1930.                              FIXED,
1940.       MIN                    BUILTIN;
1950.  0  DEPTH, NBR( V ) = DEPTH + 1;
1960.     LOW0( V ), LOW1( V ) = NBR( V );
1970.  0  DO IX = 1 TO VX( V ).DEG;
1980.          W = AV( V, IX );
1990.          IF VX( W ).NBR = 0 THEN
2000.            DO;
2010.              ADJ_LIST( V,IX ).JOIN = '1'B;
2020.              CALL DFS( W, V, DEPTH );
2030.              IF LOW0( W ) < LOW0( V ) THEN
2040.                DO;
2050.                  LOW1( V ) = MIN( LOW0(V), LOW1(W) );
2060.                  LOW0( V ) = LOW0( W );
2070.                END;
2080.              ELSE IF LOW0( W ) = LOW0( V ) THEN
2090.                  LOW1( V ) = MIN( LOW1(V), LOW1(W) );
2100.              ELSE
2110.                  LOW1( V ) = MIN( LOW1(V), LOW0(W) );
2120.            END;
2130.          ELSE IF NBR( W ) < NBR( V )  &  W ¬= P THEN
2140.            DO;
2150.              ADJ_LIST( V,IX ).JOIN = '1'B;
2160.              IF NBR(W) < LOW0(V) THEN
2170.                DO;
2180.                  LOW1( V ) = LOW0( V );
2190.                  LOW0( V ) = NBR( W );
2200.                END;
2210.              ELSE IF NBR( W ) > LOW0( V ) THEN
2220.                  LOW1( V ) = MIN( LOW1(V), NBR(W) );
2230.            END;
2240.     END;
2250.   END DFS;
2260.  1ORDER_ADJ_LISTS:   PROC;
2270.  0/*-------------------------------------------------------------------*/
2280.  0/* THIS ROUTINE ORDERS THE ADJ STRUCTURE ACCORDING TO THE            */
2290.   /* DEPTH OF THE VTX IN P(G), THE ROOT BEING THE FIRST LIST           */
2300.   /* IN "AL".                                                         */
2310.   /*    WE ORDER THE VTXS IN EACH ADJ LIST ACCORDING TO THE           */
2320.   /* "PHI_FCN" OF THE CORR EDGE, USING A RADIX SORT.                  */
2330.  0/*-------------------------------------------------------------------*/
2340.  0  DECLARE
2350.       CELL( 2*NU+1 )          PTR,
2360.       1 EDGE                  BASED,
2370.         2 ( V0, V1 )          FIXED,
2380.         2 NXT                 PTR,
2390.     ( Y,
2400.       Z )
2410.                               FIXED,
2420.     ( P, SAVE )               PTR;
2430.  0  CELL( * ) = NULL;
2440.     ALLOCATE AL( NU, MAXDEG ), NUM( NU );
2450.     DO Y = 1 TO NU;
2460.         DO Z = 1 TO VX( Y ).DEG;
2470.           IF JOIN( Y, Z ) THEN
2480.               CALL ADD_EDGE( Y, AV(Y, Z) );
2490.         END;
2500.     END;
2510.  0  NUM(*).DEG = 0;
2520.     DO Z = 1 TO ( 2*NU+1 );
2530.       DO P = CELL( Z )   REPEAT SAVE   WHILE( P ¬= NULL );
2540.           NUM( P-> V0 ).DEG = NUM( P-> V0 ).DEG + 1;
2550.           AL( P->V0, NUM(P->V0).DEG ) = P-> V1;
2560.           SAVE = P-> EDGE.NXT;
```

```
2570.           FREE P-> EDGE;
2580.       END;
2590.     END;
2600.  1  ADD_EDGE:   PROC( V, W );
2610.  0/* FOR THE RADIX SORT:  PLACE AN EDGE IN THE APPROPRIATE CELL        */
2620.  0    DECLARE
2630.       ( V, W,
2640.         C# )
2650.                          FIXED,
2660.         NEW              PTR;
2670.  0    C# = PHI_FCN( V, W );
2680.       ALLOCATE EDGE   SET( NEW );
2690.       NEW-> EDGE.V0  = NBR( V );
2700.       NEW-> EDGE.V1  = NBR( W );
2710.       NEW-> EDGE.NXT = CELL( C# );
2720.       CELL( C# )     = NEW;
2730.     END ADD_EDGE;
2740.  -  PHI_FCN:   PROC( V, W );
2750.  0/*-------------------------------------------------------------------*/
2760.  0/* FOR 2 EDGES VX AND VY WE WANT X AHEAD OF Y IN "AL(V,X*)"          */
2770.   /* IF FRONDS FROM X OR DESCENDANTS OF X REACH VTXS LOWER             */
2780.   /* THAN FRONDS FROM Y OR DESCENDANTS OF Y.                           */
2790.  0/*-------------------------------------------------------------------*/
2800.  0    DCL   ( V, W, R )   FIXED;
2810.       R = 0;
2820.       IF NBR(V) > NBR(W) THEN
2830.           R = NBR( W ) + NBR( W );
2840.       ELSE DO;
2850.           R = LOW0( W ) + LOW0( W );
2860.           IF LOW1( W ) < NBR( V )   THEN   R = R + 1;
2870.           END;
2880.       RETURN( R );
2890.     END PHI_FCN;
2900.   END ORDER_ADJ_LISTS;
2910.  1EMBED:   PROC( AL, DEG, EPS, NU );
2920.  0/*-------------------------------------------------------------------*/
2930.  0/* THIS ROUTINE TEST THE PLANARITY OF G BY ATTEMPTING TO EMBED       */
2940.   /* THE PATHS ONE AT A TIME IN THE PLANE.                            */
2950.   /*    THE FIRST PATH GENERATED IN "PATHFINDER" IS A CYCLE, C.       */
2960.   /* G-C IS DISCONNECTED.  CALL THE CONNECTED COMPONENTS OF G-C        */
2970.   /* SEGMENTS.  THE PATHS ARE GENERATED IN SUCH A WAY THAT ALL         */
2980.   /* THE PATHS IN ONE SEGMENT ARE FOUND BEFORE THE FIRST PATH OF       */
2990.   /* THE NEXT SEGMENT.                                                */
3000.   /*                                                                  */
3010.   /* FIND THE FIRST PATH, P1, OF A SEGMENT S.                         */
3020.   /* IF P1+C+ALL PREV SEGMENTS IS PLANAR, AND,                        */
3030.   /* IF S+C IS PLANAR THEN S+C+ALL PREV SEGMENTS IS PLANAR.           */
3040.   /*                                                                  */
3050.   /* A PATH IS EMBEDED EITHER ON THE LEFT (EXTERIOR) OR RIGHT         */
3060.   /* (INTERIOR) OF C.  IF P1 IS EMBEDDED ON THE LEFT (RIGHT) THEN     */
3070.   /* ALL PATHS IN S MUST BE EMBEDDED ON THE LEFT (RIGHT).             */
3080.   /*                                                                  */
3090.   /* KEEP A STACK L OF PATHS EMBEDDED ON THE LEFT AND STACK R OF      */
3100.   /* PATHS EMBEDDED ON THE RIGHT.  WE NEED STORE ONLY THE LAST        */
3110.   /* VTX OF THE PATH ON L AND R FOR THE FOLLOWING REASON:             */
3120.   /*                                                                  */
3130.   /*    WE MAY EMBED A PATH STARTING AT V1, ENDING AT V2, ON THE      */
3140.   /*    LEFT (RIGHT) IF AND ONLY IF NO FROND, XY, PREVIOUSLY EM-      */
3150.   /*    BEDDED ON THE LEFT (RIGHT) SATISFIES V2 < Y < V1.             */
3160.   /*                                                                  */
3170.   /* THE PATHS ON L AND R CAN BE GROUPED INTO BLOCKS SUCH THAT THE     */
3180.   /* PLACEMENT OF ANY ONE PATH DETERMINES THE POSITION OF ALL          */
3190.   /* PATHS IN THE BLOCK.                                              */
3200.   /*                                                                  */
```

```
3210.   /* THE ACTUAL EMBEDDING IS DONE THROUGH "PATHFINDER"; ONLY          */
3220.   /* THE INITIALIZATION OF VBLS IS DONE HERE.                        */
3230.  0/*-------------------------------------------------------------------*/
3240.  1  DECLARE
3250.     ( STACK( 0 : EPS ),              /* L AND R STACKS               */
3260.       NEXT (-1 : EPS ),              /* PTS TO NXT "STACK" ENTRY     */
3270.                                      /* NEXT(-1) PTS TO TOP OF R     */
3280.                                      /* NEXT( 0) PTS TO TOP OF L     */
3290.       F    ( 1 : EPS-NU+1 ),         /* F(I)=TERMINAL VTX OF PATH I  */
3300.       FPATH( 1 : NU ),               /* # OF 1ST PATH CONTAINING VTX I*/
3310.       FINV ( 0 : NU ),               /* INVERSE OF "F"               */
3320.       CURR ( NU+1 ),                 /* LIST OF VTXS IN CURR PATH    */
3330.       FREE,                          /* 1ST AVAIL POSN IN "STACK"    */
3340.       P,                             /* CURR PATH #                  */
3350.       S                              /* START OF CURR PATH           */
3360.     )                        FIXED,
3370.     ( AL( *,* ),
3380.       DEG( * ),
3390.       EPS, NU
3400.     )                        FIXED,
3410.       BTOP                   FIXED,
3420.       1 B( 0 : NU ),
3430.         2 ( X, Y )
3440.                              FIXED,
3450.     ( LEFT                   INIT(  0 ),
3460.       RIGHT                  INIT( -1 )
3470.     )                        FIXED STATIC,
3480.       ALLOCATION             BUILTIN;
3490.  0  P, S,
3500.     NEXT ( RIGHT ), NEXT( LEFT ), STACK( 0 ) = 0;
3510.  0  B(0).X, B(0).Y,
3520.     BTOP           = 0;
3530.     FINV( * )      = 0;
3540.  0  FREE, FPATH( ROOT ) = 1;
3550.     ALLOCATE   DEP( EPS-NU+1, EPS-NU+1 );
3560.     DEP( *,* ) = 0;
3570.     SEGTOP, SEGEND = NULL;
3580.  0  CALL PATHFINDER( (ROOT) );
3590.D  DUMP: PROC;
3600.D    DCL Z FIXED; PUT EDIT('RIGHT:')(SKIP(3),A);
3610.D    DO Z=NEXT(RIGHT) REPEAT NEXT(Z) WHILE(Z¬=0);
3620.D      PUT EDIT(STACK(Z))(F(3));  END;
3630.D    PUT EDIT('LEFT :')(COL(1),A);
3640.D    DO Z=NEXT(LEFT) REPEAT NEXT(Z) WHILE(Z¬=0);
3650.D      PUT EDIT(STACK(Z))(F(3));  END;
3660.D    PUT EDIT('BLOCKS:')(COL(1),A);
3670.D    DO Z=BTOP TO 1 BY -1;
3680.D      PUT EDIT('  (',STACK(B(Z).X),STACK(B(Z).Y),')')(A,(2)F(3),A);
3690.D  END;  END DUMP;
3700.  1  PATHFINDER:   PROC( V )   RECURSIVE;
3710.  0/*-------------------------------------------------------------------*/
3720.  0/* PROCESS THE ADJ LIST OF EACH VTX.                                */
3730.   /*    IF VW IS AN ARC THEN ISSUE A RECURSIVE CALL TO CONTINUE       */
3740.   /* ALONG THE PATH.  AFTER RETURING FROM PROCESSING AN ENTIRE        */
3750.   /* SEGMENT, ATTEMPT TO PUT ALL NEW BLOCKS ON L THEN GATHER          */
3760.   /* THEM INTO ONE BLOCK.                                            */
3770.   /*    IF VW IS A FROND THEN WE HAVE REACHED THE END OF A PATH.      */
3780.   /* THEREFORE WE ATTEMPT TO EMBED THE PATH.                         */
3790.  0/*-------------------------------------------------------------------*/
3800.  0    DECLARE
3810.       ( V, W,                      /* "W" IS ADJ TO "V"             */
3820.         IX,                        /* TO INDEX THRU ADJ LIST OF V   */
3830.         LP, RP,                    /* LEFT & RIGHT "STACK" PTRS     */
3840.         SAVE_S                     /* LOCAL VBL TO SAVE GLOBAL "S"  */
```

```
3850.        )                          FIXED;
3860.  0     SAVE_S = S;
3870.        DO IX = 1 TO DEG( V );
3880.            W = AL( V, IX );
3890.            IF S = 0 THEN CALL NEW_PATH( V, SAVE_S );
3900.            PATHLEN = PATHLEN + 1;
3910.            CURR( PATHLEN ) = W;
3920.  0         IF V < W THEN
3930.                                       /* VW IS A TREE ARC                  */
3940.              DO;
3950.                FPATH( W ) = P;
3960.                CALL PATHFINDER( W );
3970.                IF PLANAR THEN
3980.                  DO;
3990.                    CALL REMOVE_HI_VTXS( V );
4000.                    IF FPATH( W ) ¬= FPATH( V ) THEN
4010.                        CALL MOVE_NEW_BLOCKS( LP, RP, W );
4020.                  END;
4030.              END;
4040.  0         ELSE
4050.                                       /* VW IS A FROND                     */
4060.              DO;
4070.                IF PLANAR THEN
4080.                  DO;
4090.                    F( P )    = W;
4100.                    CALL SWITCH_BLOCKS( LP, RP, W );
4110.                    CALL PATH_EPILOGUE;
4120.                  END;
4130.                CALL STORE_PATH;
4140.                S = 0;
4150.              END;
4160.        END;
4170.  1  PATH_EPILOGUE:   PROC;
4180.  0/*-------------------------------------------------------------------*/
4190.  0/* - IF THE PATH IS NOT C THEN STACK THE PATH ON L.                  */
4200.   /* - ADD NEW BLOCK THAT CORRESPONDS TO UNION OF OLD BLOCKS           */
4210.   /*   DELETED IN "SWITCH_BLOCKS."                                    */
4220.   /* - FINALLY, IF CURR PATH IS NOT A SINGLE FROND, ADD AN            */
4230.   /*   END OF STACK MARKER TO R TO PREPARE FOR A RECURSIVE             */
4240.   /*   CALL.                                                          */
4250.  0/*-------------------------------------------------------------------*/
4260.  0     IF W > 1 THEN
4270.          DO;
4280.            CALL ADD_STACK( LEFT, W );
4290.            IF LP = LEFT   THEN   LP = NEXT( LEFT );
4300.          END;
4310.  0     IF RP = RIGHT THEN RP = 0;
4320.        IF ( LP ¬= 0 ) | ( RP ¬= 0 ) | ( V ¬= S ) THEN
4330.            CALL ADD_B( LP, RP );
4340.  0     IF V ¬= S THEN CALL ADD_STACK( RIGHT, 0 );
4350.        FINV( W ) = P;
4360.     END PATH_EPILOGUE;
4370.   END PATHFINDER;
4380.  1  REMOVE_HI_VTXS:   PROC( V );
4390.  0/*-------------------------------------------------------------------*/
4400.  0/* AFTER ALL SEGMENTS STARTING AT VTX V(I+1) HAVE BEEN EXPLORED      */
4410.   /* AND EMBEDDED, SEGMENTS STILL TO BE EXPLORED START AT VTXS NO      */
4420.   /* GREATER THAN V(I).  THEREFORE REMOVE ALL OCCURRENCES ON L         */
4430.   /* AND R (AND THE CORRESPONDING BLOCKS) OF VTXS >= V(I).            */
4440.  0/*-------------------------------------------------------------------*/
4450.  0     DCL   V   FIXED;
4460.  0     DO WHILE
4470.        ( ( STACK( B(BTOP).X ) >= V  |  B(BTOP).X = 0 )
4480.        & ( STACK( B(BTOP).Y ) >= V  |  B(BTOP).Y = 0 )
```

```
4490.        &   BTOP > 0 );
4500.              CALL POP_B( 1 );
4510.        END;
4520.        IF STACK( B(BTOP).X ) >= V THEN B(BTOP).X = 0;
4530.        IF STACK( B(BTOP).Y ) >= V THEN B(BTOP).Y = 0;
4540.  0     CALL POP_STACK( RIGHT, V, 0 );
4550.        CALL POP_STACK(  LEFT, V, 0 );
4560.      END REMOVE_HI_VTXS;
4570.  1   MOVE_NEW_BLOCKS:   PROC( LP, RP, W );
4580.  0/*-------------------------------------------------------------------*/
4590.  0/* ALL OF SEGMENT WITH 1ST EDGE VW (FROM "PATHFINDER") HAS BEEN      */
4600.   /* EMBEDDED, THEREFORE NEW BLOCKS MUST BE MOVED FROM R TO L IF       */
4610.   /* POSSIBLE.  (THIS IS BECAUSE ALL NEW SEGMENTS ARE EMBEDDED ON      */
4620.   /* THE LEFT, THEN MOVED TO THE RIGHT LATER IF NECESSARY.)            */
4630.  0/*-------------------------------------------------------------------*/
4640.  0     DCL   ( LP, RP, W )   FIXED;
4650.  0     LP = LEFT;
4660.D   CALL DUMP;
4670.  0     LOOP:   DO WHILE
4680.        (
4690.          ( STACK( B(BTOP).X ) > F( FPATH( W )) )
4700.        |   ( STACK( B(BTOP).Y ) > F( FPATH( W ))
4710.          & ( STACK( NEXT( RIGHT )) ¬= 0 ) )
4720.        );
4730.            IF STACK( B(BTOP).X ) > F( FPATH( W )) THEN
4740.              DO;
4750.                IF STACK( B(BTOP).Y ) > F( FPATH( W )) THEN
4760.                  DO;
4770.                    CALL NONPLANAR( P );
4780.                    LEAVE LOOP;
4790.                  END;
4800.                ELSE
4810.                    LP = B(BTOP).X;
4820.              END;
4830.  0         ELSE                      /* STACK( Y ) > F( FPATH( W ))     */
4840.              DO;
4850.                CALL SWITCH_NEXT( LP, RIGHT );
4860.                CALL SWITCH_NEXT( RIGHT, B(BTOP).Y  );
4870.                LP = B(BTOP).Y;
4880.              END;
4890.            CALL POP_B( 1 );
4900.        END;
4910.  0     IF B(BTOP).X = 0 THEN
4920.          IF LP ¬= 0   |  B(BTOP).Y ¬= 0
4930.            THEN B(BTOP).X = LP;
4940.            ELSE CALL POP_B( 1 );
4950.  0     CALL POP_STACK( RIGHT, 0, 1 );
4960.D   CALL DUMP;
4970.      END MOVE_NEW_BLOCKS;
4980.  1   SWITCH_BLOCKS:   PROC( LP, RP, W );
4990.  0/*-------------------------------------------------------------------*/
5000.  0/* ATTEMPT TO EMBED CURR PATH, P, ON THE LEFT.  IF P MUST CROSS       */
5010.   /* A PATH Q ALREADY ON L, THEN MOVE THE BLOCK THAT CONTAINS Q TO      */
5020.   /* R.  THIS MAY CAUSE A BLOCK ON R TO BE MOVED BACK TO L.            */
5030.   /*     IF PATHS ON BOTH L AND R CONFLICT WITH P THEN G IS            */
5040.   /* NON-PLANAR.                                                       */
5050.  0/*-------------------------------------------------------------------*/
5060.  0     DECLARE
5070.        ( LP, RP,
5080.          W,
5090.          TL, TR
5100.        )                       FIXED;
5110.  0     LP = LEFT;
5120.        RP = RIGHT;
```

```
5130.  0    LOOP:   DO WHILE
5140.       (
5150.         ( NEXT( LP ) ¬= 0  &  STACK( NEXT( LP )) > W )
5160.       | ( NEXT( RP ) ¬= 0  &  STACK( NEXT( RP )) > W )
5170.       );
5180.           IF B(BTOP).X ¬= 0  &  B(BTOP).Y  ¬= 0  THEN
5190.  0          IF STACK( NEXT( LP )) > W THEN
5200.               DO;
5210.                 IF STACK( NEXT( RP )) > W THEN
5220.                   DO;
5230.                     CALL NONPLANAR( P );
5240.                     LEAVE LOOP;
5250.                   END;
5260.                 ELSE
5270.                     CALL EXCHANGE;
5280.               END;
5290.  0          ELSE
5300.               DO;
5310.                 LP = B(BTOP).X;
5320.                 RP = B(BTOP).Y;
5330.               END;
5340.  0        ELSE IF B(BTOP).X ¬= 0 THEN
5350.               CALL MOVE_TO_RIGHT;
5360.           ELSE IF B(BTOP).Y ¬= 0 THEN
5370.               RP = B(BTOP).Y;
5380.  0        CALL POP_B( 1 );
5390.       END;
5400.  1    EXCHANGE:   PROC;
5410.  0/*                    EXCHANGE BLOCKS ON TOP OF L AND R                  */
5420.         CALL SWITCH_NEXT( RP, LP );
5430.         CALL SWITCH_NEXT( B(BTOP).X, B(BTOP).Y );
5440.         LP = B(BTOP).Y;
5450.         RP = B(BTOP).X;
5460.       END EXCHANGE;
5470.  -    MOVE_TO_RIGHT:   PROC;
5480.  0/*                    MOVE BLOCK ON L TO R                               */
5490.         CALL SWITCH_NEXT( B(BTOP).X, RP );
5500.         CALL SWITCH_NEXT( RP, LP );
5510.         RP = B(BTOP).X;
5520.       END MOVE_TO_RIGHT;
5530.     END SWITCH_BLOCKS;
5540.  1NEW_PATH:   PROC( V, SS );
5550.  0/*-------------------------------------------------------------------------*/
5560.  0/* WHEN A NEW PATH IS FOUND, INCR "P" AND RESET "CURR".                   */
5570.   /* IF THE NEW PATH INITIATES A NEW SEGMENT THEN ALLOCATE                  */
5580.   /* A NEW "SEG" STRUCTURE.                                                */
5590.  0/*-------------------------------------------------------------------------*/
5600.  0  DECLARE
5610.       ( V, SS )              FIXED,
5620.         NEW                  PTR;
5630.       IF SS = 0  |  SS = ROOT THEN
5640.         DO;
5650.           ALLOCATE SEG   SET( NEW );
5660.           IF SEGTOP = NULL
5670.             THEN SEGTOP = NEW;
5680.             ELSE SEGEND-> SEG.NXT = NEW;
5690.           NEW-> SEG.NXT,
5700.           NEW-> SEG.PTOP,
5710.           NEW-> SEG.PEND = NULL;
5720.           SEGEND         = NEW;
5730.         END;
5740.       P = P + 1;
5750.       PATHLEN = 1;
5760.       CURR( 1 ), S = V;
```

```
5770.    END NEW_PATH;
5780.  1 STORE_PATH:   PROC;
5790.  0/*-------------------------------------------------------------------------*/
5800.  0/* A FROND SIGNALS THE END OF A PATH.  ADD THIS PATH                       */
5810.   /* TO THE "PATH" LIST.                                                     */
5820.  0/*-------------------------------------------------------------------------*/
5830.  0     DECLARE
5840.          NEW                    PTR,
5850.          X                      FIXED;
5860.        ALLOCATE PATH   SET( NEW );
5870.        IF SEGEND-> SEG.PTOP = NULL
5880.          THEN SEGEND-> SEG.PTOP = NEW;
5890.          ELSE SEGEND-> SEG.PEND-> PATH.NXT = NEW;
5900.        SEGEND-> SEG.PEND = NEW;
5910.        NEW-> PATH.NXT    = NULL;
5920.  0     DO X = 1 TO PATHLEN;
5930.           NEW-> PATH.NODE( X ) = CURR( X );
5940.        END;
5950.     END STORE_PATH;
5960.  1  ADD_B:   PROC( NX, NY );
5970.  0/*                    ADD NEW BLOCK TO "B" STACK                          */
5980.  0     DCL   ( NX, NY )   FIXED;
5990.        BTOP = BTOP + 1;
6000.        B(BTOP).X = NX;
6010.        B(BTOP).Y = NY;
6020.        IF NX ¬= 0  &  FINV( STACK( NX ) ) ¬= 0   THEN
6030.            DEP( P, FINV( STACK( NX )) ),
6040.            DEP( FINV( STACK( NX )), P ) = 1;
6050.        IF NY ¬= 0  &  FINV( STACK( NY ) ) ¬= 0   THEN
6060.            DEP( P, FINV( STACK( NY )) ),
6070.            DEP( FINV( STACK( NY )), P ) = -1;
6080.     END ADD_B;
6090.  -  POP_B:   PROC( CNT );
6100.  0/*                    REMOVE "CNT" BLOCKS FROM "B" STACK                  */
6110.  0     DCL   CNT   FIXED;
6120.        BTOP = BTOP - CNT;
6130.     END POP_B;
6140.  1  ADD_STACK:   PROC( SIDE, ENT );
6150.  0/*-------------------------------------------------------------------------*/
6160.  0/* ADD LAST VTX ("ENT") OF PATH TO "SIDE" (= L OR R) STACK.               */
6170.  0/*-------------------------------------------------------------------------*/
6180.  0     DCL   (SIDE, ENT)   FIXED;
6190.        STACK( FREE ) = ENT;
6200.        NEXT ( FREE ) = NEXT( SIDE );
6210.        NEXT ( SIDE   ) = FREE;
6220.        FREE          = FREE + 1;
6230.     END ADD_STACK;
6240.  -  POP_STACK:   PROC( SIDE, V, CNT );
6250.  0/*-------------------------------------------------------------------------*/
6260.  0/* REMOVE VTXS >= "V" FROM "SIDE" STACK,                                   */
6270.   /* THEN REMOVE "CNT" VTXS FROM THE SAME STACK.                            */
6280.  0/*-------------------------------------------------------------------------*/
6290.  0     DECLARE
6300.        ( SIDE,
6310.          V,
6320.          CNT,
6330.          Z )                FIXED;
6340.        IF V ¬= 0 THEN
6350.        DO WHILE( NEXT( SIDE ) ¬= 0  &  STACK( NEXT( SIDE )) >= V );
6360.            NEXT( SIDE ) = NEXT( NEXT( SIDE ));
6370.        END;
6380.  0     DO Z = 1 TO CNT;
6390.            NEXT( SIDE ) = NEXT( NEXT( SIDE ));
6400.        END;
```

```
6410.      END POP_STACK;
6420.   1SWITCH_NEXT:   PROC( ONE, OTHER );
6430.   0/*                    EXCHANGE TWO PTRS TO "STACK" ENTRIES.          */
6440.   0  DCL    ( ONE, OTHER, SAVE )   FIXED;
6450.      SAVE           = NEXT( ONE );
6460.      NEXT( ONE )    = NEXT( OTHER );
6470.      NEXT( OTHER)   = SAVE;
6480.    END SWITCH_NEXT;
6490.D   HALT;
6500.    END EMBED;
6510.   1  WRITE_GRAPH:   PROC( ADJ, ID, NU, TITLE );
6520.   0/*-------------------------------------------------------------------*/
6530.   0/* PRINT A GIVEN ADJ MATRIX ALONG WITH A TITLE AND THE VTX NAMES     */
6540.    /* WHICH LABEL THE ROWS AND COLS OF THE MATRIX.                     */
6550.    /* ASSIGN VTX NAMES, "ID", IF NONE ARE GIVEN.                       */
6560.   0/*-------------------------------------------------------------------*/
6570.   0     DECLARE
6580.         ( ADJ( *,* ),
6590.           ID( * ),
6600.           NU,
6610.           R, C )              FIXED,
6620.           TITLE               CHAR(*),
6630.           REPEAT              BUILTIN;
6640.   0     IF NU < 0 THEN DO;
6650.             NU = -NU;
6660.             DO R = 1 TO NU;
6670.               ID( R ) = R;
6680.         END;    END;
6690.   0     PUT FILE( SYSPRINT ) EDIT
6700.           ( TITLE) ( SKIP,A)
6710.           ( '|', (ID(R) DO R=1 TO NU), REPEAT('---', NU))
6720.           ( SKIP(2), COL(7),A, (NU)F(3), COL(4),A )
6730.           (( ID(R),' |',( ADJ(R,C) DO C=1 TO NU ) DO R=1 TO NU ))
6740.           ( COL(3),F(3),A, (NU)F(3) );
6750.      END WRITE_GRAPH;
6760.   1NONPLANAR:    PROC( P );
6770.   0/*                    RESET "PLANAR" SWITCH.                         */
6780.   0  DCL    P    FIXED;
6790.      BADPATH = P;
6800.      PLANAR = '0'B;
6810.    END NONPLANAR;
6820.    END PLANE;
6830.   //GO.SYSIN DD *
6840.     5
6850.         0 1 1 1 1
6860.         1 0 1 1 1
6870.         1 1 0 1 1
6880.         1 1 1 0 1
6890.         1 1 1 1 0
6900.     6
6910.         0 1 0 1 0 1
6920.         1 0 1 0 1 0
6930.         0 1 0 1 0 1
6940.         1 0 1 0 1 0
6950.         0 1 0 1 0 1
6960.         1 0 1 0 1 0
6970.     8
6980.         1 1 0 0 1 0 0 0
6990.         1 0 1 1 1 0 0 0
7000.         0 1 0 0 1 1 0 0
7010.         0 1 0 0 0 1 1 1
7020.         1 1 1 0 0 0 1 0
7030.         0 0 1 1 0 0 1 0
7040.         0 0 0 1 1 1 0 1
```

```
7050.       0 0 0 1 0 0 1 1
7060.    16
7070.       1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
7080.       1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0
7090.       0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0
7100.       0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 0
7110.       0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0
7120.       0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0
7130.       0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0
7140.       0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1
7150.       1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0
7160.       0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0
7170.       0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0
7180.       0 0 0 0 0 1 1 1 0 0 0 0 0 1 0 0
7190.       0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0
7200.       0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0
7210.       0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 1
7220.       0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1
7230.    16
7240.       1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
7250.       1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
7260.       0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0
7270.       0 0 1 0 0 0 1 0 0 1 0 0 0 0 0 0
7280.       0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0
7290.       0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
7300.       0 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0
7310.       0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0
7320.       0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0
7330.       0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0
7340.       0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
7350.       0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0
7360.       0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0
7370.       0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0
7380.       0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1
7390.       0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
```

Appendix F

FURTHER DESIGN DETAILS ON THE CASCADE LAYOUT

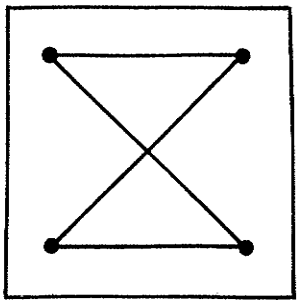
The purpose of this appendix is to present, in a terse form, the design details behind the development of the architecture presented in Figure 4.1.

TIMING DIAGRAM NOTION

In the timing diagram, illustrated on the next two pages, the operand contents of each FIFO and PROCESSOR are presented in the following format:

FIFO	P_j	FIFO
X X X X	X X X X	X X X X

where each P_j internally consists of a butterfly (i.e., one-step trellis component) which for the binary case would be traditionally represented as:

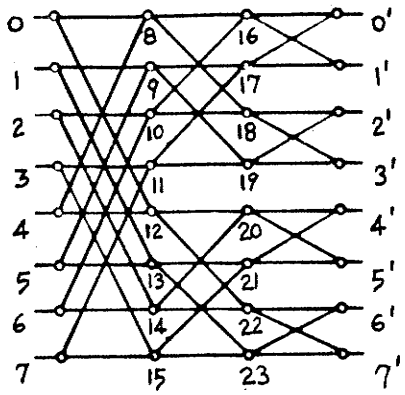


time k	P_1	P_2	P_3	N→A
0	0 X X X / X X / X X X X			
1	1 0 X X / X X / X X X X			
2	2 1 0 X / X X / X X X X			
3	3 2 1 0 / X X / X X X X			
* 4	X 3 2 1 / 0 12 / 4 8 / 12 X X X	8 X / X X / X X		
5	X X 3 2 / 1 13 / 5 9 / 13 12 X X	9 8 / X X / X X		
* 6	X X X 3 / 2 14 / 6 10 / 14 13 12 X	X 9 / 8 18 / 10 16 / 18 X	16 / X X / X	
* 7	X X X X / 3 15 / 7 11 / 15 14 13 12	X X / 9 19 / 11 17 / 19 18	X / 16 1 / 17 0 / 1	0

* = Input y_k delivered to a path metric generator

8	0	X	X	X	X	X	X	X	15	14	13	12	X	X	X	X	19	18	X	X	X	1
					X	X								X	X				X	X		
9	1	0	X	X	X	X	X	X	15	14	13	12	X	X	X	X		X	18	3	3	
					X	X								X	X				19	2		2
10	2	1	0	X	X	X	X	X	X	15	X	13	12	22	22	X		20	X	X	X	3
					X	X							14	20					X	X		
11	3	2	1	0	X	X	X	X	X	X	X	X	13	23	23	22		X	20	5	5	
					X	X							15	21					21	4		4
* 12	X	3	2	1	0	12	12	X	X	X	8	X	X	X	X	23		22	X	X	X	5
					4	8							X	X					X	X		
13	X	X	3	2	1	13	13	12	X	X	9	8	X	X	X	X		X	22	7	7	
					5	9							X	X					23	6		6
* 14	X	X	X	3	2	14	14	13	12	X	X	9	8	18	18	X		16	X	X	X	7
					6	10							10	16					X	X		
* 15	X	X	X	X	3	15	15	14	13	12	X	X	9	19	19	18		X	16	1	1	
					7	11							11	17					17	0		0
16	0	X	X	X	X	X	X	X	15	14	13	12	X	X	X	X	19	18	X	X	X	1
					X	X								X	X				X	X		

The rearranged, three baud interval trellis diagram associated with Figure 4.1 is presented below.



The correspondence between the node numbers in this diagram (what is referred to as operands), and the actual state metrics they represent is presented in the table below.

OPERAND	STATE	OPERAND	STATE	OPERAND	STATE
0	0	8	0	16	0
1	1	9	2	17	4
2	2	10	4	18	1
3	3	11	6	19	5
4	4	12	1	20	2
5	5	13	3	21	6
6	6	14	5	22	3
7	7	15	7	23	7

The clock rate is defined as follows:

$$\text{Clock rate} = \frac{\frac{\#\text{ states}}{\#\text{ processors}}}{\text{baud interval}} = \frac{\#\text{ states}}{(\#\text{ processors}) \cdot (T)}$$

$$= \frac{m^{\nu}}{\nu \cdot T}$$

Derivation of Switch Algorithms

Procedure S1 is a special case to handle operands destined for P_1 which are fed back from processor P_3. The intention is to initially fill the FIFO first with the first half of the naturally ordered states, then feed operands directly into P_1 with the last half of the naturally ordered states.

Procedure S2, S3, S4

Each procedure is associated with a switch which in turn is responsible for the routing of operands in one stage of the trellis.

The switches are controlled by an algorithm similar to that developed in ref [118, p.1052].

Procedure S5 feeds y_k samples to the appropriate path metric generators so P_j can compute state metrics as soon as possible.

In general, the switch setting is controlled by:

If $k \bmod(m^{\nu}) = m^{\nu-1}$ then Path Metric $(y_k) \rightarrow P_1$

If $k \bmod(m^{\nu}) = m^{\nu-1} + m^{\nu-2}$ then Path Metric $(y_k) \rightarrow P_2$

$\vdots$

If $k \bmod(m^{\nu}) = m^{\nu-1} + m^{\nu-2} + m.... + m^{0}$ then Path Metric $(y_k) \rightarrow P_{\nu}$

Appendix G

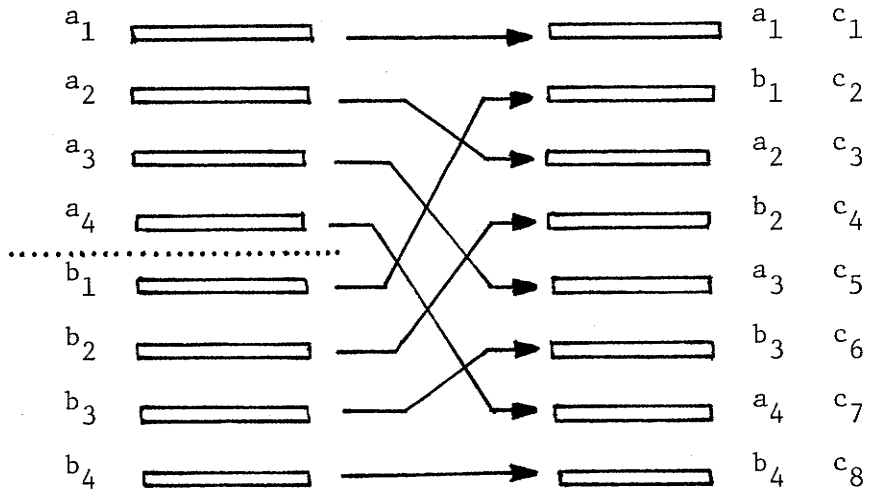
A BRIEF TUTORIAL ON THE SHUFFLE-EXCHANGE CONSTRUCT

The purpose of this appendix is to informally provide additional background material on the SE organization, for the communication engineer not especially well-versed in its function. We pursue the development of the 2-SE organization throughout, to illustrate concepts. In the following pages we move towards getting a mathematical description of such a network and ultimately to a VLSI grid model representation. In conclusion, Figure G.1 demonstrates how the SE construct can be used as a Viterbi receiver for coded modulation.

There are two ways of boosting the efficiency of a processing network: (i) pipelining and (ii) recirculation. The pipelining approach introduces a row of registers between each set or row of processing cells. A new problem can be fed into the network as soon as the previous problem inputs have emerged from the first row of processors. In the recirculation technique, one row of processing cells is used many times during the solution of a single problem. In our case, during each stage of the computation this row simulates the action of one of the stages in the trellis diagram. The state metrics needed by the row as it simulates the $(k+1)^{st}$ stage of the trellis are obtained from the outputs of the k^{th} stage of computation. If the connectivity is not precisely right the state metrics will have to circulate more than once through the network of processors.

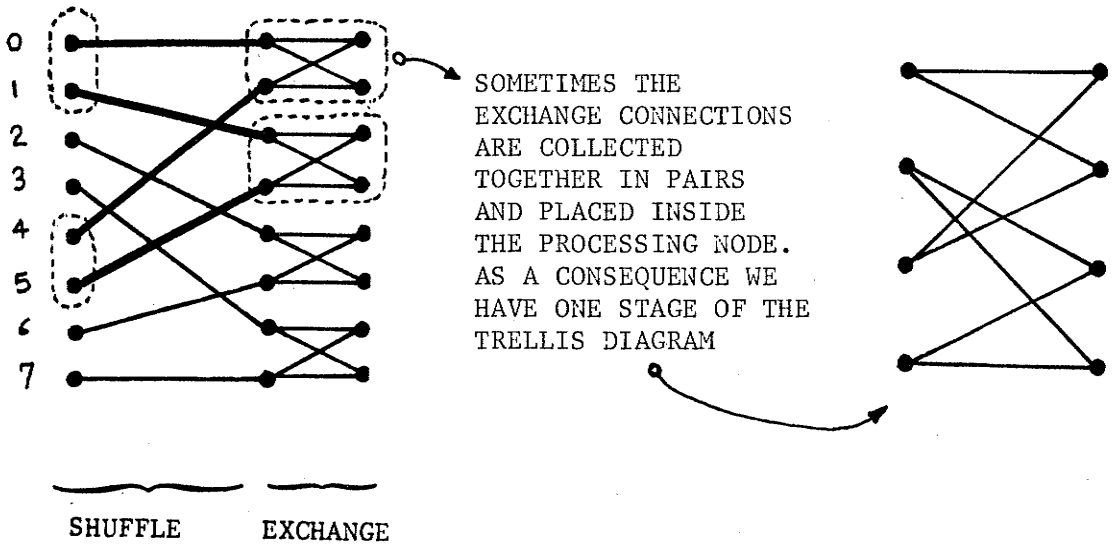
A recirculation network of particular interest is the shuffle-exchange network. Initially, the shuffle-exchange operation itself is presented.

THE SHUFFLE OPERATION CAN BE VISUALIZED BY CONSIDERING A DECK OF CARDS (WHICH MATHEMATICALLY COULD BE REPRESENTED AS A n-ELEMENT VECTOR). THE DECK IS DIVIDED INTO TWO HALVES a AND b, HENCE WE REFER TO THIS AS A 2-SHUFFLE. THE PERFECT SHUFFLE OPERATION CONSISTS OF INTERLEAVING THESE CARDS TO PRODUCE THE STACK OF CARDS c, AS SHOWN BELOW:

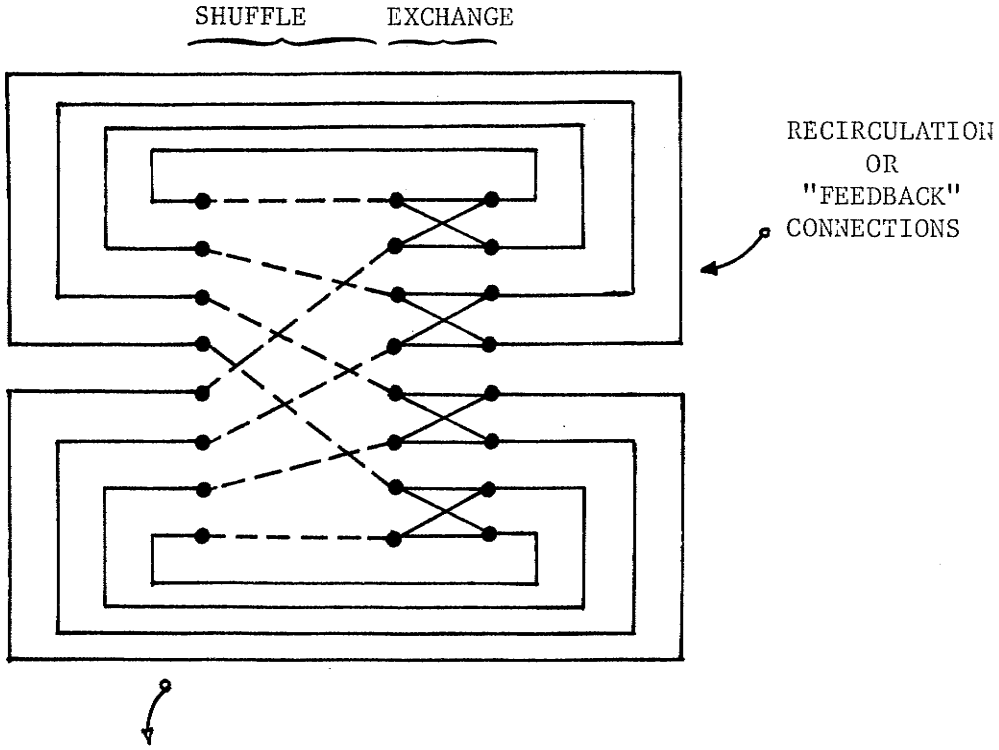


THE EXCHANGE OPERATION CONSISTS OF SWAPPING OR EXCHANGING PAIRS OF CARDS, IN ORDER, THROUGHOUT THE DEPTH OF THE STACK c.

NOW LET'S CONCATENATE THE SHUFFLE AND EXCHANGE OPERATIONS IN ONE DIAGRAM, AS ILLUSTRATED BELOW.

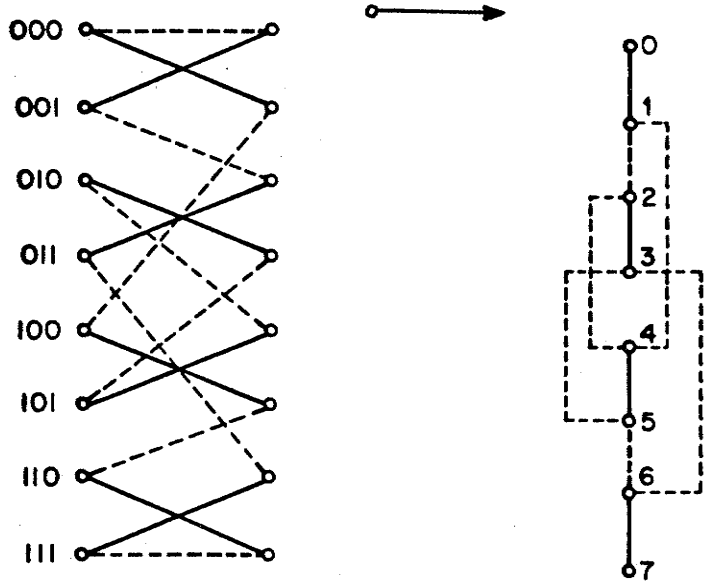


AN EXAMPLE OF A SHUFFLE-EXCHANGE RECIRCULATION NETWORK FOR n=8



FOLD BACK EXCHANGE CONNECTIONS
BY MAKING THE RECIRCULATION
EDGES OF ZERO LENGTH

FOLD THE BIPARTITE GRAPH
ABOUT A VERTICAL AXIS



EACH NODE IS A
PROCESSOR

THERE IS A
STRAIGHTFORWARD
MATHEMATICAL
DESCRIPTION OF
SUCH A NETWORK

Mathematically, the shuffle-exchange recirculation network may be described as follows:

Suppose one has n processors numbered 0, 1, 2, ... , n-1, where n is a power of two (i.e., $n=2^k$)

Associate each processor or node with a unique k-bit binary string or label represented by $a_{k-1}, \cdots, a_0$.

The Perfect Shuffle Operation (PS)

Two nodes w and w' are linked by a shuffle edge if w' is a left or right cyclic shift of w.

i.e.: If $w = a_{k-1}, \cdots, a_0$ then $w' = a_{k-2}, \cdots a_0, a_{k-1}$ or $w' = a_0, a_{k-1}, \cdots, a_1$.

Example: PS(001) = 100 or 010

The Exchange Operation (EX)

Two nodes w and w' are linked by an exchange edge if w and w' differ only in the last bit.

i.e.: If $w = a_{k-1}, \cdots, a_0$ then $w' = a_{k-1}, \cdots, a_1, \bar{a}_0$

Example: EX(001) = 000

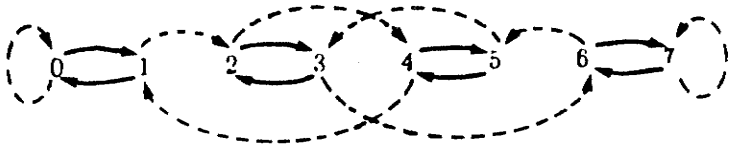
Necklaces

The collection of all cyclic shifts of a node w is called a necklace and is denoted by $< w >$.

Example: $< 001 > = \{ 001, 010, 100 \}$

Now that one can describe such networks mathematically, let us illustrate the shuffle-exchange recirculation network within the VLSI grid model

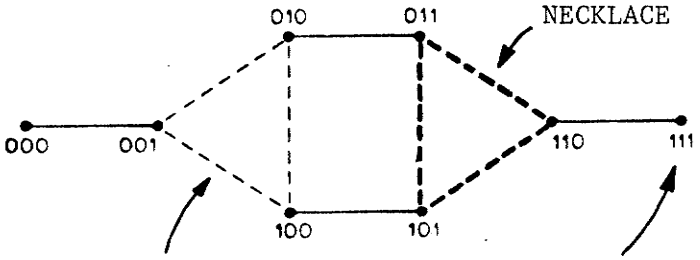
THE SHUFFLE-EXCHANGE GRAPH $n=2^k$ nodes, $\frac{3n}{2}$ edges



SHUFFLE-EXCHANGE INTERCONNECTIONS FOR n=8

---- SHUFFLE

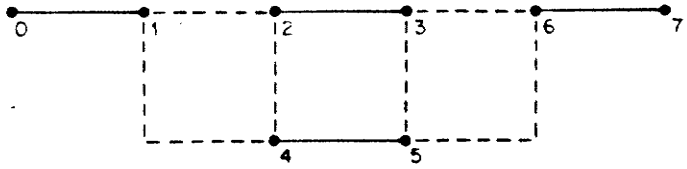
——— EXCHANGE



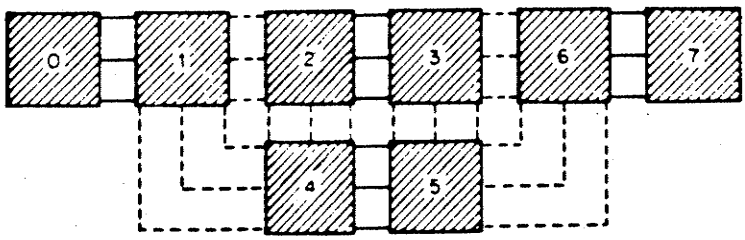
EDGES ARE UNDIRECTED FOR ILLUSTRATION

SELF-LOOPS ARE USUALLY OMITTED

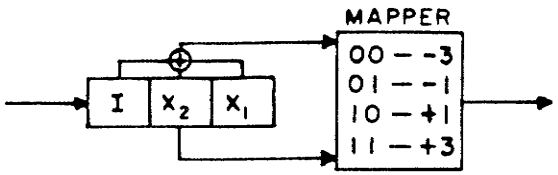
EACH NODE LABEL IS REPLACED BY ITS BINARY EQUIVALENT



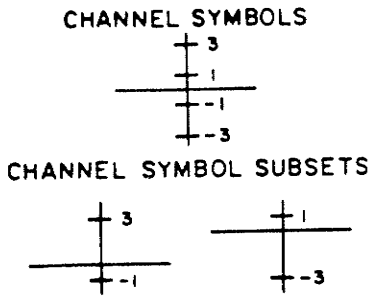
THE SHUFFLE-EXCHANGE LAYOUT IN THE VLSI GRID MODEL



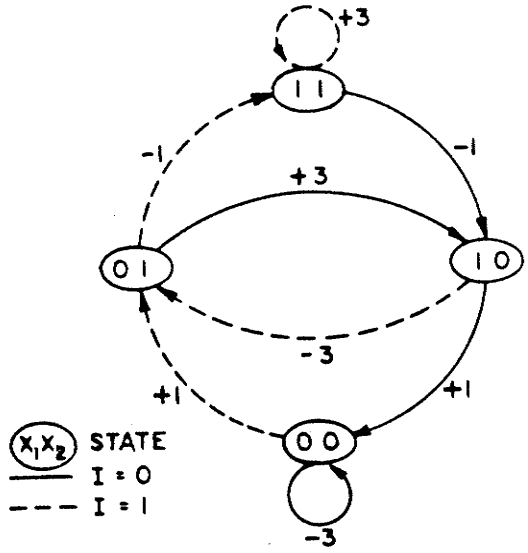
A PRACTICAL IMPLEMENTATION WHERE THE NODES ARE PROCESSORS THAT OCCUPY FINITE O(1) AREA



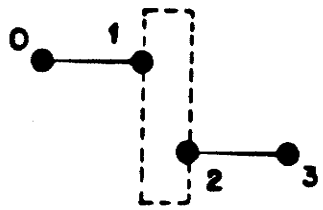
(a)



(b)



(c)



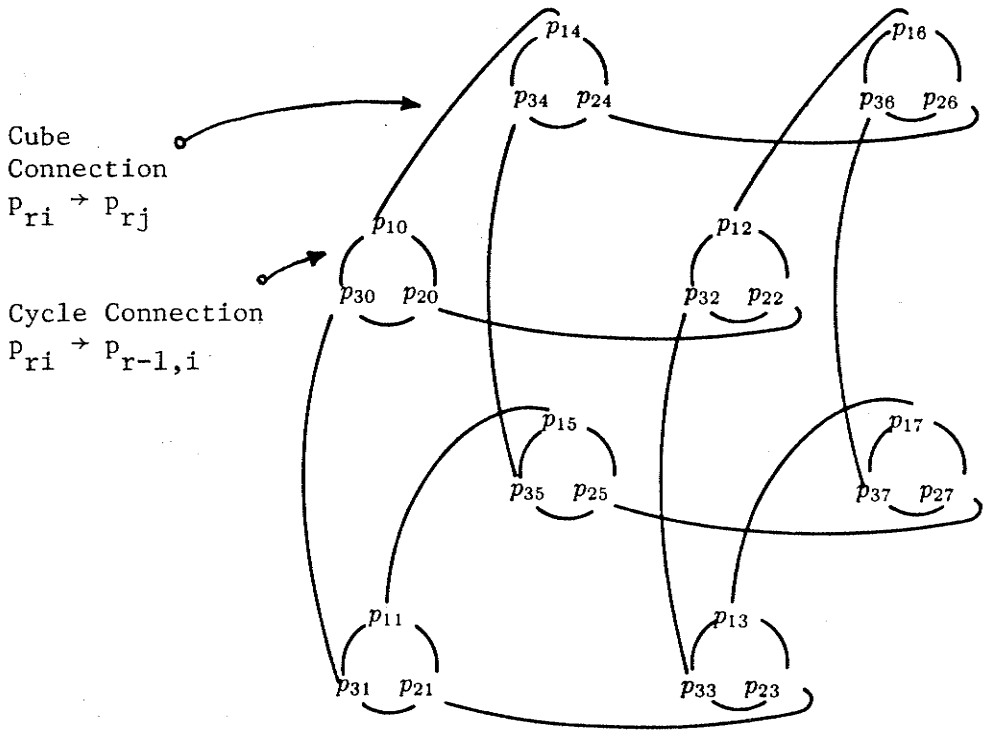
(d)

Figure G.1: USING THE SE CONSTRUCT FOR A CODED MODULATION RECEIVER
(a) The Transmitter: a convolutional encoder and mapper
(b) Mapping Channel Signals by Set Partitioning (see [142])
(c) Corresponding State Diagram for (a)
(d) VLSI Grid Model Layout of the Decoder (A SE graph for P(2) and ν=2)

Appendix H

A BRIEF TUTORIAL ON THE CCC CONSTRUCT

The purpose of this appendix is to provide additional background material on the CCC organization, for the reader not especially well-versed in its function. We pursue one instance of the CCC, that illustrated in Figure 4.16, and analyse it in depth. Shown below is a processor organization which corresponds to Figure 4.16(b). Processor p_{ri} communicates, over what is known as a cube connection, with processor p_{rj} if and only if i and j differ in the r^{th} bit from the left in their binary representations.



The Table on the following page defines the state metrics that each processor evaluates.

Again, to emphasize the power of the CCC organization, Figure H.1 demonstrates how the CCC construct can be used as a Viterbi receiver for decoding punctured convolutional codes.

OPERAND	PROCESSOR	STATE METRIC
0	p10	0
1	p11	2
2	p12	4
3	p13	6
4	p14	1
5	p15	3
6	p16	5
7	p17	7
8	p20	0
9	p21	4
10	p22	1
11	p23	5
12	p24	2
13	p25	6
14	p26	3
15	p27	7
16	p30	0
17	p31	1
18	p32	2
19	p33	3
20	p34	4
21	p35	5
22	p36	6
23	p37	7

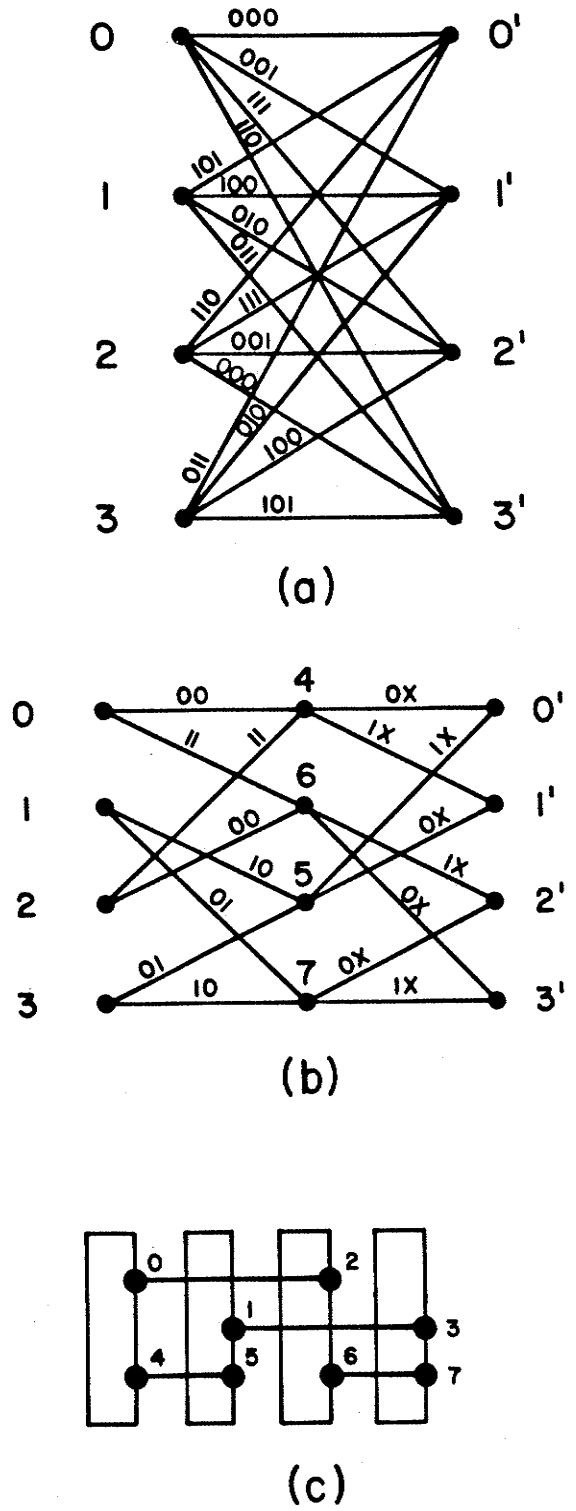


Figure H.1: THE CCC CONSTRUCT FOR DECODING PUNCTURED CONVOLUTIONAL CODES

(a) Trellis Structure for R=2/3, ν=2 code.

(b) Trellis Diagram for R=2/3, ν=2 produced by periodically deleting bits from R=2/3, ν=2 code.

(c) CCC structure corresponding to (b).

REFERENCES

1. Lucky R.W., Salz J., and Weldon E.J., Principles of Data Communication, McGraw Hill, Toronto, 1968.

2. Belfiore C.A. and Park J.H., Decision Feedback Equalization, Proc. IEEE, Vol. 67, No. 8, Aug. 1979, pp. 1143-1156.

3. Forney, G.D., Jr., Maximum - Likelihood Sequence Estimation of Digital Sequences in the Presence of Intersymbol Interference, IEEE Transactions on Information Theory, Vol. IT - 18, No. 3, May 1972, pp. 363 - 378.

4. Viterbi A.J., Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE Trans on Information Theory, Vol. IT-13, April 1967, pp. 260-269.

5. Omura J.K., On the Viterbi Decoding Algorithm, IEEE Trans. on Information Theory, IT-15, Jan. 1969, pp. 177-179.

6. Hayes, J.F., The Viterbi Algorithm Applied to Digital Data Transmission, IEEE Communications Magazine, March 1975, pp. 15-20.

7. Forney, G.D., The Viterbi Algorithm, Proc. IEEE, Vol. 61, March 1973, pp. 268-279.

8. Kobayashi, H., Correlative level coding and maximum likelihood decoding, IEEE Trans. on Information Theory, Vol. IT-17, Sept. 1971, pp. 586-594.

9. Kobayashi, H., Application of probabilistic decoding to digital magnetic recording systems, IBM Journal of Research and Development, Jan. 1971, pp. 64-74.

10. Burkhardt, H., An Event-driven Maximum - Likelihood Peak Position Detector for Run-Length-Limited Codes in Magnetic Recording, IEEE Trans. on Magnetics, Vol. MAG-17, No. 6, 1981, pp. 3337-3339.

11. Koubanitsas, T.S., Application of the Viterbi Algorithm to Adaptive Delta Modulation with Delayed Decision, Proc. IEEE, July 1975, pp. 1076-1077.

12. Tanaka, H., Hirakawa, Y., and Kaneku, S., Recognition of Distorted Patterns Using the Viterbi Algorithm, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI - 4, No. 1, Jan. 1982, pp. 18-25.

13. Hull, J.J. and Srihari, S.N., Experiments in Text Recognition with Binary n-Gram and Viterbi Algorithms, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-4, No. 5, Sept. 1982, pp. 520-530.

14. Martelli A., An Application of Heuristic Search Methods to Edge and Contour Detection, CACM, Vol. 19, No. 2, Feb. 1976, pp. 217-227.

15. Shinghal R. and Toussaint G.T., Experiments in Text Recognition with the Modified Viterbi Algorithm, IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No. 2, April 1979, pp. 184-193.

16. Magee, F.R., Proakis, J.G., Adaptive maximum likelihood sequence estimation for digital signaling in the presence of intersymbol interference, IEEE Trans. on Information Theory, Vol. IT-19, Jan. 1973, pp. 120-124.

17. Lee W., A maximum likelihood sequence estimator with decision feedback equalization, IEEE Trans. on Communications, Vol. COM - 25, No. 9, Sept. 1977, pp. 971-979.

18. Viterbi A.J. and Omura J.K., Principles of Digital Communication and Coding, McGraw-Hill, New York, 1979.

19. Bellman, R.E., Dynamic Programming, Princeton University Press, Princeton, N.J., 1957.

20. Newcombe E.A. and Pasupathy S., Error Rate Monitoring for Digital Communications, Proc. IEEE, Vol. 70, No. 8, Aug. 1982, pp. 805-828.

21. Devijver P.A. and Kittler J., Pattern Recognition: A Statistical Approach, Prentice Hall, Toronto, 1982, pp. 257-273.

22. Andrews H.C., Introduction to Mathematical Techniques in Pattern Recognition, Wiley, 1972, p.35, p.124.

23. Bezdek J.C., Pattern Recognition with Fuzzy Objective Function Algorithms, Plenum Press, 1981, p.48. (Density Functionals).

24. Dubois D. and Prade H., Fuzzy Sets and Systems: Theory and Applications, Academic Press, 1980, p.324.

25. Kailath T., The Divergence and Bhattacharyya Distance Measures in Signal Selection, IEEE Trans. on Communication, Vol. COM-15, No. 1, Feb. 1967, pp. 52-60.

26. Iyanaga S. and Kawada Y. (editors), Encyclopedic Dictionary of Mathematics, MIT Press, Cambridge, Mass., 1980, p. 927.

27. Tou J.T., Gonzalez R.C., Pattern Recognition Principles, Addison-Wesley, 1974, pp.145-151.

28. Portny S.E., Large Sample Confidence Limits for Binary Error Probabilities, Proc. IEEE, Vol. 54, No. 12, 1966, p.1993.

29. Pearl J., Theoretical Bounds on the Complexity of Inexact Computations, IEEE Trans. on Information Theory, IT-22, No. 5, Sept. 1976, pp. 580-586.

30. Dunhan J.G. and Tzou K.-H., Performance Bounds for Convolutional Codes with Digital Viterbi Decoders in Gaussian Noise, IEEE Trans. on Communications, Vol. COM-31, No. 10, Oct. 1983, pp. 1124-1132.

31. Mead, C.A. and Conway, L.A., Introduction to VLSI Systems, Addison-Wesley, 1979.

32. Kung, H.T., Let's Design Algorithms for VLSI, Proc. of the Caltech Conference on Very Large Scale Integration, Jan. 1979, pp. 65-90.

33. Haavind R.C. (ed.), Submicron techniques point to four megabit memory chip, High Technology, February 1983, p. 11.

34. Mead, C.A. and Lewicki, G., Silicon Compilers and foundries will usher in user-designed VLSI, Electronics, August 11, 1982, pp. 107-111.

35. E.G. & G. Reticon Corporation, Product Summary: Discrete Time Analog Signal Processing Devices, Product Note 97050, 1978.

36. E.G. & G. Reticon Corporation, R5403 Analog Correlator/convolver, Product Note 97100, 1979.

37. Eldon J., Convolution - A powerful technique for digital signal processing, LSI publication TP17A - 1/81 TRW LSI products, Dec. 1980.

38. Sawai A., Programmable LSI Digital Signal Processor Development, in VLSI Systems and Computations, edited by H.T. Kung, B. Sproull and G. Steele, Computer Science Press, 1981, pp. 29-40.

39. Bowen B.A. and Brown W.R., VLSI Systems Design for Digital Signal Processing. Vol. 1 - Signal Processing and Signal Processors, Prentice Hall, 1982, pp. 256-287.

40. Chapman R.C., Editor, Digital Signal Processor, Bell System Technical Journal, Vol. 60, No. 7, Part 2, Sept. 1981.

41. Intel Corporation, 2920 Analog Signal Processor Design Handbook, August 1980.

42. Ahmed H.M. and Morf M., Synthesis and Control of Signal Processing Architectures Based on Rotations, VLSI 81, edited by J.P. Gray, Academic Press, 1981, pp. 43-52.

43. Ahmed H.M., Delosme J.M. and Morf M., Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing, Computer, January 1982, pp. 65-82.

44. Lyon F.L., A Bit-Serial VLSI Architectural Methodology for Signal Processing, VLSI 81, edited by J.P. Gray, Academic Press, 1981, pp. 131-140.

45. Buric M.R. and Mead C.A., Bit-Serial Inner Product Processor in VLSI, Proc. of Second Caltech Conference on Very Large Scale Integration, Caltech Computer Science Department, 1981, pp. 155-164.

46. Denyer P.B. and Meyers D.J., Carry-Save arrays for VLSI Signal Processing, VLSI 81, edited by J.P. Gray, Academic Press, 1981, pp. 151-160.

47. Bloch R., Bottcher K., Lacroix A. and Talmi M., Architecture for VLSI - Circuits in Digital Signal Processing, ICC 80, IEEE CH1541 - 5/80, pp. 1184-1187.

48. Elmasry M., Digital Bipolar Integrated Circuits, Wiley, 1983, pp. 225-232.

49. Elmasry M., Thompson P.M., Two-Level Emitter Function Logic Structures for Logic-in-Memory Computers, IEEE Transactions on Computers, Vol. C-24, No. 3, March 1975, pp. 250-259.

50. Kohonen T., Content Addressable Memories, Springer-Verlag, New York, 1980, p. 282.

51. Denny W., Buley E.R., Hatt E., Logic Enhanced Memories: An overview and some examples of their application to a radar tracking problem, Caltech conference on VLSI, January 1979, pp. 173-186.

52. Tanimoto S.L., Systolic Cellular Logic: Inexpensive Parallel Image Processors, IEEE Conf. on Pattern Recognition and Image Processing, 1981, pp. 306-309.

53. Western Digital Corp., "WD8206 Error Detection and Correction Unit, Product Specification", 1983.

54. Kung H.T., Why Systolic Architectures?, Computer, January 1982, pp. 37-46.

55. Kung H.T., The Structure of Parallel Algorithms, Advances in Computers, Vol. 19, M.C. Yovits (editor), Academic Press, 1981, pp. 65-112.

56. Cappello P.R., Steiglitz K., Digital Signal Processing Applications of Systolic Algorithms, VLSI Systems and Computations, edited by H.T. Kung, B. Sproull, and G. Steele, Computer Science Press, 1981, pp. 245-254.

57. Kung H.T., Ruane L.R., Yen D.W., A Two-Level Pipelined Systolic Array for Convolutions, ibid, pp. 255-264.

58. Baudet G.M., Preparata F.P., Vuillemin J.E., Area-Time Optimal VLSI Circuits for Convolution, IEEE Trans. on Computers, Vol. C-32, No. 7, July 1983, pp. 684-688.

59. Fisher A.L., Systolic Algorithms for Running Order Statistics in Signal and Image Processing, VLSI Systems and Computations, edited by H.T. Kung, B. Sproull, and G. Steele, Computer Science Press, 1981, pp. 265-272.

60. Chiang A.M., A New CCD Parallel Processing Architecture, ibid, pp. 408-415.

61. Kerkhoff H.G., Tervoert M.L., Stemerdink J.A., The design and application of a CCD four-valued full adder circuit, Compcon 81, pp. 96-99.

62. Kerkhoff H.G., Tervoert M.L., Multiple-Valued Logic Charge Coupled Devices, IEEE Transactions on Computers, Vol. C-30, No. 9, Sept. 1981, pp. 644-652.

63. Advanced Micro Devices, "Am9518 Data Ciphering Processor, Product Specification", April 1981. (Similar products are produced by Fairchild Corp. and Western Digital Corp.)

64. Rivest, R.L., A Description of a Single-Chip Implementation of the RSA Public-Key Cryptosystem, IEEE 1980 National Telecommunications Conference, Houston, Texas, pp. 49.2.1-49.2.5.

65. Hindin, H.J., Bell algorithm speeds decryption of public-key coding schemes, Electronics, Vol. 54, No. 16, Aug. 11, 1981, pp. 39-40.

66. Krishnan M.S., A Structured Approach to VLSI Layout Design, Proceedings of the Second Caltech Conference on Very Large Scale Integration, Caltech Computer Science Department, January 1981, pp. 413-432.

67. Liu K.Y., Architecture for VLSI Design of Reed-Solomon Encoders, ibid, pp. 539-553.

68. Brown K.O., Dynamic Programming in Computer Science, Report Number, CMU-CS-79-106, Department of Computer Science, Carnegie-Mellon University, February 1979, 44 pages.

69. Ciminera L., Dermartin C., Serra A., Valenzano A., VLSI Structures for Speech Analysis and Pattern Recognition, IEEE Conf. on Pattern Recognition and Image Processing, 1982, pp. 692-697.

70. Guibas L.J., Kung H.T., Thompson C.D., Direct VLSI Implementation of Combinatorial Algorithms, Proc. of Caltech Conference on Very Large Scale Integration, January 1979, pp. 509-525.

71. Chu K.H. and Fu K.S., VLSI Architectures for High Speed Recognition of Context-Free Languages, 9th Annual Symposium on Computer Architecture, April 1982, pp. 43-49.

72. Rader C.M., Memory Management in a Viterbi Decoder, IEEE Transactions on Communications, Vol. COM-29, No. 9, Sept. 1981, pp. 1399-1401.

73. Gilhousen K.S., Heller J.A., Jacobs I.M. and Viterbi A.J., Coding Systems Study for high data rate telemetry links, NASA Report, Jan. 1971, Prepared under contract NAS2-6024 by Linkabit Corp., San Diego, CA., 235 pages.

74. Synder J.S., High-Speed Viterbi Decoding of High-Rate Codes, Sixth International Conference on Digital Satellite Communications, Sept. 1983, pp. XII-16 - XII-23.

75. Crozier S., Wilson M., Moreland K.W., Camelon J. and McLane P., Microprocessor based implementation and testing of a simple Viterbi detector, Canadian Electrical Engineering Journal, Vol. 6, No. 3, 1981, pp. 3-8.

76. Conan J., An F8 Microprocessor-Based Breadboard for the Simulation of Communication Links Using Rate 1/2 Convolutional Codes and Viterbi Decoding, IEEE Transactions on Communications, Vol. COM-31, No. 2, February 1983, pp. 165-171.

77. Conan J., Implementation of Microprocessor based Viterbi decoders, Proc. Int. Symp. Mini and Microprocessors, Zurich, Switzerland, June 1978, pp. 87-93.

78. Orndorff R.M., Krcmarik J.D., Colesworthy R.J., Doak T.W., and Koralek R., Viterbi Decoder VLSI Integrated Circuit for Bit Error Correction, 16th Annual Asilomar Conference on Circuits, Systems and Computers, Nov. 8-10, 1982, 4 pages.

79. Acampora A. and Gilmore R.P., Analog Viterbi Decoding for High Speed Digital Satellite Channels, IEEE Trans. on Communications, Vol. COM-26, No. 10, October 1978, pp. 1463-1470.

80. Thompson C.D. A Complexity Theory for VLSI, Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon University, 1980.

81. Bilardi G., Pracchi M., Preparata F.P., A critique and an Appraisal of VLSI Models of Computation, VLSI Systems and Computations, edited by H.T. Kung, B. Sproull and G. Steele, Computer Science Press, Oct. 1981, pp. 81-88.

82. Seitz C.L., Self timed VLSI Systems, Caltech Conference on VLSI, January 1979, pp. 345-355.

83. Kissin G., Measuring Energy Consumption in VLSI circuits, Ph.D. thesis, Dept. of Computer Science, University of Toronto, in preparation.

84. Preparata F.P. and Vuillemin J.E., The Cube-Connected-Cycles: A Versatile Network for Parallel Computation, Communications of the ACM, Vol. 24, No. 5, May 1981, pp. 300-309.

85. Thompson C.D., Area-Time Complexity for VLSI, Proceedings of the 11th Annual ACM Symposium on Theory of Computing, May 1979, pp. 81-88.

86. Mead C. and Rem M., Cost and Performance of VLSI Computing Structures, IEEE Journal of Solid State Circuits, Vol. SC-14, April 1979, pp. 455-462.

87. Brent R.P. and Kung H.T., On the Area of Binary Tree Layouts, Information Processing Letters, No. 11, 1980, pp. 44-46.

88. Leiserson C.E., Area-efficient graph layouts (for VLSI), Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, Oct. 1980, pp. 270-281.

89. Vuillemin J.E., A combinatorial limit to the computing power of VLSI circuits, ibid, pp. 294-300.

90. Baudet G.M., On the area required by VLSI circuits, VLSI Systems and Computations, edited by H.T. Kung, B. Sproull and G. Steele, Computer Science Press, Oct. 1981, pp. 100-107.

91. Kleitman D., Leighton F.T., Lepley M., Miller G.L., New Layouts for the shuffle-exchange graph, Proc. of the 13th ACM Symposium on Theory of Computing, May 1981, pp. 278-292.

92. Leighton F.T., New Lower Bound Techniques for VLSI, Proc. of the 22nd Annual Symposium on Foundations of Computer Science, 1981, pp. 1-12.

93. Patterson M.S., Ruzzo W.L., Snyder L., Bounds on minimax edge length for Complete Binary trees, Proc. 13th Annual ACM Symposium on Theory of Computing, May 1981, pp. 293-299.

94. Ruzzo W.L. and Snyder L., Minimum edge planar embeddings of trees, VLSI Systems and Computations, edited by H.T. Kung B. Sproull and G. Steele, Computer Science Press, Oct. 1981, pp. 119-123.

95. Valiant L.G., Universality considerations in VLSI circuits, IEEE Trans. on Computers, Vol. C-30, No. 2, February 1981, pp. 135-140.

96. Hoey D. and Leiserson C.E., A layout for the shuffle - exchange network, Proc. 1980 Int. Conf. Parallel Processing, Aug. 1980, pp. 329-336.

97. Steinberg D., Rodeh M., A Layout for the Shuffle Exchange Network with 0(N**2/(log N)**(3/2)) Area, IEEE Trans. on Computers, Vol. C-30, No. 12, Dec. 1981, pp. 977-982.

98. Leighton F.T., A Layout Strategy for VLSI which is Provably Good, 14th Annual ACM Symposium on the Theory of Computing, 1982 pp. 85-98.

99. Yao A.C., The Entropic Limitations on VLSI Computations, 13th Annual ACM Symp. on Theory of Computing, May 1981, pp. 309-311.

100. Burks A.W. (editor), Essays on Cellular Automata, University of Illinois Press, Urbana, 1970, 375 pages.

101. Gulak, P.G., Self-Reproducing Automata - Implications for VLSI, A Seminar presented to the E. E. Departments of the University of Manitoba, Winnipeg, Nov. 3, 1981, and the University of Toronto, Toronto, Dec. 14, 1981, 36 pages, 22 references.

102. Thompson C.D., VLSI Complexity of Sorting, IEEE Trans. on Computers, Vol. C-32, No. 12, Dec. 1983, pp. 1171-1184.

103. Rabiner L.R. and Gold B., Theory and Application of Digital Signal Processing, Prentice-Hall, pp. 609-616.

104. Leighton F.T. and Leiserson C.E., Wafer Scale Integration of Systolic Arrays, Proc. IEEE 23rd Annual Symposium on the Foundations of Computer Science, 1982, pp. 297-311.

105. Leiserson C.E., Area Efficient VLSI Computation, ACM-MIT Press Doctoral dissertation award series, MIT Press, 1983, 136 pages. (Origin: Com-Sci Dept., CMU, October 1981)

106. Aho A., Hopcroft J., Ullman J., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.

107. Thompson C.D. and Kung H.T., Sorting on a Mesh-Connected Parallel Computer, Communications of the ACM, Vol. 20, 1977, pp. 263-271.

108. Ullman J.D., Computational Aspects of VLSI, Computer Science Press, Rockville, Maryland, 1984.

109. Knuth D.E., The Art of Computer Programming, Vol. 3: Sorting and Searching, Reading, MA: Addison-Wesley, 1973.

110. Stone H.S., Parallel processing with the Perfect Shuffle, IEEE Trans. on Computers, Vol. C-20, No. 2, Feb. 1971, pp. 153-161.

111. Stanley W.D., Digital Signal Processing, Reston Publishers, New York, 1975, p. 269.

112. Haynes L.S., Lau R.L., Siewiorek D.P., Mizell D.W., A Survey of Highly Parallel Computing, Computer, Jan. 1982, pp. 9-24.

113. Chen P-Y., Lawrie D.H., Yew P-C., Padua D.A., Interconnection Networks using Shuffles, Computer, December 1981, pp. 55-64.

114. Golomb S.W., Permutations by Cutting and Shuffling, SIAM Review, Vol. 3, No. 4, October 1961, pp. 293-297.

115. Morris S.B., Hartwig R.E., The Generalized Faro Shuffle, Discrete Mathematics, Vol. 15, 1976, pp. 333-346.

116. Kodandapani K.L., Pradan D.K., A Generalization of Shuffle-Exchange Networks, Proc. Information Science and Systems, Princeton, 1980, pp. 523-530.

117. Bhuyan L.N., Agrawal D.P., Design and Performance of a General Class of Interconnection Networks, Proc. of the 1982 Conference on Parallel Processing, Aug. 24-27, 1982, pp. 2-9.

118. Thompson C.D., Fourier Transforms in VLSI, IEEE Trans. on Computers, Vol. C-32, No. 11, Nov. 1983, pp. 1047-1057.

119. Leighton F.T. and Miller G.L., Optimal Layouts for Small Shuffle Exchange Graphs, VLSI 81, J.P. Gray (Editor), Academic Press, pp. 289-300.

120. Hopcroft J. and Tarjan R., Efficient Planarity Testing, JACM, Vol. 21, No. 4, October 1974, pp. 549-568.

121. Deo N., Graph Theory with Applications to Engineering and Computer Science, Prentice Hall, 1974.

122. Leighton F.T., New Lower Bound Techniques for VLSI, Ph.D. dissertation, Mathematics Department, Massachusetts Institute of Technology, Aug. 1981.

123. Pomerance C., The Search for Prime Numbers, Scientific American, Dec. 1982, pp. 136-147.

124. Lin S. and Costello D.J., Error Control Coding, Prentice Hall, 1983, p.441.

125. Wann D.F. and Franklin M.A., Asynchronous and Clocked Control Structures for VLSI Based Interconnection Networks, IEEE Trans. on Computers, Vol. C-32, No. 2, March 1983, pp. 284-293.

126. Browning S.A., Communication in a Tree Machine, Proc. of the Second Caltech Conf. on VLSI, Jan. 1981, pp. 510-526.

127. Sabety T.M., Ibrahim H. and Shaw D.E., VLSI Design for the NON-VON Project at Columbia University, VLSI Design, March/ April 1983, pp. 71-72.

128. Thompson C.D., Area-Time Complexity for VLSI, Caltech Conf. on VLSI, Jan. 1979, pp. 495-508.

129. Peltzer D.L., Wafer-Scale Integration: The Limits of VLSI?, VLSI Design, September 1983, pp. 43-47.

130. Grinberg J., Nudd G.R. and Etchells R.D., A Cellular VLSI Architecture, Computer, Jan. 1984, pp. 69-81.

131. Frenette N., Peppard L.E. and Lodge J.H., A CMOS Implementation of a Binary Correlator for Digital Communications, 1983 Conference on Very Large Scale Integration, University of Waterloo, Oct. 24-25, 1983, pp. 108-111.

132. Rosenberg A.L., Three-Dimensional VLSI: A Case Study, Journal of the Association of Computing Machinery, Vol. 30, No. 3, July 1983, pp. 397-416.

133. Goodman J.W., Leonberger F.J., Sun-Yuan Kung and Athale R.A., Optical Interconnections for VLSI Systems, Proc. IEEE, Vol. 72, No. 7, July 1984, pp. 850-866.

134. McLane P.J., The Viterbi Receiver for Correlative Encoded MSK Signals, IEEE Trans. on Communications, Vol. COM-31, No. 2, Feb. 1983, pp. 290-295.

135. deJager F. and Dekker C.B., Tamed Frequency Modulation, A Novel Method to Achieve Spectrum Economy in Digital Transmission, IEEE Trans. on Communications, Vol. COM-26, No. 5, May 1978, pp. 534-542.

136. deBuda R., Coherent Demodulation of Frequency-Shift Keying with Low Deviation Ratio, IEEE Trans. on Communications, Vol. COM-20, June 1972, pp. 429-435.

137. Scharf L.L., Cox D.D. and Masreliez C.J, Modulo-2π Phase Sequence Estimation, IEEE Trans. on Information Theory, Vol. IT-26, No. 5, Sept. 1980, pp. 615-620.

138. Mazur B.A. and Taylor D.P., Demodulation and Carrier Synchronization of Multi-H Phase Codes, IEEE Trans. on Communications, Vol. COM-29, No. 3, March 1981, pp. 257-263.

139. Cain J.B., Clark G.C., and Geist J.M., Punctured Convolutional Codes of Rate (n-1)/n and Simplified Maximum Likelihood Decoding, IEEE Trans. on Information Theory, Vol. IT-25, No. 1, Jan. 1979, pp. 97-100.

140. Bilardi G. and Preparata F.P., An Architecture for Bitonic Sorting with Optimal VLSI Performance, IEEE Trans. on Computers, Vol. C-33, No. 7, July 1984, pp. 646-651.

141. Moldovan D.I., On the Design of Algorithms for VLSI Systolic Arrays, Proc. IEEE, Vol. 71, No. 1, Jan. 1983, pp. 113-120.

142. Ungerboeck G., Channel Coding with Multilevel/Phase Signals, IEEE Trans. on Information Theory, Vol. IT-28, No. 1, Jan. 1982, pp. 55-67.