OFFICE OF NAVAL RESEARCH

FINAL REPORT

For The Period

August 1, 1986 - July 31, 1989

Contract N00014-86-K-0726

Task No. NR4148061

SYSTEMATIC METHODS FOR DESIGN OF VLSI SIGNAL PROCESSING ARRAYS FOR COMMUNICATIONS APPLICATIONS

Thomas Kailath
Information Systems Laboratory
Department of Electrical Engineering
Stanford University
Stanford, CA 94305-4055

---

1 Introduction

This is the final report on our work under ONR Contract N00014-86-K-0726, August 1, 1986 through July 31, 1989.

The major results are in two areas:

1. Studies of systematic design procedures for a class of structured algorithms often encountered in signal processing applications. These are what we have called Regular Iterative Algorithms (RIAs) for which our results are summarized in Section 2.

   It might be mentioned that these ideas have been successfully used by one of our former students who helped to develop this theory, Dr. S. K. Rao of AT&T Bell Laboratories in Holmdel, N.J. Dr. Rao has found the RIA results helpful in designing several fast integrated circuit chips for communications and signal processing applications, some of which are being used in the AT&T – ZENITH joint effort on High Definition Television (HDTV).

2. The other major area of effort was the study of a notable family of algorithms that are not in the RIA form, viz., those associated with Viterbi decoding of convolutional and trellis codes or more generally with shortest-path problems in graphs.

   This work, which is described in Section 3, is also being followed by Dr. P. G. Gulak, a postdoctoral scholar and research associate on the contract, who is now teaching at University of Toronto, Canada. Dr. Gulak is having special chips designed and built by Bell Northern Research.

2 Summary of Our Work on Regular Iterative Algorithms

Our previous work has shown that (see *e.g.* , [14, 15, 32, 33, 34, 35, 41]) that once a Regular Iterative Algorithm is designed for a given problem, then one can use the systematic design theory developed by us to generate efficient processor arrays. However, most algorithms are not given to the designer in the RIA form and most initial representations are either sequential in nature (*e.g.*, FORTRAN or PASCAL programs) or general mathematical expressions. We have thus developed a formal methodology for systematic formulation of RIAs starting from representations that we refer to as linearly indexed Assignment Codes. It can be shown that such codes are very close to the mathematical expressions of a wide variety of problems, especially in signal processing and matrix algebra.

In this section, we shall first briefly introduce RIAs and summarize our contributions in the analysis and implementation of such algorithms. We shall then briefly summarize our formal methodology for deriving RIAs starting from general representations such as mathematical formulas.

2.1 Regular Iterative Algorithms and Our Contributions

A formal definition of RIAs can be found in [17, 35, 41]; here we shall introduce RIAs via a simple example.

Example (2-D Filtering Algorithm): It can be shown (see [32, 35]) that certain numerically stable 2-D filtering algorithms due to Deprettere and Dewilde [5], Vaidyanathan and Mitra [47], and Fettweis [6], can all be written in the form:

$$\textbf{For all } (i,j,k), \textbf{ where } 0 \le i \le n \text{ and } 0 \le j,k \le N, \textbf{ do}$$

$$x(i,j+1,k+1) = f_{x,i}(x(i,j,k), y(i,j,k), w(i,j,k))$$

$$y(i+1,j,k) = f_{y,i}(x(i,j,k), y(i,j,k), w(i,j,k))$$

$$w(i-1,j,k) = f_{w,i}(x(i,j,k), w(i,j,k))$$

where $f_{x,i}$, $f_{y,i}$, $f_{w,i}$ are linear functions that are determined by a synthesis procedure.

□

The example displays the following (characteristic) features of an RIA:

> Each variable in the RIA is identified by a label (*e.g.*, x, y or w in example 1) and an *index vector* (*e.g.* , $I = [i\ j\ k]^T$, in example 1). The set of all index points over which the variables of the RIA are defined is called the *index space*, which is a subset of the an S-dimensional integer lattice, Z^S.
>
> The dependences among the variables are regular with respect to the index points. That is, if $x_1(I)$ is computed using the value of $x_2(I - \mathbf{d_{12}})$ then the *index displacement vector* $\mathbf{d_{12}}$, corresponding to this direct dependence, is the same regardless of the index point I.

The set of computations performed at every index point is often referred to as the *iteration unit* of the RIA. Also, note that although the direct dependences among the variables in an RIA are required to be independent of the index points, the actual computations carried out to evaluate these variables can depend on the index point. In general, the index space I will be semi-infinite along certain coordinates and bounded along others. The bounds on the coordinates will be referred to as the *size parameters* of the RIA.

The regular dependences of an RIA lead to a dependence graph with an iterative structure, which can be clearly demonstrated by embedding the dependence graph within the index space. That is, a set of V nodes is defined at every index point I in the index space I, where the i^{th} node represents the variable $x_i(I)$ in the RIA. As first noted by Karp *et al.* [17] and by Waite [50], the regularity of the dependence graph of an RIA can be concisely expressed in terms of a simpler and smaller graph called the *Reduced Dependence Graph* (RDG). The RDG of an RIA (see Fig. 1) has one node for each of the indexed variables in the RIA; it has a directed arc from node x_i to node x_j,

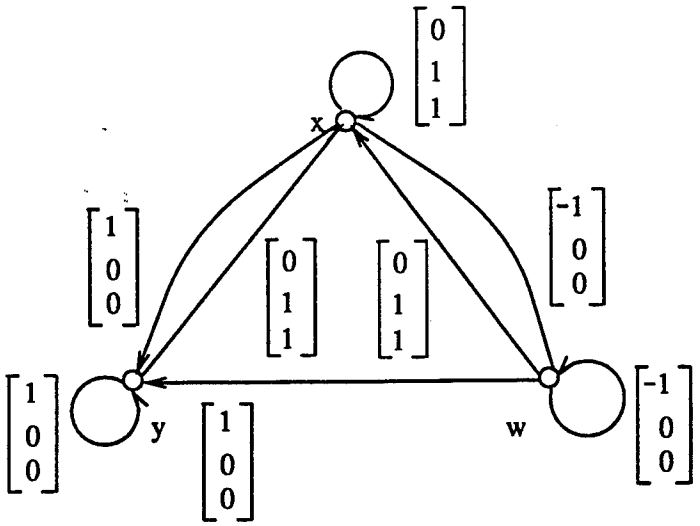

Figure 1: The RDG of the RIA in the above example.

if $x_j(I)$ is computed using the value of $x_i(I - \mathbf{d_{ij}})$ for some $\mathbf{d_{ij}}$; finally, each directed arc is assigned a vector weight representing the displacement of the index point across the direct dependence. We should note that the RDG and a specification of the index space $\mathbf{I}$, completely characterize the dependence graph of an RIA; hence, the analysis of parallelism in an RIA is based on the analysis of the RDG instead of the larger dependence graph.

Some of our important results are enumerated below; for a detailed account of the work reported here previous work please see [32, 41]

1. A formal definition of systolic arrays was obtained that captured their generally accepted properties, especially regularity (mostly identical processors), spatial locality (local interconnections), temporal locality (no delay-free operations, or more precisely, all combinational elements are latched) and pipelined operation (throughput independent of the order, suitably defined, of the system). Some authors (*e.g,* Leiserson *et al.* [24]) had used only a subset of these properties, but the consensus in the literature appeared to have required all those mentioned above (see *e.g.,* [34] and [21]).

2. A reasonable generalization of the concept of systolic arrays that allowed implementation of a larger class of algorithms (including of course all systolic algorithms) was also developed. The generalization allowed the presence of register pipelines of various lengths at different points in a regular array of (mostly) identical processors, and sometimes also some LIFO (Last-In-First-Out) buffers. Such architectures have almost all the advantages that make systolic arrays so appealing for VLSI; the only added requirement is that some of the processors may require

certain amount of memory in them. We should note here that the memory requirement is not a major bottleneck, and certain commercial products such as the WARP developed at CMU, routinely provide such on-processor memory.

Rao *et al.* called such arrays Regular Iterative Arrays, and algorithms implementable on such arrays were dubbed as Regular Iterative Algorithms. It is convenient to use the acronym RIA to stand for either of these concepts, the exact one to be inferred from the context. Using the above concepts, and their consequences, one can show for example that there are Regular Iterative Algorithms (*e.g.* , RIAs for certain classes of 2-D filtering algorithms, RIAs for certain pivoting algorithms [41, 36, 37] *etc.*) that cannot be implemented on systolic arrays, as formally defined, but can be implemented on regular iterative arrays.

3. It was also shown [15, 21, 32, 41] that many algorithms in digital filtering (convolution, correlation, autoregressive, and moving-average filtering), numerical linear algebra, discrete methods for PDEs and ODEs, graph theory (transitive closure, some coloring problems) can be reformulated as RIAs. Systematic procedures for converting algorithms into RIAs, however, remained as an open problem.

4. For any RIA, formal methods to determine lower bounds on I/O latency and memory requirements were developed; systematic procedures for implementing RIAs on regular processor arrays that can achieve the lower bound on I/O latency were also proposed (see [32, 33, 35, 41]). We should mention here that these formal mapping techniques can generate all possible architectures, though in practice one stops once a few efficient (*i.e.* , meeting certain performance lower bounds) arrays have been obtained.

5. In the design of systolic arrays, several issues such as systematic procedures for designing *multi-rate* systolic arrays were resolved. In the conventional systolic array designs all operations were assumed to take the same amount of time; this led to unrealistic and inefficient design. Our design procedure allows one to carry out the design with more realistic processor modules that can increase the throughput by exploiting the fact that the time required to carry out different operations is generally different.

2.2 Systematic Formulation of RIAs

Let us first introduce the concept of *localized algorithms* that are close to RIAs (see *e.g.* , [41, 40, 39, 16, 42]). The definition of the localized algorithms is motivated by the observation that there are certain problems that can be solved by algorithms that have *regular* dependence graphs that are not completely homogeneous. That is, the dependence graphs may have dependencies or computations that are present only in certain portions of the dependence graphs. As we shall discuss in [41], one way of handling such cases is to assume that the dependences and the computations are present

everywhere in the index space and then to apply the results for RIAs. There are several problems where this approach is reasonable; for example the Gaussian elimination algorithm without pivoting can be first written in the localized algorithm form and then can be implemented on processor arrays by modeling the localized algorithm as an RIA. The other approach is to break up the dependence graph into more than one component such that the dependence graph is homogeneous over each component. The mapping techniques can then be applied to each such component with special consideration to the dependences at the boundaries between the components. The latter approach is discussed in more detail in [41] where the example of Gauss-Jordan elimination algorithm is worked out.

The localized algorithms have statements of the form

$$x_i(I) = f_i(x_1(I - d_{i1}), \cdots, x_V(I - d_{iV})) \quad \forall I \in \mathbf{I_i}. \tag{1}$$

Thus each statement in this algorithm may have a different index space of its own; as a comparison, all statements in an RIA have the same index space.

Partial attempts have been made by several authors, including [18], [25], [20], [27], [4] and [10], to formalize the conversion procedure for going from an initial representation to an RIA or a localized algorithm. The first step always is to convert algorithms into equivalent Single Assignment Codes (SACs) and the second step tries to localize the dependences by eliminating broadcasts. Single assignment codes [2] are representations where every variable defined in the algorithm takes on a unique value during the course of execution. The fact that the dependence graph of an algorithm can be easily determined from its SAC, has made SACs a very useful starting representation for parallel implementations of algorithms. Considering its importance, a lot of work has been done in trying to convert sequential algorithms into SACs, see *e.g.*, [26]. However, sequential algorithms are not the only representations from which SACs can be derived. Often SACs can be derived systematically from given mathematical expressions. Consider a mathematical expression for matrix multiplication

$$\textbf{For all} \quad \text{tuples } (i,j)\,,\ 1 \le i,j \le n \textbf{ do}$$

$$c_{ij} := \sum_{\text{for all } 1 \le k \le n} a_{ik} \cdot b_{kj}. \tag{2}$$

and a SAC

$$\textbf{For all} \quad \text{triples } (i,j,k), \quad 1 \le i,j,k \le n \textbf{ do}$$

$$c(i,j,k+1) := c(i,j,k) + a_{ik} \cdot b_{kj} \tag{3}$$

In the mathematical expression, the ordering of operations in the inner product is not specified and in fact it can be arbitrary because of the commutativity and associativity of the operation $+$. However, in the given SAC the ordering is fixed and a degree of freedom has been lost. Since the original representation has more freedom and potential parallelism in it, it would be desirable to

make it the starting representation and then systematically derive one or more SACs from it. It turns out that a number of algorithms can be written in the form of (2) (see [41, 40, 39]), and we shall refer to such representations as *Assignment Codes* (ACs). The prefix *'Single'* has been intentionally dropped to emphasize the fact that in such representations the number of inputs for computing a variable may depend on the problem size, as opposed to a conventional SAC where the number of inputs to every variable is restricted to be some constant, independent of the problem size. From now on we shall refer to the number of inputs to a variable as its in-degree and the number of variables that a particular variable is input to as its out-degree. If the in- or out-degree of a variable depends on the problem size, then we shall define it to be unbounded. Thus, the variables in ACs can have unbounded in- and out-degrees, whereas in SACs the variables have bounded (*i.e., constant*) in-degrees but may have unbounded out-degrees.

We shall further restrict ourselves to *linearly indexed* ACs, which can be shown to be very close to mathematical expressions for a number of problems, especially in signal processing and matrix algebra. A linearly indexed AC has statements of the form

$$x(PI + \mathbf{d}) \text{ depends on } y(QI + \mathbf{e}) \text{ for all } I \in \mathbf{I} \subset \mathbf{Z}^S \tag{4}$$

where P and Q are integral matrices independent of I, $\mathbf{I}$ is an index space which is the set of all lattice points enclosed within a specified region in a S-dimensional Euclidean space and $\mathbf{d}$, $\mathbf{e}$ are constant displacement vectors. P and Q are often referred to as the *indexing matrices*. We have shown in [41, 40, 39] that in- and out-degrees of variables x and y are completely determined by the structure and dimension of the right null-space of each of the indexing matrices. Many algorithms are actually directly available as (4), and examples include the formulas for matrix multiplication, any m-dimensional convolution/correlation, matrix transposition, and solving matrix Lyapunov's equation. Algorithms that are not directly in the form of (4) can often be easily put in that form by analyzing their sequential representations (see [41]).

Example 2: The formula for matrix multiplication is:

$$\textbf{For all} \quad \text{tuples } (i,j)\,,\ 1 \le i,j \le n \ \textbf{do}$$

$$c_{ij} := \sum_{\text{for all } 1 \le k \le n} a_{ik} \cdot b_{kj}$$

The index space of the example is $\mathbf{I} = \{(i,j,k) \mid 1 \le i,j,k \le n\}$. There is one functional relation in the given AC with the dependence matrices

$$P_c = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad Q_{ac} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad Q_{bc} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

and the displacement vectors $\mathbf{d}, \mathbf{e} = 0$. □

We have shown that a linearly indexed AC can be systematically decomposed into a linearly indexed

SAC and a linearly indexed *dual SAC*. Linearly indexed dual SACs may have variables with unbounded in-degrees but bounded out-degrees, as opposed to the linearly indexed SACs, which have variables with bounded in degrees but possibly unbounded out degrees. Formal procedures will be then outlined for converting linearly indexed SACs and linearly indexed dual SACs into localized algorithms. The conversion of linearly indexed SACs to localized algorithms involves eliminating global dependencies by propagating variables in a systematic manner in the index space. The conversion of dual SACs to localized algorithms is achieved by distributing computations and introducing an ordering among the computations. The two conversion procedures turn out to be duals of each other. We should mention here that starting with linearly indexed ACs is by no means essential in our approach; if one cannot find a AC easily, then one can try to use other well-known techniques and start the procedure with a linearly indexed SAC.

In summary, we have developed a hierarchical procedure for going from a higher level representation of an algorithm to a localized algorithm, which can be described by an RIA or a localized algorithm. It can be described as follows:

> Mathematical Description → Linearly Indexed Assignment Codes → Linearly Indexed Single Assignment Codes and dual Single Assignment Codes → Localized Algorithms.

The conversion procedure is by no means unique and a number of localized algorithms can be generated starting from the same AC. To enable an efficient choice we have also developed procedures to directly schedule and analyze linearly indexed codes of the form (4). For example, we have developed necessary and sufficient conditions for determining whether a sequence of SACs of the form (4) can be scheduled using affine schedules (see [41, 16, 42]). Procedures to schedule linearly indexed codes that do not admit affine schedules are also discussed in [41].

3 Summary of Our Work on Parallel Architectures for Viterbi Decoders

In this section we shall summarize our work on the development of parallel architectures for the optimal decoding of convolutional and trellis codes using the Viterbi Algorithm.

The Viterbi Algorithm (VA) [48], [49], [9], which is widely used in communication systems using convolutional and trellis codes, is essentially an algorithm for finding a minimum distance path in a so-called trellis diagram. There have been several implementations of the VA ranging from totally sequential to fully parallel multiprocessor implementations based on the state transition diagrams of the encoders. By restricting ourselves to codes generated by linear systems over GF(q), we are able to present some apparently novel parallel architectures for decoding rate k/n linear codes, and asymptotically area-efficient implementations for the Thompson VLSI grid model. For $k = 1$,

an equivalence mapping between de Bruijn graphs and shuffle-exchange networks is used to relate previously proposed architectures for decoding rate $1/n$ codes to the straight forward architectures based on the state transition graphs of their encoders. Although these architectures result fast implementation, they require global communication, Hence, we have also studied implementations that have only local communications and require less silicon area.

In the rest of this summary we shall briefly describe our work and relate it to the work done by other researchers.

3.1 The Viterbi Algorithm and its Parallel Implementations

The basic theory behind the VA is readily available in the literature; a good survey was provided by Forney [9], who introduced the concept of the trellis exposition of the decoding algorithm. The *trellis diagram*, is a graphical representation of the state diagram of the encoder drawn as a function of discrete time. Fig. 3 shows the state transition and trellis diagrams for the binary rate 1/2 convolutional encoder shown in Fig. 2. Each time step corresponds to a single symbol interval T (defined as the time interval between two consecutive output symbols of the receiving channels), and defines one stage of the trellis. Each node at every stage of the trellis diagram represents one possible state of the encoder. If q is the number of possible alphabets and v is the number of memory elements, then each stage of the trellis has q^v nodes. There is an edge between the node S_i^t (*i.e.*, the node representing state S_i at stage t) and the node S_j^{t+1} if and only if there is a directed edge from state S_i to S_j in the state transition diagram of the encoder. If there are k inputs to the encoder circuit then each state has q^k predecessors; equivalently, the in-degree of every node in the trellis diagram is q^k.

Each edge (*i.e.*, $S_i^t \rightarrow S_j^{t+1}$) is assigned a weight, called the *branch-metric*. The VA is a dynamic programming algorithm for finding a minimum-distance path in the weighted trellis diagram [28] and can be described as follows. Each node of the trellis diagram has a path metric and a survivor sequence associated with it. The path metric P_j^t of the node S_j at time t is the weighted length of a minimum-distance path between the starting node S_0^0 and the node S_j^t in the trellis diagram; the survivor sequence Q_j^t for state S_j at time t is the state sequence associated with this minimum-distance path. Once every symbol interval, the path metrics are updated as follows:

$$P_j^{t+1} = \min(P_i^t + \lambda_{ij}^{t+1}) \quad \forall\, i \text{ such that } S_i \rightarrow S_j \tag{5}$$

where P_0^0 and $S_i \rightarrow S_j$ implies that there is a valid state transition from state S_i to S_j. The old survivor sequence of the winning ancestor is augmented with the symbol corresponding to the transition to state S_j to form the new survivor sequence for the state S_j. After a sufficiently long time L, (see *e.g.* [49]) the survivor sequence of the state with the minimum path metric is chosen to be the estimate for the state sequence of the encoder; the decoding procedure is then

completed by determining the input sequence corresponding to the estimated state sequence. [In actual implementations, many practical issues must be accommodated such as truncation of the survivor sequence, extraction of the estimated source symbols, and path metric overflow control in finite-word length registers.]

Thus, at each node of the trellis diagram, the VA has to perform q^k additions and comparisons (one for each predecessor) to update the path metrics and survivor sequences; hence, the total number of operations to be executed during a symbol interval is $O(q^{v+k})$ [1]. A sequential implementation would require q^{v+k} addition and comparison operations and q^v random accesses to the processor's memory during each symbol interval T. Hence, the throughput rate of such an implementation decreases exponentially with increase in the constraint length and may not be acceptable in applications where high data rates are required.

One can however trade time for hardware and implement the VA in parallel with increased hardware complexity but reduced T (hence, higher data rates). An intuitively obvious architecture can be obtained by projecting the trellis diagram along the time direction and implementing the VA on the resulting architecture, which will consist of a set of q^v processors connected according to the state transition diagram of the encoder. We shall refer to this architecture as the *fully parallel architecture* for implementing the VA. At every time step, each processor performs the operations represented by (5), *i.e.* , it receives the output symbol and generates the branch metrics for each of its predecessor; then it has to perform q^k additions and select the minimum among them followed by updating of the survivor sequence. Hence, for a rate k/n encoder the symbol interval T is $O(q^k)$ and a gain of $O(q^v)$ over a sequential processor has been achieved.

3.1.1 Previous Work

Kriete and Cain [1] used a fully parallel architecture (along with several tricks in order to keep the path metrics small) to implement the VA for a rate 1/2 feedforward encoder with constraint length $v = 7$ in VLSI (Very Large Scale Integrated) circuit technology. However, they did not address issues such as area-efficient implementations of Viterbi decoders for arbitrary constraint length, or alternate parallel architectures that may be simpler than the architectures based on the state diagrams. Lower silicon area enables the designer to put more processors on a single chip and yet have a high yield; it also may lead to higher speed by reducing the length of interconnecting wires. And of course, if the decoder architecture is large compared to the available resources, then one requires efficient strategies for executing the larger sized problem on the available resources.

Gulak and Shwedyk [12], [11] showed that by carefully analyzing the trellis diagram one can

[1] A quantity $f(N)$ is said to be $O(g(N))$ if there exists $k > 0$ such that $f(N) < kg(N)$ for sufficiently large N. It is $\Omega(g(N))$ if for some $k_1 > 0$, $f(N) > k_1 g(N)$ for sufficiently large N. Finally, $f(N)$ is $\Theta(g(N))$ if it is both $O(g(N))$ and $\Omega(g(N))$.

map the VA algorithm for the special case of rate $1/n$ FIR encoders onto well-known architectures for parallel processing called *shuffle-exchange* networks [44], [46]. Optimum VLSI layouts are already known [19], [22], [45], [13] for such networks and hence the same layouts can be used for Viterbi decoders. Moreover, it is well-known that shuffle-exchange networks are functionally equivalent to a whole family of other popular networks such as hypercubes, cube-connected cycles, butterflies, omega networks, *etc.*, [30], [46]; thus the VA for rate $1/n$ FIR encoders can be efficiently implemented on any of these architectures; however, among these architectures, the shuffle-exchange networks have the least VLSI area for the same number of nodes. We should also comment that, with hindsight, the relationship between trellis diagrams of rate $1/n$ FIR encoders and shuffle-exchange networks should come as no surprise. As early as 1973, Forney [9] and then Rader [31] had noted the equivalence between the trellis diagrams of rate $1/n$ feedforward encoders and the dependence graphs of FFT algorithms and the fact that FFTs can be efficiently implemented on shuffle-exchange networks had been well-known even before that [44]. However, in our work we have presented a much more direct connection between the trellis diagram and the shuffle exchange networks by pointing out a simple procedure for mapping the state transition graphs (which are known as de Bruijn graphs or Good's diagrams) to the shuffle exchange graphs.

A different family of parallel implementations was presented by Chang and Yao [3]. They interpreted the VA (both for rate $1/n$ and rate k/n feed-forward convolutional encoders) as a sequence of matrix-vector multiplications (where the usual $+$ operation is replaced by the *min* operation and the usual multiplication operation is replaced by addition) and then implemented the VA using systolic architectures already developed in the literature for matrix-vector multiplication. The implementation uses $O(q^v)$ processors; however, the symbol interval T is at best $O(q^v/v)$ and thus the gain in speed over that of the sequential processor is at most qv. Hence, with an exponential number of processors, the gain achieved in throughput rate is at best logarithmic.

3.1.2 Our Contributions

Let us first consider the fully parallel architectures where several questions remain unanswered in this area. In particular, rate k/n convolutional codes ($k > 1$) have better distance properties and lower error probabilities than rate $1/n$ codes [49] and are widely used in practice. However, fully parallel architectures for decoding rate k/n codes based on the state transition diagrams become complicated and no alternative simpler architectures or area-efficient VLSI implementations seem to have been described in the literature.

In our work we observed that the state transition diagram of any rate $1/n$ feedforward encoder, which is a de Bruijn graph (also referred to as a Good's diagram), can be directly mapped to a shuffle-exchange network. The simple mapping technique has been independently discovered by several researchers [23], [29], [38], and can be used to show that optimal VLSI layouts for de Bruijn

graphs can be obtained by suitably modifying the optimal layouts of shuffle-exchange networks. The resulting layouts of N nodes de Bruijn graphs have area of $O(N^2/\log^2 N)$ and are only a constant factor larger than the layouts of the corresponding shuffle-exchange networks. There may be, however, advantages in implementing de Bruijn networks because unlike shuffle-exchange networks, they are fault tolerant and work efficiently in the presence of a single faulty node or link [43]. We also show that the state diagrams of rate $1/n$ encoders with feedback when realized in a certain canonical form still have the structure of de Bruijn graphs; hence, the decoder architectures for feedforward encoders can be used to decode codes generated by encoders with feedback.

For rate k/n feedforward encoders realized in an 'obvious' manner, we have shown that the state diagrams can be represented as Cartesian products of k, possibly distinct, de Bruijn graphs. The resulting product graph representation is much simpler than the original state transition diagram and architectures based on the representation does not suffer any loss in performance. Minimum area VLSI layouts for the product graphs are presented using a recursive layout technique that uses the optimal layout strategy for de Bruijn networks. Also, we prove that the optimal layouts of the product graphs save at least a factor of $O(q^k)$ in silicon area compared to the direct layouts of the state transition diagrams. It is also shown that under certain conditions one may choose to implement syndrome decoding based on the state diagram of the dual encoder. The dual encoders are always feedforward [7], [8] and may have simpler state diagrams (if $k > n/2$) than the original encoder.

For the general case of rate k/n encoders with feedback, one can use a result of Forney [7] that every rate k/n convolutional code can be regarded as having been generated by a minimal encoder, which is necessarily feedforward and has minimum constraint length. Hence, any rate k/n code can be decoded by a decoder based on the state transition diagram of a corresponding feedforward minimal encoder, with the resulting codeword converted to the encoder input by a trivial linear operation. Thus in general, the VA for any rate k/n encoder can be always implemented in parallel on a set of processors connected according to a Cartesian product of de Bruijn graphs.

The fully parallel architectures, discussed above, are fast, but require global communications and large silicon area. Hence, with Dr. P. G. Gulak, a postdoctoral scholar and later a research associate, we have also studied implementations that have only local communications and require less silicon area. The study of such implementations have led us to the design of so-called *cascade architectures* that can be shown to have the best (area) × (pipeline-period) product. A paper on this and related architectures has been published. [P. G. Gulak and Thomas Kailath, "Locally Connected VLSI Architectures for The Viterbi Algorithm", *Journal on Selected Areas in Communication*, 6, pp. 527-537 April 1988.]

3.2 VLSI Implementation

In the past year, considerable effort has been put in by Dr. P. G. Gulak into utilizing theoretical insights generated by our work to obtain VLSI implementations for Viterbi decoders. The project is a rather long one and Dr. Gulak has been continuing it at University of Toronto, Canada, where he is an Assistant Professor. An experimental VLSI implementation of a soft-decision, binary rate 1/2, constraint length three, convolutional decoder in 3 μm CMOS technology has been carried out. It has been submitted to Bell Northern Research (BNR) for fabrication and a set of chips is expected to be tested by the end of this year. A brief description of the chip is presented below.

The architecture is based on a number of simple processors, each performing an Add-Compare-Select (ACS) operation, and connected according to the state transition graph of the encoder, which is a de Bruijn graph for rate $1/n$ codes. For our experimental design, the decoder has four processors (each processor corresponds to one state of the rate 1/2, constraint length three encoder) as shown in Fig. 4. Two entries, namely a path (or state) metric and a survivor sequence, are stored at each node. The survivor sequence, represents the earlier symbols which led to the presumed present state, through valid state transitions. The state metric is the weight associated with the survivor sequence. Once each symbol interval, state metrics are updated by adding a measure of unlikelihood, called a branch metric, defined by valid state transitions from each possible state into a given present state. The lowest resulting sum defines the new state metric for the given present state. The old survivor sequence in the survivor path of the winning ancestor, is appended with the symbol corresponding to the transition to this state, to form the new survivor path for this state.

During a preliminary layout to find the approximate size and shape of the various cells in the 3 μm CMOS, a four processor Viterbi decoder was determined to have 4mm square die. Various other design parameters were decided through Monte-Carlo simulations. For example, simulations demonstrated that for relatively high signal-to-noise ratios, a 6-bit survivor sequence allowed the sequences to merge before the delayed estimates were generated. A soft decision scheme was adopted and again the simulations showed that it was sufficient to maintain state metrics to 4-bit resolution. It was also decided to generate the branch metrics off-chip and use 8 four-pin ports to bring them on-chip. Although this meant that 32-pins of a 40 pin package would be used, it allows for the flexibility of using an external look-up ROM so changes in the branch metric are facilitated. Besides the branch metrics, four additional connections are made to the chip, namely, one each for power and ground, one for the system clock and one for the estimated output sequence for a total of 36 pins. The VLSI layout is presented in Fig. 5.

The operation of the chip is as follows. As shown in Fig. 4, there are two transitions into each state or processor. Thus each processor receives the two corresponding state metrics from the buses that cross the center of the chip (see Fig. 5). Meanwhile, the off chip hardware uses observations of the received symbol to look up a ROM table and generate branch metrics for both the transitions

into that state. These four-bit values arrive via busses from outside. The state metric and the branch metric are added together in separate 4 by 6-bit ripple carry adders for each of the two input paths. The two sums are compared by a 6-bit comparator and a select signal is generated, indicating which sum is lower. While these operations are in progress, the survivor sequences are delivered to the appropriate processing elements. The 6-bit survivor sequence, selected by the above comparison is saved in a D-type latch on the rising edge of the clock. The select line then gets the proper state metric into the normalizer. Only if all four of the processors select state metrics with values greater than 15, will each of the processors normalize its metric by subtracting 16 from it. This prevents wrap-around overflow while involving only the upper two bits. On the falling edge of the clock, the 6-bit state metric is latched, thus replacing the information from the previous symbol interval. The oldest bit of the survivor sequence is the estimate of the received symbol. All other bits in the survivor sequence register are shifted over by one position and the newest bit is appended to indicate the next trellis decision in the path history. The procedure then repeats with the next received symbol.

References

[1] J. B. Cain and R. A. Kriete. A VLSI r=1/2, k=7 Viterbi Decoder. *Proc. of NAECON*, May 1984.

[2] S. J. Celoni and J. L. Hennessy. SAL-A Single Assignment Language for Parallel Algorithms. Technical report, Computer Systems Laboratory, Dept. of Electrical Eng., Stanford University, Stanford, California, July 1981.

[3] C. Y. Chang and K. Yao. Systolic Array Processing of the Viterbi Algorithm. *Submitted to IEEE Transactions on Information Theory*, June 1986.

[4] J. M. Delosme and I. C. F. Ipsen. An Illustration of a Methodolgy for the Construction of efficient Systolic Architectures in VLSI. In *Proceedings Second Int. Symp. on VLSI Tech.* Taipei, Taiwan, 1985.

[5] E. Deprettre and P. Dewilde. Orthogonal cascade realization of real multi-port digital filters. Technical report, Network Theory Section, Delft Univ., The Netherlands, 1980.

[6] A. Fettweis. Digital Filter Structures Related to classical Filter Networks. *Arch. Eleck. Ubertragung*, 25:79–89, Feb. 1971.

[7] G. D. Forney. Convolutional Codes I: Algebraic Structure. *IEEE Transactions On Information Theory*, IT-16, No. 6:720–738, Nov. 1970.

[8] G. D. Forney. Structural Analysis of Convolutional Codes via Dual Codes. *IEEE Transactions on Information Theory*, IT-19:512–518, July 1973.

[9] G. D. Forney. The Viterbi Algorithm. *Proceedings of The IEEE*, 61, No. 3:268–278, March 1973.

[10] J. A. B. Fortes and D. I. Moldovan. Data broadcasting in linearly scheduled array processors. *Proc. 11th Ann. Symp. on Computer Architecture*, pages 224–231, June 1984.

[11] P. G. Gulak. *VLSI Structures For Digital Communications.* PhD thesis, University of Manitoba, Winnipeg, Canada, Dec. 1984.

[12] P. G. Gulak and E. Shwedyk. VLSI Structures for Viterbi Receivers: Part I- General Theory and Applications. *IEEE Journal on Selected Areas in Communications*, SAC-4:142–154, Jan. 1986.

[13] Dan Hoey and C. E. Leiserson. A layout for shuffle-exchange network. *Proc. 1980 Int. Conf. on Parallel Processing*, August 1980.

[14] H. V. Jagadish. *Techniques for the Design of Parallel and Pipelined VLSI Systems for Numerical Computations.* PhD thesis, Stanford University, Stanford, California, Dec. 1985.

[15] H. V. Jagadish, S. K. Rao, and T. Kailath. Multi-processor architectures for iterative algorithms. *Proceedings of the IEEE*, 75, No. 9:1304–1321, Sept. 1987.

[16] T. Kailath and V. P. Roychowdhury. Scheduling Lineary Indexed Assignment Codes. In *SPIE Symp. on High Speed Computing II, Los Angeles, CA.*, pages 118–129, Jan. 1989.

[17] R. M. Karp, R. E. Miller, and S. Winograd. The organization of computations for uniform recurrence equations. *Journal of the ACM*, 14:563–590, 1967.

[18] R. M. King. Research on the synthesis of concurrent computing systems. *Proc. of the 10th Symp. on Computer Architecture*, pages 39–46, 1983.

[19] D. Kleitman, F. T. Leighton, M. Lepley, and G. L. Miller. New Layouts for the shuffle-exchange graph. *Proc. of the 13th ACM Symposium on Theory of Computation*, pages 278–292, May 1981.

[20] R. H. Kuhn. Transforming algorithms for single-stage and VLSI architectures. *Proc. Workshop on Interconnection Networks for Parallel and Distributed Processing*, pages 11–19, Apr. 1980.

[21] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1987.

[22] F. T. Leighton. *Layouts for the shuffle-exchange graph and lower bound techniques for VLSI.* PhD thesis, Department of Mathematics, Massachusetts Institute of Technology, 1981.

[23] F. T. Leighton. Theory of parallel and VLSI computation. Seminar Notes; available as MIT/LCS/RSS from Lab. of Computer Science, MIT, MA 02139, 1986.

[24] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing synchronous circuitry by retiming. *Proceedings of the CalTech Conference on VLSI*, pages 87–116, March 1983.

[25] D. I. Moldovan. On the design of algorithms for VLSI systolic arrays. *Proceedings of the IEEE*, pages 113–120, Jan. 1983.

[26] Y. Muraoka. *Parallelism Exposure and Exploitation in Programs.* PhD thesis, University of Illinois, Urbana-Champaign, Feb. 1971.

[27] H. Nelis, E. F. Deprettere, and J. Bu. Automatic design and partitioning of VLSI systolic/wavefront arrays. In *Proc. SPIE Conference 1987, San Diego*, 1987.

[28] J. K. Omura. On the Viterbi Algorithm. *IEEE Transactions Information Theory*, IT-15:171–179, Jan. 1969.

[29] D. K. Pradhan and M. R. Samatham. The de bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI. *to appear in IEEE Transactions on Computers.*

[30] F. P. Preparata and Jean Vuillemin. The Cube-Connected Cycles: A Versatile Network for Parallel Computation. *Communications of the ACM*, 24:300–309, May 1981.

[31] C. M. Rader. Memory management in a Viterbi decoder. *IEEE Trans. on Communications*, COM-29:1399–1401, 1981.

[32] S. K. Rao. *Regular Iterative Algorithms and their Implementation on Processor Arrays.* PhD thesis, Stanford University, Stanford, California, 1985.

[33] S. K. Rao. *Systolic Arrays and their Extensions.* Prentice Hall, to appear in 1989.

[34] S. K. Rao and T. Kailath. What is a Systolic Algorithm? In *SPIE Proceedings.* Real-Time Signal Processing, 1986.

[35] S. K. Rao and T. Kailath. Regular Iterative Algorithms and their Implementations on Processor Arrays. *Proceedings of the IEEE*, 6, no.3:259–282, March 1988.

[36] V. P. Roychowdhury and T. Kailath. Regular Processor Arrays for Matrix Algorithms with Pivoting. *Int. Conf. on Systolic Arrays*, San Diego, CA,:237–246, May 1988.

[37] V. P. Roychowdhury and T. Kailath. Regular Processor Arrays for Matrix Algorithms with Pivoting. Revised and resubmitted to Comm. of ACM, July, 1989.

[38] V. P. Roychowdhury, T. Kailath, and P. G. Gulak. de bruijn Networks and their Optimal VLSI layouts. *Submitted to IEEE Trans. on Computers*, 1988.

[39] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. *IEEE Workshop on VLSI Signal Processing*, Monterey, CA,:459–470, Nov. 1988.

[40] V. P. Roychowdhury, L. Thiele, S. K. Rao, and T. Kailath. On the Localization of Algorithms for VLSI Processor Arrays. Submitted to IEEE Trans. on Computers, September, 1988.

[41] Vwani P. Roychowdhury. *Derivation, Extensions, and Parallel Implementations of Regular Iterative Algorithms.* PhD thesis, Stanford University, Stanford, California, Dec. 1988.

[42] Vwani P. Roychowdhury. *Derivation, Extensions, and Parallel Implementations of Regular Iterative Algorithms.* PhD thesis, Stanford University, Stanford, California, Dec. 1988.

[43] M. L. Schlumberger. *de Bruijn Communications Networks.* PhD thesis, Stanford University, Stanford, California, Dec. 1985.

[44] Harold S. Stone. Parallel Processing with the Perfect Shuffle. *IEEE Transactions On Computers*, c-20, No. 2:153–161, Feb. 1971.

[45] C. D. Thompson. *A Complexity Theory for VLSI.* PhD thesis, Dept. of Comp. Science, Carnegie Mellon University, Pittsburgh,PA, 1980.

[46] J. D. Ullman. *Computational Aspects of VLSI.* MIT Press and John Wiley Sons, Inc., 1981.

[47] P. P. Vaidyanathan and S. K. Mitra. A general theory and synthesis procedure for low sensitivity digital filters. ECE Report, Dept. of Electrical Eng., UCSB, California, Sept. 1982.

[48] A. J. Viterbi. Error Bounds for Convolutional Codes and an asymptotically optimum decoding algorithm. *IEEE Transactions Information Theory*, IT-13:260–269, Apr. 1967.

[49] A. J. Viterbi and J. K. Omura. *Principles of Digital Communication and coding.* New York: McGraw Hill, 1979.

[50] W. M. Waite. Path detection in multi-dimensional iterative arrays. *Journal of the ACM*, 14, 1967.

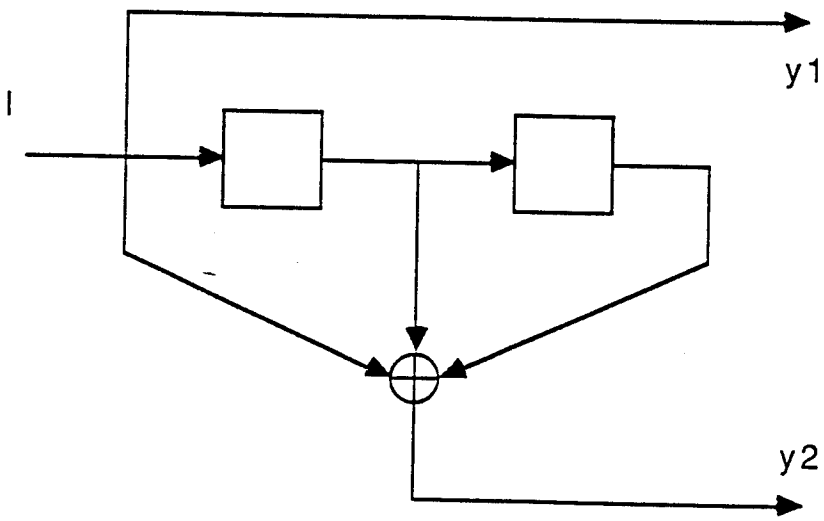

Figure 2: A binary rate 1/2 convolutional encoder with constraint length 2.

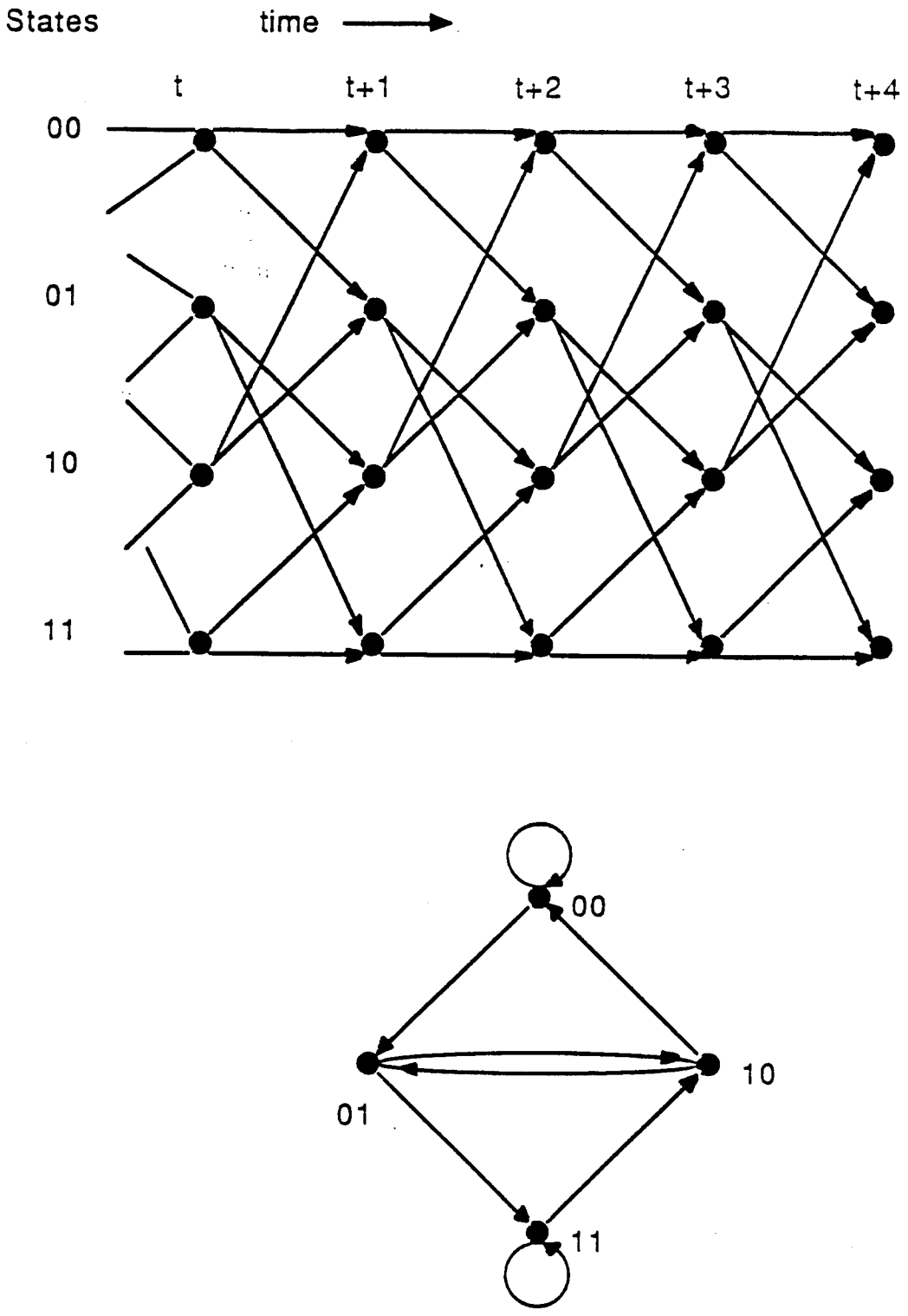

Figure 3: The trellis diagram and the state transition diagram of the encoder in Fig. 1

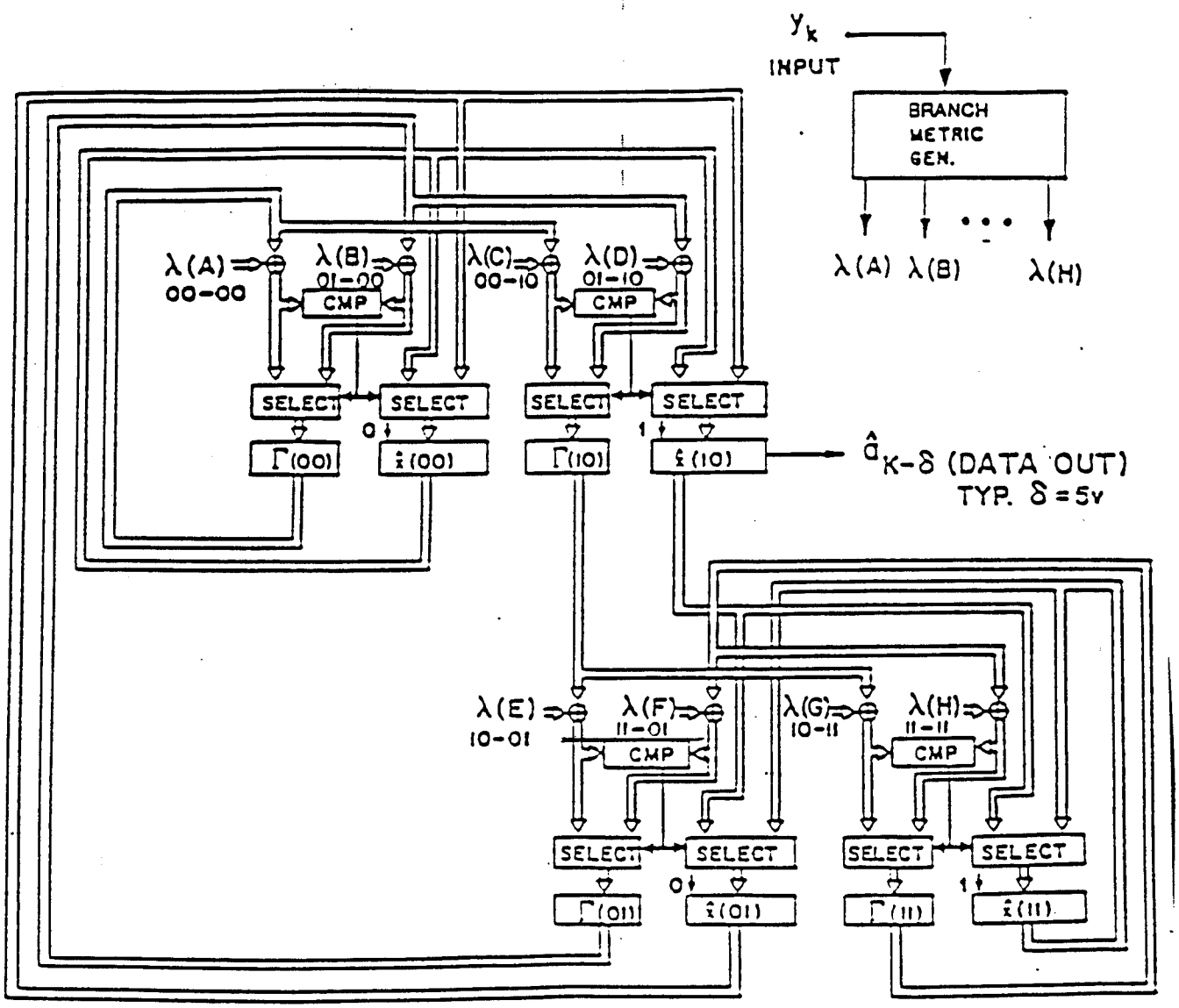

Figure 4: A fully parallel Viterbi decoder.

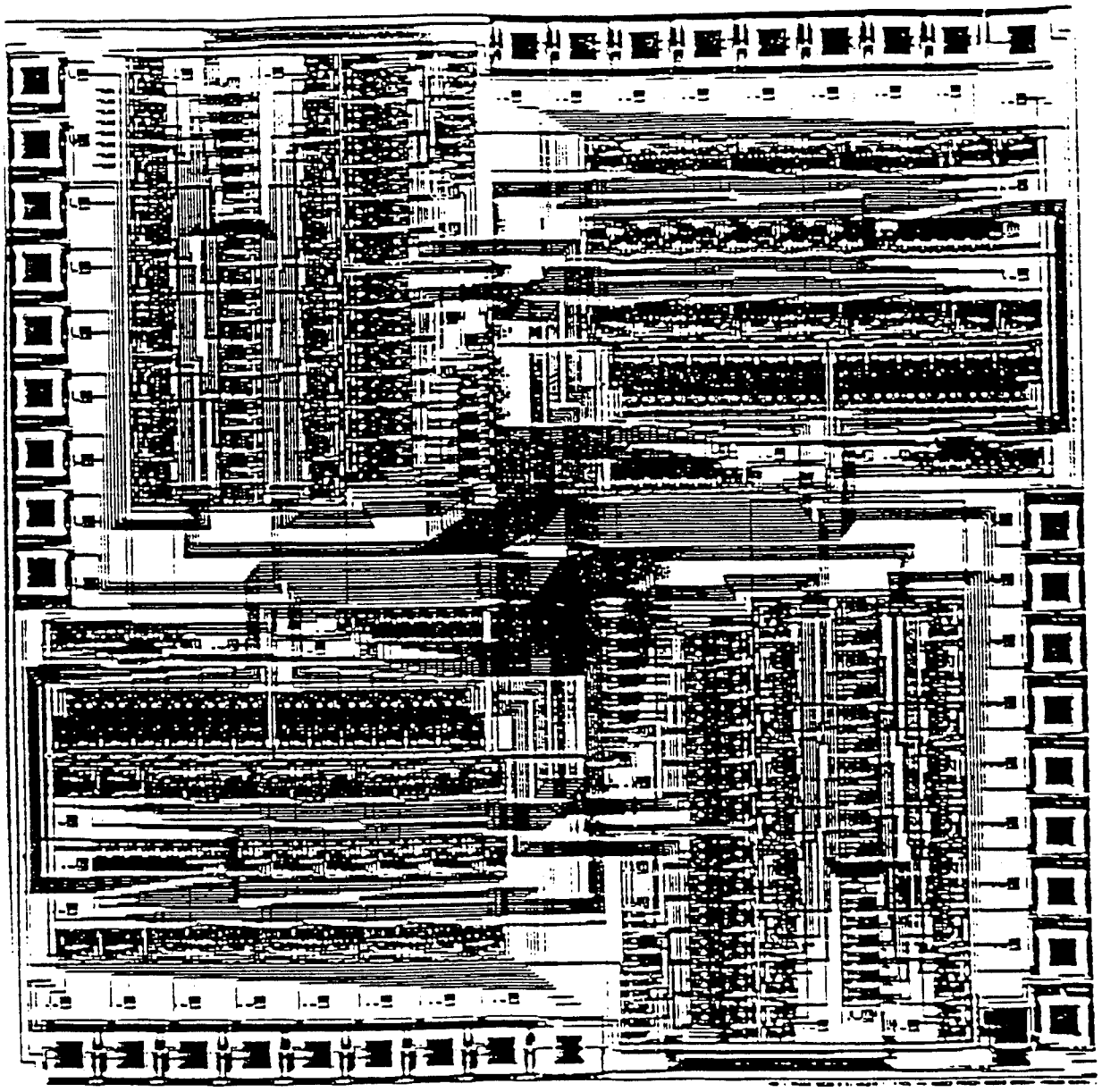

Figure 5: VLSI layout.

a) Papers Submitted to Refereed Journals (and not yet published)

1. V. P. Roychowdhury and T. Kailath, "Subspace Scheduling and Parallel Implementation of Non-Systolic Regular Iterative Algorithms," *Journal of VLSI Signal Processing*.
2. T. Varvarigou, V. P. Roychowdhury and T. Kailath, "Some New Algorithms for Reconfiguring VLSI/WSI Arrays," *IEEE Computer Society Conference*, Jan. 1990.
3. V. P. Roychowdhury, J. Bruck and T. Kailath, "Efficient Algorithms for Reconfiguration in VLSI/WSI Arrays," *IEEE Trans. Computers*.
4. V. P. Roychowdhury, L. Thiele, S. K.Rao and T. Kailath, "On the Localization of Algorithms for VLSI Processor Arrays," *IEEE Trans. Computers*.
5. V. P. Roychowdhury, P. G. Gulak, A. Montalvo and T. Kailath, "Decoding of Convolutional Codes in VLSI," *IEEE Trans. on Inform. Thy.*.
6. V. P. Roychowdhury and T. Kailath, "Regular Processor Arrays for Matrix Algorithms with Pivoting," *Communications of ACM*.

b) Contributed Presentations at Topical or Scientific/Technical Conferences

7. T. Kailath, "VLSI Array Processors for Communications, Control and Signal Processing," *TENCON 87*, pp. 96-98, Seoul, Korea, August 25-28, 1987.
8. P. G. Gulak, T. Kailath, A. Montalvo and V. P. Roychowdhury, "Decoding of Rate K/N Convolutional Codes in VLSI," *Dayton Conference on Systems Engineering*, Dayton, OH, September 1987.
9. V. P. Roychowdhury and T. Kailath, "Regular Processor Arrays for Matrix Algorithms with Pivoting," *Inter'l. Conference on Systolic Arrays*, San Diego, CA, May 1988.
10. V. P. Roychowdhury, P. G. Gulak, A. Montalvo, and T. Kailath, "Decoding of Rate *k/n* Convolutional Codes in VLSI," *1987 Princeton Workshop on Algorithms, Architectures and Technology Issues for Models of Concurrent Computation*, N.J., September 1987. Reprinted in *Concurrent Computations: Algorithms, Architecture and Technology*, Chapter 33, eds. S.K. Tewksbury, B. W. Dickinson and S. C. Schwartz, Plenum Press, N.Y., July 1988.
11. V. P. Roychowdhury, S. K. Rao, L. Thiele and T. Kailath, "On the Localization of Algorithms for VLSI Processor Arrays", 1988 IEEE Workshop on VLSI Signal Processing, pp. 459-470, Monterey, CA, November 1988.
12. T. Kailath and V. P. Roychowdhury, "Scheduling Linearly Indexed Assignment Code," *Proc. SPIE*, pp. 118-129, Los Angeles, CA, January 1989.

c) Books and sections thereof) Published:

1. V. P. Roychowdhury, P. G. Gulak, A. Montalvo, and T. Kailath, "Decoding of Rate *k/n* Convolutional Codes in VLSI," *1987 Princeton Workshop on*

Algorithms, Architectures and Technology Issues for Models of Concurrent Computation, N.J., September 1987. Reprinted in *Concurrent Computations: Algorithms, Architecture and Technology*, Chapter 33, eds. S.K. Tewksbury, B. W. Dickinson and S. C. Schwartz, Plenum Press, N.Y., July 1988.

d) Honors/Awards/Prizes

1. Engineering Achievement Award, National Federation of Asian Indian Organizations in America, 1986.
2. 1987 International Federation of Automatic Control Citation for Outstanding Contribution, J. M. Jover and T. Kailath, "A Parallel Architecture for Kalman Filter Measurement Update and Parameter Estimation," *Automatica*, Vol. 22, no. 1, pp. 43-57, 1986.
3. Hitachi America Professorship of Engineering, Jan. 1988.
4. 1988 Electrical Engineering Department Distinguished Service Award, June 1988.
5. Centennial Lecturer, American Math Society of Industrial and Applied Mathematics, July 1988.
6. 1989 Technical Achievement Award of the IEEE Acoustics, Speech and Signal Processing Society.
7. Royal Society Guest Research Fellowship, Imperial College of Science, Technology and Medicine, Department of Electrical Engineering, London, England, Summer 1989.

e) Special/Invited Conference Lectures

- Plenary Lecture, Symposium on the Fortieth Anniversary of the Joint Services Electronics Program, Washington, D.C., September 25, 1986.
- Plenary Lecture, Platinum Jubilee Conference on Systems and Signal Processing, Indian Institute of Science, India, December 11-13, 1986.
- Lecturer, IEEE Course on Array Signal Processing, Osmania University, Hyderabad, India, December 14-15, 1986.
- Plenary Lecture, White House Review of SDI Innovative Science and Technology Program, Washington, D.C., January 7, 1987.
- Lecturer, American Mathematical Society, 39th Annual Meeting, Short Course, *Moments in Mathematics*, San Antonio, TX, January 20-22, 1987.
- One-Day Presentation to Hughes Research Panel, "New Methods for Direction Finding and Spectral Algorithm -- ESPRIT," Los Angeles, CA, May 1, 1987.
- Plenary Lecture, IST/SDIO Workshop on Fundamental Issues in Communication, Computing and Signal Processing, MD, July 13-15, 1987.
- One-Day Presentation to JASON Group, "New Methods for Direction Finding," La Jolla, CA, July 16, 1987.

- Plenary Lecture, 1987 IEEE Region 10 Conference (TENCON) on Computers and Communications Technology Toward 2000, Seoul, Korea, August 25-26, 1987.
- Plenary Lecture, International Conference on Linear Algebra and Applications, Valencia, Spain, September 12-16, 1987.
- Keynote Lecture, 1988 Indo-US Workshop on Systems and Signal Processing, Bangalore, India, January 8-13, 1988.
- Plenary Lecture, 3rd SIAM Conference on Applied Linear Algebra, Madison, WI, May 23-27, 1988.
- Keynote Lecture, Sixth Army Conference on Applied Mathematics and Computing, Boulder, CO, May 31 - June 3, 1988.
- Plenary Lectures, Summer Program on Signal Processing, Institute of Math and Its Applications, Minneapolis, MN, June 1988.
- Lecturer, SDIO Innovative Science and Technology Annual Information Processing Symposia, Arlington, VA, June 1988.
- Centennial Lecture, American Math Society, Society of Industrial and Applied Math, Minneapolis, MN, July 14, 1988.
- Keynote Lecture, NATO Advanced Study Institute on Linear Algebra, Digital Signal Processing and Parallel Algorithms, Leuven, Belgium, August 1-12, 1988.
- Plenary Lecture, International Conference on Operator Theory: Advances and Applications, University of Calgary, Alberta, Canada, August 21-26, 1988.
- SDIO/IST Workshop on Sensor Signal Processing, Washington, D.C., April 24-27, 1989.
- Symposium on Applied Mathematics and Scientific Computing, Computer Science Department, Stanford University, April 21, 1989.
- School of Engineering High Noon, High Tech Lecture, April 28, 1989.
- Special Lecture Series, Imperial College of Science Technology and Medicine, Department of Electrical Engineering, London, England, Summer 1989.
- Plenary Lecture, International Symposium on the Mathematical Theory of Networks and Systems (MTNS-89), Amsterdam, The Netherlands, June 22, 1989.
- Keynote Lecture, International Symposium on Systems Engineering, Wright State University, Dayton, OH, August 23, 1989.

f) Conferences Attended

1) SPIE 30th Annual Technical Symposium, San Diego, CA, August 18-21, 1986.
2) NSF Workshop on Future Directions of Research in System Theory and Applications, Santa Clara, CA, September 18-20, 1986.
3) 1986 USC Workshop on VLSI and Signal Processing, Los Angeles, CA, November 5, 1986.
4) 20th Annual Asilomar Conference on Circuits and Systems, Monterey, CA, November 10-12, 1986.

5) Platinum Jubilee Conference on Systems and Signal Processing, Indian Institute of Science, India, December 11-13, 1986.

6) International Conference on Acoustics, Speech and Signal Processing, Dallas, TX, April 6-9, 1987.

7) 3rd Joint USSR - Swedish International Workshop on Information Theory, Sochi, USSR, May 22 - June 3, 1987.

8) Operator Theory Workshop, Phoenix, AR, June 10-12, 1987.

9) International Symposium on Mathematical Theory of Networks and Systems, Phoenix, AR, June 15-17, 1987.

10) IEEE Region 10 Conference on Computers and Communications Technology Toward 2000, Seoul, Korea, August 25-26, 1987.

11) International Conference on Linear Algebra and Applications, Valencia, Spain, September 1988.

12) 21st Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, November 1988.

13) 26th Conference on Decision and Control, Los Angeles, CA, December 1987.

14) Indo-US Workshop on Systems and Signal Processing, Bangalore, India, January 1988.

15) 1988 IEEE Communication Theory Workshop, Sedona, AZ, April 1988.

16) International Conference on Acoustics, Speech and Signal Processing, New York, April 1988.

17) SIAM Conference on Applied Linear Algebra, Madison, WI, May 1988.

18) Systolic Arrays Conference, San Diego, CA, May 1988.

19) Sixth Army Conference on Applied Mathematics and Computing, Boulder, CO, May 1988.

20) SIAM Annual Meeting, Minneapolis, MN, July 1988.

21) Institute of Math and Its Application, Summer Program on Signal Processing, Minneapolis, MN, June-July 1988.

22) SDIO Innovative Science and Technology Annual Information Processing Symposia, Arlington, VA, June 1988.

23) NATO Advanced Study Institute, Leuven, Belgium, August 1-7, 1988.

24) International Conference on Operator Theory: Advances and Applications, University of Calgary, Alberta, Canada, August 1988.

g) Researchers

Dr. A. Dembo (Research Associate, 1988-1989)
Dr. P. G. Gulak (Research Associate 1987)
Dr. H. Lev-Ari (Senior Research Associate, 1987-1988))
Dr. N. Weyland (Visiting Scholar, 1986-1987)
R. Ackner (Research Assistant, 1986-1987)
A. Montalvo (Research Assistant, 1986-1988)
K-Y. Siu (Research Assistant, 1988-1988)
V. Roychowdhury (Research Assistant/Postdoctoral Scholar, 1989)

h) Thesis

1. V. Roychowdhury, "Derivation, Extensions and Parallel Implementation of Regular Iterative Algorithms," Stanford University, Department of Electrical Engineering, December 1988.